Optimal Self-Healing of Service-Oriented Systems with Incomplete Information

Hongbing Wang, Xiaojun Wang
School of Computer Science and Engineering
Southeast University
Nanjing, China
Email: hbw@seu.edu.cn

Qi Yu
College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY, USA
Email: qi.yu@rit.edu

Abstract—Self-healing management of services aims to discover, diagnose, and react to disruptions as well as maintain the Quality of Service (QoS) at a desired level for a running service-oriented system. Existing approaches assume that the state of a running service-oriented system can be fully monitored. However, the dynamic nature of the Internet environment coupled with the opaque internal status of third-party services make such an assumption no longer hold. In this paper, we address the self-healing issue in service-oriented systems via a Partially Observed Markov Decision Process (POMDP). We determine the best action to minimizing the operation cost caused by the QoS failure of particular component services by computing the optimal value of the POMDP. By relying on such a flexible technology, we are able to deal with the difficulties arising from the unpredictability of external partner services and the opaqueness of their internal status. We conduct a simulation to assess the effectiveness of the proposed approach.

Keywords—Self-healing system; Quality of service; Markov decision process

I. INTRODUCTION

Web services provide self-describing and self-contained building blocks for rapid and low-cost development of service-oriented systems [11]. As a single Web service may not be sufficient at performing complex tasks, we usually need to combine multiple existing services together to meet customers’ complex requests. However, existing component services may need to be replaced and re-organized when the overall Quality of Service (QoS) of a running service-oriented system cannot meet the user requirements. As a result, self-healing and maintenance optimization in service-oriented systems have been gaining increasing attention given the highly dynamic Web environment as well as the autonomous and fast-changing nature of Web services. In particular, maintenance optimization is leveraged to identify the optimal self-healing strategy that determines what healing actions to perform and in what status of the system so that the cost of failing to meet the QoS requirements can be minimized while ensuring the proper functioning of the system.

There are several approaches that have been developed to achieve self-healing service-oriented systems. Some approaches implement self-healing after the detection of failures. When an error occurs, it stops currently running services[7][3], and repairs or replaces the malfunctioned ones. A major issue with these reactive approaches is that they cause long disruptions of currently running service systems, which in most cases will incur high revenue cost or risk to lose a large number of customers. To address this issue, Markov Decision Processes (MDPs) have been employed to achieve proactive self-healing strategies[12]. Despite effectively addressing the dynamics of the service-oriented systems and easy to implement and solve, MDPs assume full and precise knowledge on the state of component Web services. Unfortunately, such assumption no longer holds for most service-oriented systems as the system owner can only observe the communication with the third-party services and has no access to their internal status or variables. For instance, the system owner cannot know a priori the list of items available from an online sale service[14]. Even with some advanced inspection technologies that allow to monitor the internal state of third-party services, some QoS parameters, such as the response time and reliability are inherently uncertain and hence cannot be observed precisely. These parameters may be influenced by the unstable communication links or the performance of other services that the system depends on.

To address the central challenges as outlined above, we explicitly take into account the non-observability of component services’ internal states as well as the unpredictability of their QoS behaviors. We propose to exploit Partially Observed Markov Decision processes (POMDPs) to extend the standard MDPs, which allow us to develop self-healing strategies for service-oriented systems not just based the observed states of third-party services. In particular, POMDPs work with belief states that are represented by probability distributions over all states, including both observed and unobserved ones [8], [9]. We derive some key properties of the optimal value function used by a POMDP, which leads to the optimal maintenance and quality control policy that minimizes the expected total discounted system cost. By relying on such a flexible technology, we are able to deal with the difficulties that arise from the unpredictability of external partner services, the opaqueness of their internal status, and the measure errors of the service monitoring mechanisms.
The remainder of this paper is organized as follows. Section 2 reports some related work. Section 3 introduces a motivating scenario, which will be used throughout the paper. Section 4 gives a brief background overview on MDP and POMDP. Section 5 presents the self maintenance model where key decision variables are only partially observable. Section 6 shows the simulation study and reports the results. Section 7 concludes the paper.

II. RELATED WORK

In this section, we briefly describe several existing approaches that contribute to the self-healing issue in service-oriented systems.

A. Service Monitoring

In the context of autonomic computing, an autonomic manager is needed to monitor and collect information from the system. Ongoing research in the field of monitoring services in open-world scenarios has taken a number of diverse directions.

The work in [16] can be regarded as a requirement analysis approach to addressing the monitoring problem. Specifically, to address the problem of web service monitoring, it integrates requirement analysis and software execution monitoring. Requirement analysis helps derive goals and obstacles of monitoring, where the latter specify situations that prevent the system from reaching the former. Ghezzi et al. investigate how to monitor dynamic service compositions with respect to contracts expressed via assertions on services [1]. Dynamic compositions are represented as BPEL processes, which can be monitored at run-time to check whether individual services comply with their contracts. Monitors can be automatically defined as additional services and linked to the service composition. They present two alternative implementations for the monitoring approach: one based on late-binding and reflection and the other based on a standard assertion system. Baresi et al. propose Defensive Process Design (DPD) and Service run-time Monitoring (SrtM) [2]. DPD consists of designing the service-oriented business process so as to permit it to cope with erroneous behaviors. SrtM consists of using an external monitor service capable of checking whether functional and/or non-functional contracts are violated. Gurguis presents a mechanism to recover from failure based on XML-based correlating reported logs, which focuses on reactive self-healing mechanisms [7].

B. Service Failure Recovery

Casati et al. developed a system, called eFlow, to support specification, enactment and management of composite eServices [4]. In eFlow, service processes can adapt to environment changes with minimal or no user intervention. It is also possible to dynamically modify the process definition with user intervention. eFlow is enacted by a centralized process engine. However, eFlow does not have the rules for automatically switching services when failures occur. The reconfiguration (process definition modification) requires user intervention, which is not efficient since all process instances must be stopped and wait for the reconfiguration to complete. Finally, no QoS issues are considered in eFlow.

Yu et. al develop two algorithms to solve the service failures and the overloaded problem [18]. The first algorithm generates, for each service along the optimal path, a secondary path from current service to the end of the business process. During runtime, a business process component service can automatically switch to the secondary path whenever its successor becomes unavailable. The second algorithm uses a replacement path approach to reconstruct a new process by skipping a failed service.

C. Markov Decision Process Based Approaches

Eto et al. model a dynamic software system as a semi-Markov decision process and derive optimal software rejuvenation policies that minimize the expected cost when failures occur [6]. Dynamic programming is exploited to achieve the optimal policy. They also assume that the state of software system deteriorates stochastically. Sandhya et al. [12] present a proactive self-healing approach to decide when to replace a currently running service before the service fails. Their approach is based on the well-known Markov Decision Process (MDP) useful for studying a wide range of optimization problems via dynamic programming and reinforcement learning.

A partially observed Markov decision process (POMDP) has been used in [10] to model service adaptation and service re-selection as a sequential decision problem. In this approach, services that would bring the maximum expected long-term benefit would be selected and substituted into current instance during adaptation. POMDP is employed to take full advantage of preferential policies for selecting substitutable services, which has a completely different purpose as in our approach.

III. A MOTIVATING SCENARIO

We describe a motivating example in this section that will be used throughout the paper to illustrate the key challenges that may arise in service-oriented systems and the solutions we propose to cope with them.

Suppose that a client wants to make a travel plan through a service-oriented system (e.g., priceline.com) and typical services may include, flight query service, ticket reservation service, hotel booking service, and car rental service. We consider a simple scenario that just consists of two services: flight query and flight reservation. The flight reservation service accepts the order form, verify the user identity, prepare the bill, and finalize the purchase order request. Before making the final purchase order, the reservation service needs to invoke the flight query service to get the
flight information. In Fig. 1, we illustrate our motivating scenario graphically and we model the flight query service as a provider service and the flight reservation service as a consumer service.

As service-oriented applications are distributed systems, services are usually composed by means of brokers. Brokers help clients discover and select available candidate services based on their quality requirements. The simple scenario as given in Fig. 1 helps illustrate many typical faulty situations that may arise in a distributed application. For example, the flight query service may take a very long time to respond, thereby delay the overall performance. Assume that the required response time has an upper limit as 1 second. Hence, if the response time of light query service goes beyond 1 second, it causes a QoS failure. In this case, we may decide to replace the service when it does not meet required QoS criteria such as response time.

In a dynamic Internet environment, the quantity that represents the system state usually cannot be measured directly and accurately. The output of inspection on a certain service can vary for different Internet conditions. These factors can make the inspection unreliable and result in uncertainties of the output. Furthermore, uncertainties also exist in the interpretation of the output of inspection. So replacing the service at each QoS failure is not a beneficial action as it may not be a true failure caused by the service itself. Hence, a systematic approach is in need to decide when to go for replacement of potentially problematic services in a service-oriented system.

IV. PRELIMINARIES

In this section, we briefly describe the Markov decision process and partially observable Markov decision process that will lay out a technical background for the proposed approach.

A. Markov Decision Processes

A Markov Decision Process (MDP) [15] is a model for solving sequential stochastic decision problems. Specifically, an MDP is defined by a tuple: $\langle S, A, R, tr \rangle$, where

- $S$ is the set of the states of the world.
- $A$ is a set of actions an agent can use.
- $R : S \times A \rightarrow R$ is a reward function that for each state and action assigns a numeric reward (or cost, if the value is negative).
- $tr : S \times A \rightarrow \Delta(S)$ is a function that for each state and action associates a probability distribution over the possible successor($\Delta(S)$ denotes the set of all probability distributions over $S$) states. Thus, for each $s, s' \in S$ and $a \in A$ the function $tr$ determines the probability of a transition from state $s$ to state $s'$ after executing action $a$. i.e., $tr(s, a, s') = Pr(S_{n+1} = s'|S_n = s, A_n = a)$

A solution to an MDP is a policy $\pi : S \rightarrow A$ that defines which actions should be executed in each state.

Various exact and approximate algorithms exist for computing an optimal policy, and the best known are policy-iteration and value-iteration. Solving MDPs is known to be a polynomial problem in the number of states, and therefore exponential in the number of state variables. A value function assigns for each state a value $V(s)$—the expected utility from acting optimally beginning in $s \in S$ and on to infinity. Value iteration computes an optimal value function by iteratively solving the equation:

$$V_{n+1}(s) = \max_a \sum_{s'} tr(s, a, s')V_n(s')$$

for all $s$ and $s' \in S$, $a \in A$ and $V_n(s)$ is the $n$ step iterative value of state $s$.

B. Partially Observable Markov Decision Processes

A well known extension to the MDP model is the Partially Observable Markov Decision Process (POMDP) model [5]. A POMDP is a tuple $\langle S, A, R, tr, \Omega, O \rangle$, where

- $\langle S, A, R, tr \rangle$ defines an MDP.
- $\Omega$ is a set of possible observations.
- $\Omega(a, s, o)$ is the probability of executing action $a$, reaching state $s$ and observing $o$.

The observation can only partially capture the true state of the core process. There is a probabilistic relationship between the true state of $X_t$ the core process and the observation $Y_t$ at time $t$. Specifically, the probabilistic relationship is
denoted by \( \Omega(a_{t-1}) = [\Omega_{ij}(a_{t-1})] \), referred to the relationship matrix, where \( \Omega_{ij}(a_{t-1}) = Pr[Y_t = j|X_t = i, a_{t-1}] \). The transition matrix of the core process in \((t, t + 1)\) under control action \( a_t \in A \), is denoted by \( tr(a_t) = [tr_{ij}(a_t)] \), where \( tr_{ij}(a_t) = Pr[X_{t+1} = j|X_t = i, a_t] \). The initial probability distribution of the core states is expressed as the initial information vector \( q(0) = (q_1(0), q_2(0), ..., q_N(0)) \), where \( q_i(0) = Pr[X_0 = i] \). The terminal cost vector, \( C_0 = (C^{(1)}_0, C^{(2)}_0, ..., C^{(N)}_0) \) where \( C^{(0)}_0 \) is the cost incurred if the core state of the system is \( i \) and the system is terminated.

The evolution of the controlled process can then be described as follows. Assume that at time \( t \) the information vector \( q(t) = (q_1(t), q_2(t), ..., q_N(t)) \) is known to the decision maker, where \( q_i(t) = Pr[X_t = i|d_t] \), and \( d_t = (q_0, a_0, y_1, a_1, ..., y_t) \), being the history to \( t \) of the evolution process. The decision maker decides to take control action \( a_t = k \) on the system according to the information vector and his/her decision criteria. Then, the cost \( q(t) \cdot C_k \) will be incurred, where \( C_k = (C_{kk}, C_{ck}, ..., C_{nk}) \), and \( C_{ik} \) is the immediate cost incurred by action \( k \) if the core process is in state \( i \). The system will then evolve according to \( tr(k) \) in \((t, t + 1)\). The probability distributions of \( X_{t+1} \) and \( Y_{t+1} \) can be predicted based on \( q(t), tr(k) \) and \( \Omega(k) \).

Let \( p_k(t + 1) = (p_{k1}(t + 1), p_{k2}(t + 1), ..., p_{kN}(t + 1)) \), where \( p_{ki}(t + 1) = Pr[X_{t+1} = i|d_t, a_t = k] = \sum_{j \in S} q_j(t) tr_{ji}(k) \) (2)

and let \( \gamma(j|d_t, k) = Pr[Y_{t+1} = j|d_t, a_t = k] \), then

\[
\gamma(j|d_t, k) = \sum_{i \in S} \left[ \sum_{l \in S} q_l(t) tr_{li}(k) \right] \Omega_{ij}(k)
\]

(3)

where \( S \) is the state space of the structure.

At time \( t + 1 \), the decision maker gets information about the status of the system through observation on the system, say, \( Y_{t+1} = j \). This information is used to update its knowledge about the state of the core process, \( p_k(t + 1) \), using Bayesian principles. An updated information vector \( q^k_j(t + 1) \equiv (q^k_{j1}(t + 1), q^k_{j2}(t + 1), ..., q^k_{jN}(t + 1)) \) and \( q^k_{ji}(t + 1) = Pr[X_{t+1} = i|d_t, a_t = k, y_{t+1} = j] \). In matrix form

\[
q^k_j(t + 1) = \frac{q(t) tr(k) \Omega_{ij}(k)}{\gamma(j|d_t, k)}
\]

(4)

The decision maker selects the control decision at time \( t + 1 \) according to the updated information vector \( q^k_j(t + 1) \) and his/her decision criteria. This procedure is repeated until the end of the lifetime of the system.

It can be seen that the observation process \( \{Y_t\} \) is also a stochastic process. It generally is not a Markov process, even though \( \{X_t\} \) itself is a Markov chain, because the future observation depends on not only the current observation \( Y_t \), but also the complete history \( d_t \). Thus, neither \( X_t \) nor \( Y_t \) can be used directly as state variables to form a recursive function as required in the dynamic programming solution technique because the true value of \( X_t \) is not known, and \( Y_t \) does not have Markov property. A sufficient statistic is therefore needed, which should summarize all of the information necessary for decision making, possess the Markov property, and is completely "observable".

The above discussion shows that \( q(t) \) is such a statistic. First, the sequence of information vectors \( \{q(t)\} \) is a Markov process for any fixed sequence of actions. Second, it is completely "observable". Therefore, the original POMDP problem can be converted to an equivalent MDP problem.

The significant difference between this equivalent MDP problems and standard MDP problems is that the latter has a finite discrete state space while the former possesses an infinite continuous state space. This difference causes difficulties in the computation of POMDP problems.

Using the notation defined previously, the recursive optimal value function of the POMDP problem can be written as

\[
V_n(q(t)) = \min_{k \in A} \left[ q(t) C_k + \sum_{j \in M} \gamma(j|d_t, k) V_{n-1}(q^k_j(t + 1)) \right]
\]

(5)

where \( V_n(q(t)) \) is the minimum expected total cost incurred in the remaining \( n \neq 0 \) time periods beginning from time \( t \), given that at time \( t \) the knowledge about the states of the system is the information vector \( q(t) \), and \( M \) is the state space of outcomes that can result from the observation process, \( Y_t \). For \( n = 0 \), \( V_0 = \langle q(t) \rangle C_0 \), where \( C_0 \) is the terminal cost vector of the system.

V. OPTIMAL MAINTENANCE MODEL

We present the optimal maintenance model based on partially observable Markov decision process in this section. We consider the maintenance problem in self-healing web service environment, where the consumer service is assigned with the task of selecting a provider service. In order to make this decision, the consumer service maintains a belief over the quality levels of the various provider services. Only when the consumer is sufficiently sure that it has identified a provider with sufficient quality, should it go ahead and use the provider service. So the problem includes the decision of whether to replace an existing provider service. The reputation of provider services can aid decision making of the consumer service, which will be explained next.

A. Reputation of Providers

Reputation is a rich concept with many dimensions. For our purpose, we restrict the definition of the reputation of a provider as a measure of how well the provider fulfills the role of providing the required QoS. The extent to which a provider service fulfills this role is defined by how the
consumer’s expectations are met and these expectations are captured by a utility function. The consumer service has an expected QoS, which it demands when invoking with a provider. After a service is invoked, the consumer compares the expected QoS with the QoS actually received. If the received QoS surpasses the expected QoS, the consumer is satisfied and unsatisfied if otherwise. A provider that consistently fulfills its role and satisfies the consumer will attain a high reputation.

B. POMDP-based Maintenance Model

The QoS of a web service may change with time due to dynamic Internet environment. The state of a service-oriented system has to be inspected and repaired if necessary from time to time to maintain the desired QoS level. If some services consistently fail to meet the desired QoS criteria, they have to be replaced to ensure the performance of the entire system. But replacing an existing service with a new one typically incur higher cost that may disrupt the operation of the system. Hence, we aim to choose an optimal maintenance strategy.

The state evolution a service-oriented system is inherently a stochastic process as operation environment of component web services is highly dynamic. Hence, we cannot fully understand all the changes in the environment in a timely fashion. Therefore, we propose to model the maintenance decision making problem as a partially observable Markov decision process (POMDP), which places a belief distribution over the possible states and uses observations to adjust this belief. Instead of knowing exactly what the current state is, POMDP has a belief about the current state, represented by the function that assigns a probability to every possible state. In our model, the state of a service represents the actual reputation of the service and is only partially observable through some inspection approach. In this regard, an adaptation broker should use the partially observable environment states, historical adaptation processes, and the potential benefit to make current adaptation decisions.

The way how different components of the POMDP interact is illustrated in Fig. 2.

The agent is unable to identify the current state and is therefore forced to estimate the current state given the current observations and then executing action receiving a reward, reaching the next state. The POMDP based model is defined as $< S, A, R, T, \Omega, O >$, where

- $S$ denotes the QoS state of a Web service.
- $A$ denotes the maintenance actions.
- $R$ denotes the revenue function.
- $T$ denotes the state transition function.
- $\Omega$ and $O$ denote the observable information and function, respectively.

In what follows, we describe each of the key components of the model in detail.

**a) Web Service State:** Our model aims to decide whether continuing to use an existing provider service or replacing it based on the expected incurred loss. The consumer service keeps track of the QoS intimation of QoS failure by the provider if any. Hence, our state space is based on the loss incurred to the consumer. As a result we define the state of the consumer as follows. Let $R(t)$ be the QoS of the provider service at time $t$, which is a random variable because of the randomness of the running processes and uncertainty of maintenance effects. The state of a component service at time $t$ depends not only on its QoS at that time, but also the required QoS criterion $S(t)$, which is typically a constant. If the $R(t)$ is superior to $S(t)$, then we say that the provider service is in the state of good otherwise in bad state. It is in the failed state if the service failed. In real engineering applications, we may use a more precise classification e.g. very bad, bad, good, and very good.

**b) Maintenance Actions:** The erroneous behaviors of Web services are mostly unforeseeable at design-time. For this reason, the solutions presented here rely heavily on special-purpose recovery actions. Service run-time monitoring consists of checking whether functional and/or non-functional requirements are violated. Once a fault is notified, several different kinds of actions can be triggered in the attempt to reach the goals of the process, regardless of the errors. We identify three maintenance actions to tackle typical faulty behaviors that can arise in service-oriented system:

- **Continue:** This action simply keeps using the provider service regardless the violation of the QoS requirement.
- **Retry:** is to re-invoke a faulty external service is retried to verify whether the fault is transient.
**Rebind:** is to dynamically bind to another service. A dynamic binding occurs, which replaces an existing faulty service with a better one to guarantee the same functional and/or non-functional properties.

The resultant effect of a maintenance action can be described by the probability that the action upgrades the provider service from state $i$ before the maintenance to state $j$ after the action. Let $a_t$ represent the maintenance action taken at time $t$, $a_t = k \in A$, where $A$ is the maintenance action space, and $p_{i,j}(t, k)$ be the probability that action $k$ change the provider service from state $i$ at time $t$ to state $j$, then we define the effect of a maintenance action:

$$p_{i,j}(t, k) = Pr[X_{t+1} = j|X_t = i, a_t = k] \quad (6)$$

c) **Revenue Function:** $R(s, a, s')$ is the revenue function, which may take a negative value for a negative reward. In our model, it is always negative because for each action, a cost is paid regardless of the state. For example, we assign -1 cost for the continue action, -5 for retry action, and -10 for rebind action. However, for transitions to a satisfied state (i.e., from bad to good) no penalty is applied whereas transitions to an unsatisfied state yield a large penalty.

d) **State Transition Function:** $T(s, a, s')$ is the state transition function, which captures the probability that the system will enter the state $s'$ from $s$ under the action $a$, denoted by $Pr(s'|s, a)$. For example, when the provider service is in bad state, and we use the rebind action to switch to another service, which leads the system to a good state with probability 0.9. We can summarize it as $Pr(\text{good}|\text{bad}, \text{rebind}) = 0.9$.

e) **Observable Information and Function:** $O$ is the set of observable information received by the monitor, which denotes the QoS information of the provider service. $\Omega(s', a, a)$ is the observation function which is used to compute the possible observed value at the state $s'$ after adopting action $a$ presented by $Pr(o|s', a)$.

To achieve accurate evaluation on the condition of a service and make satisfactory maintenance decisions, the prior knowledge must be updated by incorporating new information, which better reflects the actual status the service. When monitoring a service, the belief $b$ of the service is maintained, which calculates the probability distribution over states via Bayes’ rule. That is, when $b(s)$ specifies the probability of $s$ (for all $s$), we can derive an updated belief $b'$ after taking some action $a$ and receiving an observation $o$ with probability $\Omega(o|s', a)$. Assuming discrete sets of states and observations, this update can be formulated as:

$$b'(s') = \eta \Omega(o|s', a) \Sigma_{s \in S} tr(s'|s, a)b(s) \quad (7)$$

where $\eta = 1/Pr(o|b, a)$ is a normalizing constant with $Pr(o|b, a) = \Sigma_{s' \in S} \Omega(o|s', a) \Sigma_{s \in S} tr(s'|s, a)b(s)$.

These beliefs are the basis for decision making because a policy $\pi$ maps beliefs to actions: $\pi(b) = a$.

We can specify the observation function to account for how the monitored QoS information changes our belief about the actual reputations of a provider service. Furthermore, we can specify the transition function that maps actions in prior states to next state. Using both the transition and observation functions, the POMDP provides a principled framework to find the best action based upon the current belief.

C. **Computing the Solution to the POMDP Model**

A POMDP provides a principled and expressive framework for self-healing modeling. However calculating optimal policies for a POMDP is nontrivial. An exact solution can be computed using the belief state MDP, which is an MDP over the belief space of the POMDP. The policy maps a belief state space into the action space. The optimal policy can be understood as the solution of a continuous space Markov Decision Process[8]. It is defined as a tuple $B, A, \tau, r$ where

- $B$: is the set of belief states over the POMDP states.
- $A$: is the same finite set of action as for the original POMDP.
- $\tau$: is the belief state transition function,
- $r$: $B \times A \rightarrow R$ is the reward function on belief states.

It writes : $r(b, a) = \Sigma_{s \in S} b(s) R(s, a)$

Therefore, value iteration which is a standard method of finding the optimal policy in MDP can also be used to calculate optimal infinite horizon POMDP policies. The difference between the optimal value function and the optimal $t$-horizon value function goes to zero as $t$ goes to infinity:

$$\lim_{t \rightarrow \infty} \max_{b \in B} |V^*(b) - V^*_t(b)| = 0 \quad (8)$$

It turns out that the optimal value function can be calculated in a finite number of steps given the Bellman error $\epsilon$, which is the maximum difference between two successive finite horizon value functions. The value iteration algorithm for POMDPs can be summarized in Algorithm 1.

**Algorithm 1 The Value Iteration for Solving POMDP**

Initialize $t=0$;

while $\max_{b \in B} |V_{t+1}(b) - V_t(b)| > \epsilon$ do

calculate $V_{t+1}(b)$ for all states $b \in B$:

$$V_{t+1}(b) = \max_{a \in A} \left[ \Sigma_{b' \in B} r(b, a) + \gamma \Sigma_{b' \in B} V_t(b') \right]$$

increment $t$

end while

The previous equation can be rewritten in terms of the original POMDP formulation as

$$V_{t+1}(b) = \max_{a \in A} \left[ \Sigma_{s \in S} b(s) R(s, a) + \gamma \Sigma_{o \in O} Pr(o|a, b) V_t(b') \right] \quad (9)$$
where

\[ Pr(o|a, b) = \sum_{s \in S} \Omega(s, a, o) \sum_{s' \in S} T(s, a, s') b(s). \]  \hfill (10)

Note that by capturing the action a that maximizes the right-hand-side of Bellman’s equation, we indirectly obtain a policy. While Bellman’s equation gives us a dynamic programming algorithm that can be used in theory to find optimal policies, in practice, the continuous belief space is problematic since it is not possible to apply Bellman’s equation to an infinite number of belief points. Fortunately, as pointed out by Smallwood and Sondik [17], optimal value functions are piecewise-linear and convex, which can be exploited to apply Bellman’s equation in a finite number of times (once per linear piece of the value function). As an alternative, approximate solutions can be efficiently computed using the point-based value iteration algorithm [13].

VI. EVALUATION

As there is no publicly available real-world datasets that can be used for evaluation purposes, we conduct a simulation study to assess the effectiveness of the proposed approach. All the experiments are implemented in Java and carried out on a Windows 2003 Server, with an Intel Xeon E7320 2.13GHz CPU and 8GB RAM.

Let us consider the example discussed in section 3. Assume that the performance of the consumer service is described using four states: (1) good, (2) medium, (3) bad, and (4) failed. The self-evolution process is modeled by a stationary Markov chain with state space \( S = \{1, 2, 3, 4\} \). The transition matrix is given by

\[
P_s = \begin{pmatrix}
0.8 & 0.1 & 0.07 & 0.03 \\
0.15 & 0.75 & 0.1 & 0.05 \\
0.05 & 0.1 & 0.7 & 0.15 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]  \hfill (11)

A monitoring service is used to measure the QoS of the provider service and evaluate which state the system is in. Therefore, the monitor service may yield the same number of outcomes as that of the system states, but errors exist in the measurement results. The relationship matrices is

\[
R = \begin{pmatrix}
0.85 & 0.1 & 0.05 & 0 \\
0.05 & 0.8 & 0.1 & 0.05 \\
0.0 & 0.1 & 0.85 & 0.05 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]  \hfill (12)

The row and column of matrices \( R \) correspond to the true state of the process and the outcome of monitor, and \( R_{ij} \) is the probability that the monitor outcome will be \( j \) if the true state is \( i \). For example, \( R_{23} = 0.1 \) means that the monitor outcome could be “system being in state 3” with probability 0.1 if the true state of the system is 2.

The maintenance actions available include: continue, retry, and rebind. The joint transition matrices corresponding to different actions are as follows

\[
T(continue) = \begin{pmatrix}
0.8 & 0.1 & 0.07 & 0.03 \\
0.1 & 0.75 & 0.1 & 0.05 \\
0.05 & 0.1 & 0.7 & 0.15 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]  \hfill (13)

\[
T(retry) = \begin{pmatrix}
0.8 & 0.15 & 0.05 & 0.05 \\
0.1 & 0.8 & 0.1 & 0.0 \\
0.15 & 0.7 & 0.05 & 0 \\
0.0 & 0.0 & 0.0 & 1
\end{pmatrix}
\]  \hfill (14)

\[
T(rebind) = \begin{pmatrix}
0.9 & 0.1 & 0.0 & 0.0 \\
0.9 & 0.1 & 0.0 & 0.0 \\
0.9 & 0.1 & 0.0 & 0.0 \\
0.9 & 0.1 & 0.0 & 0.0
\end{pmatrix}
\]  \hfill (15)

The maintenance cost matrix is defined as:

\[
C = \begin{pmatrix}
0 & 5 & 20 \\
0 & 5 & 20 \\
5 & 10 & 30 \\
40 & 60 & 80
\end{pmatrix}
\]  \hfill (16)

This problem is modeled as a POMDP model and the optimal maintenance policy can be found through the discussed approach.

Figure 3 shows the changed average cost per action execution over time. It can be seen that the model works well in the partial observable environment. The average cost does not fluctuate drastically. It can always avoid entering the failed state. The time complexity of finding a best strategy based on this model is an exponential function and will increase with the number of steps, \( t \). So, if the state increases, the computing time cannot be ignored, and the approximation technique need to be adopted.
VII. CONCLUSIONS

In this paper, we present a Partially Observed Markov Decision processes (POMDP) based model to tackle the non-observability of component services internal states and the unpredictability of their QoS behaviors. The proposed model works with belief states that are represented by probability distributions over all states via Bayes rule. In this way, the model helps determine the best maintenance policy which defines what actions to perform at what particular state of the system. By relying on such a flexible technology, we are able to deal with the difficulties stemming from the unpredictability of external partner services, the opaqueness of their internal status, and the measure errors of the monitor.

ACKNOWLEDGMENT

This work is partially supported by NSFC(61232007) and JSNSF of China (No.BK2010417).

REFERENCES


