2010

Predicting Policy Violations in Policy Based Proactive Systems Management

Khandakar Rashed Ahmed
Western University

Predicting Policy Violations in Policy Based Proactive Systems Management

(Spine Title: Policy Based Proactive Systems Management)

(Thesis format: Monograph)

by

Khandakar Rashed Ahmed

Graduate program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

The continuous development and advancement in networking, computing, software and web technologies have led to an explosive growth in distributed systems. To ensure better quality of service (QoS), management of large scale distributed systems is important. The increasing complexity of distributed systems requires significantly higher levels of automation in system management. The core of autonomic computing is the ability to analyze data about the distributed system and to take actions. Such autonomic management should include some ability to anticipate potential problems and take action to avoid them that is, it should be *proactive.*

System management should be proactive in order to be able to identify possible faults before they occur and before they can result in severe degradation in performance. In this thesis, our goal is to predict policy violations and take actions ahead of time in order to achieve proactive management in a policy based system.We implemented different prediction algorithm to predict policy violations. Based on the prediction decision, proactive actions are implemented in the system. Adaptive proactive action approach is also introduced to increase the performance of the proactive management system.

Keywords: Prediction, Policy Violation, Policy State Graph, Proactive Action.

Acknowledgements

This is a great opportunity to express my respect and thanks to Dr. Michael A. Bauer for his valuable guidance, support and supervision. His continuous motivation kept me inspired throughout this research work. He was always there with his advise and helpful direction when it was required most.

Heartfelt thanks to all the members in my research group for their valuable feedback and help on this research work.

I would like to thank Raphael M. Bahati for his great support and guidance to me throughout my research.

I would also like to thank all my friends for their motivation and cooperation till date.

I am pleased to thank my family members for being a continuous source of support and inspiration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

1.1 Introduction

The continuous development and advancement in networking, computing, software and web technologies have led to complex distributed systems. To ensure better quality of service (QoS), management of large scale distributed systems and networks has become very important [Wik10]. Distributed systems include network devices, compute nodes and resources, like memory, communication links among nodes in the network, etc. The management of these large systems is a challenging task for a variety of reasons including communications among heterogeneous components, the dynamic characteristics of distributed systems etc. [HKV+98]. Current management technologies are considered to be *reactive management*, that is, they may take action after problems occur. These technologies collect management information from the system and may take actions or provide graphical user interfaces to assist a system manager in critical situations [KHD00]. Efficient management of such complex systems continues to be an important area of research.

1.2 Proactive Management

System management should be proactive for a complex distributed system in order to be able to identify possible faults before they occur and before they can result in severe degradation in performance [FAR+97]. The aims of proactive systems management are to ensure the quality of the offered services, to increase system performance and increase reliability. Organizations can benefit from proactive management because it can help to decrease the investment in fault recovery, save time and increase service to customers. Normally, the task of management begins when any fault is encountered in the distributed system. The management team takes immediate action to try to determine the root cause of the fault. This type of management approach can be considered *reactive management.* Proactive system management attempts to predict faults in advance so that the probable fault could be avoided and the system can remain in a safe state.

The growth in network based services has increased the pressure on management systems. Rapid detection and resolution of problems has created pressure for system administrators [HZS01]. In the next few years, production systems are expected to contain tens of hundreds of thousands of computing nodes and thousands of I/O nodes [Sit]. As network size increases, the types and numbers of faults also increase. Proactive management could be a useful approach in this dynamic environment.

In order to try to identify problems in advance, one must rely on current and past data and then predict when a problem might occur. *Prediction is a statement identifying some outcome which is expected in the future [Wik10].* Some form of prediction is required for the development of a proactive management system.

1.3 Differences between Reactive and Proactive Management

The functionality of reactive and proactive system management is to identify system faults and provide solutions to rectify the problem. Though their overall objectives are the same, their approaches are different.

In reactive management, faults are detected by the management system. The management system collects real time information by monitoring the system. This information includes performance metrics, resource usage, application status, communication link performance etc. Network and system administrators can get information about the entire system from the representative information and can do maintenance operations based on that information. An alarm is triggered if any sort of anomaly or abnormal condition is detected. By understanding the nature of alarms and severity level, the management system or the system administrator can take corrective actions to resolve the abnormal situation.

In a proactive approach, the management system attempts to predict faults in advance of their occurrence in the system. In this case, alarm messages might be generated to the system administrator to notify them that the system is expecting abnormal conditions within a short time. The management system uses prediction techniques to do this based on the available data from the system. If the prediction is correct, then the administrator or system could take action to avoid the potential problem. The system information collected is the same as that collected with the reactive method. Though prediction cannot be completely correct, the main advantage is that if it is successful a sufficiently often, then it can help avoid or reduce downtime costs and any loss of quality of service (QoS).

1.4 Policy Based Proactive Management

Policies are often used to specify the required or desired behaviour of a system and its applications. Policies can be used in an autonomic management system to adjust application or system tuning parameters in order to meet operational requirements [BBV07]. In many systems, there may be multiple components where each component may have its own set of policies. When these policies are violated, the autonomic management system tries to identify the actions needed to take, based on the policies or in some cases, based on the past behaviour of the system [BB08]. This approach to management is essentially policy based reactive system management. The nature of a reactive approach is that the management system takes action only after a policy is violated which can occur when there is a decrease in system performance, availability, reliability, etc. In contrast, an approach to systems management based on a policy based proactive approach attempts to predict a policy violation and then take proactive action before the violation actually occurs. Of course, with complex system behaviour and multi-component environments, predicting policy violation is a challenge.

1.5 Thesis organization

The remaining Chapters of this Thesis are organized as follows. The second Chapter looks at previous work on different prediction techniques in system management and policy based management. Chapter Three provides details about a reinforcement learning model which is the base of this thesis. It defines our notion of *policy*, an existing system architecture and the reinforcement learning process. Chapter Four outlines the research approach to policy based proactive system management and presents several algorithms for prediction and action selection based on the reinforcement learning model. Chapter Five describes an implementation and presents experimental result. Finally, Chapter Six presents some concluding comments and possible future work.

Chapter 2

Related Work

Most existing management systems collect management information about packet throughput, delay and packet errors at input and output of network interfaces[HKV$^+$98]. To implement end-to-end proactive system management, more information is required, like the current loads on computers, the types of processes accessing file systems, types of users and their access pattern profiles, security information, etc. The amount of information collected for large distributed systems can be vast. It is very difficult for system administrators to efficiently use all of this of information to improve performance.

2.1 Background Knowledge

A large distributed system is a complex architecture where thousands of nodes are connected to each other and each node has resources like processors, memory, and storage arrays etc. These resources communicate within the node and between nodes to try to provide the best performance. Prediction in proactive system management can be roughly categorized as falling into two broad categories - Resource Usage and Fault Management.

Prediction of resource allocation or utilization is required to ensure the better system performance. Prediction of resource usage can include estimation of CPU load, memory utilization, process workload, link performance, bandwidth usage, scheduling of resource or I/O operations, etc.

In a distributed system, a fault is an abnormal condition in one or more of the system components. A fault can be two of types: application/software fault or a hardware fault. When any application performance is degraded or if the application is not working properly, it is considered to be an application fault. For example, exceeding the threshold limit of a specific parameter of a network application, an application being down, a specific application module down etc. are generally considered to be application faults. On the other hand, nodes consist of different types of hardware like processors, storage arrays, network cards etc. Hardware faults can include disk channel degradation in disk arrays, bad sectors in physical memory, processor malfunction etc.

A failure is a critical situation in a system when system components or the system itself becomes out of service status. For example, application crash is considered as application failure and abnormal system shutdown is considered as system failure.

2.2 Prediction Techniques in Proactive Systems Management

In recent years, researchers have explored different techniques to do different types of prediction. Most of the techniques have dealt with fault detection in systems where a fault is mostly an application or service problem. There have also been some prediction techniques used to predict resource usage, like CPU load, disk usage etc. The following describes some of the prediction techniques used in recent years.

2.2.1 Fuzzy Logic Controller

A fuzzy logic controller prediction method is used in [AGP06]. In this paper, the approach tried to predict computational demand or resource usage of CPU load, disk usage etc. in a utility computing environment system based on the past information. A web hosting company was considered a utility computing environment where customers create

CPU load or disk usage based on their needs.

The utility provider can sell more resources than it actually has; this is called over-subscribing. The concept of over-subscribing resources in a utility computing environment depends on predictions of the use of resources by consumers and how it handles the chance or risk of over-subscriptions.

The use of a utility computing environment is changed by user requests which actually change the data used for prediction modeling. The proposed approach is based on genetic algorithms and fuzzy logic which allows for creation of robust prediction models which can be trained by scarce training data. This model is a non-linear model.

The prediction process of computation demand is divided into two steps:

- A Fuzzy Logic Controller (FLC) is used to generate a rule set from a data set (based on an open source fuzzy engine[Saz02]).
- A genetic algorithm (learning scheme) is used to generate a ranked list of potential solutions from the rule set generated by fuzzy logic.

Fuzzy Logic and the Fuzzy Logic Controller

Fuzzy logic (FL) is an extension of classical logic which provides an effective conceptual framework for handling decisions under uncertainty and imprecision, and is especially suitable for a scarce model data scenario[Zad65]. A fuzzy logic predictor/controller (FLC) introduced in [MA75] is essentially a function interpolator based on if/then rules and fuzzy reasoning.

The controller input (state variables) and output (control variables) are represented by linguistic variables (LV) which are real valued variables. For each LV value, the range of its value is several fuzzy sets. The output value is interpreted as the degree of membership of the LV value to the corresponding fuzzy set. In Fuzzy Logic, the presentation of a LV value belongs to intervals with some degree of certainty, e.g. a value might belong to the interval [a,b] with a certainty of 90 percent and to an interval [c,d] with a degree of 30 percent. Note that Fuzzy Logic permits LVs to belong to multiple intervals which do not have to be exclusive. The rule set is defined based on the input and output variables. Classical design techniques for FLCs consist of the manual specification of the LVs, fuzzy sets, and rules by a domain expert.

An example of a rule set is given below [Saz02]:

If A is high then Z is negative
If C is low or D is high then Z is positive

In this rule set, A, C and D are LV input variables and Z is an output variable and the fuzzy sets for all variables are high, low and positive, negative. Several rule sets are prepared by a domain expert based on the past information. A family of FLCs is defined to represent all possible solutions which is required for the genetic algorithm.

Genetic Algorithm (learning scheme)

A genetic algorithm is used to obtain a ranked list of potential solutions (rule set) from an FLC family. In genetic fuzzy systems[CHW+95], a genetic algorithm (GA) is used to improve the generated fuzzy sets and/or the if/then rules. A set of potential solutions (rule set) containing appropriate encodings of FLCs is retained. These solutions are randomly changed and merged yielding a new set of solutions. The fitness of each new solution is evaluated based on historical data. The fitness value is determined to decide whether the solutions will survive in the next generation or not. The process continues for a fixed number of iterations.

Initially, the population of the GA consists of randomly chosen members of the FLC family. The mutation is done by randomly changing every encoded part of the rule set. Different set of values can be used for each mutation type. For each of these mutation types, different probabilities can be set.

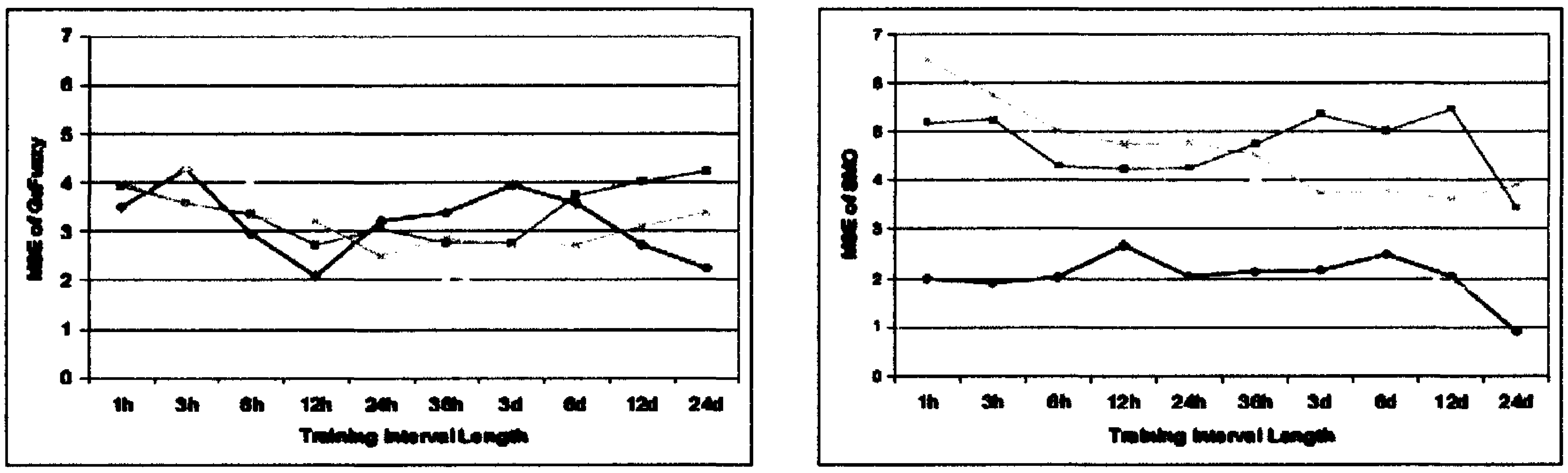


Figure 2.1: MSE of the GaFuzzy(left) and MSE of the SMO(right) [AGP06]

The benefits of this approach include easy and compact encoding of the prior knowledge, which is human readable, and shows that good predictions are possible even with small training sets. This approach provides good results for dynamic computer systems and networks, such as utility computing environments. The main disadvantage of this approach is the manual rule creation. The rule creation process should be dynamic rather than manual.

2.2.2 Bayesian Network Approach

The probabilistic framework of a Bayesian network has been used to do prediction in several research studies[HJ98, DLJ$^+$06, HE01]. We focus on the work in [HJ98] which tries to predict network anomalies that typically precede a fault. Specifically, the authors propose an approach to predict node failure. The paper [HJ98] described an intelligent agent that collects information about a network node using SNMP (Simple Network Management Protocol). The intelligent agent learns the normal behaviour of each measurement variable and combines the information into the probabilistic framework of a Bayesian network. This produces an image of the network's health from the perception of the network node, which can be used to generate local corrective action or a message to a centralized network manager.

Structural Overview

Figure 2.2 illustrates how the intelligent agent approach works. It has two steps: observation processing and combination of information.

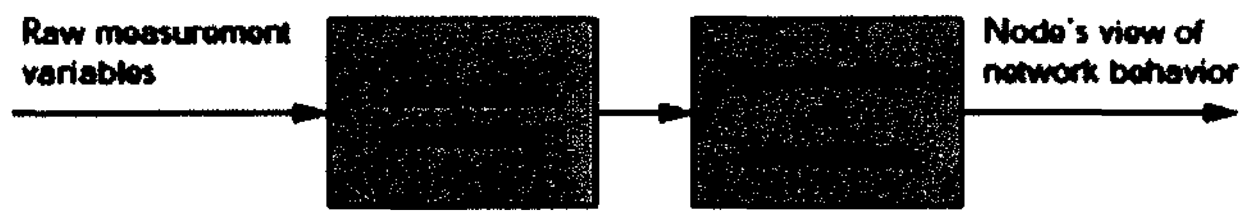


Figure 2.2: Structural Overview of the function of intelligent processing agent [HJ98]

Observation Processing

Each network node must build its own picture to generate the network's health information. To provide the picture of the node, the observation processing component processes the raw measurement variables and generates the probability of each measured variable at a given time. The component uses a change detection method to estimate the behaviour of the measurement variables. The goal is to detect the change in network behaviour which is related to a network fault. The process has three aspects: segmenting data, feature extraction and learning behaviour.

Segmenting Data

The observation processing component uses an algorithm [AB83] to segment the raw data into variable length pieces where each piece contains a portion of the time series data that is statistically similar. The algorithm assumes that the time series is from a piecewise wide-sense stationary Gaussian process [HJ98]. After segmentation, the signals are in stationary pieces. The benefit of using segmentation is that it temporally correlates the observations. Because many network signals are bursty, the temporal correlation can help the agent distinguish between a burst and a change in the signal's nature.

Feature Extraction

The most common method of detecting abnormal behaviour is thresholds. The feature is not the value of the threshold itself, but the information on whether a particular measurement variable exceeded the threshold value or not. If the threshold has upper and lower bounds, then the feature is whether the measurement variable is within the threshold boundary or not. It is a hard task to determine the correct threshold value because of the network's dynamic behaviour. That is, the objective is to capture only the signal information that is related to abnormal network conditions. In order to do that, features that can use network traffic measurements to distinguish abnormal from normal network behaviour are required. Cynthia and Chuanyi choose features that change along with changes in the network performance and allow the normal behaviour model to continually adapt. The change detection method uses the parameters of a second-order autoregressive process $AR(2)$ as features. The $AR(2)$ process is defined as:

$$y(t) = a_1 y(t-1) + a_2 y(t-2) + \in (t)$$

where $y(t)$ is the value of the signal at time t, and a_1 and a_2 are the AR parameters that are used as features, and $\in (t)$ is noise or error term. Using the $AR(2)$, the features are calculated for each node to generate the node picture.

Learning Behavior

The agent uses the features to create a description of normal behavior in the form of a probability distribution. In order to estimate the probability of each sample when the network is operating normally and when there is a fault, we need to know exactly when the network is operating normally from the node's view. Information about the network problems is generated in the system log file which is used by the intelligent agent from learning perspective. The reports from the log file are used as measurement labels that identify when the fault occurred. These labels are used to learn the probability distribution of each measurement variable when it is related to abnormal network function. The agent must also learn the probability distribution of variables which is related to normal network function. Normal behavior is defined as the variable behavior during the learning window when the agents learn about the probability distribution. During this learning period, it is assumed that agent will rarely learn about problematic behavior of network. Since problematic behavior is unknown, any sample that falls outside the range of $AR(2)$ parameters is considered as abnormal behavior of the network.

Combination of Information

The goal of this component is to combine the processed information generated by the observation processing component into higher level measures of network behavior from node's view. Measurement variables are combined in the probabilistic framework of a Bayesian network[Pea88]. The structure of the Bayesian network is illustrated in Figure 2.3.

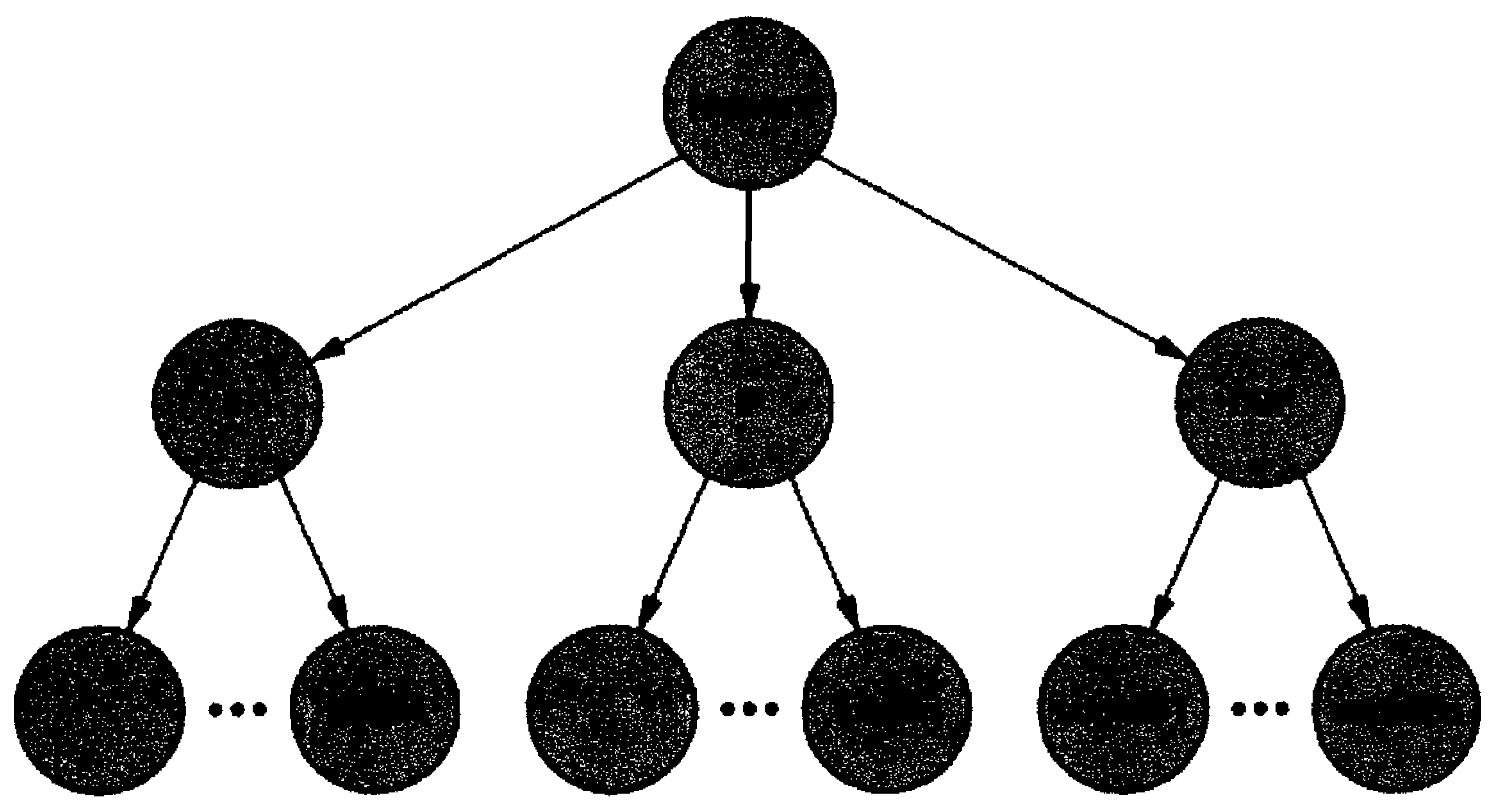


Figure 2.3: Probabilistic framework of a Bayesian network [HJ98]

The top level is a variable which describes the network's entire health. The middle levels are the internal network variables which signify different network functionality. The bottom levels are the MIB(Management Information Base) variables which are stored in the SNMP agent's management information base. The arrow from top to bottom between the nodes represents the cause and effect relationship. Network and MIB variables are observed variables. Internal variables are not observed variables and it includes IF (which represents the network interface), IP (which represents the Internet Protocol) and UDP (which represents the User Datagram Protocol). These variables are associated with the MIB groups and have different network functionality. Figure 2.3 shows that the network health information directly influences network functionality information and network functionality also influences the MIB variables.

The approach assumes that the network health and network functionalities are independent, since each network function represents an independent functional network component. This conditional independence is assumed in order to simplify the problem of combining information. All internal variables (IF, IP, UDP) and the network health variables are discrete variables and each has two states: normal and abnormal. The MIB variables are continuous. Detail probability calculation process is given in [HJ98].

Evaluation

In an experiment to evaluate the approach, seven months of log data was collected using SNMP queries. There were seven subnets and two routers in the network. Router 2 was connected to different sub networks and data was collected from Router 2. The system detected fault "Server not responding between 6:33 a.m. and 6:36 a.m. December 23, 1996, where the server is a fileserver in the subnetwork. The results for the posterior probabilities estimated are shown in Figure 2.4. The x-axis represents the time in second and y-axes are the posterior probabilities of the abnormal behavior in (a) IF variables, (b) IP variable, (c) UDP variable and (d) network variable.

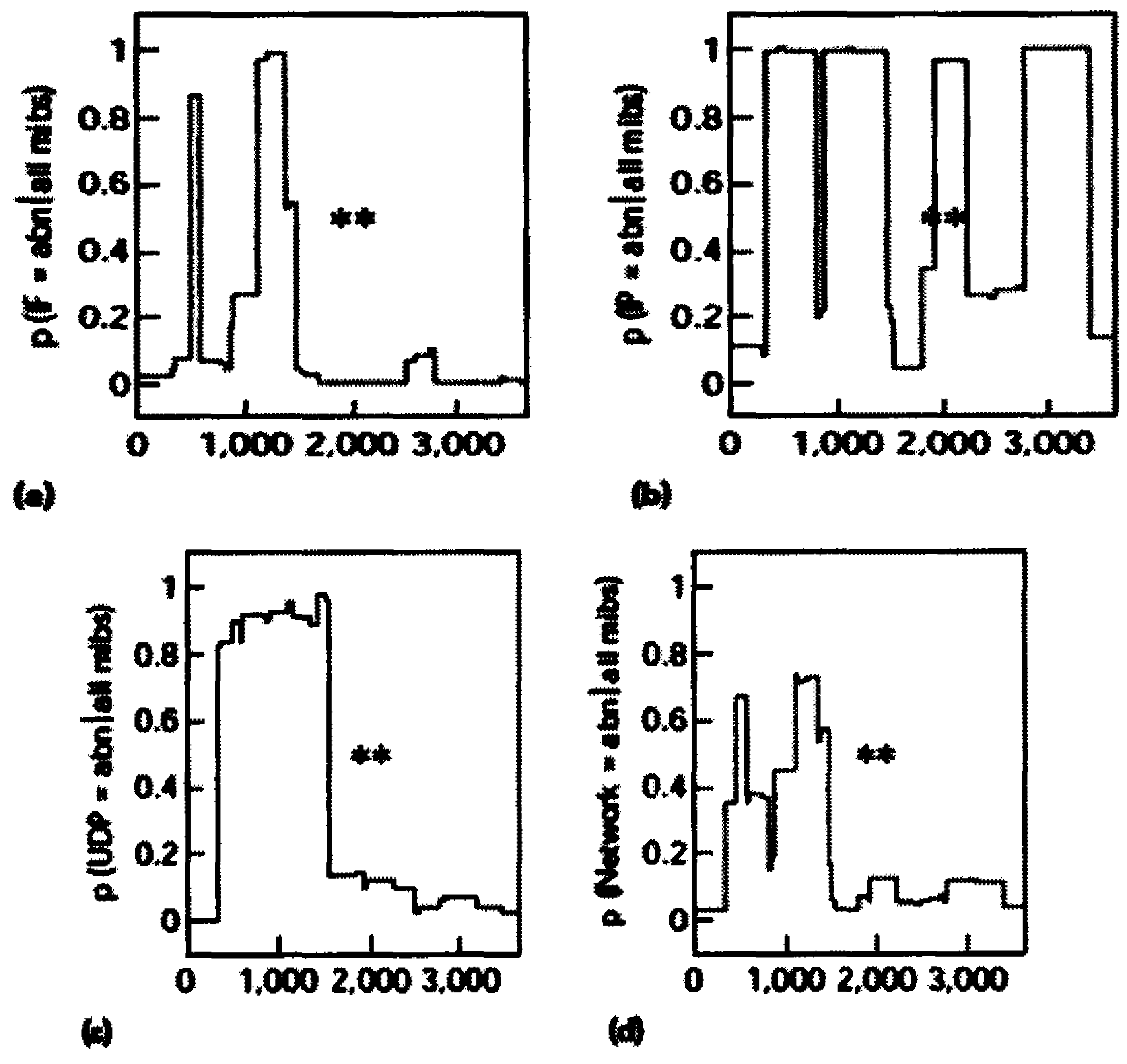


Figure 2.4: Experiment result using 1 hour learning window [HJ98]

The information from (a) to (c) is combined to determine the network information (d). The asterisk denotes the downtime period in the graph. It can be seen from the graph

(d) that the agent was able to detect abnormal behavior 12 minutes before the failure report of the server. Thus the experiment shows the validation of the approach.

2.2.3 Knowledge Based Approach

The work by Webber and Westphall[WW97] introduces an approach to identify potential problems in advance of any performance degradation of network applications using knowledge based techniques as well as providing support for future decision making actions in a networked environment. This type of prediction falls into the fault management prediction category, since it is related to application performance degradation. Their overall approach joins two different approaches to address proactive network management. First, they use remote monitoring and simulation tools[Fra96] and secondly they make use of a knowledge based approach [Roc96].

The prediction process starts by constructing baseline information (Figure 2.5, left). The network is monitored constantly to collect statistical samples and within a short period of time it is possible to establish a network profile which is called a "baseline". The main components of this approach are the Management Platform, the Communication Service and the Proactive Module. The knowledge base system has been attached to the Verification Service module (Figure 2.5, right) for the diagnosis process. The Proactive Module is programmed to notify the management platform of any parameters that indicate a decrease in the network performance. These parameters will then be analyzed in order to provide the corrective actions to be performed by the manager.

The baseline information is used to determine symptoms through comparisons with the real-time data which is provided by remote monitoring agent. The verification schema (Figure 2.5, right) collects data from remote agent in real-time and compares that with the baseline data. If some tendency toward trouble was detected, the system would notify the manager (or Management Platform) using the Communication Service. This activity is realized through the diagnosis process. The diagnosis process makes use of a set of trees which represents the knowledge base for the system. The tree has three levels.

The lowest level of the tree is the Parameter Level, where the parameter state is identified and evaluated as function of its value, average and standard deviation. The middle level is the Diagnostic Level which contains diagnostics made through the analysis of parameters. At the top of the tree, the Suggestion Level, the final diagnostics are suggested for the network administrator.

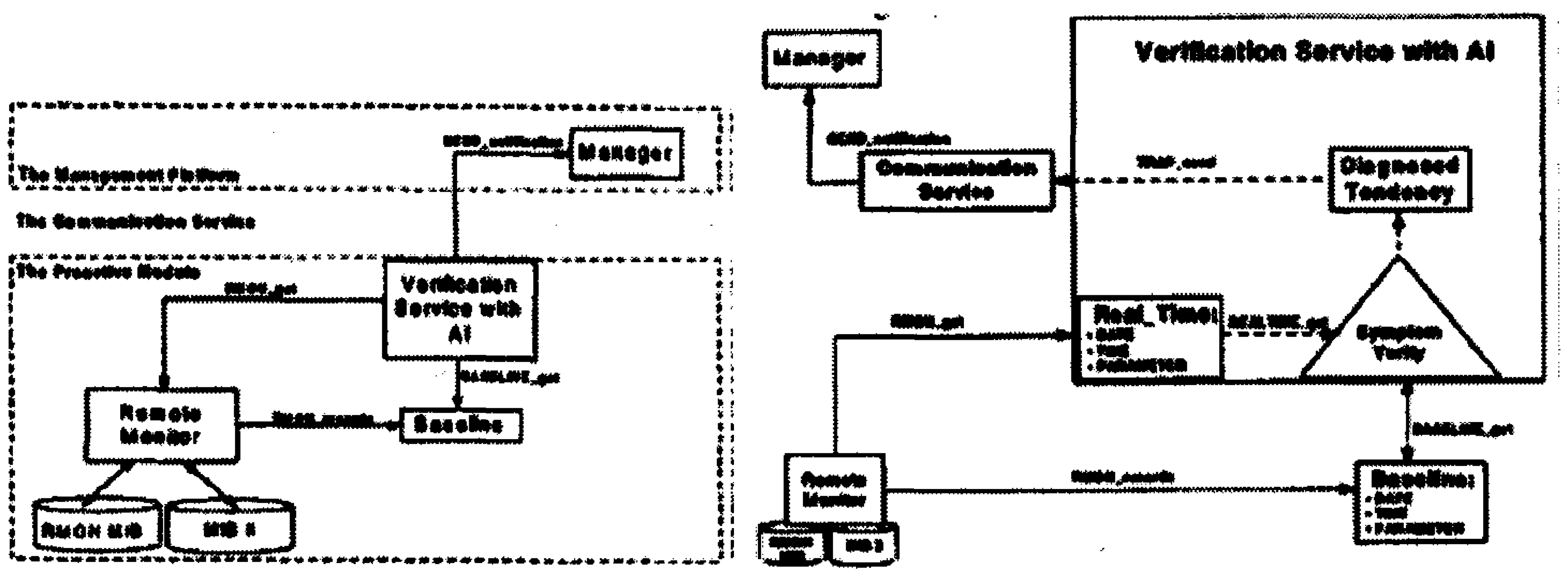


Figure 2.5: Architecture of Proactive Management Component (left) and Verification Schema (right)[WW97]

2.3 Other Approaches

Apart from the prediction technique and concept, failure prediction of large scale system can also be classified into two categories: model-based and data driven[GZL+08]. A model-based prediction method is either an analytical or probabilistic model of the system. The model generates warning messages when it detects any deviation from the model in the system[HE01, HZS01, IYS86, HSM04, TA03, TV99]. For example, in [HE01] Bayesian based algorithm is used to predict disk failures. In [SZL05], a specific analytical method is developed for fast detection of faults in I/O systems.

On the other end, data-driven methods use data mining techniques to do failure prediction. The method attempts to learn and classify failure patterns as rules from historical data rather than generating probabilistic models ahead of time[GLL+07, LZSS06, Sea03]. For example, one laboratory group used statistical learning techniques for failure diagnosis in Internet services[Lab]. Sahoo etal applied association rules to predict failure events in a 350-node IBM cluster[Sea03]. In [GLL$^+$07, LLG$^+$07], a meta learning based method is investigated by combining the merits of various data mining techniques.

The above mentioned prediction techniques mainly focus on static analysis by using one specific method. In this static analysis, the method generates static rule using fixed training set. In spite of effective fault forecasting, they do have drawbacks. Firstly, the source of failures are complex and unlimited, so it is very hard for a single method to detect all faults. Second, in order to obtain efficient failure patterns, the training set should be long (e.g. a year) which means failure prediction will start working after long training period. Thirdly, the method focuses on static analysis where the training set remain unchanged.

2.4 Policy Driven Proactive System Management

Proactive management in policy based systems or networks is a realtively new area of research. There has been very little work done in this research area. In [SML07], a mathematical predictive model is used to calculate estimated time before vertical handover in 4G heterogeneous network. The Architecture for Network Autonomy (ANA) [XTL07] attempts to realize a proactive policy based network management for autonomic communications. In [BB08], reinforcement learning model is used in policy based systems to deal with critical situations in the system. The model takes action in critical situations based on the past experience when any policy is violated. This is a reactive approach. We use this model to develop a protective approach for policy based systems. Detail about the reinforcement learning model is described in Chapter 3.

Chapter 3

Reinforcement Learning in Policy Driven Autonomic Management

3.1 A Policy Based Management System

A policy based management system has been developed by Bahati[BB08], where reinforcement learning is used to determine the best use of a set of active (enabled) policies to meet different performance goals. The learning approach is based on the analysis of past experience of the system and the learning model is used to train the system to dynamically adapt the choice of policy actions for adjusting application and system tuning parameters in response to policy violations.

Reinforcement learning is a learning paradigm[SB98], where an agent learns how to best map situations to actions through trial and error interaction with its environment(Figure 3.1). Reinforcement learning uses a "reward and punishment" approach, where for each action a numeric reward is generated by the agent which indicates the desirability of the agent being in a particular state. The only way to maximize this reward is to discover which action generates the most reward in a given state by trying them. The learning agent must also consider a trade off between whether it should use its current knowledge to select the best action to take (exploit) or to try new, perhaps as yet untried, actions (explore) in order to improve its performance in future.

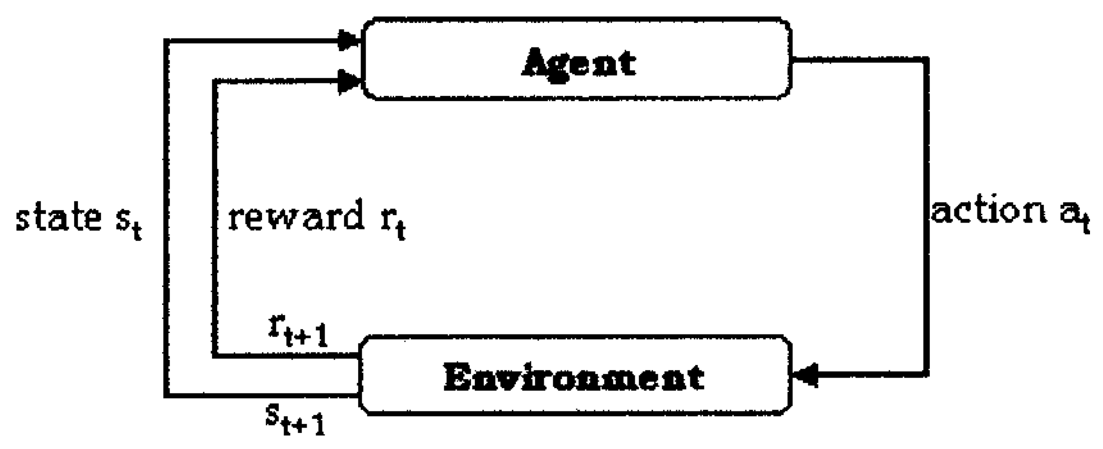


Figure 3.1: The agent-environment interaction in Reinforcement Learning [SB98]

3.2 System Architecture

A detailed architectural view of the adaptive policy driven autonomic management system of Bahati is illustrated in Figure 3.2. The adaptation strategies were evaluated on the behavior of a multi-tiered web server consisting of Linux[Lin], Apache[Pro], PHP[PHP] and MySQL[Dat]. The functionalities of different components of the architecture are described below:

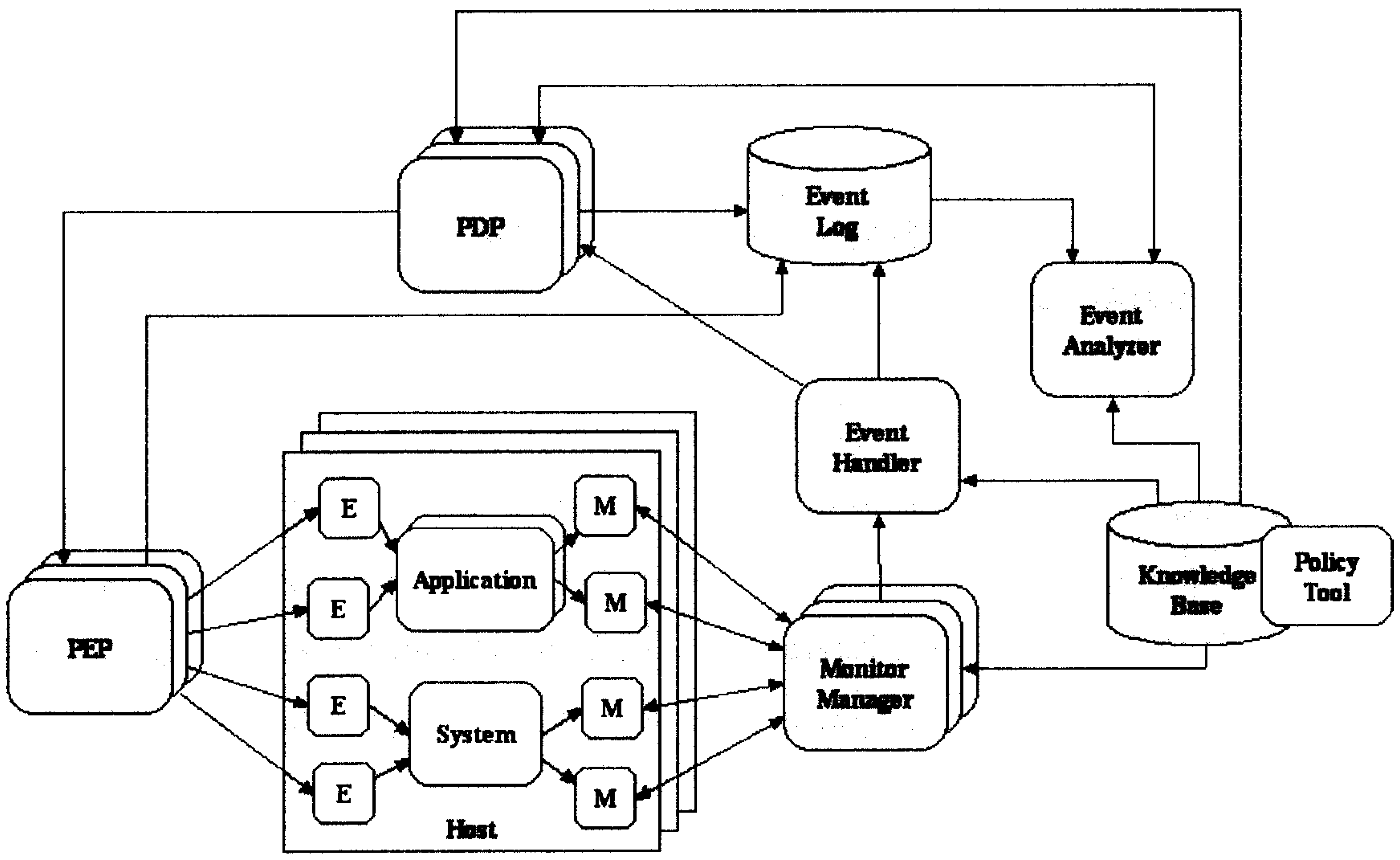


Figure 3.2: System Architecture of Reinforcement Learning Model [BB08]

Knowledge Base: The Knowledge base is a shared repository of management policies. It contains system context for events, rules for making trade offs, rules for making corrective actions, decision metrics and statistics of events. It is used to store information about event handlers and event consumers.

Monitor (M): Monitor components are used to get information about different resources (e.g. Apache, MySQL, PHP) of the system. Performance metrics are collected and then forwarded to the Monitor Manager.

Monitor Manager: The Monitor Manager gathers information from monitors. It also instantiates monitors for certain resources and reconfigures monitors at runtime by retrieving policies from the Knowledge Base. It forwards the monitor events to the Event Handler.

Event Handler: The Event Handler processes events from the Monitor Manager to determine whether the events might correspond to a policy violation. It then forwards the event information to the Policy Decision Point(PDP) and logs the information in the Event Log.

Event Log: The Event Log stores all the system activity for future reference, including information about events from the Event Handler, records decision made by PDP, actions enforced by PEP, etc.

Policy Decision Point(PDP): The PDP processes event messages to determine whether there is any policy violation in the system. It also selects corrective actions based on policy structure and on input from the EventAnalyzer and then forwards the actions to Policy Enforcement Point (PEP) to implement in the system.

EventAanalyzer: The Event Analyzer correlates events with respect to the contexts and performs statistical analysis of the information. It is also the component that is responsible for the reinforcement learning model and uses that model to help the PDP to take actions based on past experience.

Policy Enforcement Point (PEP): The PEP enforces actions suggested by the PDP. It forwards the actions to the appropriate effectors to perform the actual adjustment.

Effector (E): Effectors implement the action in the system; multiple effectors exist for the different resources.

Policy Tool: The Policy tool is an application, implemented in Java. It is used to specify policy rules governing the behaviour of the autonomous management system. It provides a console for observing the behaviour of the web server like memory utilization, response time etc. Policies can be added or modified at runtime using the policy tool.

The implemented autonomic management system involves providing quality of service support local to each host. Each local host has a single PDP which is responsible for system management according to policy specification. In a multi-tired Web-server environment, several components (i.e., a Web server, application server and a database server) may work together to provide a set of services. In that case, all those components running on a single host are managed by a local PDP, which is responsible for managing the whole system as expected by the policy structure or specification.

3.3 Autonomic Management Policies

Policies are used for management decisions in this autonomic management system. In this system, it is assumed that action policies are considered as event-triggered, a action-condition rules[DDLS00]. An event triggers the evaluation of a rule of the form 'if [conditions] then [actions]'. An event is generated when some condition of the state of a system become true. The appropriate action is chosen from the policy specification for that event.

A policy consists of several attributes including one or more conditions and an ordered list of actions to make adjustment of system tuning parameters. A policy rule has mainly three parts: a name, conditions and actions.

```
expectation policy{RESPONSETIMEViolation(PDP, PEP)}
 if(APACHE:responseTime > 2000.0) & (APACHE:responseTimeTREND > 0.0)
 then{AdjustMaxClients(+25) test{newMaxClients < 151} |
      AdjustMaxKeepAliveRequests(-30) test{newMaxKeepAliveRequests > 1} |
      AdjustMaxBandwidth(-128) test{newMaxBandwidth > 255}}
```

Figure 3.3: A sample expectation policy for resolving Apache's response time violation [BB08]

Figure 3.3 is a sample policy rule where *RESPONSETIMEViolation* is the name of the policy rule and it is an expectation type policy rule. This policy rule suggests a number of actions when Apache response time is greater than 2000 ms and the trend of the response time is increasing. The conditions of the policy rule specified in the 'if' clause and the policy actions are defined after 'then' clause; there can be one or more policy actions and an action may have a test associated with it. For example, $AdjustMaxBandwidth(-128)$ is an action which decreases the bandwidth. Before executing the suggested action, the system performs the associated test to determine the validity of the action, i.e., if the action can be executed.

Policies are two types - Configuration policies and Expectation policies. Configuration policies describe those policies that are used to specify how to configure and install applications and services.

```
configuration policy{InstallCPUMonitor(MonitorManager, localhost)}
 if(INSTALL:CPUMonitor = true)
 then{./CPUMonitor test{IsConditionEnabled(CPU:utilization) = true}}
```

Figure 3.4: A configuration policy for installing CPU Monitor[BB08]

Figure 3.4 shows a sample configuration policy specifying that a monitor is required at system start up time to collect performance metrics on CPU utilization. Expectation policies define information used to ensure that operational requirements are met and expected conditions are not violated. Figure 3.3 presents one expectation policy for the Apache web server.

3.4 Modelling Reinforcement Learning

In reinforcement learning, a model gives feedback which guides the learning agent in its interaction with its environment. In a policy-driven autonomic management domain, the actions are determined by the expectation policies that are violated. The performance and behavioral objectives are described by the conditions of the expectation policies. A policy-state-transition model is used based on a set of active policies to create a set of policy-states and the actions of the management system.

The management systems behavior is captured from the Event Log and used to build a policy-state transition graph (Figure 3.5). The structure is built dynamically as the events from different management components are recorded in the log file[BB08]. This information may include Monitor events, violation events, decisions made by PDP, actions made by PEP etc.

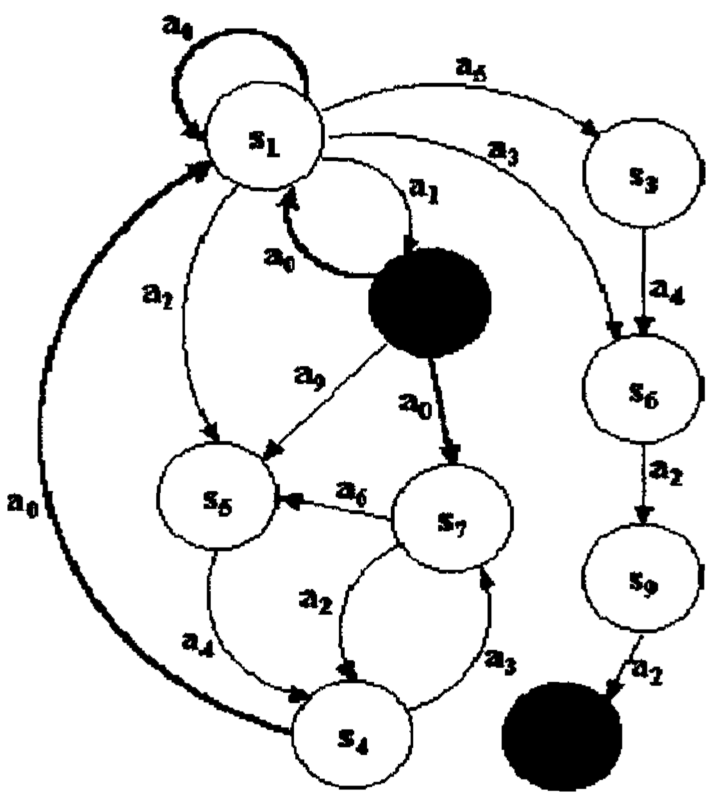


Figure 3.5: Policy-State Transition Graph [BB08]

A state s_i, consists of several attributes as given below:

$$s_i = (t, h, \{m_1, m_2, ..., m_n\})$$

where type (t) defines a state as 'violation' or 'acceptable', a health (h) value derived from the observed values for the state metrics in that policy state and a set of metrics. Each of the state metrics, m_i, corresponds to a unique condition about m_i determined from one of the enabled (active) expectation policies and consists of condition's name, severity (which depends on the observed values of the metrics), and region (which maps the condition severity into appropriate locality as defined by the policy condition). Sample state information is given below where *responseTime* and *responseTimeTrend* are two metrics of the state designated as m_1 and m_2. Metric m_1 has two regions - region $R^1_{m_1}$ represents values greater than 2000 and region $R^2_{m_1}$ represents values less than or equal to 2000. A weight is associated with each region. Region $R^1_{m_1}$ is assigned a weight of 0 and region $R^2_{m_1}$ is assigned a weight of 100. The higher weight value indicates a preferred region, i.e., where the metric is likely not to contributed to a policy violation. A lower weight indicates that the metric will cause a policy violation.

$s_1 = (t_1, h_1\{APACHE : reponseTime > 2000.0, APACHE : responseTimeTrend > 0.0)\})$

3.4.1 Learning by reinforcement

The main function of the PDP is to determine the actions we need to take when any expectation policy is violated. The PDP can request input from the Event Analyzer to help determine the actions. The Event Analyzer implements the reinforcement learning mechanism which uses two strategies - an Exploration Strategy and Exploitation Strategy:

Exploration Strategy:

The learning agent might need to take management decisions in certain situations without depending on the past experience. This could happen for several reasons. For example, this could be part of the agent learning strategy to explore the environment and discover what action can be best for the current situation. It could also be because of the system has no past reference for the current situation.

Exploitation strategy:

The learning agent can suggest an action based on past experience and would update the action reward value. The main objective of the autonomic manager is to try to learn which actions work best in certain situations and trying to move the system towards 'acceptable' states and avoid 'violation' states. In order to achieve that, a numeric reward r value for an action is determined after each time that particular action has been suggested by the learning agent. The action reward value is calculated using following equation:

$$r(s_t) = \sqrt{\sum_{i=1}^{n} m_i.w * [f(R^j_{m_i})]^2}$$

where, s_t is the state visited after taking an action a in the previous state, n is the number of metrics and $m_i.w$ and $R^j_{m_i}$ correspond to, respectively, the weight associated with metric m_i and the region where metric m_i measurement falls.

3.5 Towards Proactive Management

In the next Chapter, we elaborate on the state-graph reinforcement learning model. We describe how the state-graph is formed, what information is collected for each state and how the transitions are formed. We build on this to introduce our prediction approaches and strategy for proactive management.

Chapter 4

Approach to Proactive Management

4.1 System Architecture

A detailed view of our architecture for a policy based proactive management system is depicted in the figure 4.1. The module named **Proactive Decision Point (PRDP)** is integrated with the previous architecture which was developed for research into adaptive policy-driven autonomic management system [BB08]. The PRDP is shown in figure 4.1, as the green colored box inside the EventAnalyzer module.

The PRDP collects system information from the EventAnalyzer and performs the prediction computations for the three different approaches that we have explored (the probability approach, the probability and reward approach and the reward approach); the three approaches are described in detail in following sections. After the prediction computation, the proactive decisions made by the PRDP are transferred to the PDP which carries out the necessary operation to execute the decisions. Finally, the PDP forwards the actions to the PEP to execute the proactive decisions. Before executing any proactive action, the PEP carries out its normal action execution procedure. If the chosen proactive actions do not meet the test conditions that need to be satisfied, then those proactive actions will not be executed. Otherwise, all proactive actions will be executed in the system.

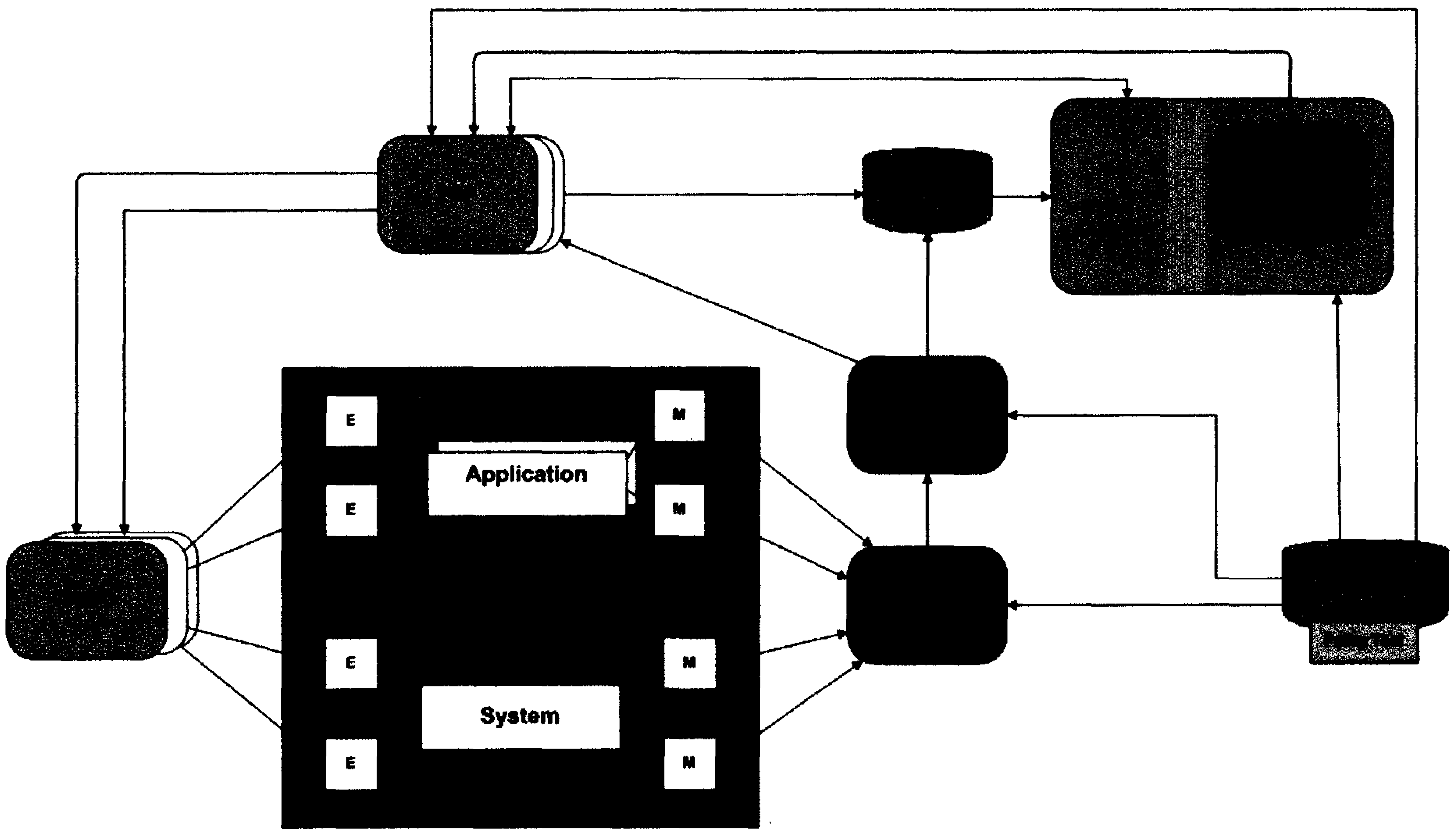


Figure 4.1: Policy Based Proactive Management System

The boxes colored purple have been modified from the previous system architecture (See Figure 3.2) in order to interact with PRDP. All blue color modules remain unchanged. All communication among proactive management modules are shown by green communication links.

4.2 Reinforcement Learning Model

The reinforcement learning model [BB08] is used as the basis for our approach to proactive management. The aim of the reinforcement learning approach within the autonomic manager is to enable the manager to determine the best use of the set of active policies to ensure different performance criteria. The learning approach is based on the analysis of past experience of the system to dynamically adapt the choice of policy actions for adjusting applications and system tuning parameters to handle policy violations.

The EventAnalyzer scans the past information of the system from Eventlog to generate the state graph of the system. The log information is scanned through a specific period of time known as a management cycle. In each management cycle, the state graph information is updated based on the events that occurred in that management cycle.

The state graph captures different policy states of the system with a set of attributes and records transitions among different states after taking policy actions. Table 4.1 shows the information recorded about a single state ($S0$) which is taken from the state graph: State $S0$ has three attributes - a set of metrics (M), a set of transitions (T) and set of actions (A), where $T \in A$.

```
State[0]: ID[0]:

METRICID[0]: REGION[100]:
METRICID[1]: REGION[0]:
METRICID[2]: REGION[100]:
METRICID[3]: REGION[0]:
METRICID[4]: REGION[100]:
METRICID[5]: REGION[0]:

Transition[0]: ACTION[-1]: FREQUENCY[1]: NEXT STATE[0]
Transition[1]: ACTION[-1]: FREQUENCY[1]: NEXT STATE[1]
Transition[2]: ACTION[-1]: FREQUENCY[1]: NEXT STATE[2]

Action[42]: Q[3.64]: AdjustEaccMemSize - 1
Action[47]: Q[3.64]: AdjustKeyBufferSize - 1024000
Action[52]: Q[2.84]: AdjustMaxBandwidth + 64
Action[48]: Q[0.00]: AdjustQueryCacheSize - 1024000
Action[-1]: Q[85.08]: GammaAction
```

Table 4.1: A State Information of State Graph

Detail about these attributes have been discussed in more detail in Chapter 3. Transition attributes indicate the next possible states ($S0, S1, S2$) and the corresponding policy actions (the action identifier is -1 in this example) from state $S0$. Those actions are selected from the set of Actions(A) attribute which is shown in third row of the table. The action list is generated from the set of policies for the state $S0$. The particular action frequency in the transition attribute indicates the number of times that action has been taken by the system. The Action reward value is represented in the third row of the table by the symbol Q. Notice that, the action -1 has the highest reward value since the system has only taken action -1 from the state $S0$.

After each management cycle, the system updates state graph information either by adding a new state or by updating the previous state information which includes an update of the transition frequency and reward value of actions. After a certain number of management cycles the system may have a number of states and transition lists.

A sample transition graph is presented in the figure 4.2 where there are 9 states from $S1$ to $S9$ and a number of transitions. Blue colored states illustrate violation states and indicate that one or more policies were violated when the system was in those particular states. The arrow from one state to another represents the transition. Our approach uses this state graph information, which is generated by the reinforcement learning model to achieve proactive management in a policy based system. Our aim is to predict a policy violation in the system and take corrective action to avoid violations.

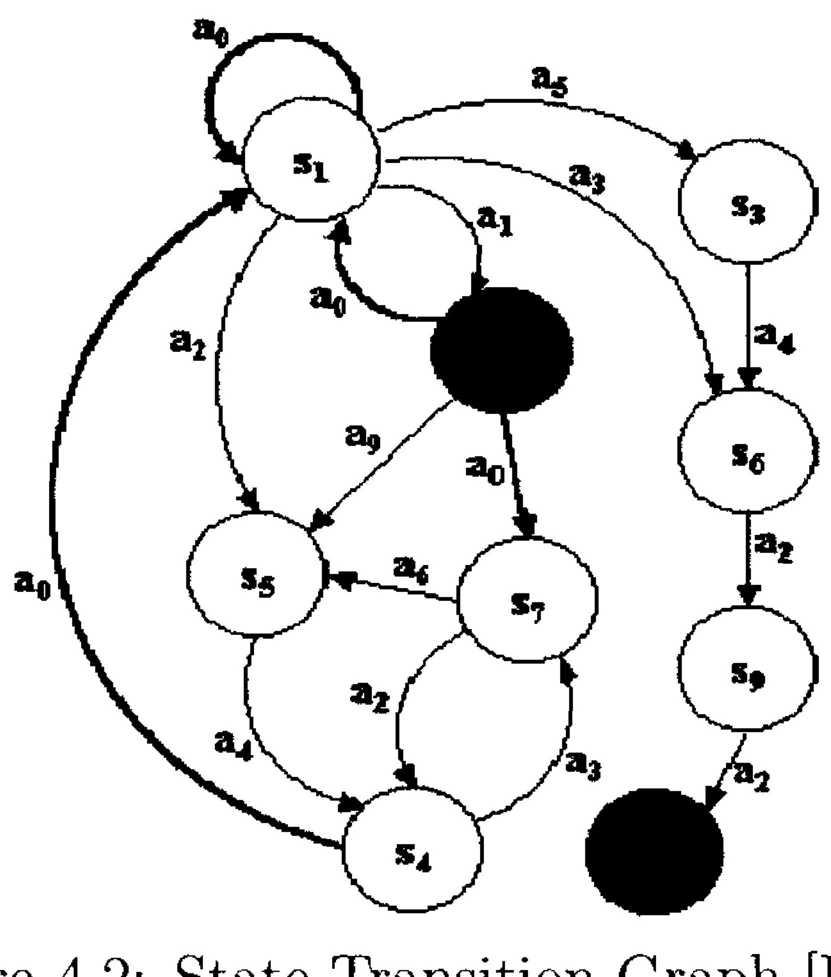


Figure 4.2: State Transition Graph [BB08]

In this policy based system, a policy violation is considered as notion of fault though in reality a 'fault' in the system may result in a policy violation. In this thesis, our aim is to predict policy violation and take action ahead of time in order to achieve proactive management in policy based system. Thesis objective is to avoid policy violations and thus avoid faults. Failures are not considered in thesis. The proactive management part is divided into two parts - Policy Violation Prediction(PVP) and Proactive Action Execution(PAE). In the policy violation prediction part, the main function is predicting the state and which policy will be violated. In the PAE part, the activity depends on the decision from the PVP part. If the PVP predicts that a policy is likely to be violated then it will suggest an action to take which can hopefully take the system to a good or safe state and thus avoiding a policy violation. The PAE will try to execute proactive actions based on the suggestion from PVP.

4.3 Policy Violation Prediction (PVP)

The PVP is responsible for policy violation prediction and determining proactive actions. PVP only suggests proactive actions when it predicts a policy violation or violation state. If the PVP does not predict the violation of a policy then the proactive management component plays no role in that particular management cycle. We consider two different approaches in the way that the PVP performs for prediction - predicting a specific state with a policy violation and predicting only when there might be some policy violation.

For our current work, we have decided to predict two management cycles ahead. This means that if we are currently at management cycle t, then we will try to predict whether there will be any policy violations at management cycle $t + 2$, i.e., management cycle $[t, t+1, t+2]$. In particular, we look at predicting and what will be the system state or the state id at cycle $t+2$. We consider three approaches to do this prediction: the probability approach, an approach that combines probability and reward and an approach based only on reward.

4.3.1 Probability Approach

The probability approach determines the policy violation and state based on the action probability value. The action probability value is calculated from the action frequency value. The action frequency value indicates the number of times that an action has been taken from a particular state. Figure 4.3 illustrates the action probability value approach. The state graph is only a portion of the large graph of the system. There are number of policies are involved in each state of the graph 4.3. For example, expectation policy of Figure 3.3 is involved for violation state $S4$. A complete set of policies are given in Appendix A.Assume that, state $S0$ is the current state at management cycle t, then we have the following information about $S0$ from state graph (Figure 4.3):

$$A_{t+1}^{S0}\ \{\text{ a0, a1, a2 }\}$$
$$F_{t+1}^{S0}\ \{\ 1,\ 2,\ 3\ \}$$
$$NS_{t+1}^{S0}\ \{\text{ S0, S1, S2 }\}$$

where, A is a set of transitions (actions), F is a set of frequencies and NS is a set of next states from current state $S0$. The values of sets A, F and NS are related to each other within state $S0$, that is, if we take action $a1$ then we will go to state $S1$ and the frequency of action $a1$ is 2. The action frequency values are shown in the Figure 4.3 within brackets. In this particular state graph, we have 5 states, where normal or good states are $S0, S1, S2, S3$ and the policy violation state is $S4$ which is colored red. If we find that the system has a high probability to go to state $S4$ then we will take proactive actions to take the system to one of the good states $S0, S1, S2, S3$.

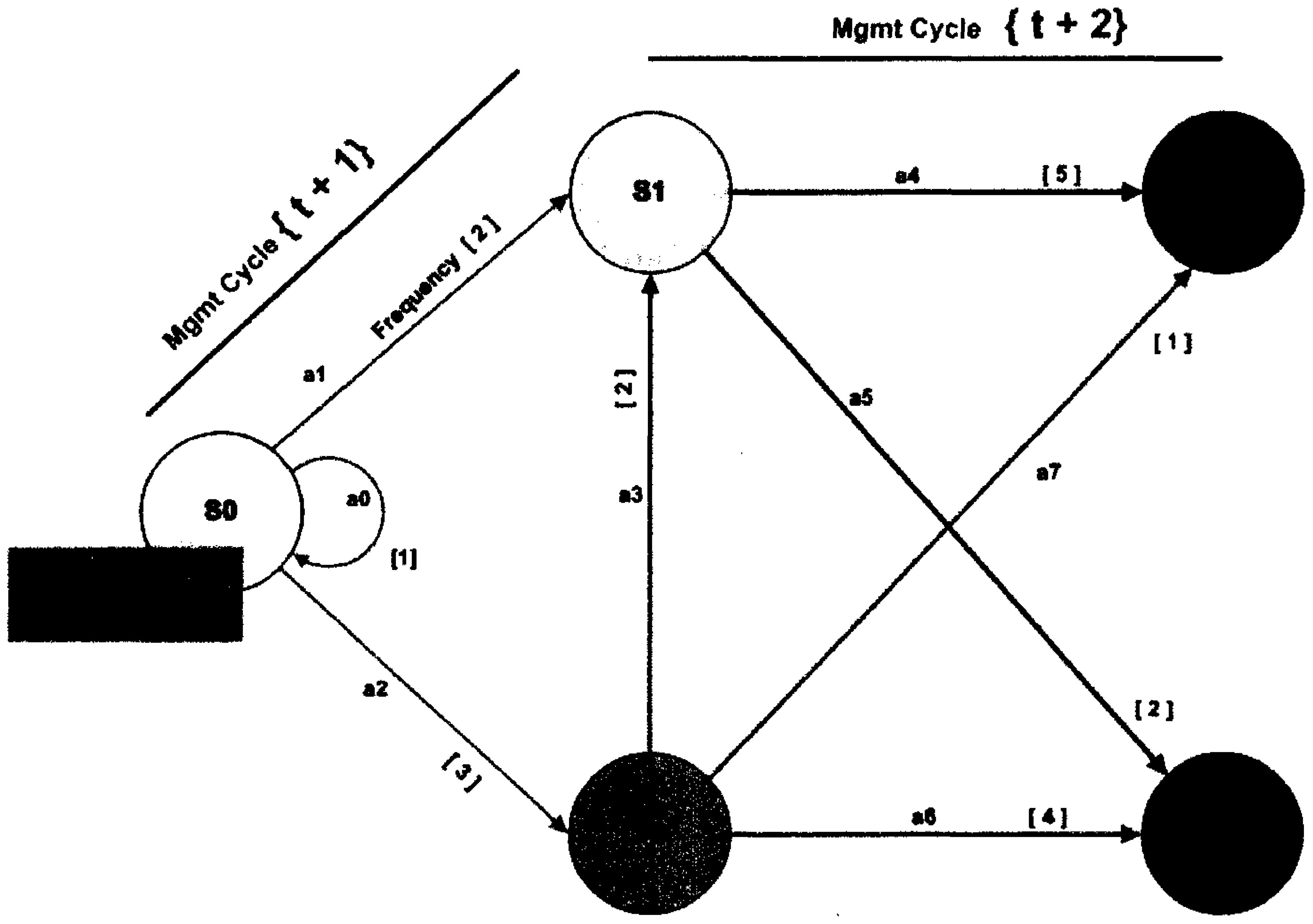


Figure 4.3: Prediction Based on Action Probability Value

In order to find the action probability value, we sum all the action frequency values for a particular state. The total action frequency value for $S0$ is 6 where actions $a0$, $a1$ and $a2$ have frequency values of 1,2 and 3 respectively. So, the probability value for actions $a0$, $a1$ and $a2$ are 0.17, 0.33 and 0.50. The probability values indicate that if the system is in state $S0$ at management cycle t, then the system may take action $a2$ to try to move forward to state $S2$ at management cycle $t+1$ since action $a2$ has the highest probability value.

The action probability value is based on the systems past behavior in that particular state. If any action frequency value is high, then it means the system has been most likely to take that action from that specific state. As we want to predict two cycles ahead (t + 2), we will do a similar action probability calculation for all possible next states ($S0, S1, S2$) from our current state which is $S0$. At t + 1, the state information for $S1$ and $S2$ is as follows:

Attribute	**State S1**	**State S2**
Transition	A^{S1}_{t+2}\{ a4, a5 \}	A^{S2}_{t+2}\{ a3, a6, a7 \}
Frequency	F^{S1}_{t+2}\{ 5, 2 \}	F^{S2}_{t+2}\{ 2, 4,1 \}
Next State	NS^{S1}_{t+2}\{ S3, S4 \}	NS^{S2}_{t+2}\{ S1, S4, S3\}

The probability values from state $S1$ for action $a4$ and $a5$ are 0.71, 0.28. Similarly, the probability values from state $S2$ for actions $a3$,$a6$ and $a7$ are 0.28, 0.57 and 0.14, respectively. This means that from state $S1$ and $S2$ the system is most likely to take action $a4$ and $a6$ to move forward to state $S3$ and $S4$.

Predicting Only Policy Violations

When we want to predict only a policy violation then we multiply probability value of the $t + 1st$ management cycle with $t + 2nd$ management cycle for each action. At the $t + 2$ management cycle, if we end up with a violation state, then we add that action probability value to the violation prediction score. On the other hand, if we do not end up with a violation state then we add that action probability value to the safe prediction score. At the end of the computation, if the violation prediction score is equal or higher than the safe prediction score, then our prediction is that the system is expecting a policy violation at the $t + 2nd$ management cycle. The complete calculation process is illustrated in following table:

Current Sate at time (t)	Action to time (t+1) (A^S_{t+1})	Prob. to time (t+1) [$p1$] (p^S_{t+1})	Next State to time (t+1) (NS^S_{t+1})	Action to time (t+2) (A^S_{t+2})	Probability to time (t+2) [$p2$] (p^S_{t+2})	Next State to time (t+2) [ns] (NS^S_{t+2})	Total Probability Value at t+2 [P] $(p1 * p2)$	Violation Prediction Score at t+2 $\sum P_V$ If ns is a violation state	Safe Prediction Score at t+2 $\sum P_S$ If ns is not a violation state
S0	a0	0.17	S0	a0	0.17	S0	0.029	0.00	0.029
				a1	0.33	S1	0.057	0.00	0.086
				a2	0.50	S2	0.085	0.00	0.171
	a1	0.33	S1	a4	0.72	S3	0.238	0.00	0.409
				a5	0.28	■	0.092	0.092	0.409
	a2	0.50	S2	a3	0.28	S1	0.14	0.092	0.549
				a6	0.58	■	0.29	0.382	0.549
				a7	0.14	S3	0.07	■	■

Table 4.2 : Policy Violation Prediction based on Probability value

Table 4.2 shows the complete prediction computation process. Since the violation prediction(P_V) score (0.382) is lower than the safe prediction (P_S) score (0.619), the PVP is not expecting any policy violation at the $t+2$ management cycle. In the opposite case, the PVP would expect a policy violation and would generate a proactive action list and provide it to the PAE to execute the actions. The algorithm for predicting the policy violation based on only the probabilities is given below:

Algorithm 1: Predicting Policy Violation using Probability Approach

```
Input: Initialize ProactiveAnalysis(s) for s ∈ S, where s is the current state
1: state ← store the current state and its status information
2: $transitionCount_{t+1}$ ← get the total number of transitions for s
3: $totalFrequency_{t+1}$:= total action frequency for all actions from state s.
4: $P_V$:=$P_S$:=$l$:=0 {$l$ is used for how many management cyles ahead to predict}
```

```
5: for $i := 0$ to $transitionCount_{t+1}$ do
6:   $A_{t+1}^{S}[l][i]$:=getProactiveActions(s,i)
7:   $NS_{t+1}^{S}[l][i]$:=getProactiveNextState($A_{t+1}^{S}[l][i]$)
8:   $F_{t+1}^{S}[l][i]$:=getProactiveActionsFrequency($A_{t+1}^{S}[l][i]$)
9:   $p_{t+1}^{S}[l][i]$:=$F_{t+1}^{S}[l][i]/totalFrequency_{t+1}$
10:  $transitionCount_{t+2} \leftarrow$ get the total number of transitions for $NS_{t+1}^{S}[l][i]$
11:  $totalFrequency_{t+2}$:= total action frequency for all actions from state $NS_{t+1}^{S}[l][i]$
12:  for $k = 0$ to $transitionCount_{t+2}$ do
13:    $A_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=getProactiveActions($NS_{t+1}^{S}[l][i]$,k)
14:    $NS_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=getProactiveNextState($A_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$)
15:    $F_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=getProactiveActionsFrequency($A_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$)
16:    $p_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=$F_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]/totalFrequency_{t+2}$
17:    $P$:=$p_{t+1}^{S}[l][i]$ * $p_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$
18:    if $NS_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$ = VIOLATION then
19:      $P_V = P_V + P$
20:    else
21:      $P_S = P_S + P$
22:    end if
23:  end for
24: end for
25: if $P_V \geq P_S$ then
26:   proactiveActionExecution()
27: end if
```

The functions(*getProactiveActions*(),*getProactiveNextState*() etc.) get the respective information and return the associated values. The decision for a state to be considered a VIOLATED STATE is made by searching in the information of the state list which is initialized at the beginning of the algorithm. The function of *proactiveActionExecution*() is described in greater detail when we discuss the PAE (Section 5.3). The parameters of that function are the action ID and action strength (reward value) of the management cycles $t+1$ and $t+2$.

Predicting State and Policy Violation

When we want to predict the state (or state identifier), the computation is different than above. In this approach, we try to find the highest probability value of actions at management cycles $t+1$ and $t+2$ and consider the specific states that could be reached at management cycle $t+2$ from the current state. Using our previous example, if we consider the action with the highest probability value among all actions at management cycle $t+1$ then we get action $a2$ which has the highest probability value(0.50) from the current state $S0$. If we then move forward through $a2$, we move to state $S2$. At management cycle $t+2$, then action $a6$ has the highest probability value(0.58) among all possible actions from state $S2$. So, if we move from the current state $S0$ through $a2$ then take action $a6$, this would take the system to state $S4$ at management cycle $t+2$ where $S4$ is a violation state. In this case, since we are predicting that the system will enter the violation state $S4$, we will generate a list of actions to provide to the PAE. The algorithm for this process is given below:

Algorithm 2: Predicting State and Policy Violation using Probability Approach

```
Input: Initialize ProactiveAnalysis(s) for all s ∈ S, where s is the current state
1: state ← store the current state and its status information for VIOLATION checking
2: $transitionCount_{t+1}$ ← get the total number of transition for s
3: $totalFrequency_{t+1}$:= total action frequency for all action from state s.
4: $l$:=0 {$l$ is used for number of management cyle ahead for prediction}
5: for $i = 0$ to $transitionCount_{t+1}$ do
6:     $A^S_{t+1}[l][i]$:=getProactiveActions(s,i)
7:     $NS^S_{t+1}[l][i]$:=getProactiveNextState($A^S_{t+1}[l][i]$)
8:     $F^S_{t+1}[l][i]$:=getProactiveActionsFrequency($A^S_{t+1}[l][i]$)
9:     $p^S_{t+1}[l][i]$:=$F^S_{t+1}[l][i]/totalFrequency_{t+1}$
```

```
10:     $transitionCount_{t+2} \leftarrow$ get the total number of transition for $NS^{S}_{t+1}[l][i]$
11:     $totalFrequency_{t+2}$:= total action frequency for all action from state $NS^{S}_{t+1}[l][i]$
12:     for $k = 0$ to $transitionCount_{t+2}$ do
13:         $A^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActions($NS^{S}_{t+1}[l][i]$,k)
14:         $NS^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveNextState($A^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$)
15:         $F^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsFrequency($A^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$)
16:         $p^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$:=$F^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]/totalFrequency_{t+2}$
17:         $P$:=$p^{S}_{t+1}[l][i]$ * $p^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][k]$
18:     end for
19:     $transitionCount_{t+2}$:=0
20:     $flag^{1ST\_STAGE}_{t+2} \leftarrow$ get the index of array NS which has highest probability value at
        $t+1st$ management cycle
21:     $flag^{2ND\_STAGE}_{t+2} \leftarrow$ get the index of array NS which has highest probability value
        at $t+2nd$ management cycle
22: end for
23: $transitionCount_{t+1}$:=0
24: if $NS^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][flag^{1ST\_STAGE}_{t+2}] = VIOLATION$ || $NS^{NS^{S}_{t+1}[l][i]}_{t+2}[l+1][flag^{2ND\_STAGE}_{t+2}]$
    = VIOLATION then
25:     proactiveActionExecution()
26: end if
```

We analyze the accuracy of prediction of each method in Chapter 5.

4.3.2 Probability and Reward Approach

This probability and reward approach is similar to the probability approach. The only difference is that the action reward value is multiplied with action probability value in management cycles $t+1$ and $t+2$. Decisions on policy violation and the state are based on the combined values. Figure 4.4 describes the reward value for all actions e.g. 23.34 is the reward value of action $a1$ at state $S0$ in management cycle $t+1$. It also shows the frequency value(2) of action $a1$ which is used to determine the action probability value.

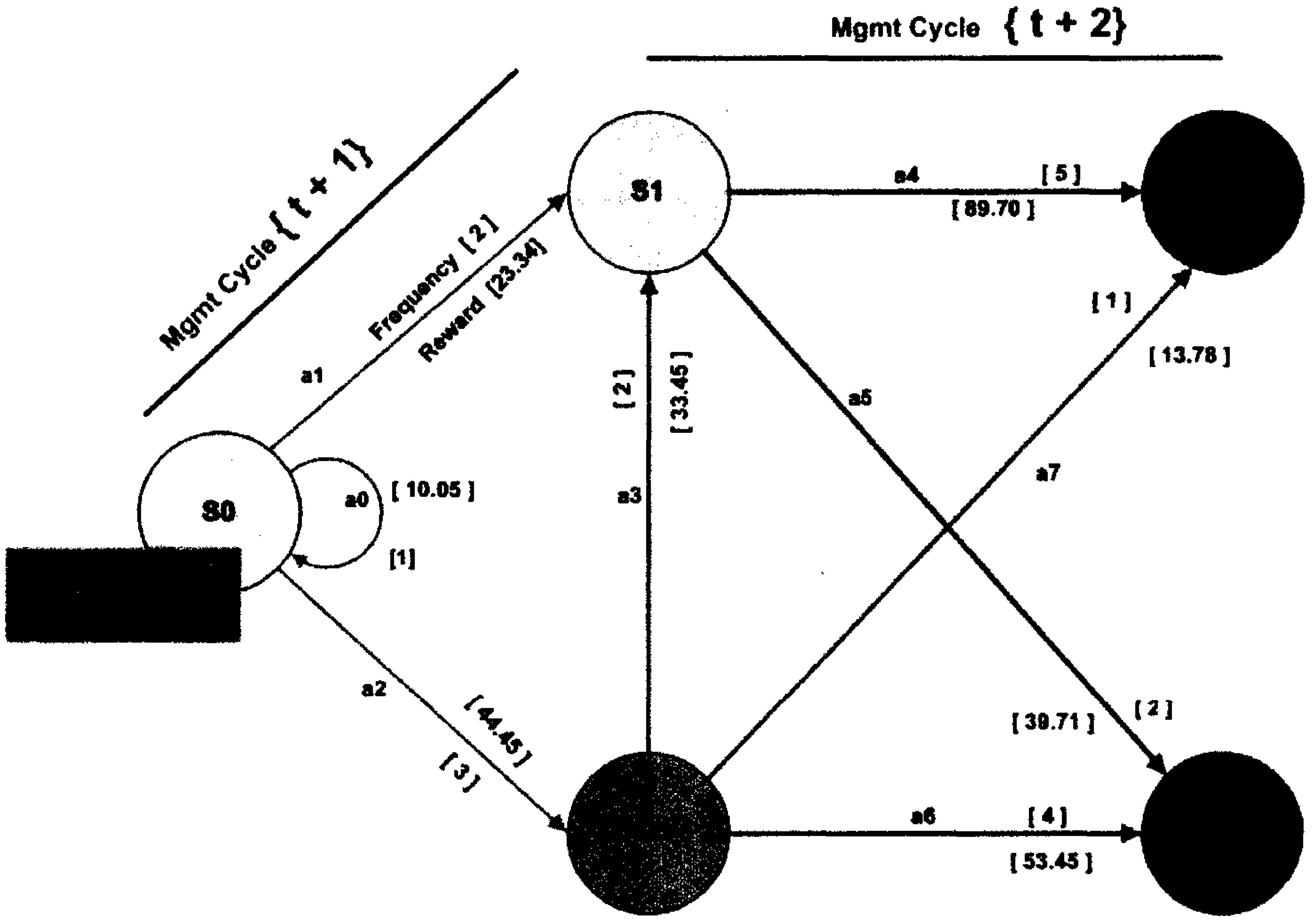


Figure 4.4: Prediction Based on Action Probability and Reward Value

The prediction computation process is almost that same as the probability approach. Table 4.3 shows the complete prediction computation process. Since the violation prediction(PR_V) score (72.03) is lower than the safe prediction (PR_S) score (164.72), the PVP assumes no policy violation at management cycle $t + 2$. The same kind of decisions made here are the same as in the probability-only approach.

Current Sate at time (t)	Action to time (t+1) (A^S_{t+1})	Probability and reward value to time (t+1) [$pr1$] $p^S_{t+1} * r^S_{t+1}$	Next State to time (t+1) (NS^S_{t+1})	Action to time (t+2) (A^S_{t+2})	Probability and reward value to time (t+2) [$pr2$] $p^S_{t+2} * r^S_{t+2}$	Next State to time (t+2) [ns] (NS^S_{t+2})	Total Probability and Reward Value at t+2 [PR] $(pr1 + pr2)$	Violation Prediction Score at t+2 $\sum PR_V$ If ns is a violation state	Safe Prediction Score at t+2 $\sum PR_S$ If ns is not a violation state
S0	a0	0.17 * 10.05	S0	a0	0.17 * 10.05	S0	3.40	0.00	3.40
				a1	0.33 * 23.34	S1	9.40	0.00	12.80
				a2	0.50 * 44.45	S2	23.92	0.00	36.72
	a1	0.33 * 23.34	S1	a4	0.72 * 89.70	S3	72.28	0.00	109.00
				a5	0.28 * 39.71	■	18.81	18.81	109.00
	a2	0.50 * 44.45	S2	a3	0.28 * 33.45	S1	31.58	18.81	140.58
				a6	0.58 * 53.45	■	53.22	72.03	140.58
				a7	0.14 * 13.78	S3	24.14	■	■

Table 4.3: Policy Violation Prediction based on Probability and Reward Value

Algorithms for this approach are virtually the same as the probability-only approach except for the use of both the reward and probability values.

Algorithm 3: Predicting Policy Violations using Probability and Reward Approach

Input: Initialize Proactive Analysis(s) for all s $\in$ S, where s is the current state

```
1: state ← store the current state and its status information
2: $transitionCount_{t+1}$ ← get the total number of transition for s
3: $totalFrequency_{t+1}$:= total action frequency for all action from state s.
4: $PR_V$:=$PR_S$:=$l$:=0 {$l$ is used for number of management cyle ahead for prediction}
5: for $i$ = 0 to $transitionCount_{t+1}$ do
6:     $A^S_{t+1}[l][i]$:=getProactiveActions(s,i)
```

```
7:      $r^S_{t+1}[l][i]$:=getProactiveActionsReward($A^S_{t+1}[l][i]$)
8:      $NS^S_{t+1}[l][i]$:=getProactiveNextState($A^S_{t+1}[l][i]$)
9:      $F^S_{t+1}[l][i]$:=getProactiveActionsFrequency($A^S_{t+1}[l][i]$)
10:     $p^S_{t+1}[l][i]$:=$F^S_{t+1}[l][i]/totalFrequency_{t+1}$
11:     $pr1^S_{t+1}[l][i] = r^S_{t+1}[l][i] * p^S_{t+1}[l][i]$
12:     $transitionCount_{t+2} \leftarrow$ get the total number of transition for $NS^S_{t+1}[l][i]$
13:     $totalFrequency_{t+2}$:= total action frequency for all action from state $NS^S_{t+1}[l][i]$
14:     for $k = 0$ to $transitionCount_{t+2}$ do
15:        $A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActions($NS^S_{t+1}[l][i]$,k)
16:        $r^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsReward($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
17:        $NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveNextState($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
18:        $F^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsFrequency($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
19:        $p^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=$F^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]/totalFrequency_{t+2}$
20:        $pr2^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$=$r^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$*$p^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$
21:        if $NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$ = VIOLATION then
22:           $PR_V = PR_V + pr1^S_{t+1}[l][i] + pr2^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$
23:        else
24:           $PR_S = PR_S + pr1^S_{t+1}[l][i] + pr2^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$
25:        end if
26:     end for
27:  end for
28:  if $PR_V \geq PR_S$ then
29:     proactiveActionExecution()
30:  end if
```

Algorithm 4: Predicting State and Policy Violation using Probability and Reward Approach

Input: Initialize ProactiveAnalysis(s) for all s ∈ S, where s is the current state

```
1: state ← store the current state and its status information
```

```
2: $transitionCount_{t+1} \leftarrow$ get the total number of transition for s
3: $totalFrequency_{t+1}$:= total action frequency for all action from state s.
4: $l$:=0 {$l$ is used for number of management cyle ahead for prediction}
5: for $i = 0$ to $transitionCount_{t+1}$ do
6:    $A^S_{t+1}[l][i]$:=getProactiveActions(s,i)
7:    $r^S_{t+1}[l][i]$:=getProactiveActionsReward($A^S_{t+1}[l][i]$)
8:    $NS^S_{t+1}[l][i]$:=getProactiveNextState($A^S_{t+1}[l][i]$)
9:    $F^S_{t+1}[l][i]$:=getProactiveActionsFrequency($A^S_{t+1}[l][i]$)
10:   $p^S_{t+1}[l][i]$:=$F^S_{t+1}[l][i]/totalFrequency_{t+1}$
11:   $pr1^S_{t+1}[l][i] = r^S_{t+1}[l][i] * p^S_{t+1}[l][i]$
12:   $transitionCount_{t+2} \leftarrow$ get the total number of transition for $NS^S_{t+1}[l][i]$
13:   $totalFrequency_{t+2}$:= total action frequency for all action from state $NS^S_{t+1}[l][i]$
14:   for $k = 0$ to $transitionCount_{t+2}$ do
15:      $A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActions($NS^S_{t+1}[l][i]$,k)
16:      $r^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsReward($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
17:      $NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveNextState($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
18:      $F^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsFrequency($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
19:      $p^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=$F^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]/totalFrequency_{t+2}$
20:      $pr2^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$=$r^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$*$p^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$
21:      $PR := pr2^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k] + pr1^S_{t+1}[l][i]$
22:   end for
23:   $transitionCount_{t+2}$:=0
24:   $flag^{1ST\_STAGE}_{t+2} \leftarrow$ get the index of array NS which has highest probability value at
      $t + 1st$ management cycle
25:   $flag^{2ND\_STAGE}_{t+2} \leftarrow$ get the index of array NS which has highest probability value
      at $t + 2nd$ management cycle
26: end for
27: $transitionCount_{t+1}$:=0
28: if $NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][flag^{1ST\_STAGE}_{t+2}] = VIOLATION \parallel NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][flag^{2ND\_STAGE}_{t+2}]$
    = VIOLATION then
```

```
29:     proactiveActionExecution()
30: end if
```

4.3.3 Reward Approach

The reward approach only considers the action reward value for prediction analysis. The action reward value from management cycle $t+1$ is added to the reward values of actions at management cycle $t+2$. Policy violations are based only on the action reward value. Figure 4.5 describes the reward value for all actions e.g. 23.34 is reward value of action $a1$ at state $S0$ in management cycle $t+1$.

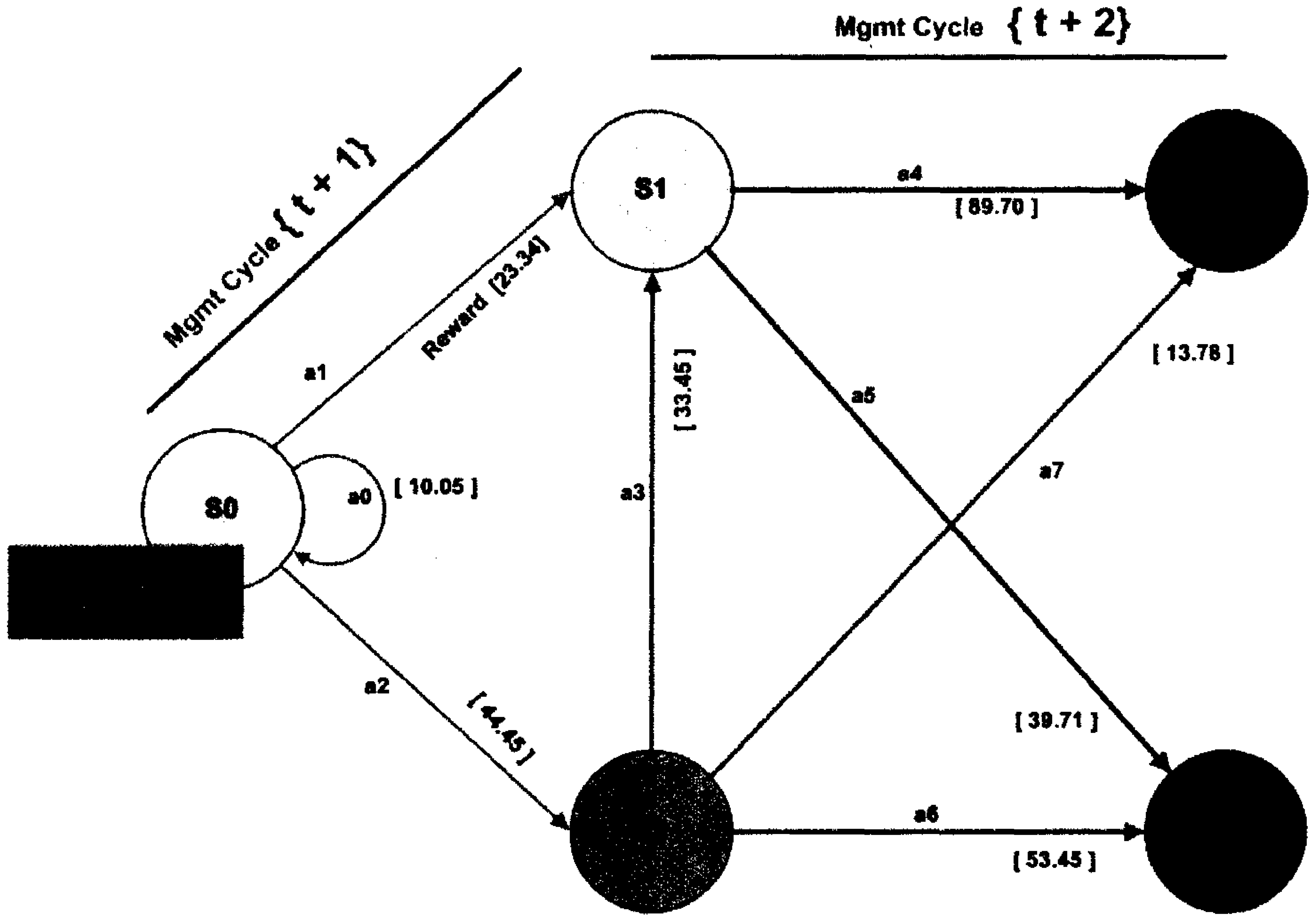


Figure 4.5: Prediction Based on Action Reward Value

The prediction computation process for this approach is illustrated in Table 4.4. Since the violation prediction (R_V) score (160.95) is lower than total safe prediction(R_S) score (357.09), the PVP is not expecting any policy violation at management cycle $t+2$; decisions here are similar to the previous approaches but only depend on the reward value.

Current Sate at time (t)	Action to time (t+1)	Reward value to time (t+1)	Next State to time (t+1)	Action to time (t+2)	Reward value to time (t+2)	Next State to time (t+2)	Total Reward Value at t+2	Violation Prediction Score at t+2	Safe Prediction Score at t+2
		$[r1]$			$[r2]$	$[ns]$	$[R]$	$\sum R_V$	$\sum R_S$
	(A^S_{t+1})	r^S_{t+1}	(NS^S_{t+1})	(A^S_{t+2})	r^S_{t+2}	(NS^S_{t+2})	$(r1 + r2)$	If ns is violation state	If ns is not violation state
S0	a0	10.05	S0	a0	10.05	S0	20.10	0.00	20.10
				a1	23.34	S1	33.39	0.00	53.49
				a2	44.45	S2	54.49	0.00	107.98
	a1	23.34	S1	a4	89.70	S3	113.04	0.00	221.02
				a5	39.71	S4	63.05	63.05	221.02
	a2	44.45	S2	a3	33.45	S1	77.90	63.05	298.92
				a6	53.45	S4	97.90	160.95	298.92
				a7	13.72	S3	58.17	160.95	357.09

Table 4.4 : Policy Violation Prediction based on Reward Value

Algorithms for this approach are similar to the others:

Algorithm 5: Predicting Policy Violation using Reward Approach

Input: Initialize Proactive Analysis(s) for all s ∈ S, where s is the current state

```
1: state ← store the current state and its status information
2: $transitionCount_{t+1}$ ← get the total number of transition for s
3: $R_V$:=$R_S$:=$l$:=0 {$l$ is used for number of management cyle ahead for prediction}
4: for $i = 0$ to $transitionCount_{t+1}$ do
5:     $A^S_{t+1}[l][i]$:=getProactiveActions(s,i)
6:     $r^S_{t+1}[l][i]$:=getProactiveActionsReward($A^S_{t+1}[l][i]$)
7:     $NS^S_{t+1}[l][i]$:=getProactiveNextState($A^S_{t+1}[l][i]$)
```

```
8:    $F_{t+1}^S[l][i]$:=getProactiveActionsFrequency($A_{t+1}^S[l][i]$)
9:    $transitionCount_{t+2}$ ← get the total number of transition for $NS_{t+1}^S[l][i]$
10:   for $k = 0$ to $transitionCount_{t+2}$ do
11:     $A_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k]$:=getProactiveActions($NS_{t+1}^S[i+1][k]$,k)
12:     $r_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k]$:=getProactiveActionsReward($A_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k]$)
13:     $NS_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k]$:=getProactiveNextState($A_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k]$)
14:     if $NS_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k]$ = VIOLATION then
15:       $R_V = R_V + r_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k] + r_{t+1}^S[l][i]$
16:     else
17:       $R_S = R_S + r_{t+2}^{NS_{t+1}^S[l][i]}[l+1][k] + r_{t+1}^S[l][i]$
18:     end if
19:   end for
20: end for
21: if $R_V \geq R_S$ then
22:   proactiveActionExecution()
23: end if
```

Algorithm 6: Predicting State and Policy Violation using Reward Approach

```
Input: Initialize ProactiveAnalysis(s) for all s ∈ S, where s is the current state
1:  state ← store the current state and its status information
2:  $transitionCount_{t+1}$ ← get the total number of transition for s
3:  $l$:=0 {$l$ is used for number of management cyle ahead for prediction}
4:  for $i = 0$ to $transitionCount_{t+1}$ do
5:    $A_{t+1}^S[l][i]$:=getProactiveActions(s,i)
6:    $r_{t+1}^S[l][i]$:=getProactiveActionsReward($A_{t+1}^S[l][i]$)
7:    $NS_{t+1}^S[l][i]$:=getProactiveNextState($A_{t+1}^S[l][i]$)
8:    $transitionCount_{t+2}$ ← get the total number of transition for $NS_{t+1}^S[l][i]$
9:    $totalFrequency_{t+2}$:= total action frequency for all action from state $NS_{t+1}^S[l][i]$
10:   for $k = 0$ to $transitionCount_{t+2}$ do
```

```
11:     $A_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=getProactiveActions($NS_{t+1}^{S}[i+1][k]$,k)
12:     $r_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=getProactiveActionsReward($A_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$)
13:     $NS_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$:=getProactiveNextState($A_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k]$)
14:     $R := r_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][k] + r_{t+1}^{S}[l][i]$
15:   end for
16:   $transitionCount_{t+2}$:=0
17:   $flag_{t+2}^{1ST\_STAGE}$ ← get the index of array NS which has highest probability value at
      $t + 1st$ management cycle
18:   $flag_{t+2}^{2ND\_STAGE}$ ← get the index of array NS which has highest probability value
      at $t + 2nd$ management cycle
19: end for
20: $transitionCount_{t+1}$:=0
21: if $NS_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][flag_{t+2}^{1ST\_STAGE}] = VIOLATION \parallel NS_{t+2}^{NS_{t+1}^{S}[l][i]}[l+1][flag_{t+2}^{2ND\_STAGE}]$
    = VIOLATION then
22:   proactiveActionExecution()
23: end if
```

The accuracy of prediction is important. The result of proactivity depends on the accuracy of the prediction technique. In Chapter 5, we examine the prediction accuracy of our prediction approaches.

Chapter 5

Proactive Action Execution and Result Analysis

Before starting discussion about proactive action and result analysis (PAE), we first examine the accuracy of our prediction algorithms. The next section describes the testing environment and the section after that discusses prediction accuracy.

5.1 Experimental Environment

The experimental environment consists of networked workstations, each connected via an Ethernet switch (see Figure 5.1). A Linux workstation with a 2.0 GHz processor and 2.0 Gigabytes of memory is used to host Apache Web Server, the Knowledge Base and the MySQL database server. An administrative console is used to run the policy tool. Three network workstations are used to run the traffic load tool for generating server requests. The three workstations represent load for gold, silver and bronze users and their service classes.

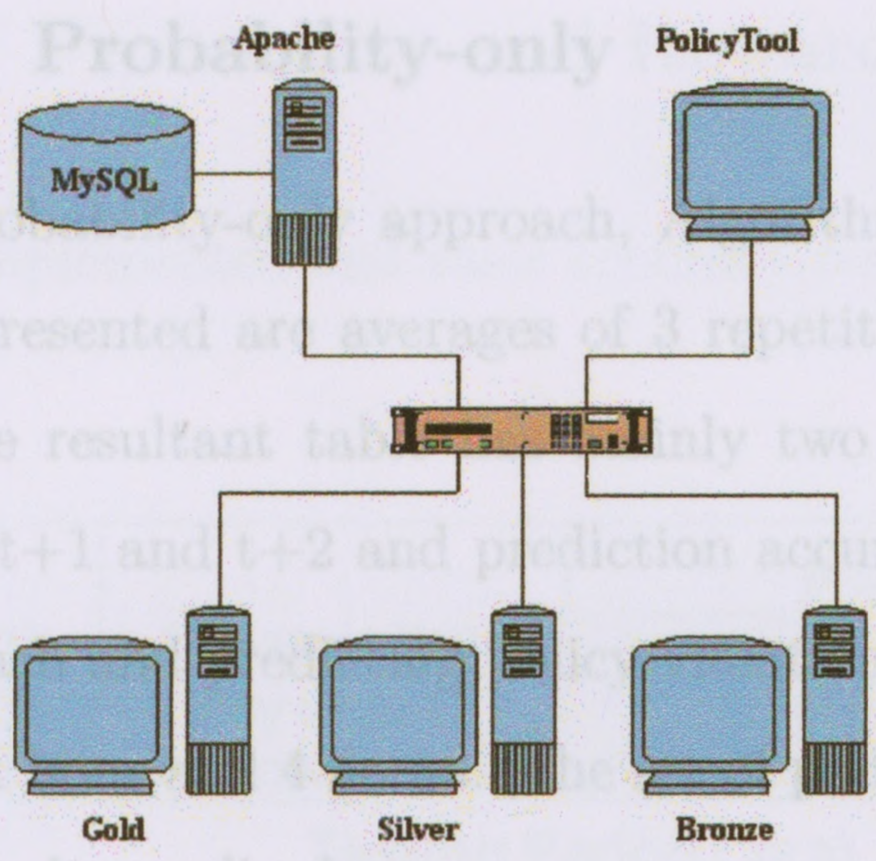


Figure 5.1: Experimental Environment[BB08]

In order to implement service classes a Linux Traffic Controller (TC) Tool [Lin] is used to control the bandwidth associated with the gold, silver, and bronze service classes. Thus, given a ratio of bandwidth for each of the service classes, the bandwidth is shared accordingly; for our experiments this ratio was 85:10:5. A tuning parameter MaxBandwidth determines bandwidth which need to be assigned to each service class.

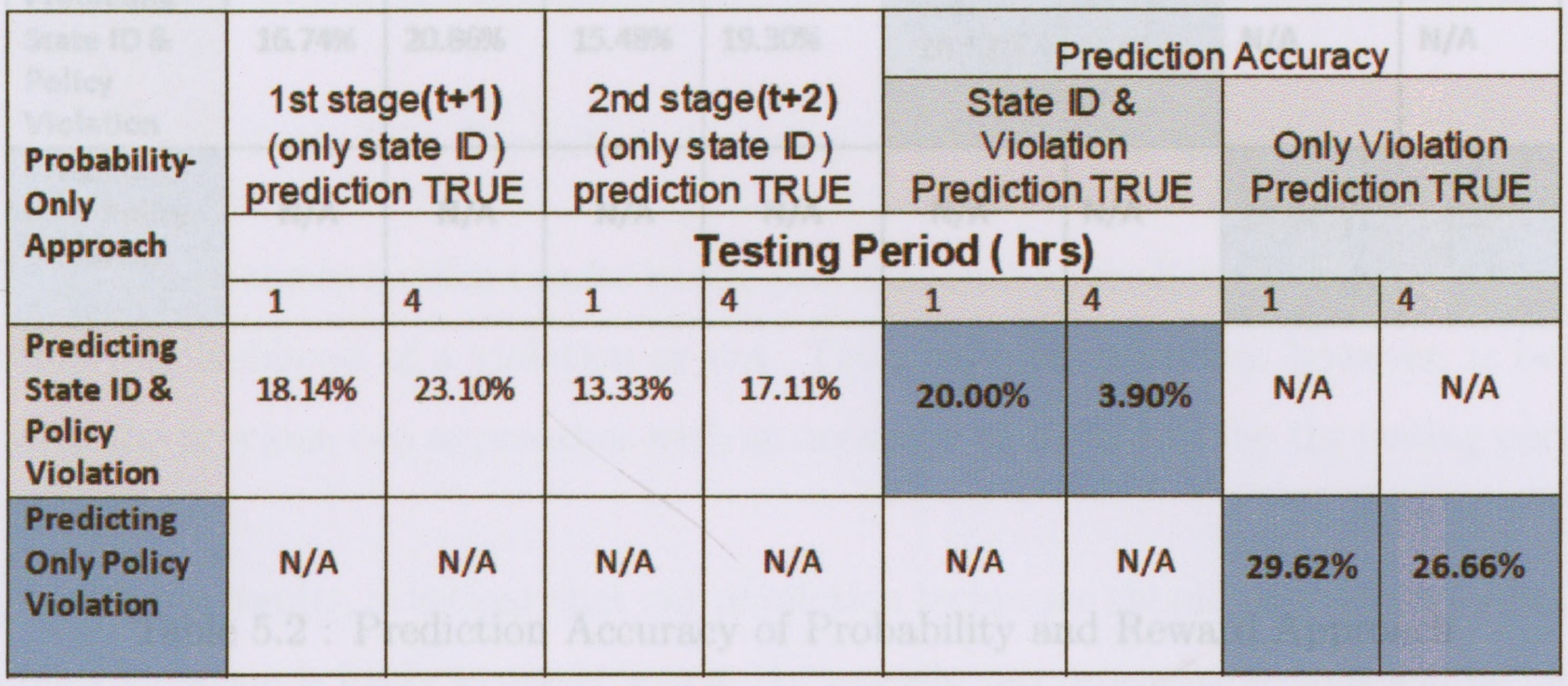
Apache Jmeter [Jme] is used as a traffic load generator. The Jmeter application runs in each of the workstations where each has a dynamic load testing plan. All workstation generate traffic load using the same plan. The load plan contains dynamic requests which create situations where the system resource usage is increased at a significant rate. We ran the experiment for 1 and 4 hour periods. Basically, 1 hour seemed to be a good amount of time to run through a number of requests so that the learning model would have some useful information. Then 4 hours was somewhat arbitrarily chosen.

5.2 Prediction Accuracy

In order to determine the accuracy of prediction techniques, we implemented algorithms discussed in Chapter 4. We consider three scenarios corresponding to the probability-only approach, probability and reward approach and reward-only approach.

5.2.1 Scenario - 1: Probability-only

To test the accuracy of probability-only approach, Algorithm 1 and Algorithm 2 were implemented.The results presented are averages of 3 repetitions. The results are summarized in Table 5.1. The resultant table has mainly two sections - state prediction true at management cycle t+1 and t+2 and prediction accuracy of the predicting state and policy violation approach and predicting policy violation approach. There were two length of testing periods - 1 hour and 4 hours. The set of policies which were used in the experiment are given in the Appendix A.



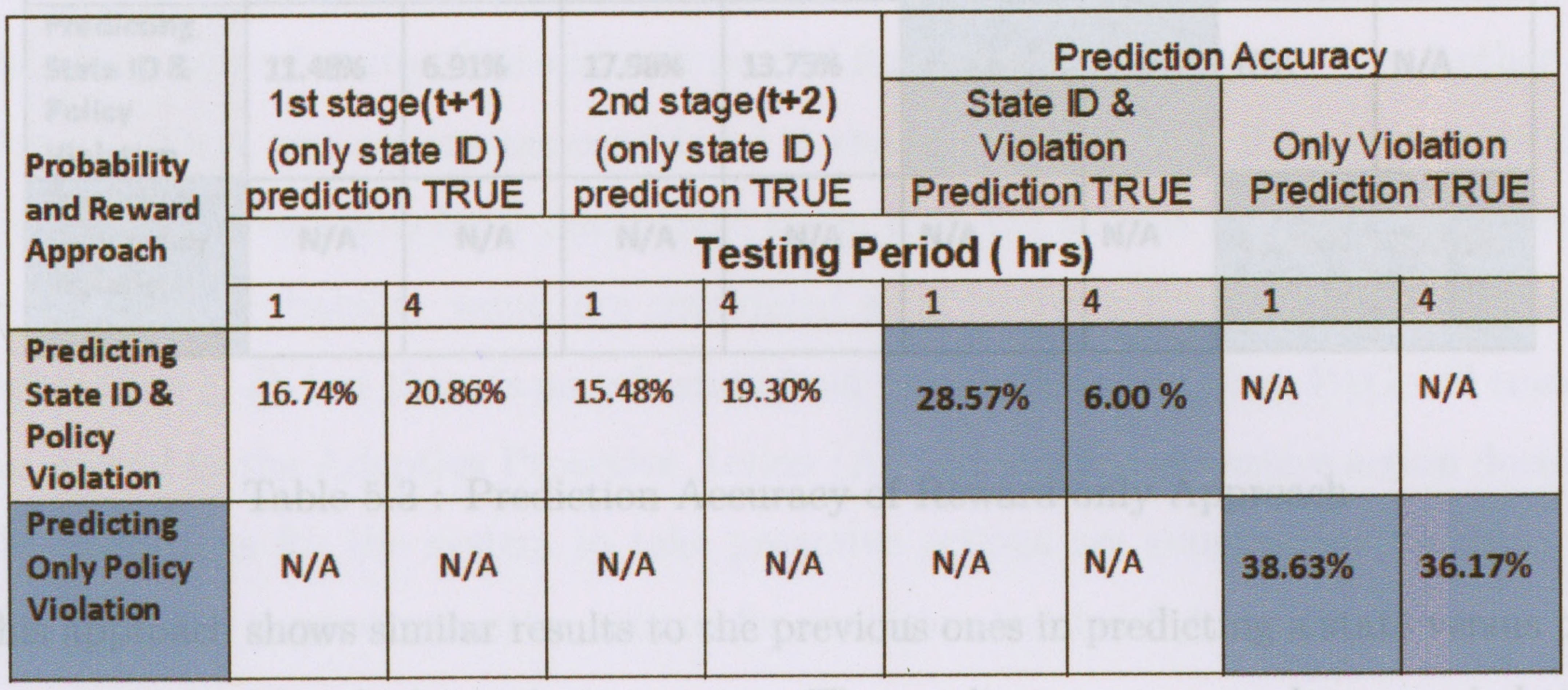
Probability-Only Approach	1st stage(t+1) (only state ID) prediction TRUE		2nd stage(t+2) (only state ID) prediction TRUE		Prediction Accuracy: State ID & Violation Prediction TRUE		Prediction Accuracy: Only Violation Prediction TRUE	
	Testing Period (hrs)							
	1	4	1	4	1	4	1	4
Predicting State ID & Policy Violation	18.14%	23.10%	13.33%	17.11%	20.00%	3.90%	N/A	N/A
Predicting Only Policy Violation	N/A	N/A	N/A	N/A	N/A	N/A	29.62%	26.66%

Table 5.1 : Prediction Accuracy of Probability-only Approach

The first part of the table shows the results of true predictions for predicting a state at management cycle $t{+}1$ and $t{+}2$. In looking at the table, we see that the correct prediction of the state at management cycle $t + 1$ is slightly more than the management cycle at $t + 2$ in both testing periods. The result is not that good - 20.00% in the 1hr testing period, but only 3.90% in the 4hr test period. In contrast, predicting the likelihood of a policy violation or not, shows better results - the accuracy is 29.62% and 26.66% in the 1hr and 4hr test periods, respectively. This is not surprising, since predicting a specific state versus a condition over many states (violation or not) is much harder. Predicting only whether some policy violation is likely to occur has a better prediction accuracy.

5.2.2 Scenario - 2: Probability and Reward

Algorithms 3 and 4 were implemented and their accuracy results measured; the result is shown in Table 5.2.



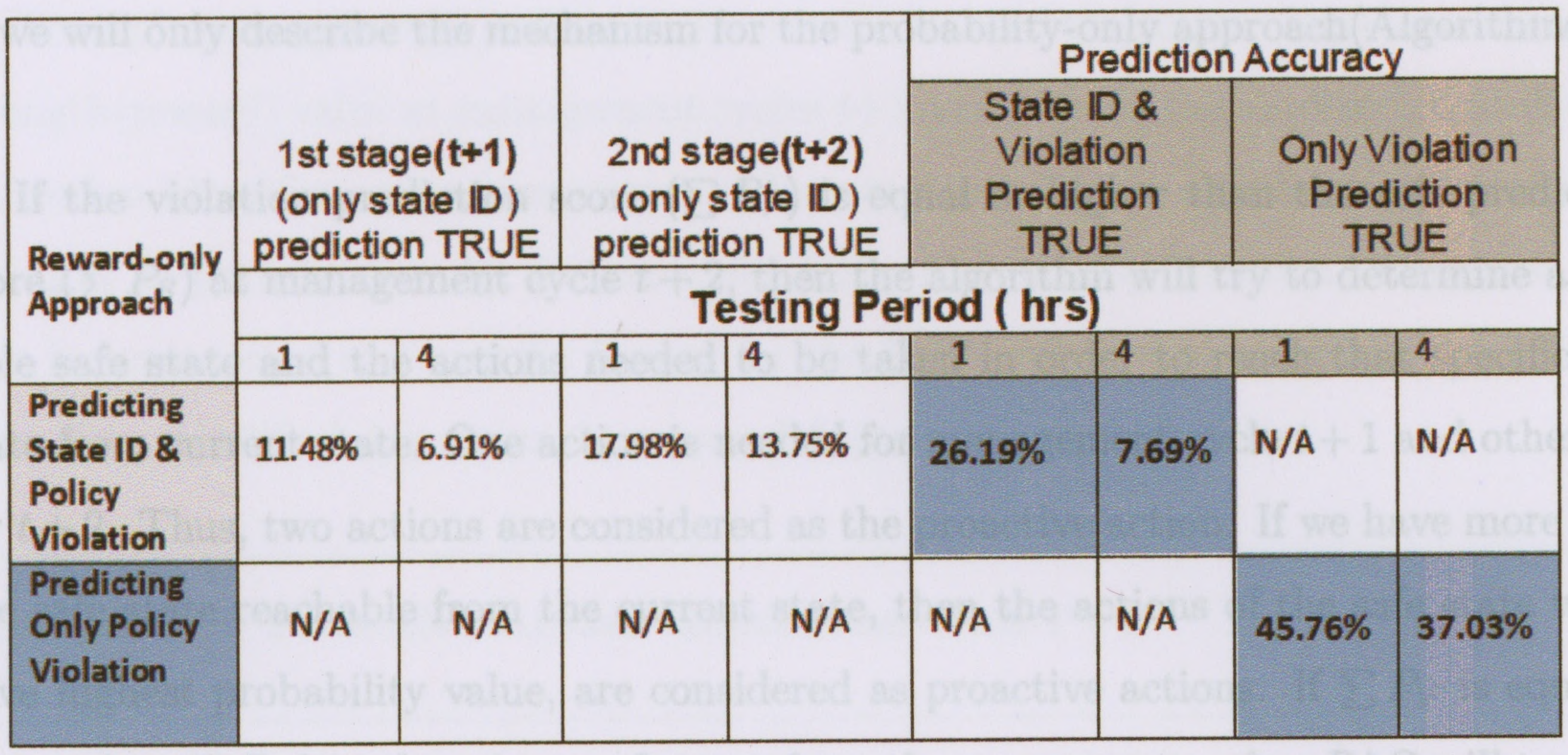
Probability and Reward Approach	1st stage(t+1) (only state ID) prediction TRUE		2nd stage(t+2) (only state ID) prediction TRUE		Prediction Accuracy			
					State ID & Violation Prediction TRUE		Only Violation Prediction TRUE	
	Testing Period (hrs)							
	1	4	1	4	1	4	1	4
Predicting State ID & Policy Violation	16.74%	20.86%	15.48%	19.30%	28.57%	6.00 %	N/A	N/A
Predicting Only Policy Violation	N/A	N/A	N/A	N/A	N/A	N/A	38.63%	36.17%

Table 5.2 : Prediction Accuracy of Probability and Reward Approach

This approach shows results similar to the probability-only approach. Predicting states is harder and results in lower prediction accuracy, though better than the prediction-only approach. Predicting only whether a policy violation occurs or not is better and is better than in the probability-only approach.

5.2.3 Scenario - 3: Reward-only

To test the accuracy of reward-only approach, Algorithms 5 and 6 were implemented and the result is shown in Table 5.3.



Reward-only Approach	1st stage(t+1) (only state ID) prediction TRUE		2nd stage(t+2) (only state ID) prediction TRUE		Prediction Accuracy: State ID & Violation Prediction TRUE		Prediction Accuracy: Only Violation Prediction TRUE	
	Testing Period (hrs)							
	1	4	1	4	1	4	1	4
Predicting State ID & Policy Violation	11.48%	6.91%	17.98%	13.75%	26.19%	7.69%	N/A	N/A
Predicting Only Policy Violation	N/A	N/A	N/A	N/A	N/A	N/A	45.76%	37.03%

Table 5.3 : Prediction Accuracy of Reward-only Approach

This approach shows similar results to the previous ones in predicting a state versus prediction the likelihood of a violation or not. The prediction accuracy, however, is better than in the previous two approaches with an accuracy of 45.76% in the 1hr testing period.

Given the results, it is clear that our prediction technique should only predict whether a policy violation is likely to occur or not. As a result, our proactive actions will be based on this.

5.3 Proactive Action Execution

Proactive action execution (PAE) depends on the decisions of the PVP and the accuracy of the predictions. Based on the analysis of accuracy presented in the previous section, we focus on Algorithms 1,3 and 5. Our approach consists of two parts: Proactive Action Generation and Adaptive Proactive Action.

5.3.1 Proactive Action Generation

In order to determine actions to take, the Proactive Action Generation (PAG) is included in each of Algorithms 1,3 and 5. The decision mechanism is the same for all approach, so we will only describe the mechanism for the probability-only approach(Algorithms 1) .

If the violation prediction score ($\sum P_V$) is equal or higher than the safe prediction score ($\sum P_S$) at management cycle $t+2$, then the algorithm will try to determine a possible safe state and the actions needed to be taken in order to reach that specific safe state from current state. One action is needed for management cycle $t+1$ and other one for $t+2$. Thus, two actions are considered as the proactive action. If we have more than one safe state reachable from the current state, then the actions of the safe state which have highest probability value, are considered as proactive actions. If $\sum P_V$ is equal or higher than $\sum P_S$ but there is no safe state from the current state, then PAG will transfer the control to the Adaptive Proactive Action (APA) part for a proactive action decision. The conditions for the system to take proactive actions are summarized in Table 5.4. The APA will suggest proactive actions for condition number 2,4 and 6.

Condition No.	**Curr. STATE** (Policy Violation)	$\sum P_V \geq \sum P_S$	**Safe state Avail.** (From Curr. State)	**Proac. Actions** Suggestion
1	Yes	Yes	Yes	Yes
2	Yes	Yes	No	Yes
3	Yes	No	Yes	Yes
4	Yes	No	No	Yes
5	No	Yes	Yes	Yes
6	No	Yes	No	Yes
7	No	No	Yes	No
8	No	No	No	No

Table 5.4 : Conditions for PAE to take proactive actions

No proactive actions are suggested for conditions 7 and 8, where neither the current state is a policy violation state nor $\sum P_V \geq \sum P_S$ condition is true. The APA proactive action generation process is described in the next sections.

To implement proactive actions in the system, we need the action identifiers and action strength(reward) value at management cycles $t+1$ and $t+2$. In the modified Algorithm-1, the action identifier and action reward value are passed to *proactiveActionExecution*() function as parameters. In the *proactiveActionExecution*() function, an action message is constructed in a specific format for the PDP and then is passed to the PEP for execution. Modified Algorithm-1, renamed Algorithm7 is given below:

Algorithm 7: Proactive Action Generation with PVP

```
Input: Initialize ProactiveAnalysis(s) for all s ∈ S, where s is the current state
1: state ← store the current state and its status information for VIOLATION checking
2: $transitionCount_{t+1}$ ← get the total number of transition for s
3: $totalFrequnecy_{t+1}$:= total action frequency for all action from state s.
4: $P_V$:=$P_S$:=$l$:=0 {$l$ is used for number of management cyle ahead for prediction}
5: for $i := 0$ to $transitionCount_{t+1}$ do
6:   $A^S_{t+1}[l][i]$:=getProactiveActions(s,i)
7:   $Aid^S_{t+1}[l][i]$:=getProactiveNextState($A^S_{t+1}[l][i]$)
8:   $r^S_{t+1}[l][i]$:=getProactiveActionsReward($A^S_{t+1}[l][i]$)
9:   $NS^S_{t+1}[l][i]$:=getProactiveNextState($A^S_{t+1}[l][i]$)
10:  $F^S_{t+1}[l][i]$:=getProactiveActionsFrequnecy($A^S_{t+1}[l][i]$)
11:  $p^S_{t+1}[l][i]$:=$F^S_{t+1}[l][i]/totalFrequnecy_{t+1}$
12:  $transitionCount_{t+2}$ ← get the total number of transition for $NS^S_{t+1}[l][i]$
13:  $totalFrequnecy_{t+2}$:= total action frequency for all action from state $NS^S_{t+1}[l][i]$
14:  for $k = 0$ to $transitionCount_{t+2}$ do
15:    $A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActions($NS^S_{t+1}[l][i]$,k)
16:    $Aid^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveNextState($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
17:    $r^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsReward($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
18:    $NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveNextState($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
19:    $F^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=getProactiveActionsFrequnecy($A^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$)
20:    $p^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$:=$F^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]/totalFrequnecy_{t+1}$
```

```
21:       $P$:=$p^S_{t+1}[l][i]$ * $p^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$
22:       if $NS^{NS^S_{t+1}[l][i]}_{t+2}[l+1][k]$ = VIOLATION then
23:          Compute the set of VIOLATION states
24:          $P_V = P_V + P$
25:       else
26:          Compute the set of GOOD or SAFE states
27:          get the action ID and action strength (proactiveActionID1,proactiveActionID2,
28:          proactiveActionStrength1,proactiveActionStrength2) to be in this SAFE STATE
             which has highest probability value among all possible good states from Cur-
             rent state
29:          $P_S = P_S + P$
30:          safeCount ← count the number of safe states
31:       end if
32:    end for
33: end for
34: if $P_V \geq P_S$ then
35:    if safeCount > 0 then
36:       proactiveActionExecution(proactiveActionID1,proactiveActionStrength1,
37:       proactiveActionID2,proactiveActionStrength2)
38:       adaptiveProactiveAction(1,proactiveActionID1,proactiveActionStrength1,
39:       proactiveActionID2,proactiveActionStrength2)
40:    else
41:       adaptiveProactiveAction(0,proactiveActionID1,proactiveActionStrength1,
42:       proactiveActionID2,proactiveActionStrength2)
43:    end if
44: else
45:    if $s$ = VIOLATION then
46:       adaptiveProactiveAction(0,proactiveActionID1,proactiveActionStrength1,
47:       proactiveActionID2,proactiveActionStrength2)
48:    end if
```

```
49: end if
```

5.3.2 Adaptive Proactive Action

The Adaptive Proactive Action (APA) part starts working as soon as the system takes its first proactive actions. It is invoked when PAG is not able to generate any proactive actions for a specific state. APA generates a proactive action list to keep track of all proactive actions taken in the past. This information includes the action identifier, the action reward value and its frequency.

The proactive action list is updated every time PAG generates new proactive actions. Since PAG generate the actions based on the probability and reward value, the generated proactive actions may not actually achieve our target e.g. PAG may generate a proactive action only because of their high probability or reward value which may throw the system in a violation state or may have no impact in critical situation. During the creation of the proactive action list, APA filters actions based on conditions and excludes those actions which have no effect (Gamma Actions) or which increase the resource usage limit when the system is experiencing low resource usage.

PAG uses the APA for proactive actions conditions 2,4 and 6 (see Table 5.4). The APA suggests two proactive actions, one for management cycle $t + 1$ and for cycle $t + 2$, from the proactive action list based on the highest frequency value. The logic behind this is that proactive actions which have been executed most of the time in the past are likely to be useful in critical situations. The APA algorithm is given below:

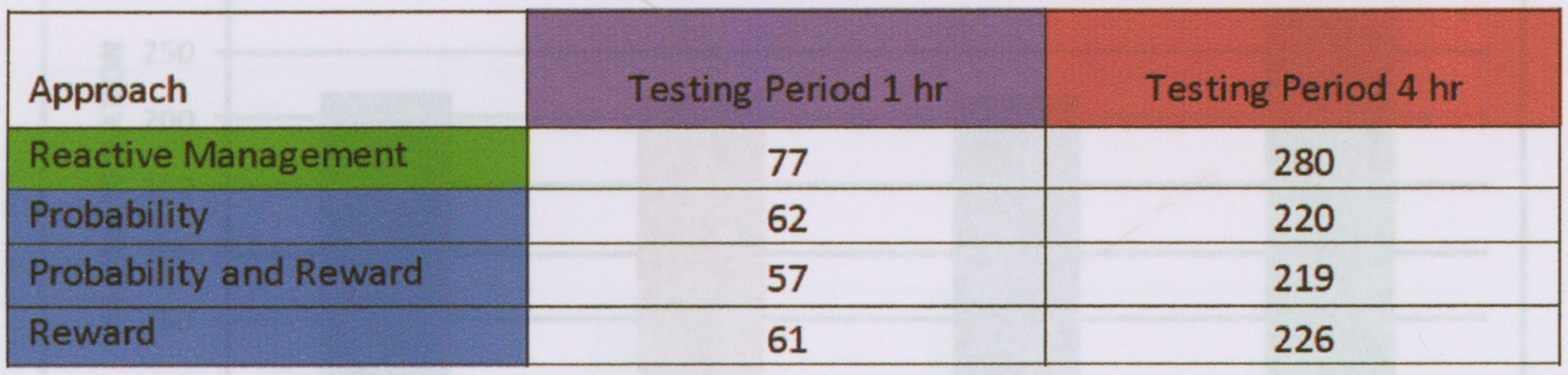
Algorithm 8: Adaptive Proactive Action

```
adaptiveProactiveAction($update$,$proactiveActionID1$,$proactiveActionStrength1$,
$proactiveActionID2$,$proactiveActionStrength2$)
 1: $countProactiveActionList$ ← get the number of proactive action from List
 2: $negateAction$ ← list of action which increase the resource usage and Gamma action
 3: if update = 1 then
 4:     add or update action ID($proactiveActionListID$), frequency($proactiveActionListFrequency$)
        and strength($proactiveActionListStrength$) to proactive action arrays
 5:     $pointerFirstProactiveAction$ ← get the index of proactive action ID ($proactiveActionListID$)
        list array which has highest action frequency value
 6:     $pointerSecondProactiveAction$ ← get the index of proactive action ID ($proactiveActionListID$)
        list array which has second highest action frequency value
 7:     $countProactiveActionList$ ← get the number of proactive actions in the list
 8: else
 9:     if $countProactiveActionList$ > 0 and update = 0 then
10:         proactiveActionExecution($proactiveActionListID[pointerFirstProactiveAction]$,
11:         $proactiveActionListStrength[pointerFirstProactiveAction]$,
12:         $proactiveActionListID[pointerSecondProactiveAction]$,
13:         $proactiveActionListStrength[pointerSecondProactiveAction]$)
14:     end if
15: end if
```

Algorithms 7 and 8 have been implemented and the experimental results are discussed in the next section where we compare policy based reactive management and proactive management approaches.

5.4 Comparison of Proactive Approaches

In this section, we compare different proactive approaches with reactive management. The experiments are done for each of the 1hr and 4hr testing periods for both the reactive management approach and the different proactive management approaches. The traffic load varies each experiment and so we run each experiment 3 times and average the result. Table 5.5 shows the average results. Since our aim is to reduce policy violations, we counted the number of policy violations that occurred during the testing period. When we use reactive management, we have 77 and 280 policy violations in the 1hr and 4hr time periods, respectively. But when we use the different approaches for proactive management, the number of policy violations is reduced to 60 and 221 around in 1hr and 4hr time periods. The different proactive approaches produce very similar results. The combined probability and reward approach shows somewhat better results when compared to the other proactive approaches.



Approach	Testing Period 1 hr	Testing Period 4 hr
Reactive Management	77	280
Probability	62	220
Probability and Reward	57	219
Reward	61	226

Table 5.5 : Number of Policy Violation by different approach

Figures 5.2 and 5.3 show a graph for the 1hr and 4hr test periods where the x axis of the graph represents the different management approaches and the y axis represents the number of policy violations that occurred.

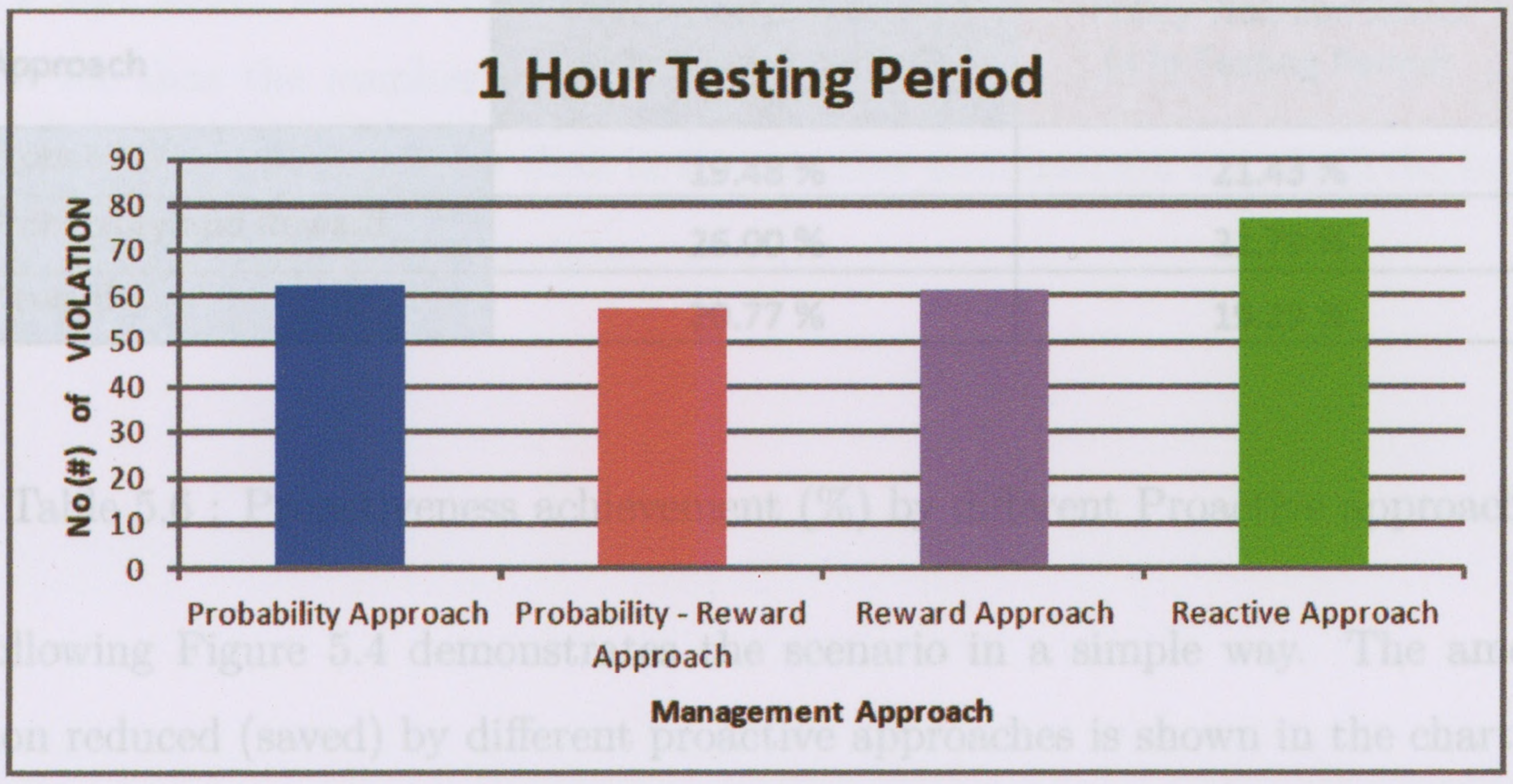


Figure 5.2: Number of Policy Violations by Different Approaches (1hr Test Period)

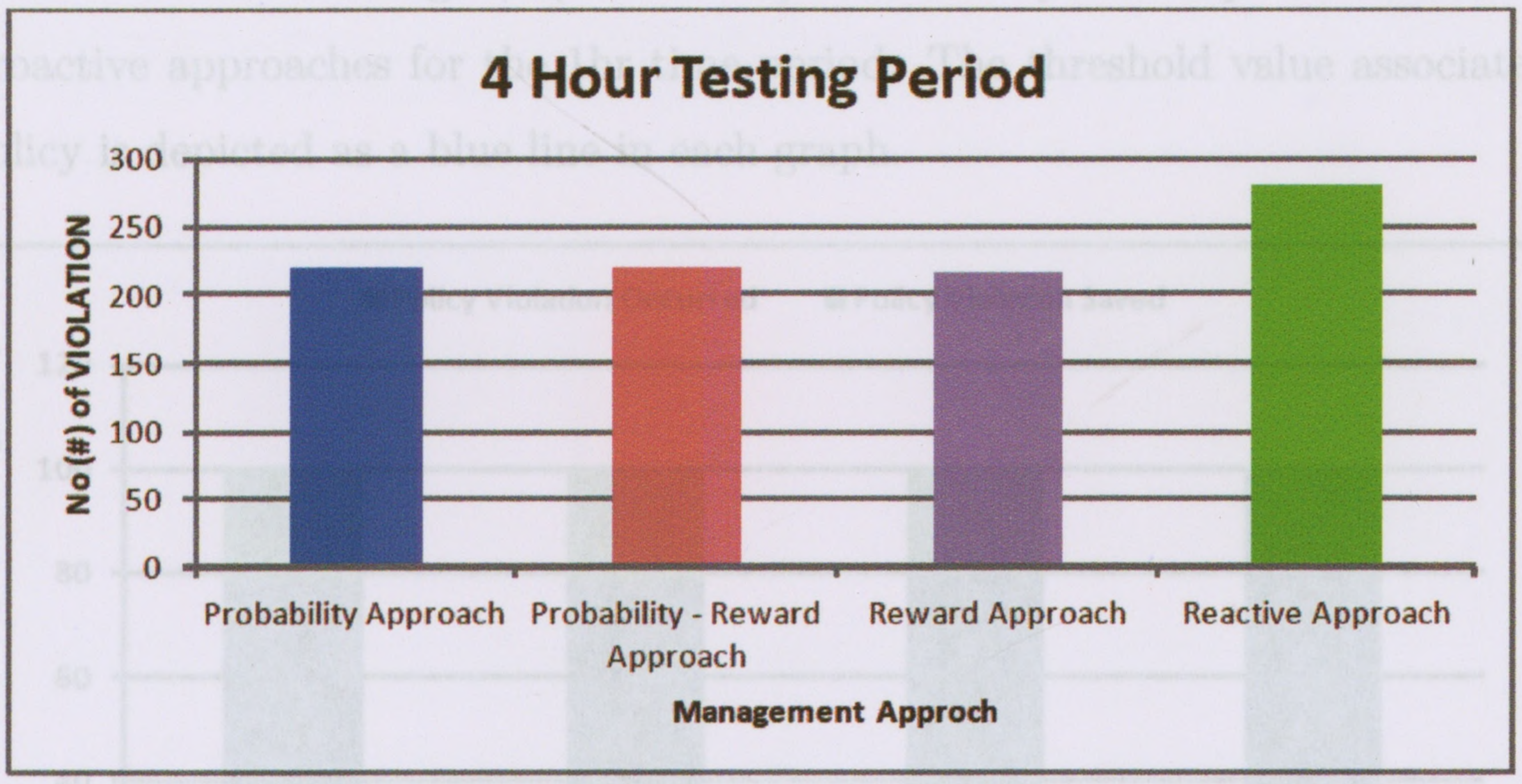


Figure 5.3: Number of Policy Violation by Different Approach (4hr Test Period)

In order to calculate the proactiveness achievement of different proactive management approaches on top of reactive management approach, the following table is given here. Though prediction accuracy of the reward approach is best among all three approach, when we implement proactive action; it is found that proactiveness of probability and reward approach achievement is slightly higher (21.79 %) than other two approach.

Approach	Policy Violation Saved (1 hr Testing Period)	Policy Violation Saved (4 hr Testing Period)
Probability	19.48 %	21.43 %
Probability and Reward	26.00 %	21.79 %
Reward	20.77 %	19.29 %

Table 5.6 : Proactiveness achievement (%) by different Proactive approach

The following Figure 5.4 demonstrates the scenario in a simple way. The amount of violation reduced (saved) by different proactive approaches is shown in the chart.

CPU utilization (Scenario-1) and the Apache response time (Scenario-2) are the two main metrics used to specify the policies in the experiments. Figures(5.5 to 5.8) show the CPU utilization and Figures (5.9 to 5.12) show the Apache response time for each of the proactive approaches for the 1hr time period. The threshold value associated with the policy is depicted as a blue line in each graph.

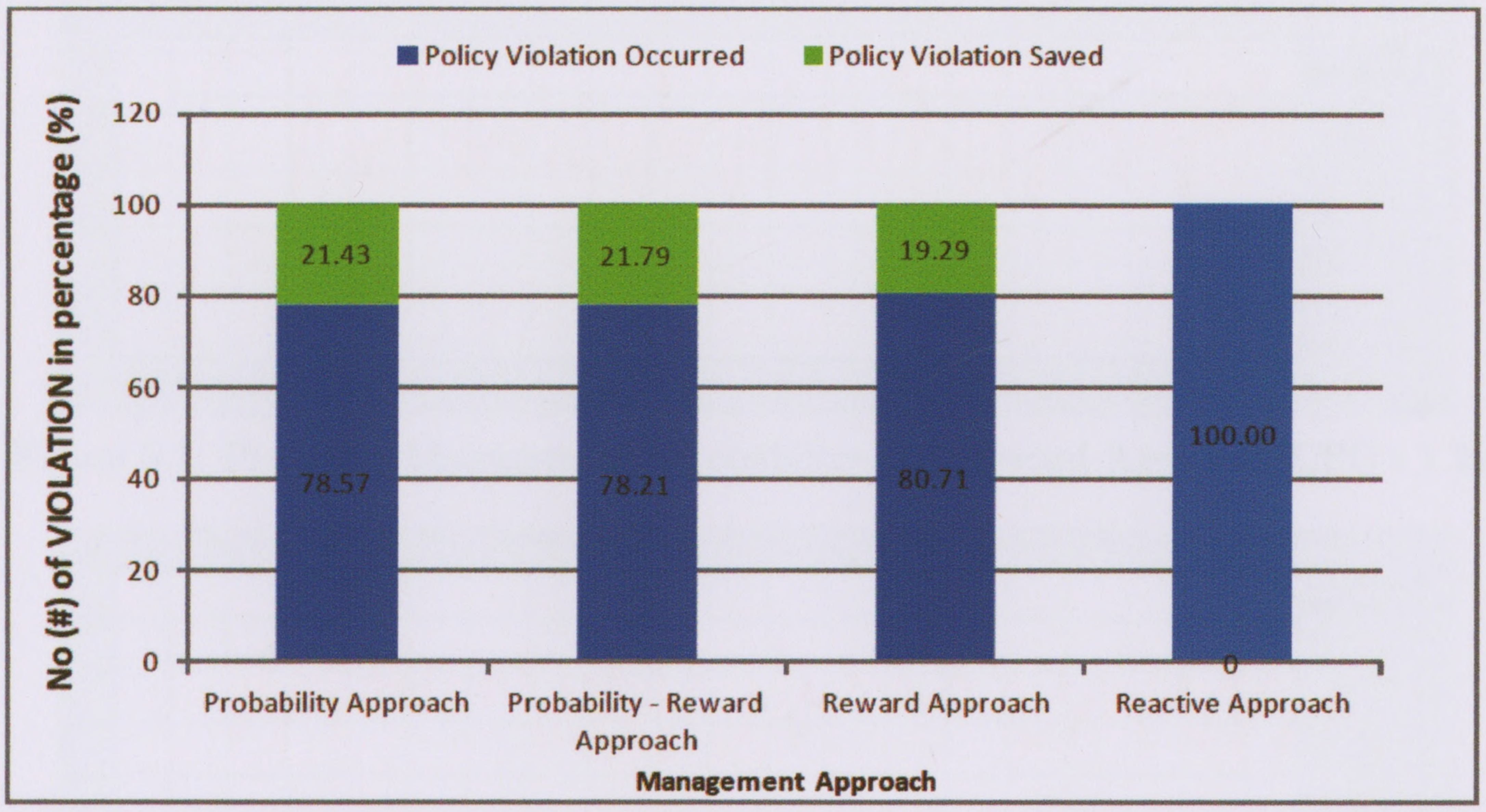


Figure 5.4: Policy Violations Saved by Different Proactive Management Approaches

In the graphs(see 5.4.1 and 5.4.2), the x axis represents the time in seconds and the y axis represents value of CPU utilization and the Apache response time in milliseconds (ms). Peaks above the threshold line indicate policy violations. Observing the graphs one can see that the number of high peaks above the threshold value in the proactive management approaches is less than in the reactive management approach for both CPU utilization and Apache response time.

5.4.1 Scenario - 1: CPU Utilization [Test Period: 1 hr]

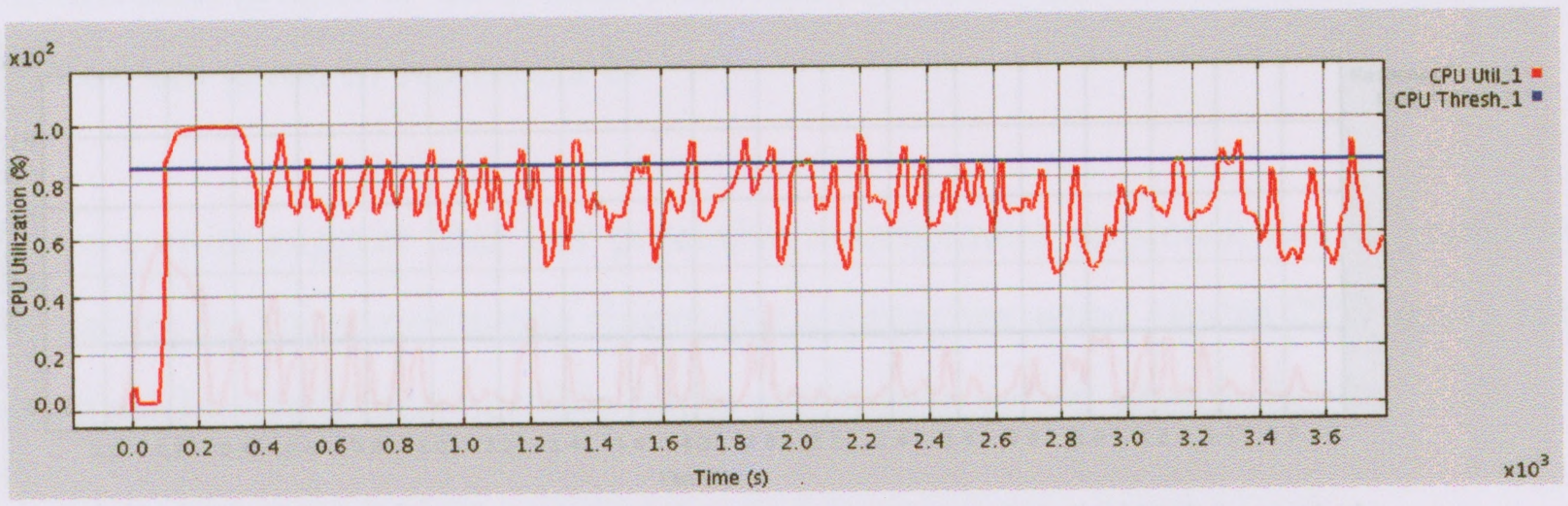


Figure 5.5: Reactive Management Approach(CPU - 1 hr)

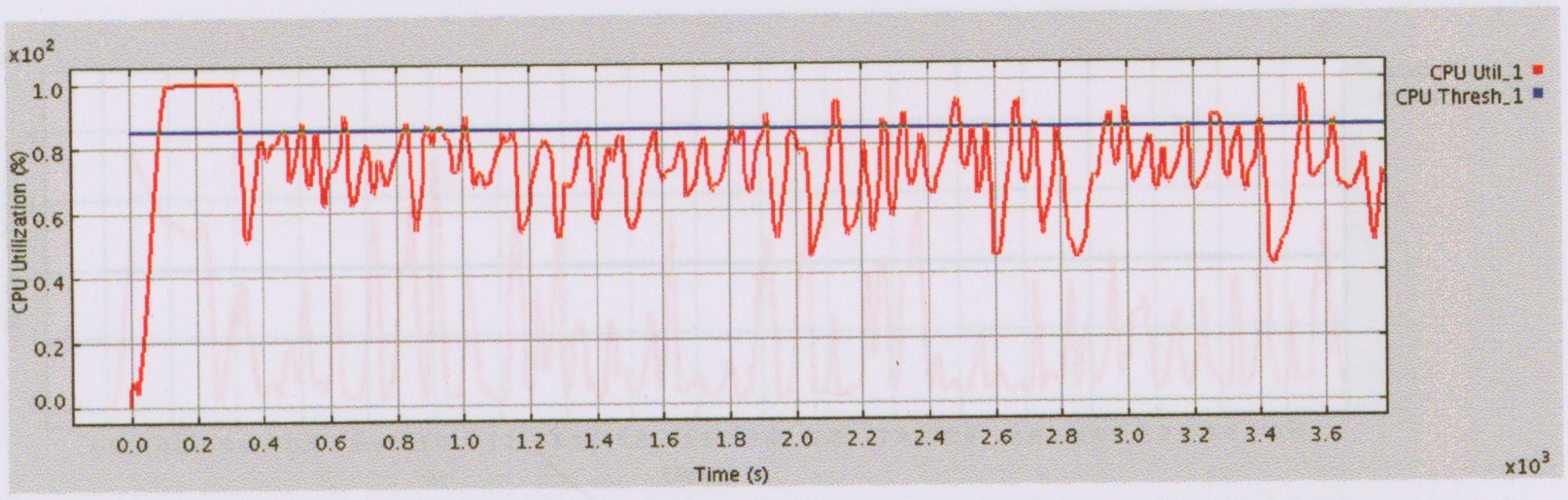


Figure 5.6: Proactive Management: Probability Approach(CPU - 1 hr)

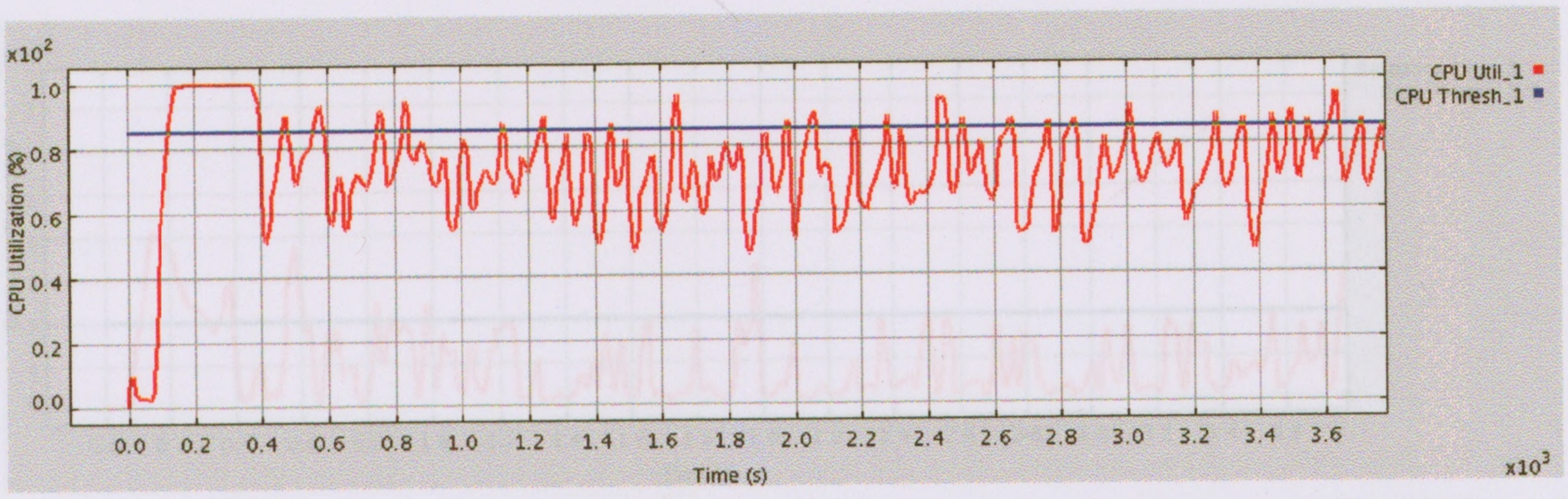


Figure 5.7: Proactive Management: Probability and Reward Approach(CPU - 1 hr)

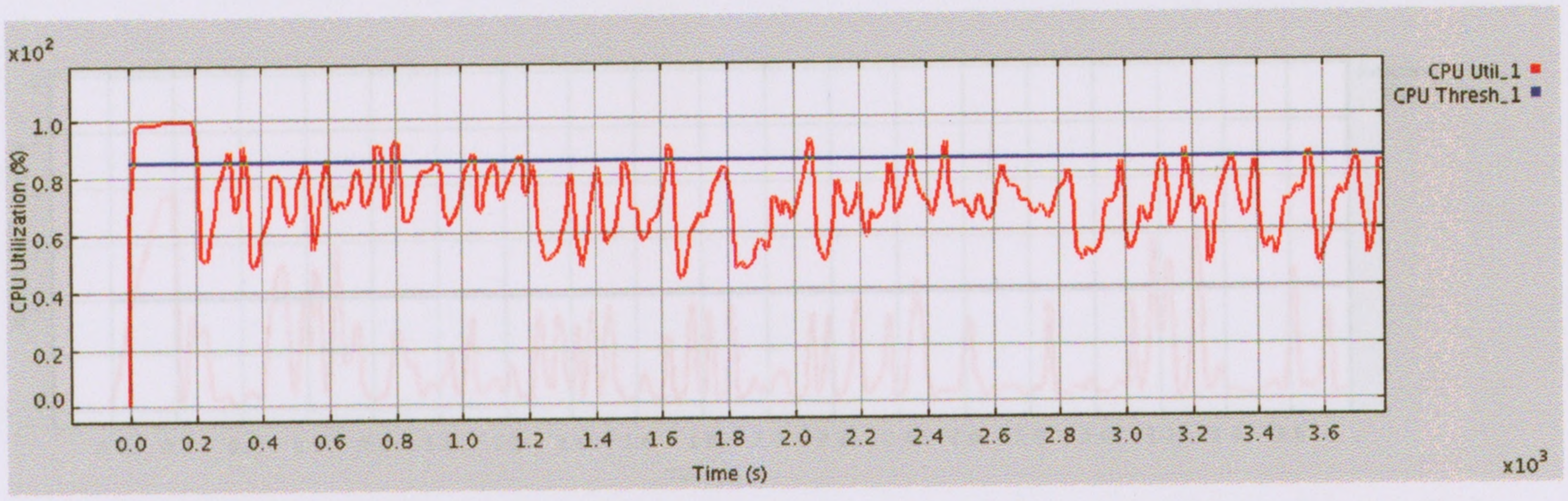


Figure 5.8: Proactive Management: Reward Approach(CPU - 1 hr)

5.4.2 Scenario - 2: Apache Response Time [Test Period: 1 hr]

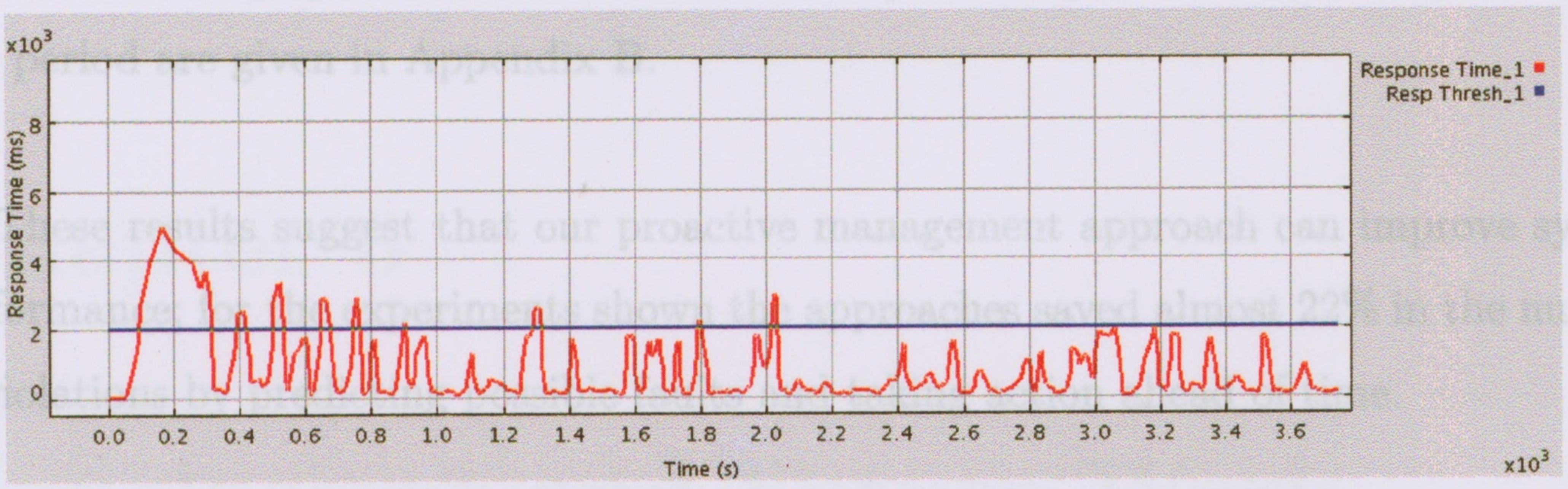


Figure 5.9: Reactive Management Approach(Apache - 1 hr)

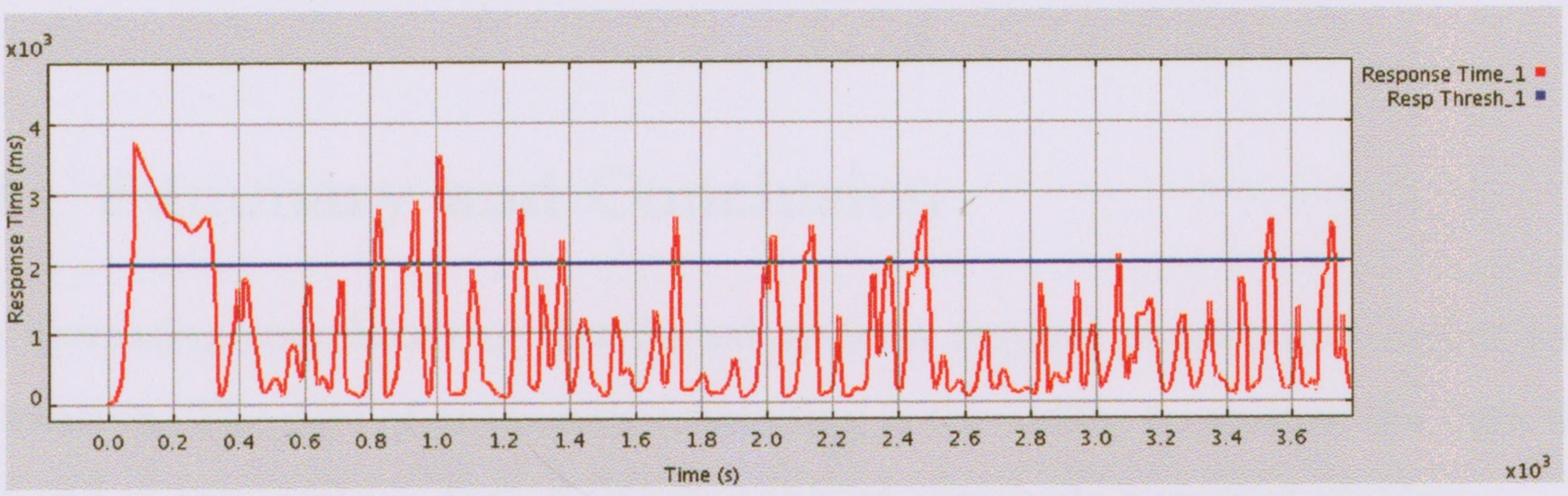


Figure 5.10: Proactive Management: Probability Approach(Apache - 1 hr)

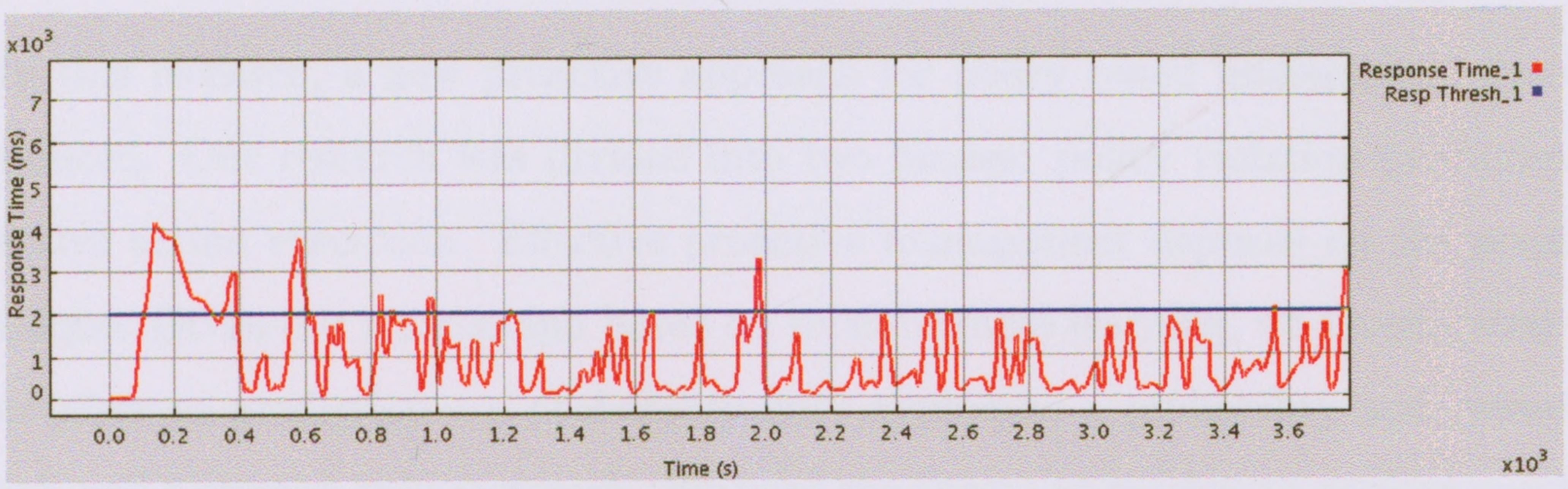


Figure 5.11: Proactive Management: Probability and Reward Approach(Apache - 1 hr)

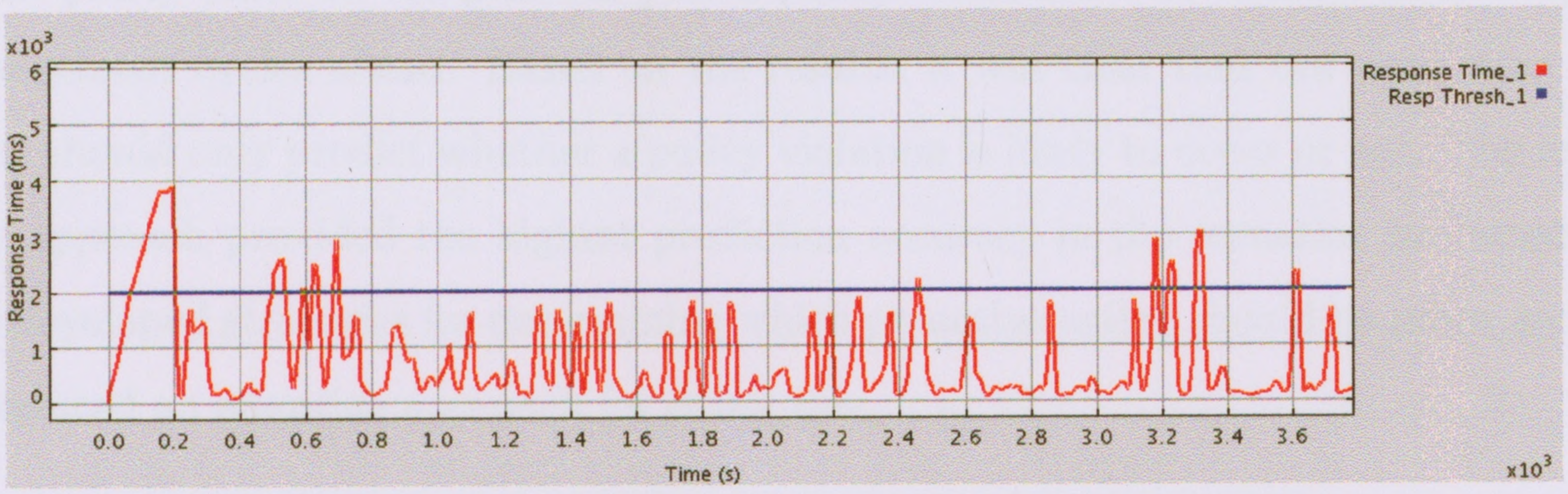


Figure 5.12: Proactive Management: Reward Approach(Apache - 1 hr)

In both scenarios, the combined probability and reward approach shows better results. The traffic load graphs of CPU utilization and Apache response time parameter for 4 hrs test period are given in Appendix B.

These results suggest that our proactive management approach can improve system performance; for the experiments shown the approaches saved almost 22% in the number of violations by predicting possible faults and taking action ahead of time.

Chapter 6

Conclusions and Future Work

6.1 Summary and Conclusion

The increasing complexity of distributed systems requires significantly higher level of automation. The core of autonomic computing is the ability to analyze the data in real time and to take actions, including being able to anticipate potential problems.

In this research, a new proactive approach for policy based management system is introduced. Our research was divided into two phases: policy violation prediction and proactive action execution. Effective proactive management depends on the prediction technique. Given our state-graph based on reinforcement learning, we investigated three approaches to prediction - a probability only approach, a probability and reward approach and a reward only approach. We used the three approaches to explore prediction of the policy state or the possibility of a policy violation. We focused predicting two management cycles ahead. Based on the results, it was clear that our prediction technique should only predict whether a policy violation is likely to occur or not. The reward only approach provided the highest prediction accuracy in the scenarios we tested. We also developed strategies for determining which proactive action should be taken and also introduced an adaptive approach for doing this.

A comparative analysis was done between proactive and reactive policy based management. Based on the scenarios and experiments carried out, our proactive management system is able to reduce the number of policy violations by almost 22% by predicting policy violations and by taking action ahead of time.

6.2 Future Work

Proactive management in policy based systems is a relatively new area to explore. The work undertaken in this thesis suggests that further research in this area could be helpful to reduce or avoid policy violations. As a result, there are number of areas which can be the focus of future work:

- Our prediction technique performs prediction for two management cycles ahead. Prediction can be done for three or four management cycles ahead, though likely at a reduction of accuracy. This would be interesting to explore.
- We have not fully explored the implications of the proactive actions taken. Some of these might be beneficial while others could be harmful, though no evidence to that effect showed up in our experiments. Nevertheless, an important area for future work would be to incorporate strategies to assess actions and perhaps even consider whether certain actions should be 'undone'.
- We explored policy violation prediction only. It might also be interesting to include prediction of resource usage by predicting the values of the metrics in the states Such an approach might assist in better determining just which action to take.
- There may be alternative strategies that could be implemented as part of the Adaptive Proactive Action approach. Different techniques might be used to generate alternative actions and/or do more to evaluate the result of those actions.

Proactive management seems promising in policy-based autonomic management. Predicting future conditions is a challenging task and further work is needed to explore the potential of this approach and to better understand some of the risks.

Bibliography

[AB83] U. Appel and A. Brandt. Adaptive sequential segmentation of piecewise stationary time series. In *Information Sciences*, pages 27–56, 1983.

[AGP06] Artur Andrzejak, Sven Graupner, and Stefan Plantikow. Predicting resource demand in dynamic utility computing environments. In *International Conference on Autonomic and Autonomous Systems (ICAS)*, pages 6–6, 2006.

[BB08] Raphael M. Bahati and Michael A. Bauer. Modelling reinforcement learning in policy-driven autonomic management. In *Proceedings of IEEE/IARIA International Journal On Advances in Intelligent Systems*, pages vol. 1, no. 1, 54–79, 2008.

[BBV07] Raphael M. Bahati, Michael A. Bauer, and Elvis M. Viera. Adaptation strategies in policy-driven autonomic management. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, page 16, 2007.

[CHW$^+$95] O. Cordn, F. Herrera, G. Winter, J. Periaux, M. Galan, P. Cuesta (Eds.), John Wiley, and Sons. General study on genetic fuzzy systems. In *Genetic Algorithms in Engineering and Computer Science*, pages 33–57, 1995.

[Dat] MySQL Database. Available: http://www.mysql.com/.

[DDLS00] 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. In *Technical Report, ImperialCollege*, 2000.

[DLJ$^+$06] Jianguo Ding, Xiaoyong Li, Ningkang Jiang, B.J. Kramer, Davoli, and F. Prediction strategies for proactive management in dynamic distributed systems. In *International Conference on Digital Telecommunications*, pages 74–74, 2006.

[FAR$^+$97] De Franceschi, A.S.M., Da Rocha, M. A., Weber, H. L., and Westphall C. B. Proactive network management using remote monitoring and artificial intelligence techniques. In *Proceedings of the 2nd IEEE Symposium and Communications ISCC*, pages 587–592, 1997.

[Fra96] A.S.M. De Franceschi. An application to validate the proactive network management. In *M. Sc. Dissertation, Federal University of Santa Catarina*, 1996.

[GLL+07] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White. A meta-learning failure predictor for bluegene/l systems. In *Proceedings of International Conference on Parallel Processing (ICPP)*, pages 40–40, 2007.

[GZL+08] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B. Park. Dynamic meta-learning for failure prediction in large-scale systems: A case study. In *Proceedings of International Conference Parallel Processing (ICPP)*, pages 157–164, 2008.

[HE01] G. Hamerly and C. Elkan. Bayesian approaches to failure prediction for disk drives. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 202–209, 2001.

[HJ98] Cynthia S. Hood and Chuanyi Ji. Intelligent agents for proactive network fault detection. In *IEEE Internet Computing, Vol.2,*, pages 65–72, 1998.

[HKV+98] Hariri, S. Kim, Y. Varshney, K. Kaminski, R. Hague, D. Maciag, and C. A framework for end-to-end proactive network management. In *IEEE Network Operations and Management Symposium (NOMS),vol.1,*, pages 280–286, 1998.

[HSM04] G. Hoffmann, F. Salfner, and M. Malek. Advanced failure prediction in complex software systems. In *Proceedings of IEEE Symposium on Reliability in Distributed Software (SRDS)*, 2004.

[HZS01] J. Hellerstein, F. Zhang, and P. Shahabuddin. A statistical approach to predictive detection. In *Computer Networks: The International Journal of Computer and Telecommunications Networking*, pages 77–95, 2001.

[IYS86] R. K. Iyer, L. T. Young, and V. Sridhar. Recognition of error symptoms in large systems. In *Proceedings of 1986 ACM Fall joint computer conference*, pages 797–806, Dallas, TX, 1986.

[Jme] Apache Jmeter. Available: http://jakarta.apache.org/jmeter/.

[KHD00] Yoonhee Kim, Salim Hariri, and Muhamad Djunaedi. Expermental results and evaluation of the proactive application management system (pams). In *IEEE Proceedings Heterogeneous Computing Workshop, 9th Workshop*, pages 53–59, 2000.

[Lab] RAD Lab:. Reliable adaptive distributed systems laboratory. In *http://radlab.cs berkeley.edu/.*

[Lin] Linux. Available: http://www.linux.org/.

[LLG+07] Z. Lan, Y. Li, P. Gujrati, Z. Zheng, R. Thakur, and J. White. A fault diagnosis and prognosis service for teragrid clusters. In *Proceedings of TeraGrid*, 2007.

[LZSS06] Y. Liang, Y. Zhang, A. Sivasubramanium, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *Proceedings of Dependable Systems and Networks (DSN)*, pages 425 - 434, 2006.

[MA75] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. In *International Journal of Man-Machine Studies*, pages 1–13, 1975.

[Pea88] J. Pearl. Probabilistic reasoning in intelligent systems. In *Networks of Plausible Inference.* Morgan Kaufmann Publishers, 1988.

[PHP] PHP. Available: http://www.php.net/.

[Pro] Apache Http Server Project. Available: http://www.apache.org/.

[Roc96] M.A. Rocha. A strategy to implement the proactive network management using knowledge-based systems. In *M. Sc. Dissertation, Federal University of Rio Grande do SUI*, 1996.

[Saz02] E. S. Sazonov. Open source fuzzy inference engine for java. In *http://www.intelligent-systems.info/FuzzyEngine.htm*, 2002.

[SB98] R. S. Sutton and A. G. Barto. Reinforcement learning:an introduction. Bradford Books,MIT Press, 1998.

[Sea03] R.K. Sahoo and A.J. Oliner et al. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of Knowledge Discovery and Data Mining (KDD)*, pages 426–435, 2003.

[Sit] The TOP500 Supercomputer Sites. Available: www.top500.org.

[SML07] F. Shaikh, G. Mapp, and A. Lasebae. Proactive policy management using tbvh mechanism in heterogeneous networks. In *Proceedings of the International Conference and Exhibition on Next Generation Mobile Application,Services and Technologies*, pages 151–157, 2007.

[SZL05] Kai Shen, Ming Zhong, and Chuanpeng Li. I/o system performance debugging using model-driven anomaly characterization. In *4th USENIX Conference on File and Storage Technologies*, pages 23–23, Berkeley, CA, USA, 2005.

[TA03] D. Turnbull and N. Alldrin. Failure prediction in hardware systems. In *UCSD CSE221 Project*, 2003.

[TV99] K. Trivedi and K. Vaidyanathan. A measurementbased model for estimation of resource exhaustion in operational software systems. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, page 84, 1999.

[Wik10] Wikipedia. Available: http://www.wikipedia.com. 08 November 2010.

[WW97] Henrique L. Weber and Carlos B. Westphall. A proactive network management using remote monitoring and artificial intelligence techniques. In *Proceeding of the 2nd IEEE Symposium and Communications (ISCC)*, pages 167–171, 1997.

[XTL07] Gu X, Klie T, and Wolf L. A proactive policy-based management approach towards autonomic communications. In *Proceedings of 4th IEEE Consumer Communications and Networking Conference (CCNC)*, pages 587–592, 2007.

[Zad65] L. A. Zadeh. Fuzzy sets - information and control. pages vol.8, pp. 338–353, 1965.

Appendix A

Policy List

A.1 Policy List

```
if (APACHE:responseTime > 2000.0) & (APACHE:responseTimeTrend > 0.0)
then { AdjustMaxClients (+25) test {newMaxClients < 151 } |
AdjustMaxKeepAliveRequests (-30) test {newMaxKeepAliveRequests > 0.0} |
AdjustMaxBandwidth (-128) test {newMaxBandwidth > 255.0}}
```

```
if (CPU:utilization > 85.0) & (CPU:utilizationTrend > 0.0)
then { AdjustMaxKeepAliveRequests (-30) test {newMaxKeepAliveRequests > 0.0} |
AdjustMaxBandwidth (-128) test {newMaxBandwidth > 255.0}}
```

```
if (MEMORY:utilization > 50.0) & (MEMORY:utilizationTrend > 0.0)
then { AdjustMaxClients (-25) test {newMaxClients > 49.0 }
```

```
if (APACHE:responseTime > 2000.0) & (APACHE:responseTimeTrend > 0.0)
& (MEMORY:utilization > 50.0) & (MEMORY:utilizationTrend > 0.0)
then { AdjustMaxKeepAliveRequests (-30) test {newMaxKeepAliveRequests > 0.0}}
```

```
if (CPU:utilization > 85.0) & (CPU:utilizationTrend > 0.0)
& (MEMORY:utilization > 50.0) & (MEMORY:utilizationTrend > 0.0)
then { AdjustMaxClients (-25) test {newMaxClients > 49.0 } }
```

```
if (CPU:utilization > 85.0) & (CPU:utilizationTrend > 0.0)
& (APACHE:responseTime > 2000.0) & (APACHE:responseTimeTrend > 0.0)
then { AdjustMaxKeepAliveRequests (-30) test {newMaxKeepAliveRequests > 0.0} |
AdjustMaxBandwidth (-128) test {newMaxBandwidth > 255.0}}

if (APACHE:responseTime > 2000.0) & (APACHE:responseTimeTrend > 0.0)
then { AdjustEaccMemSize (+1) test {availableEaccMem > 1
& newEaccMemSize < 32} }

if (CPU:utilization > 85.0) & (CPU:utilizationTrend > 0.0)
then { AdjustEaccMemSize (+1) test { availableEaccMem > 1
& newEaccMemSize < 32} }

if (MEMORY:utilization > 50.0) & (MEMORY:utilizationTrend > 0.0)
then { AdjustEaccMemSize (-11) test {newEaccMemSize > 16}}

if (APACHE:responseTime > 2000.0) & (APACHE:responseTimeTrend > 0.0)
then { AdjustKeyBufferSize (+1024000) test { newKeyBufferSize < 3.2768E7
& availableKeyBlocks < 1000.0} |
AdjustQueryCacheSize (+1024000) test {newQueryCacheSize > 3.2768E7
& availableQueryCacheMem > 1024000.0} }

if (CPU:utilization > 85.0) & (CPU:utilizationTrend > 0.0)
then { AdjustThreadCacheSize (+50) test {newThreadCacheSize < 201.0} }

if (MEMORY:utilization > 50.0) & (MEMORY:utilizationTrend > 0.0)
then { AdjustKeyBufferSize (-1024000) test {newKeyBufferSize > 1.6384E7} |
AdjustQueryCacheSize (-1024000) test {newQueryCacheSize > 1.6384E7} }
```

Appendix B

Traffic Load Graph

B.1 CPU Utilization [Test Period: 4 hrs]

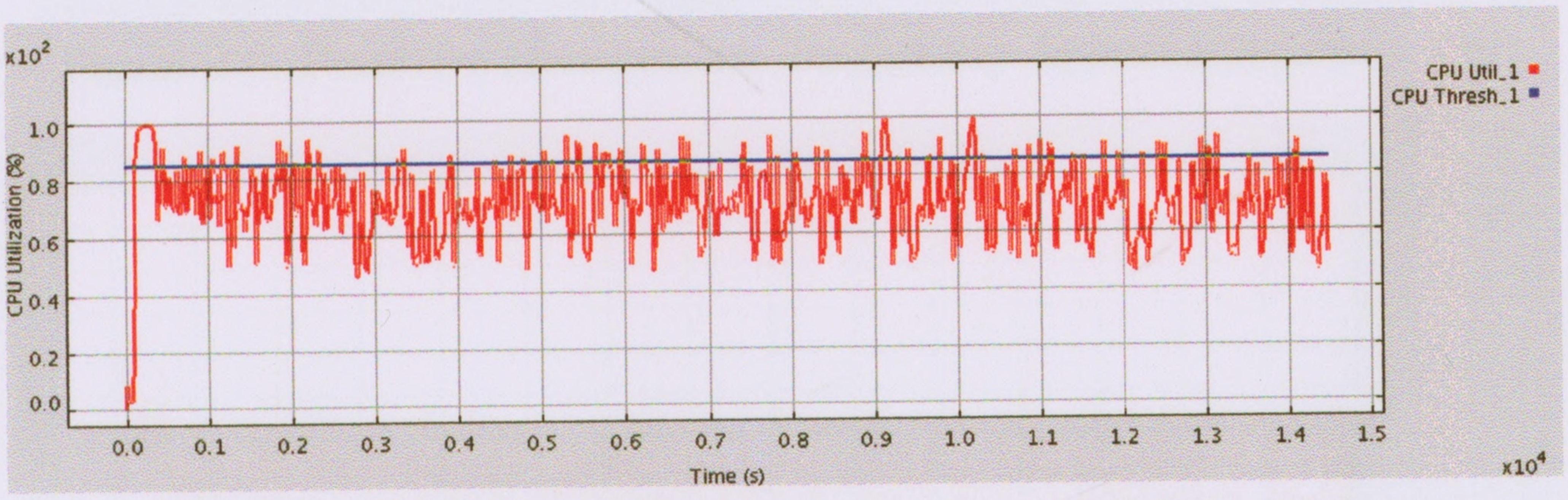


Figure B.1: Reactive Management Approach(CPU - 4 hr)

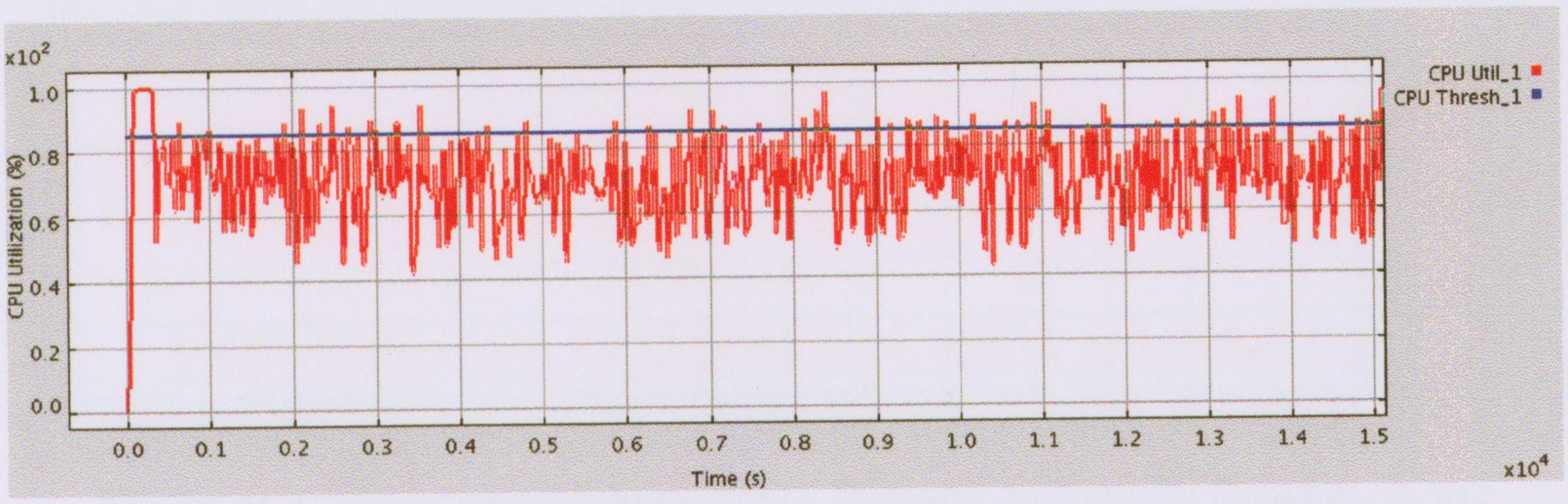


Figure B.2: Proactive Management: Probability Approach(CPU - 4 hr)

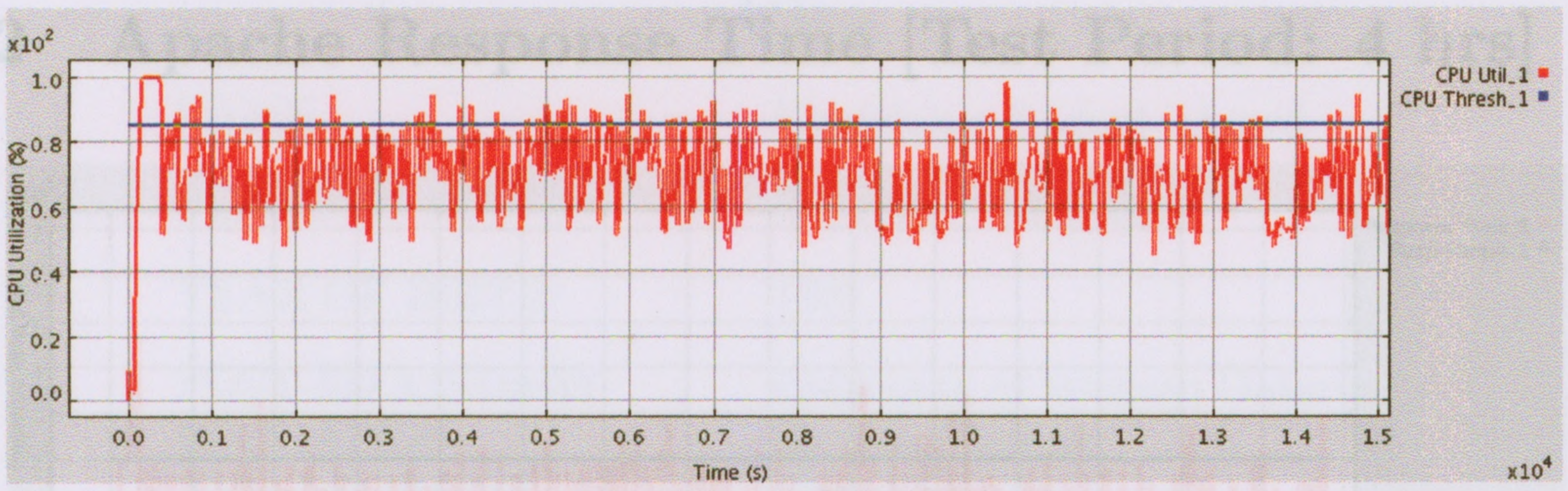


Figure B.3: Proactive Management: Probability and Reward Approach(CPU - 4 hr)

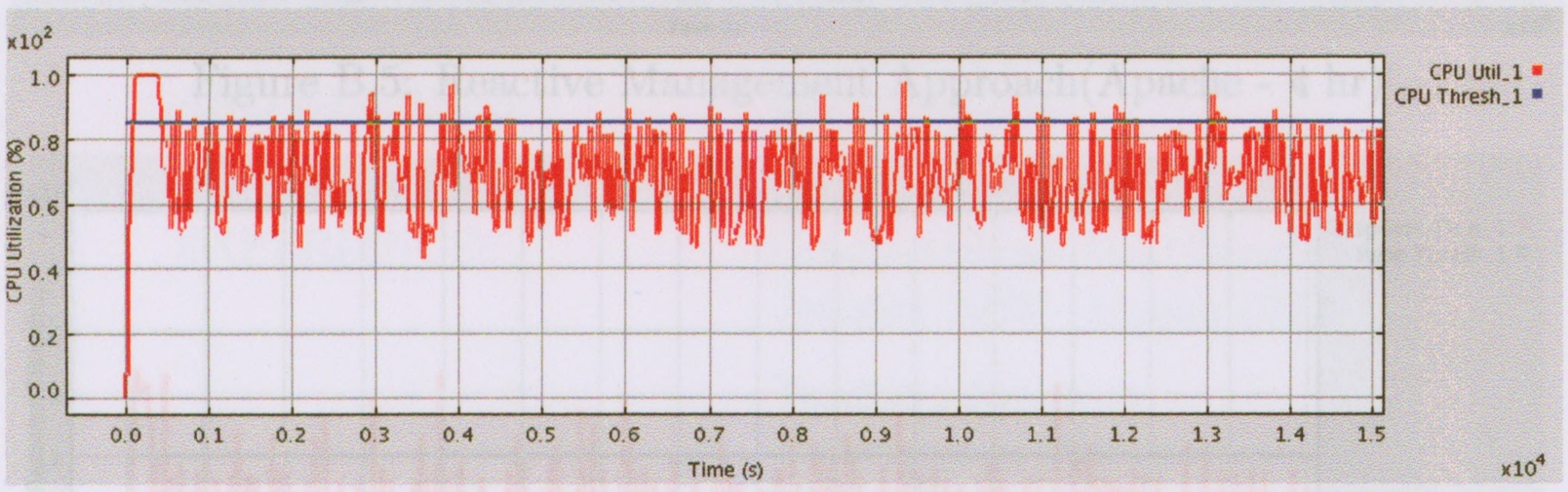


Figure B.4: Proactive Management: Reward Approach(CPU - 4 hr)

B.2 Apache Response Time [Test Period: 4 hrs]

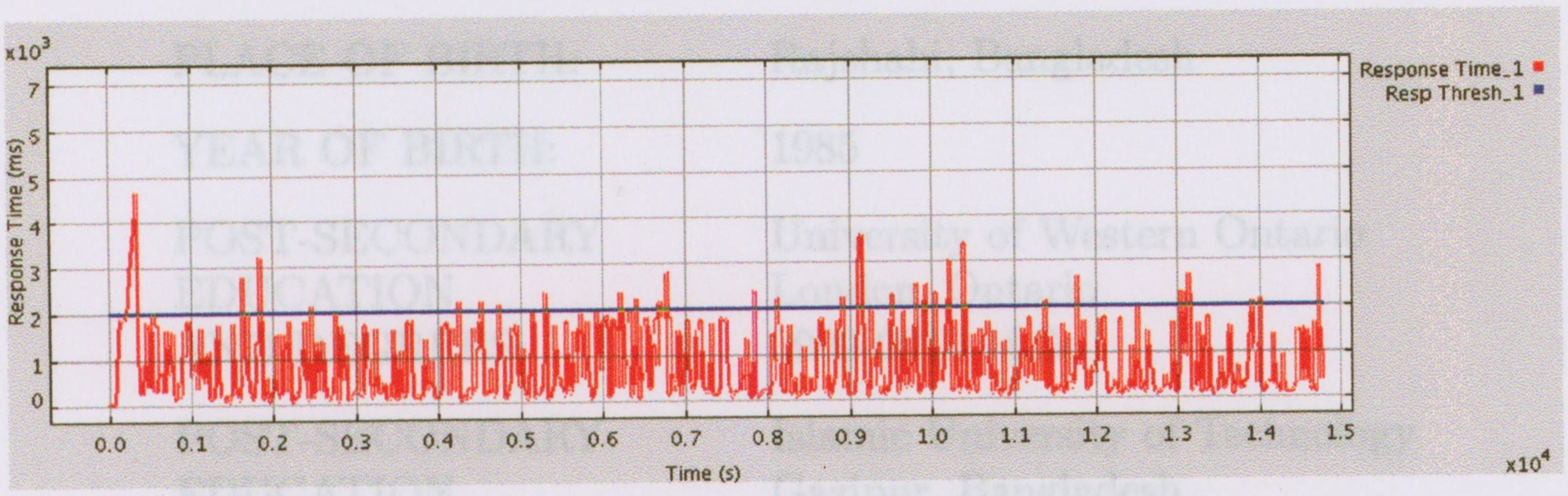


Figure B.5: Reactive Management Approach(Apache - 4 hr)

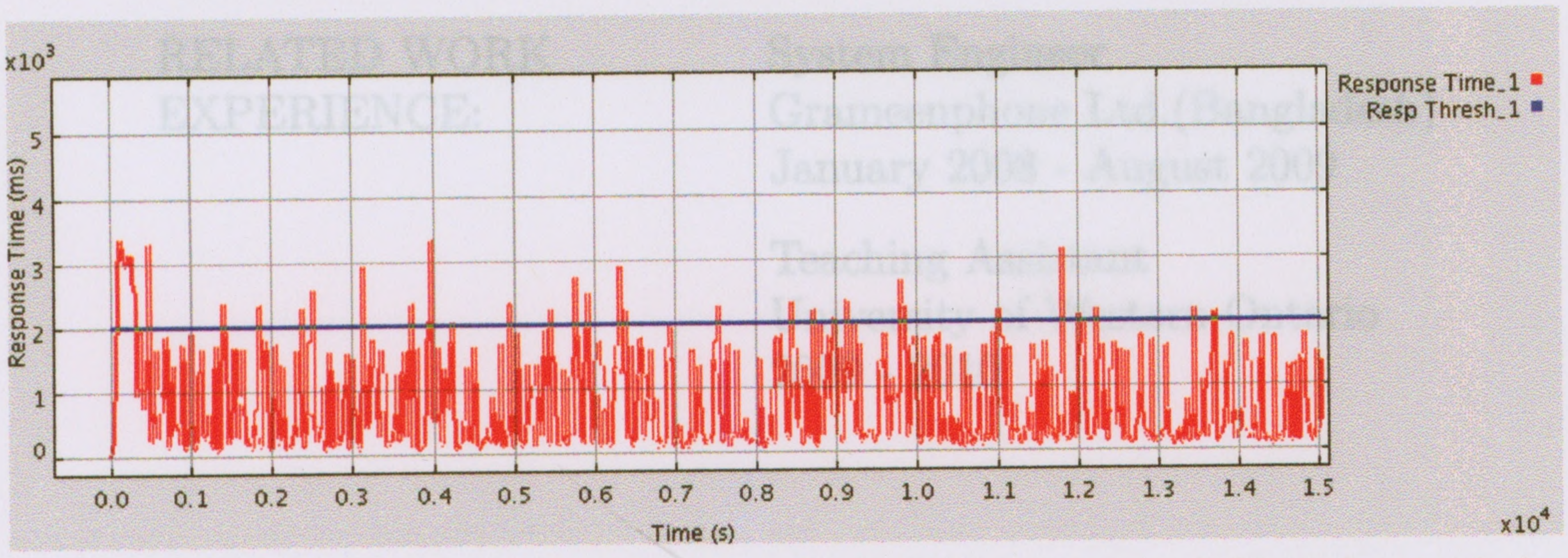


Figure B.6: Proactive Management: Probability Approach(Apache - 4 hr)

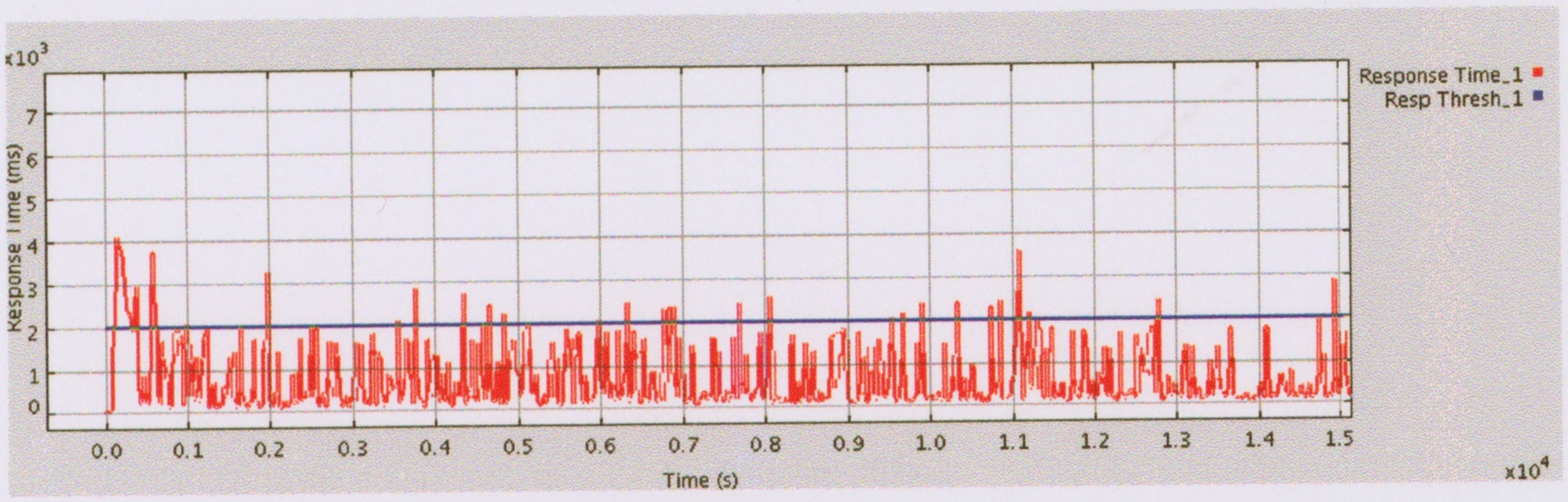


Figure B.7: Proactive Management: Probability and Reward Approach(Apache - 4 hr)

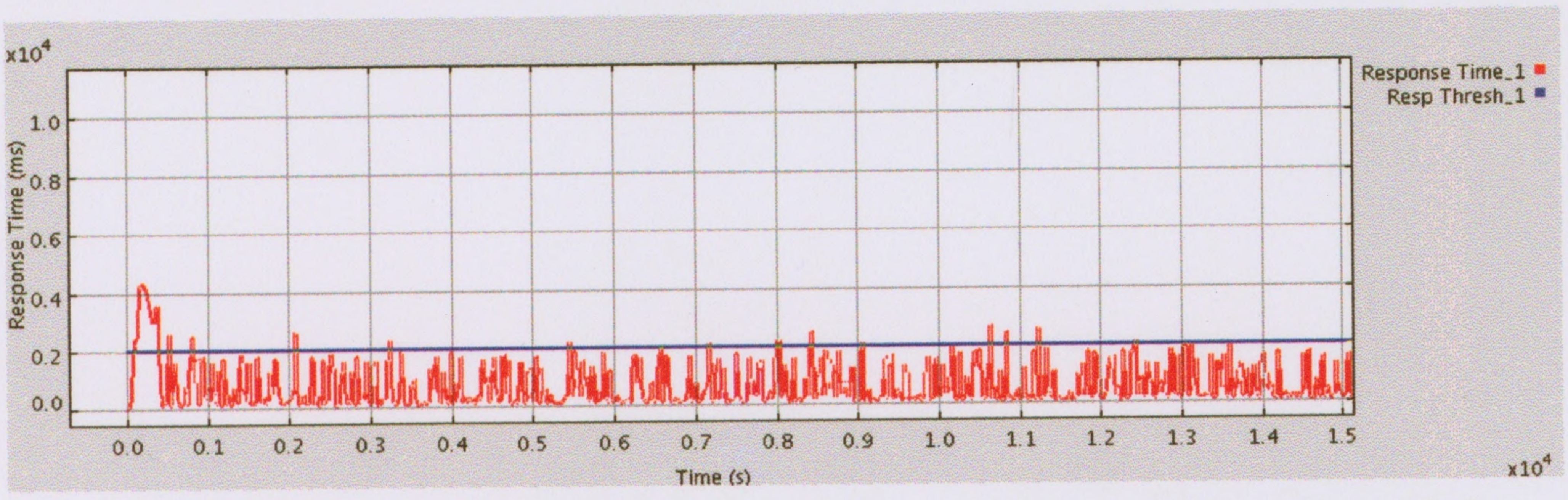


Figure B.8: Proactive Management: Reward Approach(Apache - 4 hr)