Towards adaptive policy-based management

Conference Paper · January 2010
DOI: 10.1109/NOMS.2010.5488472 · Source: DBLP

2 authors:

Raphael M. Bahati
Lawson Health Research Institute
20 PUBLICATIONS **127** CITATIONS
SEE PROFILE

Michael A. Bauer
The University of Western Ontario
263 PUBLICATIONS **1,484** CITATIONS
SEE PROFILE

Towards Adaptive Policy-based Management

Raphael M. Bahati and Michael A. Bauer
Department of Computer Science
The University of Western Ontario, London, ON N6A 5B7, CANADA
Email: {rbahati;bauer}@csd.uwo.ca

***Abstract*—The use of learning, in particular reinforcement learning, has been explored in the context of policy-driven autonomic management as a means of aiding decision making. In this context, the autonomic manager "learns" a model about what actions to take, for example, in certain situations. However, when the set of policies changes, the model is typically discarded or, if used, may yield misleading information. In contrast, this paper presents an approach for "re-using" past knowledge - by transforming a model learned from the use of one set of active policies to a new model when those policies change. This means that some of the "learned" knowledge can be utilized within the new environment. This is possible because our approach to modeling learning and adaptation is dependent only on the structure of the policies. Consequently, changes to policies can be mapped onto transformations specific to the model derived from the use of those policies. In this paper, we describe the model construction and policy modifications and elaborate, with a detailed case study, on how such changes could alter the currently learned model. Our analysis of the different kinds of policy modifications also suggest that, in most cases, most of the learned model can still be reused. This can significantly accelerate the learning process, essentially improving the overall quality of service, as the results presented in this paper demonstrate.**

I. Introduction

The use of Reinforcement Learning [1] in performance management has recently attracted significant interest within autonomic computing (see, for example, [2], [3], [4], [5]). We have also explored how reinforcement learning could be used to guide management decisions in the context of adaptive policy-driven autonomic management [6], [7]. In our approach, the model of the environment dynamics is derived from the autonomic manager's use of an active set of policies. This model, in turn, provides guidance to the autonomic system on how to effectively use the active policies. This enables the system to learn from past experience, predict possible future actions, and make appropriate trade-offs when selecting actions for resolving quality of service violations. The use of policies in this context offers significant benefits to autonomic systems in that it allows system administrators to focus on the specification of the objectives leaving it to systems to plan how to achieve them. In our approach, these objectives are captured via *expectation policies* (or *obligation policies* [8]) which are then used to derive the structure of the model. Moreover, since our strategies are dependent only on the structure of the policies, the heuristics for learning can thus be utilized in a variety of domains. This is essential in developing flexible, adaptive, and portable autonomic management solutions.

A key benefit in using policies within autonomic management is the flexibility with which systems behavior can be modified; i.e., through policy manipulations rather than through re-engineering. Policy modifications, however, bring their own set of challenges for autonomic systems. For example, a major drawback in many approaches utilizing some form of learning is that when the environment, in this case the active set of policies, change, the existing model is often discarded and the system must learn a new model from scratch. Our experience with policies, however, suggests that policy modifications are frequently "incremental" - e.g., the change of a threshold, the disabling of a policy, etc. The ability to reuse a learned model, or part there of, has the potential benefit of significantly accelerating the learning process.

To this end, we have developed mechanisms enabling policy-driven autonomic systems to adapt the model learned from the use of one set of policies to a related model when that set of policies change [9]. Our experience with the system suggests that there are many circumstances where changes to the policies have little impact on the model and, in particular, the model itself can be adapted. In contrast, there are relatively few circumstances where the impact of the changes would mean discarding the entire model and restarting the learning. In this paper, we define what constitutes a policy modification and elaborate, with a detailed case study, on how such changes are mapped to specific model transformations. We also present the results of experiments with a prototype system, demonstrating the performance benefits specific to the adaptation mechanisms in terms of the improvement in the overall quality of service of a multi-tiered Web server.

The rest of the paper is organized as follows. Section II describes our strategy for modeling reinforcement learning. We then describe our approach for adapting the learned model to run-time policy modifications in Section III and present the results of our prototype implementation in Section IV. We conclude with a discussion in Section V.

II. Modelling Reinforcement Learning

This section provides the key definitions of terms and concepts related to how we model learning; a more detailed and formal description can be found in [7]. Our work focuses on expectation (obligation) policies of the form "if [conditions] then [actions]", which are modeled as follows:

Definition 1: An *expectation policy* p is a pair $\langle C, A\rangle$ where C is a finite set of *conditions*, $C = \{c_1, \ldots, c_m\}$, and $A = \{a_1, \ldots, a_k\}$ is an ordered set of *actions*. Each *condition*, $c_i \in C$, is defined by a tuple $c_i = \langle \texttt{metricName}, \texttt{operator}, \Gamma\rangle$,

978-1-4244-5367-2/10/$26.00

where `metricName` is the name of a metric measured/monitored by the management system, `operator` is a relational operator, and Γ is a constant indicating a threshold value.

The set of active policies at any time within the management system is then modeled as a policy system:

Definition 2: A *policy system* $PS = \langle P, W_C \rangle$, where $P = \{p_1, ..., p_n\}$ is the subset of expectation policies that are active and W_C[1] associates a metric weight with each unique condition occurring in any policy of P.

As such, a *policy system* is the set of active policies where each condition occurring in some policy has an associated weight. These definitions provide the fundamental structure that is used to build our reinforcement learning model.

A. System States

To model the dynamics of the environment from the use of an active set of policies, we define a set of states whose structure is derived from the metrics associated with the enabled expectation policies (see, for example, Fig. 3).

Definition 3: A policy system $PS = \langle P, W_C \rangle$ derives a set of *system metrics*, $M = \{m_1, \ldots, m_t\}$, such that, for each $p_j = \langle C_j, A_j \rangle \in P$, $M = \bigcup_{c_i \in C_j} \{c_i.\texttt{metricName}\}$.

The set M is the set of all metrics occurring in any of the active policies. For each metric in this set, there are a finite number of threshold values to which the metric is compared; these can be ordered to form "regions".

Definition 4: Let $PS = \langle P, W_C \rangle$ be policy system with metrics set M. For each metric $m_i \in M$, let $\sigma_{m_i} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_k\}$ be the set of thresholds from the conditions associated with metric m_i, such that, $\Gamma_i < \Gamma_j$ if $i < j$. Then σ_{m_i} induces a set of *metric regions* associated with metric m_i: $R_{m_i} = \{R^1_{m_i}, R^2_{m_i}, \ldots, R^{k+1}_{m_i}\}$, where $R^1_{m_i} = (-\infty, \Gamma_1)$, $R^2_{m_i} = (\Gamma_1, \Gamma_2)$, etc., and, $R^{k+1}_{m_i} = (\Gamma_k, \infty)$.

In essence, for any metric, a measured value of the metric can be mapped uniquely onto one of the regions (i.e., intervals) as defined by the thresholds of the policy conditions. For example, if $\sigma_{m_i} = \langle \Gamma_1, \Gamma_2 \rangle$, then there would be three regions: $R^1_{m_i} = (-\infty, \Gamma_1)$, $R^2_{m_i} = (\Gamma_1, \Gamma_2)$, and $R^3_{m_i} = (\Gamma_2, \infty)$. This brings us to our next definition.

Definition 5: For each metric $m_i \in M$, and its set of metric regions, $R_{m_i} = \{R^1_{m_i}, \ldots, R^n_{m_i}\}$, we define a *weighting function*, $f(R^j_{m_i}) \rightarrow \mathbb{R}$, which assigns a numeric value to the j-th region in R_{m_i}, such that, $f(R^k_{m_i}) > f(R^l_{m_i})$ if $k < l$.
An example of such a mapping, which we make use of in our current implementation, is defined by Equation 1:

$$f(R^j_{m_i}) \;=\; 100 - (\frac{100}{n-1})(j-1) \qquad (1)$$

where n is the total number of regions in R_{m_i}. This function assigns a numeric value between 100 and 0 for each metric's region in R_{m_i}, starting from 100 for the most "desirable region" and decrementing at equal intervals towards the opposite end of the spectrum, whose region is assigned a value of 0. Here, we assume that smaller metric values are "more desirable", though in general this is not a necessary requirement of the weighting function. The idea is that regions of greater "desirability", i.e., preferred quality of service, are assigned higher values.

The key role of these regions is that they partition the space of values that a metric can take on with respect to the thresholds in conditions involving that metric. We use these to define a state within our model.

Definition 6: A policy system $PS = \langle P, W_C \rangle$ with metrics M derives a set of system states S, where $s_i \in S$ is defined by a tuple $s_i = \langle \mu, M(s_i), P(s_i), A(s_i) \rangle$, and where:

- μ is a `type` which classifies a state as either "violation" or "non-violation" depending, respectively, on whether or not there are any policies violated when visiting a particular state.
- $P(s_i)$ is a set of expectation policies that were violated when the system was in state s_i.
- $A(s_i)$ is a set of unique actions advocated by the expectation policies in $P(s_i)$, plus γ-action (i.e., a_0) which is a "null" or "no-op" action.
- $M(s_i)$ is the set $\{\langle \texttt{value}_1, R^{l_1}_{m_1} \rangle, \ldots, \langle \texttt{value}_n, R^{l_n}_{m_n} \rangle\}$, where $\texttt{value}_j$ is the observed measurement of metric m_j or its average value when state s_i is visited multiple times and $R^{l_j}_{m_j}$ is the region corresponding to a region in σ_{m_j} in which $\texttt{value}_j$ falls. That is, if $R^l_{m_j} = (\Gamma_1, \Gamma_2)$, then $\Gamma_1 < \texttt{value}_j < \Gamma_2$. For each such region, $f(R^l_{m_j})$ then associates a value as described by Equation 1.

Essentially, each state has a unique region for each metric based on the `value` associated with each state metric. That is, for a set of policies involving n metrics, each state would have n metrics $\{m_1, m_2, ..., m_n\}$ and, for each of those metrics, there would be a single metric region (see definition 4) associated with the metric.

B. System Transitions

Transitions are essentially determined by the actions taken by the management system and labelled by a value determined by the learning algorithm.

Definition 7: A state transition, $t_i(s_p, a_p, s_c)$, is a directed edge corresponding to a transition originating in state s_p and ending at state s_c as a result of taking action a_p while in state s_p. It is labeled by $\langle \lambda, Q_{t_i}(s_p, a_p) \rangle$ where λ is the frequency (i.e., the number of times) the transition has occurred and $Q_{t_i}(s_p, a_p)$ is the action-value estimate associated with taking action a_p in state s_p. In our current implementation, this value is computed using a one-step Q-Learning [1] algorithm, which has been described elsewhere [7].

C. A State-Transition Model

A state-transition model is then defined for a set of active expectation policies:

Definition 8: A *state-transition model* derived from the policy system $PS = \langle P, W_C \rangle$ is defined by the graph $G^P = \langle S, T \rangle$ where S is a set of system states, and T is a set of state transitions.

[1]We do not rely on W_C in this paper, but include it for completeness and consistency with some of our earlier work where it does play a role.

m_i	c_k	Policy Condition	R_{m_i}	$R^j_{m_i}$	$f(R^j_{m_i})$
m_1	c_3	APACHE:responseTime < 250.0	$m_1.value < 250.0$	$R^1_{m_1}$	100
			$250.0 \leq m_1.value \leq 2000.0$	$R^2_{m_1}$	50
	c_1	APACHE:responseTime > 2000.0	$m_1.value > 2000.0$	$R^3_{m_1}$	0
m_2			$m_2.value \leq 0.0$	$R^1_{m_2}$	100
	c_2	APACHE:responseTimeTREND > 0.0	$m_2.value > 0.0$	$R^2_{m_2}$	0

Fig. 1. A metrics structure derived from the policy system of Fig. 3.

State	$R^k_{m_j}$		$f(R^k_{m_j})$		$A(s_i)$		$t(s_i, a_l, s_*)$	
s_i	$R^k_{m_1}$	$R^k_{m_2}$	$f(R^k_{m_1})$	$f(R^k_{m_2})$	a_l	State action	λ	s_*
s_1	$R^3_{m_1}$	$R^2_{m_2}$	0	0	a_0	γ-action	1	s_1
					a_1	AdjustMaxClients(+25)	3	s_4
					a_2	AdjustMaxKeepAliveRequests(-30)	8	s_2
					a_3	AdjustMaxBandwidth(-128)		
s_2	$R^3_{m_1}$	$R^1_{m_2}$	0	100	a_0	γ-action	3	s_6
s_3	$R^2_{m_1}$	$R^2_{m_2}$	50	0	a_0	γ-action	1;1	s_3;s_6
s_4	$R^2_{m_1}$	$R^1_{m_2}$	50	100	a_0	γ-action	3	s_3
s_5	$R^1_{m_1}$	$R^2_{m_2}$	100	0	a_0	γ-action	2	s_5
					a_4	AdjustMaxClients(-25)	7	s_6
					a_5	AdjustMaxKeepAliveRequests(+30)		
					a_6	AdjustMaxBandwidth(+64)		
s_6	$R^1_{m_1}$	$R^1_{m_2}$	100	100	a_0	γ-action		
					a_4	AdjustMaxClients(-25)	2	s_2
					a_5	AdjustMaxKeepAliveRequests(+30)		
					a_6	AdjustMaxBandwidth(+64)	5;9	s_1;s_5

Fig. 2. System states based on the metrics structure of Fig. 1 and the state-transitio model of Fig. 7(a).

The construction of states and transitions is naturally done at run-time and not *a priori* given an active set of policies. In practice, many of the states may never occur, thus keeping the the size of the model manageable.

```
{p1} expectation policy{RESPONSETIMEViolation}
  if(APACHE:responseTime > 2000.0) &
    (APACHE:responseTimeTREND > 0.0)
  then{AdjustMaxClients(+25) |
       AdjustMaxKeepAliveRequests(-30) |
       AdjustMaxBandwidth(-128)}
{p2} expectation policy{RESPONSETIMENormal}
  if(APACHE:responseTime < 250.0)
  then{AdjustMaxClients(-25) |
       AdjustMaxKeepAliveRequests(+30) |
       AdjustMaxBandwidth(+64)}
```

Fig. 3. Sample policy system $PS = \langle P, W_C \rangle$ where $P = \{p_1, p_2\}$.

Suppose that, a policy system $PS = \langle P, W_C \rangle$ consists of the expectation policies depicted in Fig. 3. Such a system would derive the metrics structure shown in Fig. 1, which would then yield six states, illustrated in Fig. 2. Next, we describe our approach for adapting the state-transition model to run-time policy modifications.

III. Modeling Adaptation

Fig. 4 describes our approach to model-adaptation. The left-part of the diagram (i.e., prior to $\triangle[P]$) depicts the reinforcement learning mechanisms described in Section II. The agent begins with G^P_0 (perhaps the empty model - empty

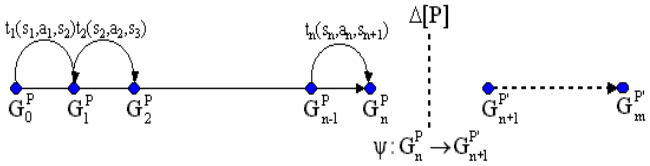


Fig. 4. Adapting to run-time policy modifications.

sets of states and transitions), and as it takes actions or encounters new states, evolves the model. Thus, the model is continuously updated based on the experience garnered with use of the same set of expectation policies, P, and its interaction with its environment. At some point there is some change to the set of policies, P, denoted by $\triangle[P]$, which results in a new set of policies P'. The diagram depicts the mapping, using the transformation function Ψ, from the current system's model (i.e., $G^P_n = \langle S, T \rangle$) to a new model (i.e., $G^{P'}_{n+1} = \langle S', T' \rangle$), as a result of $\triangle[P]$. The system then continues to use and evolve the new model.

A. Policy Modifications

By run-time policy modifications, we mean any type of modification resulting in changes to the policies driving autonomic management. Since the structure of the learned model is derived from the characteristics of the enabled policies, any run-time changes to these characteristics could have possible ramifications on the model or its use. This would depend on what kind of modification is done to the policies.

The following is the set of policy modifications that can occur to a set of expectation policies that can impact the state-transition model:

- $\triangle_1[P]$: Adding a condition to a policy in P resulting in a new metric.
- $\triangle_2[P]$: Adding a condition to a policy in P resulting in changes to the regions of an existing metric.
- $\triangle_3[P]$: Adding a condition to a policy in P resulting in no changes to the metrics regions.
- $\triangle_4[P]$: Removing a condition from a policy in P resulting in a decrease in the number of metrics.
- $\triangle_5[P]$: Removing a condition from a policy in P resulting in changes to the regions of an existing metric.
- $\triangle_6[P]$: Removing a condition from a policy in P resulting in no changes to the metrics regions.
- $\triangle_7[P]$: Modifying the threshold of a policy condition within the conditions of the policies in P.
- $\triangle_8[P]$: Adding a policy action to a policy in P.
- $\triangle_9[P]$: Removing a policy action from a policy in P.
- $\triangle_{10}[P]$: Modifying a policy action within the actions of the policies in P.

Note that more complex policy modifications, such as those involving adding/removing one or more policies from the policies in P, can be described in terms of adding/removing individual conditions and actions within policies. For example, adding policy $p_* = \langle C, A\rangle$ to the policies in P could be modeled by the following sequence of policy modifications: (i) Adding the policy without conditions or actions. (ii) Adding each condition in C, one at a time, to p_*. (iii) Adding each action in A, one at a time, to p_*.

These modifications to a policy system can be defined precisely (see [10]); we illustrate the general approach for policy modification $\triangle_4[P]$:

Policy Modification - $\triangle_4[P]$:
Given: policy system $PS = \langle P, W_C\rangle$ and condition $c_* = \langle m, \texttt{operator}, \Gamma\rangle$ such that, $c*$ is in policy p_*, p_* is in P, and where there is only a single occurrence of the condition c_* within the policies in P,
Then: removing c_* from p_* results in:

1) A new policy system $PS' = \langle P', W_C\rangle$ such that, P' is the same as P, but without condition c_*,
2) A new metrics set M' without metric m,
3) A new metrics regions structure $M_R^{P'}$ excluding the regions associated with metric m.

In the next section, we describe our approach to adaptation, elaborating on how the different policy modifications are mapped onto special instances of model transformations. We illustrate further on how the system might deal with $\triangle_4[P]$.

B. Model Transformations

Given the policy modifications outlined in the previous section, corresponding model transformations can be identified; these are summarized below:

- $\Psi_1[G_n^P]$: describes a model transformation for dealing with policy modifications that only impact the actions within states but not states or transitions (i.e., $\triangle_8[P]$ and $\triangle_{10}[P]$). The result is a new model with all the old states and transitions.
- $\Psi_2[G_n^P]$: describes a model transformation for dealing with policy modifications affecting transitions as a result of changes to the composition of policy actions within states (i.e., $\triangle_3[P]$, $\triangle_6[P]$, and $\triangle_9[P]$). The result is a new model with old states, and only the valid transitions.
- $\Psi_3[G_n^P]$: describes a model transformation for dealing with policy modifications affecting the regions within state metrics (i.e., $\triangle_2[P]$, $\triangle_5[P]$, and $\triangle_7[P]$). The result is a new model with all the old states, but with recomputed regions for the affected metric.
- $\Psi_4[G_n^P]$: describes a model transformation for dealing with policy modifications resulting in a decrease in the number of system metrics (i.e., $\triangle_4[P]$). The result is a new model with all the old states, less the affected metric.
- $\Psi_5[G_n^P]$: describes a model transformation for dealing with policy modifications resulting in an increase in the number of metrics (i.e., $\triangle_1[P]$). The result is a new model with no states and no transitions.

Our analysis of the policy modifications suggest that in the worse case scenario (i.e., $\Psi_5[G_n^P]$), $\triangle[P]$ means discarding the entire model and restarting the learning. In the best case scenario (i.e., $\Psi_1[G_n^P]$), $\triangle[P]$ has no impact on the learned model, essentially allowing the agent to continue using the model after the policy modification. This suggests that, there is significant potential for reusing some of the information about past experience prior to $\triangle[P]$. We illustrate, through an example using model-transformation $\Psi_4[G_n^P]$, how an agent is able to reuse the learned information after $\triangle_4[P]$ has occurred. As with the changes to policies, these transformations can be precisely defined (see [10]).

Model Transformation - $\Psi_4[G_n^P]$:
Given: model $G_n^P = \langle S, T\rangle$ and policy modification $\triangle_4[P]$,
Then: $\Psi_4 : G_n^P \rightarrow G_{n+1}^{P'}$ such that, $G_{n+1}^{P'} = \langle S', T'\rangle$, where

1) S' is S with m, the deleted metric, removed from all states, and the actions in each state updated, if necessary,
2) T' is T with any invalid transitions removed (a transition is said to be invalid if $\exists\ t(s_i, a, s_j) \in T : a \notin A(s_i)$),
3) Any identical states in S' are then merged,
4) Action-value estimates (part of the reinforcement learning) are updated.

Suppose, for example, that the current model of the system (i.e., $G_n^P = \langle S, T\rangle$), as depicted in Fig. 7(a), includes the six states shown in Fig. 2. Suppose also that the condition "`APACHE:responseTimeTREND > 0.0`" (which corresponds to metric m_2 in Fig. 1) is removed from policy p_1 in Fig. 3; i.e., as a result of policy modification $\triangle_4[P]$. For the system derived from visiting the states shown in Fig. 2, whose current state-transition model is shown in Fig. 7(a), such a modification would involve the following adaptation steps, which correspond to the four steps of $\Psi_4[G_n^P]$:

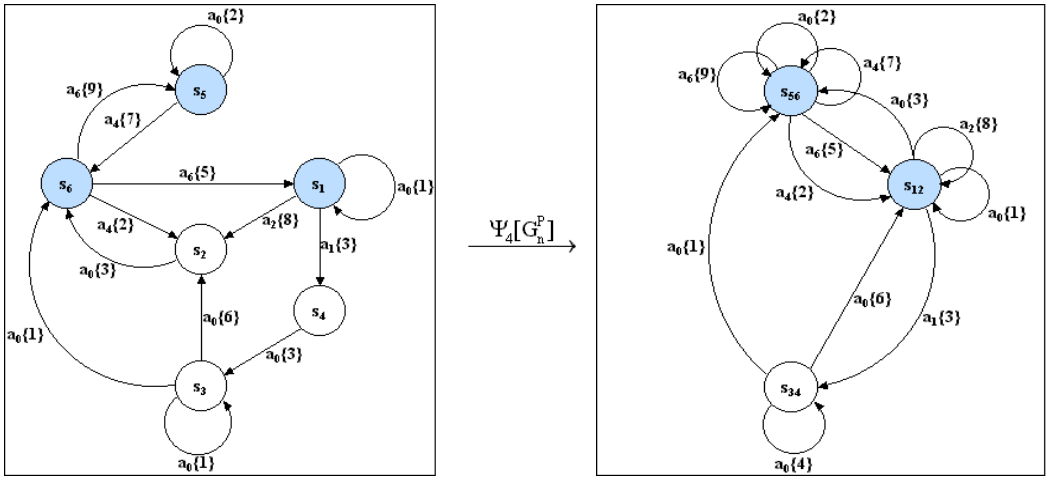
Step 1: The metric regions associated with the affected condition (in this case metric m_2) will be removed from the

m_i	c_k	Policy Condition	R_{m_i}	$R^j_{m_i}$	$f(R^j_{m_i})$
m_1	c_3	`APACHE:responseTime < 250.0`	$m_1.value < 250.0$	$R^1_{m_1}$	100
			$250.0 \leq m_1.value \leq 2000.0$	$R^2_{m_1}$	50
	c_1	`APACHE:responseTime > 2000.0`	$m_1.value > 2000.0$	$R^3_{m_1}$	0

Fig. 5. The metrics structure after removing "`APACHE:responseTimeTREND > 0.0`" from policy p_1 in Fig. 3.

State	$R^k_{m_j}$	$f(R^k_{m_j})$	$A(s_i)$		$t(s_i, a_l, s_*)$	
s_i	$R^k_{m_1}$	$f(R^k_{m_1})$	a_l	State action	λ	s_*
s_{12}	$R^3_{m_1}$	0	a_0	γ-action	1;3	s_{12};s_{56}
			a_1	`AdjustMaxClients(+25)`	3	s_{34}
			a_2	`AdjustMaxKeepAliveRequests(-30)`	8	s_{12}
			a_3	`AdjustMaxBandwidth(-128)`		
s_{34}	$R^2_{m_1}$	50	a_0	γ-action	6;4;1	s_{12};s_{34};s_{56}
s_{56}	$R^1_{m_1}$	100	a_0	γ-action	2	s_{56}
			a_4	`AdjustMaxClients(-25)`	2;7	s_{12};s_{56}
			a_5	`AdjustMaxKeepAliveRequests(+30)`		
			a_6	`AdjustMaxBandwidth(+64)`	5;9	s_{12};s_{56}

Fig. 6. System states based on the metrics structure of Fig. 5 and the state-transitio model of Fig. 7(b).



(a) Original model: $G^P_n = \langle S, T \rangle$

(b) New model: $G^{P'}_{n+1} = \langle S', T' \rangle$

Fig. 7. The impact of removing the condition "`APACHE:responseTimeTREND > 0.0`" from policy p_1 in Fig. 3.

metrics structure as illustrated in Fig. 5. For each state that has been encountered to date (see Fig. 2), the agent must ensure that all information associated with metric m_2 is removed. This would result in a system's model with six states each with a single metric; i.e., $M(s_i) = \{m_1\}$. The agent must also update the actions sets of each state. In the context of our example, this would involve updating the actions set $A(s_2) = \{a_0\}$ to $A'(s_2) = \{a_0, a_1, a_2, a_3\}$ because the policy modification would now mean policy p_1 is violated whenever the agent is in state s_2; i.e., $P(s_2) = \{p_1\}$. This was not the case prior to $\triangle[P]$ since only one of the conditions of policy p_1 was violated; a policy is said to be violated if all its conditions evaluate to true.

Step 2: The state transition model must also be updated to ensure that transitions within states are consistent with the updated state actions sets. In our example, no transition will be affected since all the transitions in state s_2 would remain valid under the new actions set $A'(s_2) = \{a_0, a_1, a_2, a_3\}$.

Step 3: The agent must then determine whether any states should be merged together since the changes above may have resulted in two or more identical states. Since states are uniquely identified by the regions values assigned to the states metrics (i.e., $f(R^l_{m_1})$), states s_1 and s_2 are essentially identical after metric m_2 is deleted. The same could also be said about the states pairs $\{s_3, s_4\}$ and $\{s_5, s_6\}$. Our approach to adaptation involves merging similar states; i.e., the two identical states in each pair would be merged into three unique states namely s_{12}, s_{34}, and s_{56} as illustrated in Fig. 6 where state s_{ij} corresponds to a state obtained by merging states s_i and s_j. This step also involves updating statistics associated with individual transitions, such as the action-value estimates and frequencies, whenever two identical transitions are merged together as a result of merging the two states. For example, transitions $t(s_3, a_0\{1\}, s_3)$ and $t(s_4, a_0\{3\}, s_3)$ from

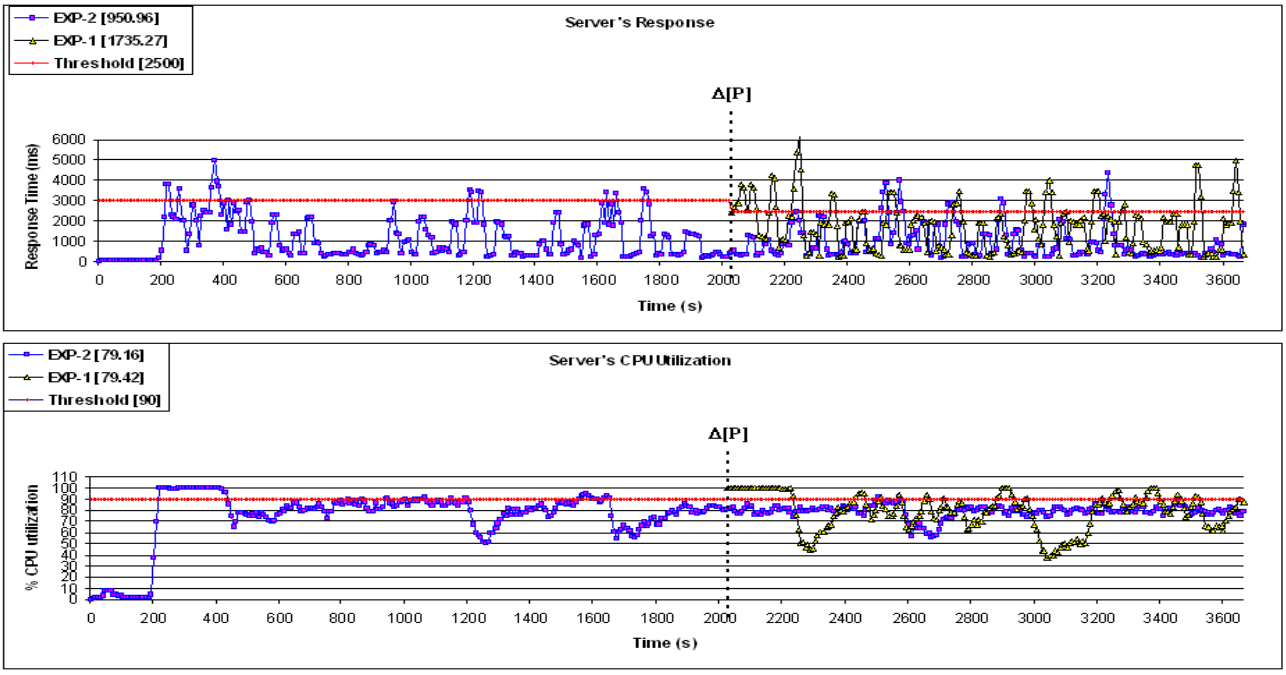


Fig. 8. Adapting to $\triangle[P]$ involving modifying the threshold of condition "`APACHE:responseTime > 3000.0`" to "`2500.0`".

the old states set (i.e., $s_i \in S$) would all map onto transition $t(s_{34}, a_0\{4\}, s_{34})$ in the new states set (i.e., $s_{ij} \in S'$) as shown in Fig. 6. Note that the frequency associated with the transition $t(s_{34}, a_0, s_{34})$ is the sum of the frequencies associated with the two merged transitions.

Step 4: The final step of the adaptation is for the agent to perform backup-updates so that any impact on the state-transition model as a result of either removing transitions or merging states is propagated across the model.

IV. Experimental Evaluation

In this section, we consider the effectiveness of the adaptation mechanisms on the behavior of a multi-tiered Web server in response to run-time policy modifications. In particular, we compare results from two experimental settings based on how the agent responded to run-time policy modifications. The first experiment (*EXP-1*) illustrates the behavior of the server where the learning agent discards everything it learned prior to $\triangle[P]$ (see Fig. 4) and begins learning from scratch; i.e., $G^{P'}_{n+1} = \{\emptyset, \emptyset\}$. The second experiment (*EXP-2*) considers the behavior of the server where the learning agent adapts the model to take into account changes to an active set of policies, essentially reusing some of the learned information; i.e., $\Psi : G^{P}_{n} \rightarrow G^{P'}_{n+1}$. The performance comparisons in terms of the averages and violation rates reported focused specifically on the measurements after $\triangle[P]$ occurred.

A. Prototype Implementation

1) Testbed Environment: Our testbed consisted of a collection of networked workstations: an administrative workstation to run the experiments; a Linux workstation with a 2.0 GHz processor and 2.0 Gb of memory, which hosted the Apache Web Server along with the PHP module and the MySQL database server; and three workstations used for generating the server's workload, which consisted of clients requests associated with gold, silver, and bronze service classes.

2) Workload Generator: To emulate the stochastic behavior of users, an Apache benchmark tool called JMeter [11] was used. The tool provides support for concurrent sampling of requests through a multi-threaded framework. It provides the ability to specify the client's think time as well as the number of concurrent - independent - *keep-alive* connections. In the experiments reported here, these values were set to ensure that the server was under overload conditions (i.e., saturated) for the entire duration of the experiments. In our current implementation, the tool has been configured to traverse a Web graph of an actual Web site: in our case, phpBB [12]. In the experiments reported in this section, only *read-only* database requests were considered.

3) Service Differentiation: In order to support service differentiation, a Linux Traffic Controller (TC) was used to configure the bandwidth associated with the gold, silver, and bronze service classes. The service classes bandwidth was assigned proportionately according to the ratio 85:10:5. Bandwidth sharing was also permitted. The tuning parameter `MaxBandwidth` is what determines how much bandwidth is assigned to each service class. It corresponds to the physical capacity (in kbps) of the network connection to the workstation hosting the servers. Thus, given that the first policy of Fig. 3 has been violated and that it is no longer possible, for example, to adjust the parameters `MaxClients` and

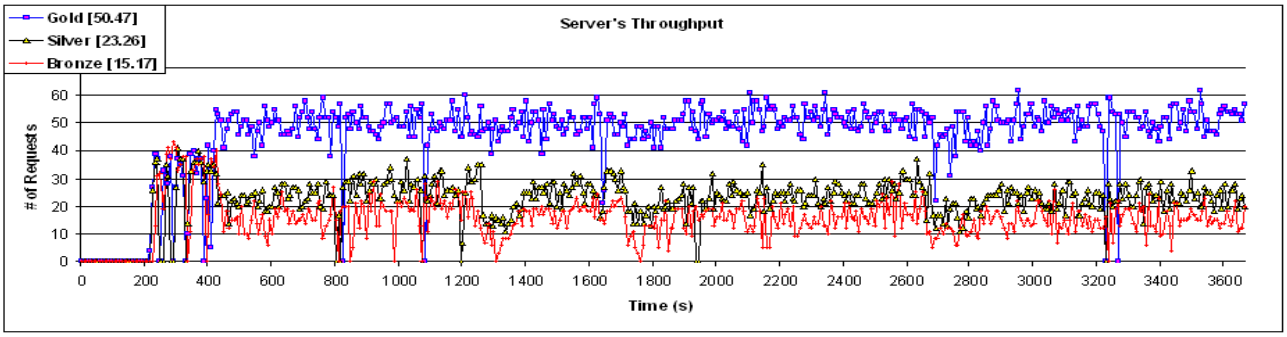


Fig. 9. Server's throughput measurements for *EXP-2*.

`MaxKeepAliveRequests`, then the last policy action (i.e., `AdjustMaxBandwidth(-128)`) would be executed. This action essentially reduces the total bandwidth by 128 kbps. The percentage of the new bandwidth is what is eventually assigned to the different service classes.

B. Results

Fig. 8 describes the behavior of the server in the two experiments where $\triangle[P]$ involved modifying the threshold of policy condition "`APACHE:responseTime > 3000.0`" to "`2500.0`". This is illustrated by the threshold line of the first graph, which shows $\triangle[P]$ occurring at about 2,000 seconds from the start of the experiment. The results of the second graph shows the behavior of the server in terms of CPU utilization. Fig. 9 shows the throughput measurements of *EXP-2* in terms of the number of requests successfully served for the gold, silver, and bronze service classes. With service differentiation mechanisms enabled, performance differences between the classes was substantial, with more gold clients requests being processed in the expense of silver and bronze clients requests; this was consistent in both experiments.

When the average throughput measurements (i.e., the number of requests successfully served) of identical service classes were compared (see the top diagram in Fig. 10), the server performed slightly better in *EXP-2* than in *EXP-1*, particularly for the gold service class where more requests were served. For the other classes, however, the performance gains were statistically insignificant since the measurements fell within the standard error, as is illustrated by the error bars associated with the mean throughput. In terms of the average response time across the service classes (see the bottom diagram), the results were a bit mixed. On the one hand, gold clients in *EXP-2* performed slightly better than their counter parts in *EXP-1*. On the other hand, with adaptation, the silver and bronze service classes performed poorer - perhaps as a result of improved service to the gold clients.

While the above results show slight improvement in the overall quality due to the adaptation mechanisms, an alternative was to compute the area occupied by the curves beyond the thresholds and divide it by the duration of the experiment (60 minutes). This is a more accurate measure

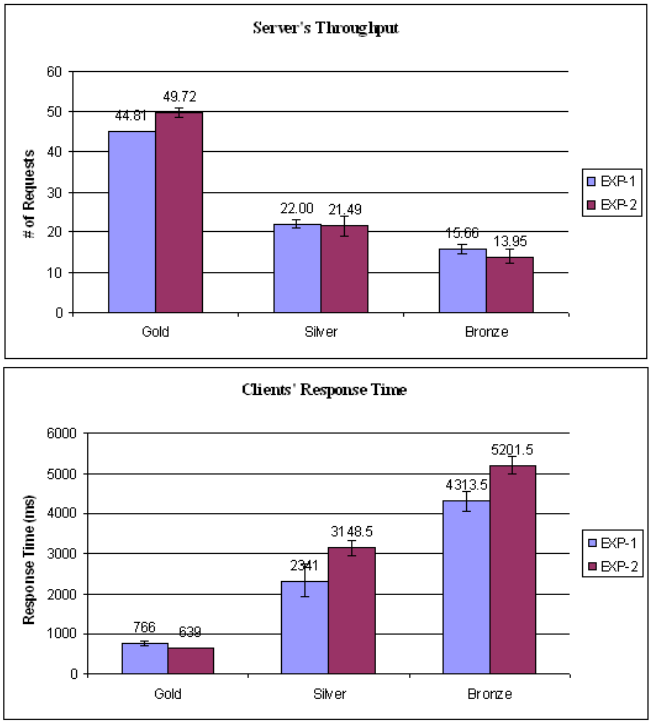
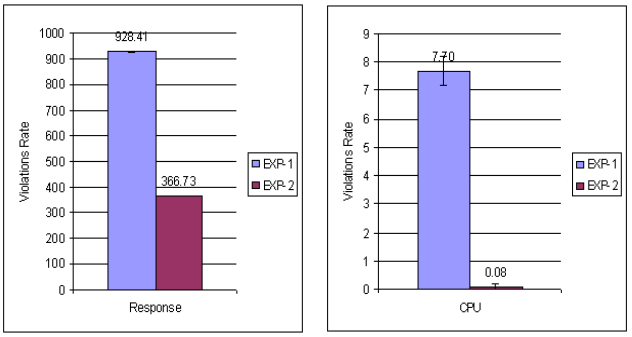


Fig. 10. Mean performance comparisons.

of the performance since the main concern was to ensure that performance objectives, as captured by the thresholds of the expectation policies, were not violated. Thus, violations magnitude captured both the severity and the frequency with which performance objectives were violated. These results are summarized in Fig. 11. Note that, *EXP-2* recorded at least a 60% improvement in the quality of service as measured in terms of response time violations (see Fig. 11(a)) and a 99% improvement when considering CPU utilization violations (see Fig. 11(b)). The performance improvements, particularly in terms of the CPU utilization violations, are consistent with the results of Fig. 8; i.e., apart from the violations recorded at about 2,600 seconds, the measured utilization fell below the threshold for the duration of the run.



(a) Response Time (b) CPU Utilization

Fig. 11. Magnitude of violations performance comparisons.

V. Discussion and Conclusion

Our previous experience in utilizing reinforcement learning methodologies to provide guidance to autonomic systems have demonstrated significant performance benefits, particularly in dynamic use of policies within autonomic management (see [6], [7]). Unlike most of the approaches where changes to policies often mean discarding the old model, however, having the ability to reuse the learned model, or part there of, as illustrated in this paper can significantly accelerate the learning process. The case study, in particular, demonstrated how transformations could be used to alter an existing reinforcement learning model (Fig. 7(a)) to form one compatible with changes in a set of policies (Fig. 7(b)). In this way, actions learned by an autonomic manager based on one set of policies can be "re-used", within the scope of some changes, for a closely related set of policies. The goal here is to re-use knowledge gained and to reduce the time that the manager may need to "re-learn" strategies. Note that in the worst case, the manager simply discards the model and restarts. Our experience, however, suggests that changes to policies sets are typically incremental. Thus, re-using past knowledge can significantly improve the overall quality of service, as the results have demonstrated. This, of course, would depend not only on the type of policy modifications, but also on the amount of time used to form the model before the changes.

A related question, however, is whether the transformation of one model to another model results in a "valid" model. More precisely, given a policies set P and a reinforcement learning model G_n^P that has been learned over some period of time, if P is changed to P' and G_n^P is transformed to $G_{n+1}^{P'}$, then is $G_{n+1}^{P'}$ a model of P'? Our construction of any model is incremental, that is, based on the behavior of the system and the actions of the autonomic manager. Thus, the states and transitions of the model are developed over time; i.e., states and transitions are only added to the model if encountered. This keeps the number of states reasonable in practice. This means, however, that we cannot ask whether $G_{n+1}^{P'}$ would have been formed if P' had been the set of policies in place when the system started instead of P. What we really wish to know is whether $G_{n+1}^{P'}$ is, in some sense, "consistent" with the policies set P'; i.e., are the states in $G_{n+1}^{P'}$ ones that could have been formed under policies set P' and are the actions on the transitions consistent with P'. The formulation of this question and the formal proofs that show that, indeed if P is changed to P' and G_n^P is transformed to $G_{n+1}^{P'}$, then $G_{n+1}^{P'}$ is a model of P', can be found in [10].

Our approach is only the first step in the broader goal of developing adaptive policy driven autonomic solutions involving multiple agents working together towards a set of common objectives. The complexity of today's IT infrastructure means that we can no longer rely on centralized approaches to performance management. It is becoming increasingly common, for example, to find multiple agents coexisting within a single computing environment. Thus, policy modifications directives could come not only from the users of the systems, but also from other autonomic managers. The fact that our approach to modeling learning and adaptation is only dependent on the structure of the policies means that the impact of policy changes could be mapped onto the original model, essentially providing the ability to transform the model according to the type of policy modification. Extending our prototype to incorporate collaborative management mechanisms looks very promising and definitely deserves further investigation.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[2] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," in *International Conference on Autonomic Computing (ICAC'06)*, Dublin, Ireland, June 2006, pp. 65–73.

[3] G. Tesauro, "Online Resource Allocation Using Decompositional Reinforcement Learning," in *Association for the Advancement of Artificial Intelligence (AAAI'05)*, 2005.

[4] D. Vengerov and N. Iakovlev, "A Reinforcement Learning Framework for Dynamic Resource Allocation: First Results," in *International Conference on Autonomic Computing (ICAC'05)*, Seattle, WA, USA, January 2005, pp. 339–340.

[5] P. Vienne and J.-L. Sourrouille, "A Middleware for Autonomic QoS Management based on Learning," in *International Conference on Sofware Engineering and Middleware*, Lisbon, Portugal, September 2005, pp. 1–8.

[6] R. M. Bahati and M. A. Bauer, "Reinforcement Learning in Policy-driven Autonomic Management," in *Network Operations & Management Symposium (NOMS'08)*, Salvador, Brazil, April 2008, pp. 899–902.

[7] ——, "Modelling Reinforcement Learning in Policy-driven Autonomic Management," in *IARIA International Journal On Advances in Intelligent Systems*, vol. 1, no. 1, January 2008, pp. 54–79.

[8] N. Damianou, N. Dulay, E. C. Lupu, and M. S. Sloman, "Ponder: A Language for Specifying Security and Management Policies for Distributed Systems: The Language Specification," Technical Report, Imperial College, London, England, Version 2.1, April 2000.

[9] R. M. Bahati and M. A. Bauer, "An Adaptive Reinforcement Learning Approach to Policy-driven Autonomic Management," in *International Conference on Autonomic and Autonomous Systems (ICAS'09)*, Valencia, Spain, April 2009, pp. 135–141.

[10] R. M. Bahati, "Towards Adaptive Policy-based Autonomic Management," Ph.D. dissertation, The University of Western Ontario, London, ON, Canada, 2010.

[11] Apache JMeter. [Online]. Available: http://jakarta.apache.org/jmeter/

[12] PHP Bulletin Board. [Online]. Available: http://www.phpbb.com/