PreCog: A Robust Machine Learning System to Predict Failure in a Virtualized Environment

By

Adam Rogal

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING
IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2008

Signature of Author:_____
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by:_____
Sridhar Rajagopal
Staff Engineer, R&D: Virtual Infrastructure Mgmt, VMware
VI-A Thesis Supervisor

Certified by:_____
Larry Rudolph
Principal Research Scientist of Electrical Engineering, MIT CS and AI Lab
MIT Thesis Supervisor

Accepted by:_____
Arthur C. Smith
Professor of Electrical Engineering and Computer Science
Chairman, Department Committee on Graduate Theses

PreCog: A Robust Machine Learning System to Predict Failure in a Virtualized Environment

by

Adam Rogal

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008 in partial fulfillment of the
requirements for the Degree of Master of Engineering in
Electrical Engineering and Computer Science

ABSTRACT

The research in this work addresses the need for a warning system to predict future application failures. PreCog, the predictive and regressional error correlating guide system, aims to aid administrators by providing a robust future failure warning system statistically induced from past system behavior. In this work, we show that with the use of machine learning techniques such as Adaptive Boosting and Correlation-based Feature Selection, PreCog, without any prior knowledge of its target, can be accurately and reliably trained within a virtual environment using past system metrics to predict future application in a variety of domains.

VI-A Company Thesis Supervisor: Sridhar Rajagopal
Title: Staff Engineer, R&D: Virtual Infrastructure Mgmt

MIT Thesis Supervisor: Larry Rudolph
Title: Principal Research Scientist of Electrical Engineering,

ACKNOWLEDGEMENTS

As are most projects of this magnitude, I could have not accomplished what I have without the guidance and support of a number of individuals…

Foremost, I would like to thank the unwavering support of VMware, the company for which the work of my thesis was done. Through the VI-A program, VMware gave me an unparalleled experience to any previously of my career. I was given the opportunity to design and pursue a project of my own. Despite my inexperience and lofty goals, I was given respect, patience, understanding, and support. Over the last 2 years while working for VMware, this has truly been a wonderful experience.

Among my colleagues at VMware, I would like to give thanks to those who helped throughout the project – from the primordial days to the final weeks of result verification and thesis editing. To Sridhar Rajagopal, you have been a true mentor in every sense of the word. Your continued guidance and support allowed me to push through even the hardest of times. To Eddie Ma, thank you for providing every resource I needed to complete my work. To the rest of my group, thank you for your input and contribution throughout the project. To Christian Hammond, thank you for giving me access to Review Board. Without your help, I would not have been able to develop my wok as far as I did.

Although the bulk of my work took place at VMware, my work could not be possible without the help of those from MIT. To my thesis advisor, Larry Rudolph, you are and have been an infinite source of wisdom and advice. Thank you for the many hours of guidance and mentorship. To Kimberle Koile, a mentor in all regards, thank you for the many years of advice, support, and friendship. Regardless of any future project you may take on, you can count on me

to be there, computer cart in tow. To the VI-A program, I believe VI-A is an amazing program, unmatched by any other at MIT. I am grateful for the opportunity it has provided me and implore more students to explore the VI-A program.

To my friends, family and loved ones: A project of this magnitude cannot be done alone. Thank you to my friends who always supported me. To Steve, Tim, Dan, Sun, Jenna, Monica, and Aron: these four years could not have been the same without you. To my brother, Justin, thank you for the occasional reminder that there is more to life than computers. To Jia, thank you for your love and support. Finally, thank you to my parents. To my father, thank you always for your pride in me and the many lessons you have taught me, especially that one is never done learning. Thank you to my mother, who every Sunday, lovingly never let me forget that "you should probably be working on my thesis right now..." When an individual goes on any journey, the ones to whom they are connected join them. It is only through their support that one can succeed at any undertaking. I dedicate this thesis to them. For without them, none of this would be possible.

We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes.

- Pierre-Simon Laplace, ***Philosophical Essays*** *on* ***Probability***. New York: Springer-Verlag, 1995.

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

As networked systems continue to grow in size and complexity, the technologies developed to manage and facilitate these infrastructures struggle to keep up. Not only do these systems need to be highly available to users, but also their performance must be accurately tracked and maintained. Many tools exist to help administrators to this end, but the ability to model these complex networked systems may be too difficult for any human to accomplish, as the systems' behaviors depend on not only workload, but also software structure, hardware, and external variables [1]. We address this challenge through PreCog, the predictive and regressional error correlating guide system, which aims to aid administrators by providing a robust future warning system statistically induced from past system behavior. This work shows that with the use of machine learning techniques such as Adaptive Boosting [2], PreCog, without any prior knowledge of its target, can be accurately trained using past and current system behavior to predict future application in a variety of domains.

2 BACKGROUND

Machine learning has been shown to be effective in this domain, specifically in work by *Ira Cohen et al* [3]. Their work shows that application failures determined by Service Level Objectives (SLO) – a metric of application performance in relation to a set threshold – could be accurately classified by current system behavior, regardless of application or hardware and without any prior knowledge of the system. Their work achieves accuracies as high as 95% for failure detection utilizing machine learning algorithms such as the Tree Augment Naïve Bayes (TAN) [4] classifier. Although preliminary work shows that future SLO violations can be predicted using similar techniques [5], it was still inconclusive as to their adaptability into the

prediction domain. PreCog extends their work by using similar machine learning algorithms to create a concrete human understandable warning system for future failure.

2.1 MOTIVATION

In most large-scale systems, the downtime incurred is not only brought upon by the unavailability of applications, but the aftermath that comes with it. When a failure finally has been diagnosed and repaired, large amounts of data have often already been lost or corrupted in the process. If an application were developed that could provide an early warning system with an estimated time until failure, administrators could have a multitude of options at the ready, including repartitioning resources, shutting down non-vital components, or, in the worst case, gracefully shutting down the system, thus avoiding any significant data loss.

Even if a warning system could be developed, the adaptability given the almost infinite permutations of system and software components that exist in even the most simple of environments is unknown. Although PreCog has the potential to be applied to physical machine environments, a virtual environment is leveraged such that each system shares a common architecture. It is often difficult, and perhaps impossible, to expect a human to take the correct evasive action in any environment. We believe that PreCog is more useful in a system that can perform such actions automatically, such as a virtual environment in which each system shares a common architecture and supports virtual machine migration. To this end, a warning system can be trained, and acted upon, regardless of specific underlying hardware or software implementations.

2.2 THE SUPERMARKET DILEMMA

To understand the need for PreCog, we can model the virtual infrastructure and the prediction engine as a supermarket and its manager directing cashiers and baggers. In this analogy, we model the available resources of the supermarket as the available resources of a virtualized environment. A virtualized system of a typical 3-tier e-commerce website, for example, may consist of separate virtual machines including a web server, an application server, and a database, all hosted on the same ESX sever. Each one of these virtual machines competes for an equal share of resources much as customers compete for a limited number of cashiers and checkout lines.

With only so many resources available, management systems are a necessity. In the supermarket, the manager lays out a detailed plan for the baggers and cashiers. It may be a trivial plan such as "each checkout lane will always just have one cashier and one bagger" or more complex such as "each checkout lane will have one cashier and as many baggers as needed to maximize efficiency." Likewise, many current system management suites offer similar tools to create such rules. VMware's current resource management system (DRS) repartitions resources based on rules defined by system administrators, such as minimum resource values or VM resource priority. These naïve systems can cover a large range of use scenarios; yet, they are still limited by the foresight of the administrator (be it system administrator or store manager) that defines these rules. PreCog offers an intelligent foundation for a potential resource management system by learning behavior from analysis of previous failures. In the real world, a manager who notices and remembers patterns is able to adapt his plans using past experience. In essence, PreCog aims to capture and mimic that behavior in order to predict when a failure may occur.

We now consider a manager who has been in the business for his entire life. He recognizes key traits in both the flow and the character of the shoppers, and has an uncanny ability to predict how many cashiers and baggers he will need. More importantly, he has learned that the time to recognize these traits is not while the customers are in line, but as soon as soon as they enter the store. By the time the customer has entered the line it is too late – the manager can only guess if this congestion will continue, a risky decision at best.

We return to the source of our analogy. If we were to use the traditional method of monitoring application workload, we would be acting much like the naïve manager. If only the number of visits to a web page per minute were monitored, for example, it would be impossible to learn from previous failures; what might seem a rush of customers one minute could just as well die down the next. Continuing this analogy, even after costumers have entered the store, many of them might linger down aisles or hold up aisles. In other words, by not using the vast amounts of information contained within the character of the load (or the type of customers, not just the number of them), naïve systems can only guess at future load, by which time it may already be too late.

3 OVERVIEW

3.1 FAILURE

Before being able to predict when failures may occur, one must first define what is meant by "failure." Failure can mean anything from a hardware crash, software fault, or missing deadlines. One can strengthen the definition to include any event in which the system is not performing at its expected level. Even this definition may be too vague; yet, it rules out many of the failures

that should not be included in the scope of PreCog. A plug being pulled, for example, or an employee accidently stopping a vital program are failures. These failures are too extreme to fit our definition; not only is the system not performing at its expected level, but also not performing at all.

PreCog leaves the definition of failure up to the user, although is mostly concerned with "soft" failures. In particular, a failure is defined as not meeting Service Level Objectives (SLO). SLOs may gauge the performance of, for example, a database by the response time of a certain SQL request, or any SQL request for that matter. SLOs may also be applied to the resources of a system – gauging over-utilization based on expected levels. Many high availability solutions, in fact, cater to strictly monitoring SLOs and reporting failures to system administrators based on these values. For example, a popular product Hyperic utilizes an open source plug-in architecture that allows the author to create SLOs for any different number applications and operating systems. Another example, VMware offers high availability rule management via SLOs in the form of available system resources and server response to gauge proper management of virtual machines. SLOs, though simple, offer an elegant way of tracking and gauging system performance and failure.

3.2 PREDICTION

The ultimate goal of a highly accurate prediction system is to be able to give as input the current system behavior and receive a real number indicating the amount left until failure and the probability associated with that failure occurring. Generating this function, however, through machine learning is intractable. Although linear regression can be used along the metric by which the SLO is defined, it is likely to perform poorly when given sharp spikes in user load.

PreCog, instead, aims to produce as output a descretized range of warning levels that can not only be accurately classified, but be human readable as well. Thus, we define a warning system as follows:

- Level 1: 0-5 minutes until first occurrence of failure
- Level 2: 5-15 minutes until first occurrence of failure
- Level 3: 15-30 minutes until first occurrence of failure
- Level 4: 30-60 minutes until first occurrence of failure
- Level 5: 60+ minutes until first occurrence of failure

Not only does this warning system give a time range in which the system may enter a fail state, but also gives a likelihood that the system may enter a fail state at any given moment. If, for instance, PreCog outputs a level 5 warning, it is much more unlikely that it will begin to fail than if it were a level 2 or 1 warning. The further definition of likely is explained in context of specific classifiers in Section 5.5.

3.3 PreCog System Design

The PreCog system design, shown in Figure 1, is a set of components that allows for application of failure prediction within a virtual environment. The virtual environment, the shaded region in this example diagram, consists of a set of servers, the large bounding boxes, each running a multitude of virtual machines (VM), the smaller bounded boxes. Of these virtual machines, at least one hosts an application that is the target of PreCog's warning system. In Figure 1, this VM is in bold and being polled by the SLO monitor. The PreCog system consists of the following components:

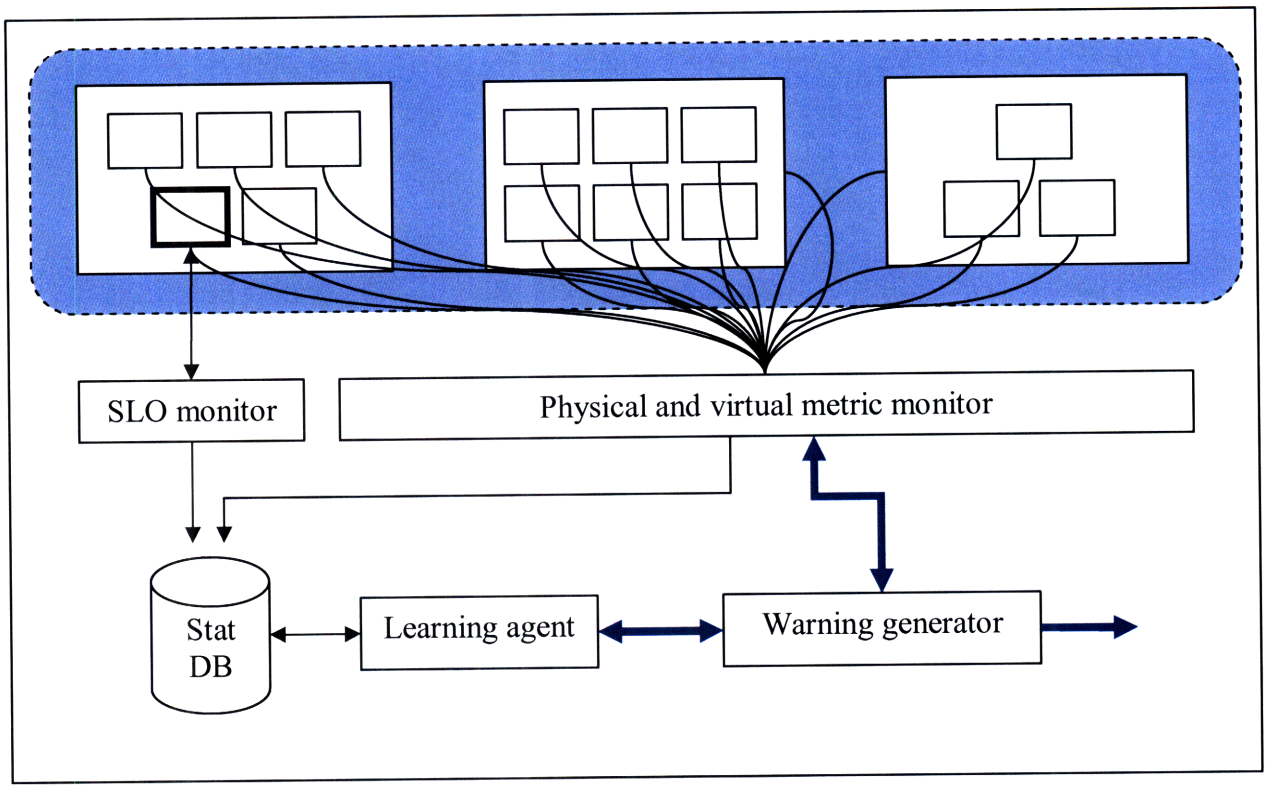


Figure 1: A layout of the PreCog system. The shaded region represents the virtual environment and the bold square represents the virtual machine that is being monitored for failure. The bold lines represent a query made to predict a failure after the initial training stage.

- **SLO monitor:** The SLO monitor poles all monitored virtual machines for failures. The failures are domain specific and defined by their SLOs. In the experiments completed in this work, the monitored virtual machine is an Apache web server and the SLO is defined by the average response time per page. This SLO is monitored by scraping the Apache HTTP server log.
- **Physical and virtual metric monitor:** The metric monitor poles all virtual entities within the virtual infrastructure (shaded in the diagram above) for all available metrics. These metrics are collected in timestamped five-minute intervals and averaged. From a normal virtual machine, 50 or so metrics can be scraped. These range from the absolute value usage of a virtual CPU, to the percent change in memory usage over the last 5

minutes. A full listing of all metrics scraped from the JPetStore testbed can be viewed in Appendix B. From the hosts in the virtual infrastructure, the metrics that are scraped are similar to the virtual machine's. There are, however, superfluous and often misleading metrics such as system uptime. If the same experiment were performed a number of times, and these metrics were left in during training, the utility of these metrics might be over-emphasized in the learning stage as each experiment has a planned time of failure.

- **Statistical database:** This database serves two functions: First, it stores the structure of the virtual environment per timestamp. That is, as VMs are powered down or moved from our virtual environment, the overall structure changes. The models induced over a set of metrics from the past may no longer apply. Therefore, the virtual environment must be tracked such that PreCog can adapt to these changes. Second, it stores both the timestamped metrics from all polled hosts and VMs correlated with SLO violations. These paired metrics and SLO violations serve as training data for the learning engine.
- **Learning agent:** This component gathers data from the statistical database for a given virtual environment structure and produces a classifier that is trained for a specific SLO and set of warning levels. When queried with a set of system metrics from the virtual environment, the classifier chooses the most likely warning level.
- **Warning generator:** The warning generator produces a warning level for a given virtual environment, set of metrics values, and SLO by polling the learning agent. This warning is generated only after an adequate amount of data has been collected for the classifier to be trained accurately.

3.4 MACHINE LEARNING

The core of the PreCog system is a machine-learning engine that correlates system level metrics produced from any of the virtual machines and hosts in the virtual environment, with application SLO failures to forecast future events, the internals of which are built upon the open source WEKA library [6]. Building on previous research [7], PreCog determines not only if supervised machine learning algorithms are a viable choice, but also how these algorithms must be applied to retrieve accurate and efficient failure predictions. That is to say, in the scope of this research, accuracy and efficiency are defined as to provide valuable and concrete evaluations.

PROBLEM FORMALIZATION

The formalization for each classifier follows the general approach of pattern classification [8]. We define the feature set, the input, and the classes to which an input may belong. Each classifier, in general terms, uses a supervised machine learning algorithm to train itself on system metrics collected from virtual machines and hosts, and correlates them with the failure class.

Feature set

The feature set is all available system-level metrics. That is, each virtual machine and host, internally, keeps track of metrics such as CPU, memory, network, and disk usage. Furthermore, the absolute value, first order change, minimum, and maximum are also collected. A full list of the 214 metrics collected from the JPetStore testbed may be found in Appendix B.

Input

Let $\vec{M_t}$ be a vector of real numbers for a set of n collected metrics $< m_0, m_1, \ldots, m_n >$ for time t. We may consider each vector $\vec{M_t}$ at a given time as the input. In other words, the classifiers goal is to label this set with a classification determined by the warning level system.

Classification

The goal of PreCog is, given a vector of metrics $\vec{M_t}$ representing the current system state, output a range of time until a failure occurs. As defined earlier, the ranges that PreCog may output can be considered a warning level system. The warning system is defined by a range until a failure state may occur such as 0-5, 5-15, 15-30, 30-60, and 60+ minutes. We define the set of offsets $\vec{p}$ as $p_0 = 0, p_1 = 5, p_2 = 15$, etc..., and define a valid range as $P_i = \{p_i, p_{i+1}\}$. Given that there are n offsets, the catchall range extending beyond the last offset (represented in the example above as 60+), is $P_{n-1,\infty} = P_\infty$. All valid ranges form the set $\mathcal{P}$. The final output is one of the elements of $\mathcal{P}$.

Training

The goal of the training stage is to label all input data points, all vectors of $\vec{M_t}$ in a specific data set, with the soonest range in $\mathcal{P}$ during which a failure occurred. Let S_t be an indicator variable representing the state of failure for time t defined by an SLO. The random variable S_t takes on the values of $\{0,1\}$, representing compliance or violation, respectively, of a SLO. A fail state is defined by an indicator variable F_{t_1,t_2} as true if there exists any failures within a given time period. In other words, let $F_{t_1,t_2} = \left(\sum_{t=t_1}^{t_2} S_t\right) > 0$. Or, if we let the range $P_0 = \{p_0, p_1\}$, the notation F_{t,P_0} would be equivalent to $F_{t+p_0,t+p_1}$. For each $\vec{M_t}$ and each available class in $\mathcal{P}$, the minimum range for which $F_{P,t}$ is true is the label for the data point. Thus, each classifier conducts supervised training based upon sets in the form of $\langle \vec{M_t}, P \rangle$.

CLASSIFIERS

Most likely, there is no single best machine-learning algorithm that performs both accurately and efficiently over all possible domains. Given this, we explore a multitude of algorithms for supervised machine learning and feature selection to determine one that is likely to perform on average best in an unknown domain. The parameters and explanation of each are listed in Appendix C.

- **J48 Decision Tree**: The J48 [6] decision tree is the WEKA toolkit's implementation of the C4.5 decision tree [9]. The C4.5 decision tree builds upon the ID3 decision tree [10] algorithm with accuracy and speed improvements. The tree is built with the basic process of partitioning sets of data down a tree according to the information gain made by each split. In other words, each branch is made in hopes of partitioning the set into the most predictive correlating ranges of features. In the scope of our feature set, each split is on the value of a system metric such as CPU usage. The leaves of the tree are the most likely warning level P, the class, given the path to that leaf. To classify a new set of metrics, we simply follow the value of each metric down the tree until a leaf is reached. The warning level associated with that leaf is the class for the queried set of metrics.
- **Naïve Bayes**: The naïve Bayes classifier [11] estimates the probability distribution of warning levels given a set of system metrics, $p(P|\vec{M}_t)$, by making strong independence assumptions on each metric. Because it is very unlikely that all metrics are independent from each other, this is considered the naïve approach. Given the probability distribution $p(P|\vec{M}_t)$, to classify a new set of metrics, we determine the probability for each warning level given these metrics and choose the warning level with the highest probability. The

probability distribution, during the training stage, over all warning levels is calculated in the form of $p(P|\vec{M}_t) = \frac{1}{Z} p(P) \prod_{i=1}^{n} p(\vec{M}_{t,i}|P)$. That is, the conditional probability of each warning level given the current set of metrics is the normalized product of the prior probabilities of each warning level and all likelihood probabilities of each metric value given the warning level. In the case of continuous variables, such as metric values in the scope of this work, descretization is performed to increase accuracy of the overall classifier. Thus, $p(\vec{M}_{t,i}|P)$ represents the likelihood probability that $\vec{M}_{t,i}$ falls within a given range. Most likely, the naïve Bayes classifier perform well on data sets where there is no dependence between metrics. That is not to say that the metrics derived from a CPU do not depend on the metrics derived from the network card. The naïve Bayes algorithm merely capitalizes on the notion that more accurate results may be achieved by assuming that there is no dependence.

- **Tree Augmented Naïve Bayes (TAN)**: The TAN classifier [3] works much in the same way the Naïve Bayesian classifier does, except that two dependencies are allowed per metric between metrics. That is, the likelihood probability of a metric $\vec{M}_{t,i}$ that depends on $\vec{M}_{t,j}$ and $\vec{M}_{t,k}$ is determined by the conditional probability of $p(\vec{M}_{t,i}|\vec{M}_{t,j}, \vec{M}_{t,k})$. In context, this states that, for example, the probability that the range in which the current usage metric of the CPU from one virtual machine may depend on the range in which the current usage metric of the network card and CPU from the host server are. As it is naturally likely for this to occur, allowing dependence between metrics has been proven to increase accuracies [5] in domains such as PreCog. The same goal of the TAN's training is to determine $p(P|\vec{M}_t)$. Training is performed similarly to Naïve Bayes except that we recursively determine the probabilities for each metric until we backtrack back to

the root node. Classification is performed as it is in the Naïve Bayes classifier: Given the probability distribution $p(P|\vec{M}_t)$, to classify a new set of metrics, we determine the probability for each warning level given these metrics and choose the warning level with the highest probability.

- **Naïve Bayes Decision Tree (NBTree):** The NBTree classifier [12] is a hybrid of the C4.5 decision tree and the Naïve Bayes classifier. It has been shown to be an effective classifier for maintaining large-scale models, performing better than both a C4.5 and a NB classifier separately. The outer structure of the NBTree is a decision tree that is trained using the C4.5 algorithm. Where it differs from a full decision tree is that, while training, at a threshold number of metrics and classification gain, the decision tree creates a leaf which is a trained a Naïve Bayes classifier of all data points partitioned by the path to that leaf instead of a deterministic class label. That is, each leaf now has a probabilistic choice of warning level determined by the NB classifier at the leaf. To classify a set of metrics, a path is followed down the tree given a set of metrics. When a leaf is reached, the NB model is queried and the warning level with the most likely probability is chosen.
- **Tree Augmented Naïve Bayes Tree (TANTreee):** The TANTree is similar to the NBTree except that its NB leaves are replaced with TAN models. Much in the same way that the NBTree improved accuracies over the C4.5 decision tree and the NB classifier, the TANTree hopes to perform better than either the J48 decision tree (which is trained using C4.5) or the TAN classifier.
- **Adaptive Boosting (AdaBoost):** The AdaBoost classifier [2] is a meta-classifier that attempts to maximize accuracy for a classifier through analysis of and close attention to misclassifications over several iterations of training. After each trial, the elements in the

dataset are weighted according to how much they contributed to error; the weights of incorrectly classified examples are weighted higher. That is, the interesting data points gain more attention than to those that do not add to an increase of classification accuracy. In the following experiments, we use the AdaBoosting algorithm on the J48 decision tree. Although the AdaBoost classifier is susceptible to noise and outliers, it is resilient against overfitting datasets. Therefore, we should see an increase in performance over the J48 decision tree without boosting.

FEATURE SELECTION

Given the large dimensionality of the feature set within the virtual environment, a goal of the PreCog system is to reduce the feature set to a tractable subset. The benefits of this are twofold: Not only are the results more accurate and resilient to overfitting, but also we receive a substantial decrease in training time. In our analysis of metric selection we evaluate the three methods listed below combined with the choice of maximum number of metrics:

- **Number of metrics:** The range of the number of metrics over which to permute is empirically determined through analysis of classification. We find that at 25 metrics, accuracy no longer significantly increases. Furthermore, training times for datasets that include over 25 metrics take far too long for realistic experimentation purposes (an hour to two hours). Experiments are conducted over the choices of {0, 1, 5, 10, 15, 25} metrics.
- **Information Gain (IG):** Information Gain feature selection [13] deduces which metrics are most valuable to classification by first measuring the entropy for the data partitioned by labeled class as well as on whole, and then determining which metrics offer the

highest discrimination. Metrics with a high variance that correlate well with class are favored most.

- **Gain Ratio (GR):** Gain Ratio feature selection [14], much like IG, deduces which features are most relevant to the data through analysis of a specific feature's entropy in relation to each class. Unlike IG, however, GR favors features that have fewer values.
- **Correlation-based Feature Selection (CFS):** CFS [15] determines the worth of a feature by a correlation-based heuristic that aims to select features that are highly predictive of the class and are not correlated with each other. CFS tends to select a core subset of features that has low redundancy and is strongly predictive of the class.

SUBSAMPLING

Large datasets are subsampled to reduce redundancy before evaluation to provide more efficient approximations to true data evaluation. In large datasets, there will most likely be many more instances of the class "no error predicted to occur" than others. In practice, we find that these classes outnumbered others 100:1. Most of these data points are redundant as well. Large datasets are, thus, subsampled by limiting the number of data points to each class to a certain percentage of the "no error" class. As the results section shows, not only does this improve training and evaluation times, but also the results are more accurate in most cases.

3.5 ACCURACY

As previous researchers have shown [3], accuracy can be a hard metric to judge. While one can make the argument that simply the percentage of correctly classified data points should be a suitable enough indicator, it falls vastly short of a rigorous and thorough tool for analysis. We, therefore, develop two separate metrics, balanced accuracy and false positive cost analysis,

which can be used independently to gauge the accuracy for a choice of classifier, attribute selection, and sampling of data.

Balanced Accuracy

Balanced accuracy is an averaging of the true positive rate for each class. If we use the warning system outlined at the beginning of this section, for example, the balanced accuracy of the results is the average of the percentage of correctly classified data points for each period. As a baseline, if a classifier simply guessed the class for each classifier, it would have a $1/n$ chance, where n is the number of classes, of choosing the correct class. This naïve classifier would thus have a balanced accuracy of $1/n$. Furthermore, if a classifier is able to guess one specific class very well, but falls short in the remaining classes, given that the baseline is $1/n$, the shortcomings of the worst performing classes will be amplified.

False Positive Cost Analysis

For a domain, such as PreCog, that estimates a time until failure with discrete ranges, a misclassification cannot simply be regarded as incorrect; there are varying degrees of incorrectness that must be examined. If a data point, for example, is classified as the level indicating an error will occur in 0-5 minutes, but in fact, the real error did not occur until 5-15 minutes, this should not be counted the same as if that same data point were classified as the level indicating an error will occur in 60+ minutes. Thus, the process of false positive cost analysis adds weight to misclassifications that were further from their true class.

The total cost for a dataset is calculated as the sum of each cost for every classification. Given a cost matrix C and a classification matrix M, where $M_{i,j}$ may be interpreted as "the

number of times class i was classified as class j" and $C_{i,j}$ may be interpreted as "the cost of classifying i as j", the total cost is $T = \sum_{i,j} C_{i,j} * M_{i,j}$

RESULT ANALYSIS

Both balanced accuracy and total cost paint different pictures of how choices of classifiers and other parameters affect classification. While balanced accuracy gives an indication of how well a classifier may do, total cost determines how a classifier may perform when expectations are relaxed. These two indicators can be likened to multidimensional versions of average true positive weighting and false positive analysis, which are widely used in similar works.

3.6 IMPLEMENTATION

The classifiers and data analysis tools are implemented with the open source toolkit WEKA [6]. This choice was made as it is a highly extensible, highly customizable interface that allows for quick and thorough experimentations. All classifiers but the TANTree were package with the WEKA toolkit. The TANTree was developed as a combination of the implementation of the NBTree and TAN classifiers. Furthermore, WEKA implements a robust data mining API such that many forms of input are valid. In this work, arff files are used to store datasets.

3.7 EXPERIMENT EVALUATION

To garner a better understanding of the classification results, for each coupling of experiment and testbed, we perform two separate types of evaluation – 10-fold-cross validation of all acquired data and single week-to-week or day-to-day train and test classification. While the week-to-week evaluation gives a reasonable baseline of performance, the 10-fold-cross validation evaluation delivers a projected accuracy given enough datasets. Furthermore, by

examining the shortcomings in one evaluation compared to the other, we shall derive an estimate for how much data one would need to collect to produce accurate and robust results.

4 EXPERIMENTATION

The goal of this work is not only to determine what types of failures can be predictable, but also how the randomness of the real world affects PreCog's accuracy. To explore both of these goals, two testbeds are used: The controlled environment, JPetStore, gives us a glass box understanding of potential failures and methods to predict them. The independent testbed, Review Board, an up-and-coming server within VMware's network used for code peer review, delivers a black box method to explore PreCog's potential to adapt to new domains.

4.1 EXPERIMENTATION PROCESS

The same process for experimentation is conducted for both testbeds: The testbed runs for a set time with either simulated or real workloads. During this time, the metric collector inserts time-stamped metrics for all available entities. When the experiment is over, the Apache server's log are scraped to insert time-stamped failures into the statistical database. The requests per minute are also recorded for future analysis.

FAILURE

Failure for the experiments is defined by the average response time per page; the SLO for the experiment is defined by the minimum time it would take for the page to load, i.e. the sum of response time for every request per page visits. This metric is calculated by scraping the Apache HTTP server's log, summing the response time for each line, and dividing the sum by the number of pages per average interval. The SLO takes into account server failure codes as well. If

more than 20% of the requests were denied or returned codes other than 200, or any other "OK" code, the server should be considered in a fail state. For both testbeds, the average is calculated over five minute intervals.

The threshold for the SLO is defined by typical user behavior to a website; if a user visits a web page, he will give up after 10 seconds if it has not loaded yet. Although the user may wait longer if parts begin to load, we may still use 10 seconds as hard upper bound on the time we are willing to allow the server to load the entire page. The SLO threshold for JPetStore experiment is thus set at 10 seconds. Review Board, however, sees far fewer long wait times, even for modest definitions such as four seconds. This may be either because coders are impatient and close their browser as soon as they receive any hint that the server is stalling, or there is simply not enough load on the Review Board servers to warrant a large page request time. Thus, the SLO threshold for Review Board is set at 2.5 seconds

4.2 JPETSTORE

A controlled environment was paramount for providing the rigorous and well-understood experimentation necessary for understanding the limitations of any approach. We use a typical 3-tier setup of an ecommerce website, called JPetStore, as the basis of this control testbed. The load was simulated to induce failures arising from varying requests per minute and tasks.

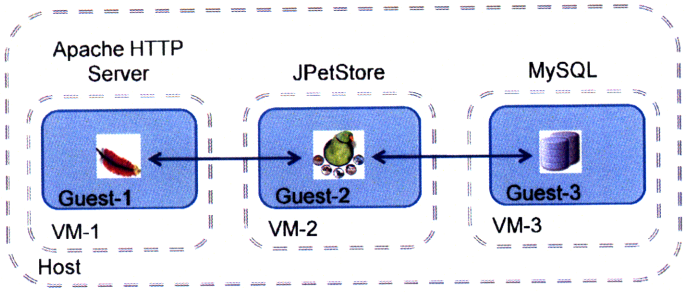


Figure 2: The three-tiered setup of the JPetStore website. The monitored application is the Apache HTTP server hosted on VM-1. When the average response time per page has surpassed 10 seconds, the SLO is violated.

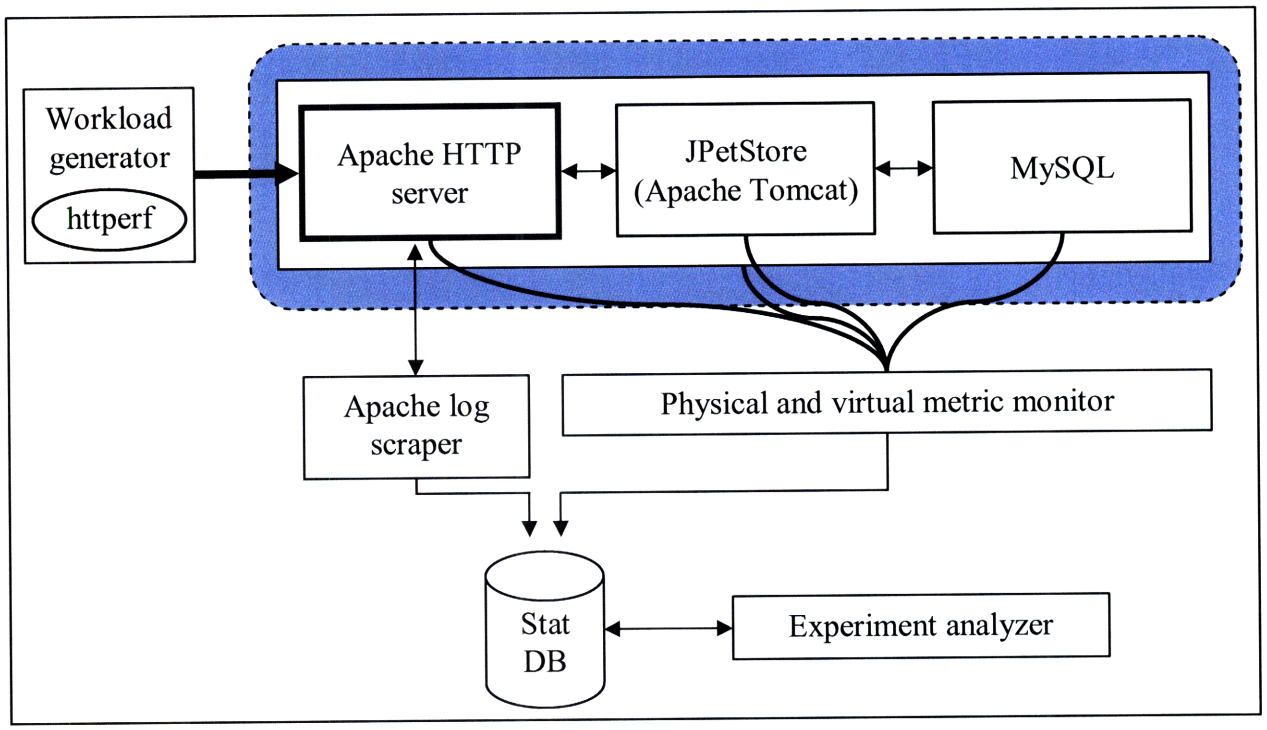


Figure 3: The JPetStore testbed augmented to include a workload generator as well as an experiment analyzer implemented in WEKA. The workload generator perturbs the JPetStore testbed over a set time to generate data. The experiment analyzer performs offline assessment of all classifiers, feature selection methods, and number of features.

- **System configuration:** The JPetStore testbed consists of three virtual machines hosted on a single ESX server. The ESX server is a quad-core 2 GHz AMD Opteron with 4 Gigabytes of available ram running version 3.0.2 of ESX. The Apache HTTP server is version 2.2 running on Linux and configured to optimal performance for high loads. The Tomcat server is version 6.0 running on Windows Server 2003. The MySQL database is version 5.0 on Windows 2003. Each virtual machine is given an equal share of resources.
- **Workload generator:** The workload generator runs on a separate computer, generating load against the JPetStore website. This machine simulates user load by running a Perl script wrapper for httperf [16]. The workload generator combines two parameters every five minutes to create a trial: First, a script for simulated users is created, which is eventually fed to httperf. To recreate realistic code coverage and load, each individual

user within the script performs a logical set of actions generated from the flow chart shown in Appendix A. The workload generator as well uses httperf's ability to create unique cookies such that each user may add items to their shopping cart and check out successfully. The second parameter of the workload generator is the average requests per minute that is put on the server over each period of 5 minutes. This load is a function of the specific workload type – daily rhythm, step, and realistic. Each type is discussed further in the next section.

- **Apache log scraper:** The Apache log scraper is a Perl script, which, given an Apache log, runs through the log and determines, for a given time period, if the SLO has been violated. As mentioned earlier, this is done by summing the total request time for all lines in a given five minute interval, and dividing by the number of page requests to give the average page request time per minute. As performing offline analysis of these logs entails that these logs may be several Gigabytes in size, the scraper works by buffering sections of the log into a lightweight database such as PostgreSQL and determining the SLO by querying this database. In practice, we found the time for offline analysis dropped from 24 hours to three using this multiple stage method.
- **Physical and virtual metric monitor:** The physical and virtual metric monitor polls all entities in the virtual environment for available system metrics, a typical set of which is listed in Appendix B. In the JPetStore testbed, four sets of metrics were received every five minutes – those from the three virtual machines and one from the ESX host. The monitor, written in Python, utilizes public APIs to request real-time metrics from the entities. This method is used so that PreCog, in its experimental stages, can be plugged in

to any VMware virtualized environment in a secure read-only manner, such as we did for Review Board. The collected real-time stats are averaged into five minute period values.

- **Experiment analyzer:** The experiment analyzer performs statistical induction on the data retrieved from the statistical database using WEKA. As discussed in the Machine Learning section, the metrics and SLOs were retrieved from the statistical database and combined form arff files that could be read by the WEKA toolkit. All permutations of metric selection, number of metrics, classifiers, and SLO thresholds were tested in the evaluation process, the results of which were recorded for later analysis.

WORKLOADS

Three workloads test various conditions a typical e-commerce website might endure. Each workload would run over a 24-hour period. Regardless of which workload is chosen, during each five minute interval, noise is added with uniform probability for {-1000, -500, 0, 500, 1000} requests per minute. In each of the figures that follow, a tick mark indicates 30 minutes.

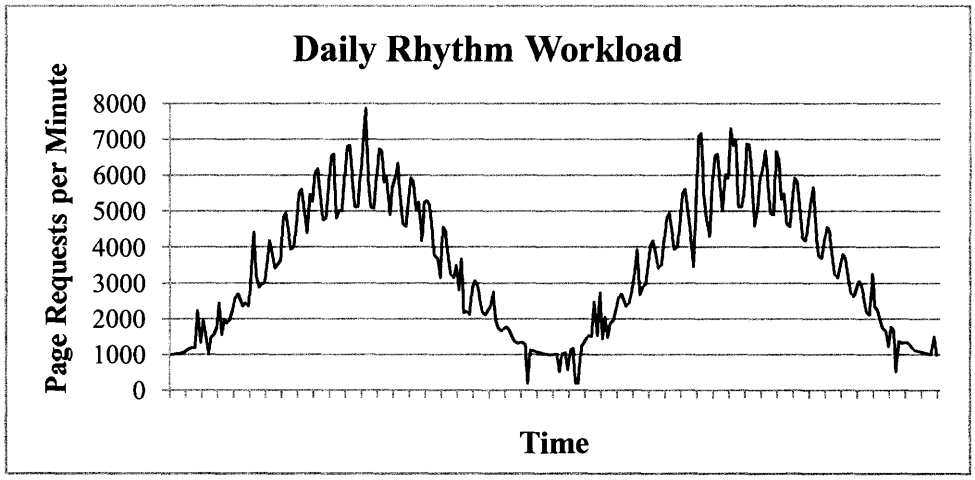


Figure 4: The daily rhythm workload page requests per minute. This workload is considered the easiest to predict.

- **Daily Rhythm:** The daily rhythm workload, shown in Figure 4, simulates a slow rise to a peak of 7000 requests per minute over the course of 24 hours. On top of the outer sinusoid, there is a higher frequency sinusoid to simulate the randomness of typical usage. In analyzing the experiment, we would like to see that an hour before the load reaches its critical amount (empirically this number was found to be around 6000 requests per minute), a gradual escalation of warning levels begins. Furthermore, in an accurately functioning system, we would not like to see warnings after the failure, even though the requests per minute may be at similar levels before a failure occurred. As we show in the results section, it was imperative to include metrics such as first degree derivates of metrics that would differentiate negative instead of positive slope of metric changes.

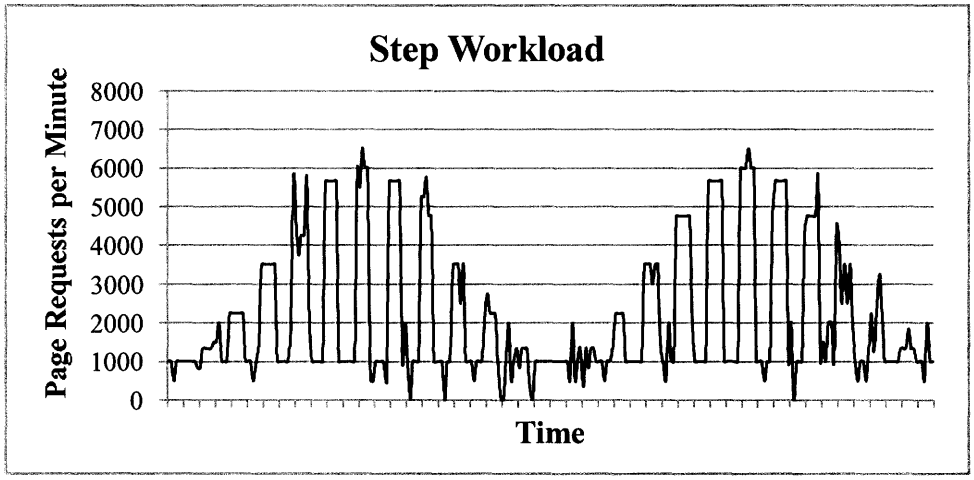


Figure 5: The step workload page requests per minute. This workload is considered harder than the daily rhythm workload because of its sharp steps.

- **Step**: The step workload's overall structure, shown in Figure 5, is very similar to the daily rhythm workload. The major difference is that there is no longer any gradual

increase of requests per minute. Every 30 minutes, the workload switches from 1000 requests per minute to what it would be if it were performing the daily rhythm workload. These large steps are meant to jar the system and test sudden increases in user load. The step workload serves two purposes: Firstly, training and testing with just step workloads should reveal that although PreCog is not able to predict these sudden changes, it still accurately identifies failure as it occurs. Secondly, training with daily rhythm workloads and testing with step workloads should show that training across domains is possible. That is, although user load may be different across different weeks, the system should still be trainable with an amalgamated input.

- **Realistic:** The realistic workload, shown in Figure 6, represents a load that a website may see if website users are in two distinct time zones. This workload proved very reliable for testing false positives, i.e. this workload allows us to examine if PreCog generates any warning levels along the first large hump, as it does not quite reach failure.

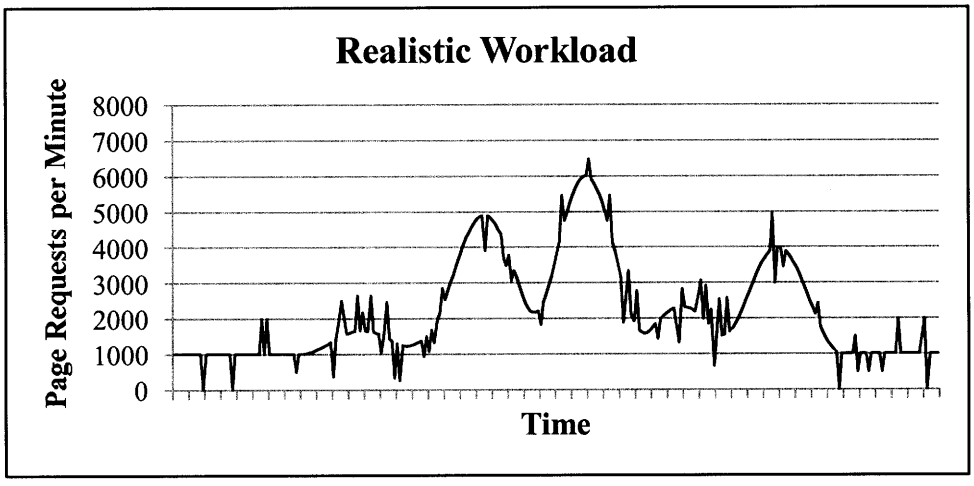


Figure 6: The realistic workload page requests per minute. This workload represents the load a typical website may see if its users were in two separate time zones. The shorter first hump allows for testing of false positives.

EXTERNAL VARIABLES

One of the paramount goals of PreCog is to determine how well classification works given variables external to the system. One such variable that we have already discussed is noise. As VMs do move to and from hosts in typical large environments, it is critical to determine how these moves affect classifications. If a new VM were to appear on an already fully utilized host, performance of the remaining VMs will drastically decrease. Obviously, there is no way to predict such a sudden of a change, but we can use experiments that mimic this scenario to ensure that we may still accurately detect if any machines have violated their SLOs.

- **Realistic and hog experiment:** The realistic and hog experiment explores the scenario that an unknown virtual machine moves onto the same host server running JPetStore. The hog VM is given an equal share of resources as the other VMs. We simulate random behavior by having the hog VM run one of four different tasks periodically for 30 minutes every two hours. These tasks are chosen to simulate random loads a new VM may put on the system. They include maximizing CPU usage, copying large amounts of data over the network, creating a large amount of disk read/writes, and ballooning use of memory. In the results we hope to see that although the sudden change of the system to a fail state could not be predicted, while the failure occurred, PreCog is able to determine that the SLO had been violated.

4.3 REVIEW BOARD

To explore PreCog's ability to perform in a variety of domains, we evaluated its accuracy on a real world server in a black box manner. The Review Board experiment is very similar to the JPetStore experiment in that the same offline analysis is performed. Instead of generating a

workload, Review Board is monitored in one-week intervals, during which time it was used internally within the VMware network.

EXPERIMENT SETUP

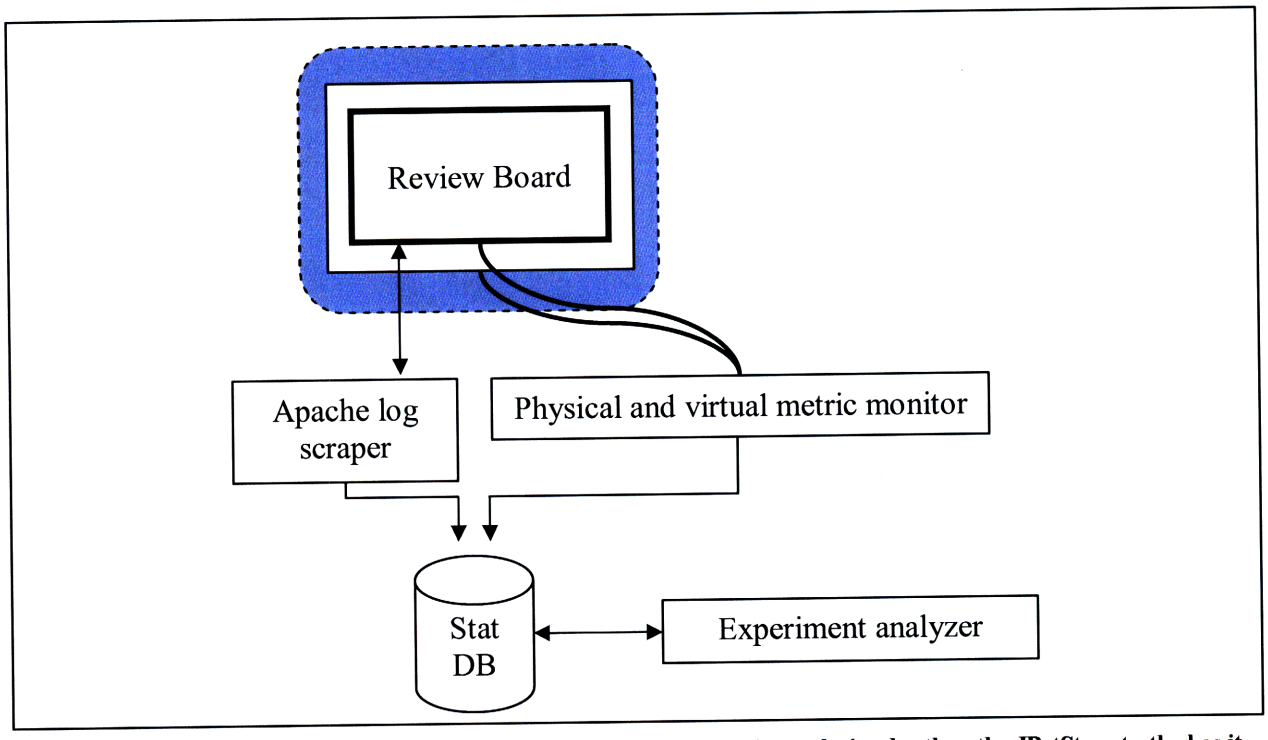


Figure 7: The Review Board experiment setup. The Review Board testbed is much simpler than the JPetStore testbed as it only has one host and virtual machine. It was used day-to-day internally within VMware to collect data for analysis.

- **Review Board**: Review Board is a peer code-review website designed to allow management of code differentiations and bugs. In its initial stages, Review Board was very responsive to user input. As more users discovered Review Board, however, the server slowed to unacceptable levels during peak usage times. Furthermore, the authors of Review Board determined that SLO violations occur not only when the load of Review Board increased to levels near 300 requests per minute, but also when those users began committing large code differentiations. It was not merely enough to monitor the user load

to determine when a failure might occur. A sample of a typical week of usage can be seen in Figure 8.

- **System configuration:** To maintain a black box nature of the testbed, not much is known about the configuration of the Review Board setup. Although it is known that the virtual environment only consists of a single host and VM running Linux, the specifications of the host are unknown.
- **Experiment analyzer:** The experiment analyzer is used much in the same way it was for the JPetStore testbed for offline analysis of PreCog. Because far fewer encounters of failure occur, however, the data must be subsampled to increase accuracy. The effects of this process, as described in the Machine Learning section, are discussed further in the Results section.

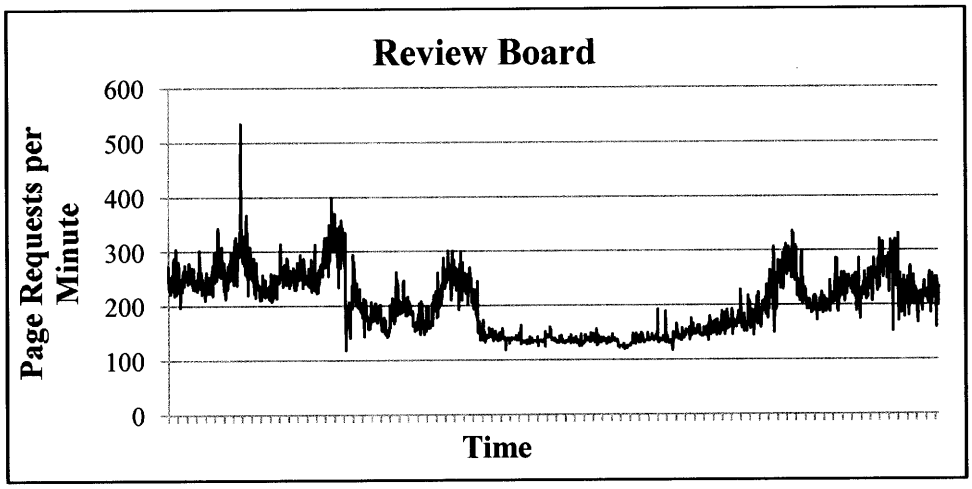


Figure 8: Review Board page requests per minute for a single week beginning on a Wednesday and ending on the following Tuesday.

5 RESULTS

The purpose of the experiment evaluation is to determine which machine learning techniques are most useful in each of our testbeds. Our results show that, indeed, classification accuracy is domain-specific; parameters that worked well for JPetStore did not fare as well when evaluating Review Board. We examine the results of classification over the parameters of metric selection, number of metrics chosen, classifier, and, in the case of Review Board, the amount of subsampling.

5.1 DOMAIN ANALYSIS

The results of classification showed that overall accuracy of the PreCog system is very dependent on domain; i.e., a scheme that works well for one testbed may not work for another. We may gain insight into the dependence on domain by first analyzing the effects each workload had on the classification of warning levels. In each of the figures below, pairs of dashed lines represent periods of failure. A system that classifies these failures accurately outputs a warning level of 4 during these periods as well as the correct levels leading up to the point of failure.

- **Daily Rhythm:** The daily rhythm workload is the most predictable workload of the three. When the system violates the SLO, the average response time per page being greater than 10 seconds, it remains in violation until the load has subsided. The classification of this workload is very accurate. As shown in Figure 10, the SLO violation is accurately predicted around an hour before the first failure period. Furthermore, there are few false positives on the downward slope of the hump, indicating that first order derivates of metrics came into play.

- **Step:** The step workload is the least predictable of the three; its sharp rise and falls work against any strong correlation of first order derivative metrics. Furthermore, as shown in Figure 11, the SLO violation period is not stable; in the figure, black bars represent the short noncontiguous periods. Although the system may reject all requests immediately following a failure, and as such violate its SLO, no requests are being made to the system during this off period. We see in Figure 12 that while classification still identifies failure while it is occurring, the classifier is much more prone to false positives.
- **Realistic**: The realistic workload showed that although false positives were rare in this domain, they most likely could not be avoided and could even offer valuable information to an administrator. As seen in Figure 13, although the only failure period occurred during the second large hump, the first hump produces a slight rise in the average response time. During classification, shown in Figure 14, this rise was represented by a sudden level 2 warning. Although no error occurs 15 to 30 minutes from this point, instead, the warning level is correctly indicating the health of the system. In other words, given the current system state, it is likely that within the next 15 minutes the system will fall into an SLO violation given the current type of load.
- **Review Board:** The Review Board testbed offered the most varied domain. Of the three weeks in which data was collected, only one week offered a significant number of failures, defined by the average response time per page exceeding 2.5 seconds, an example of a single day is shown in Figure 15. Although, classification, shown in Figure 16, was accurate, the SLO seemed to move rapidly across the threshold, which may cause future classification to not be as accurate.

5.2 SUBSAMPLING

As mentioned, subsampling the data to reduce the number of level 0 classes improves accuracy as shown in Figure 9. By reducing the number of data points labeled as level 0, we still maintain high accuracies within that class, but also normalize the prior probabilities such that it would reduce the threshold to move from the more probable level 0 class to any other class. As well, empirical testing showed no difference for values larger than 20%. It is still unknown, however, why there is a significant dip at subsampling by 20%. This data point may be due to reaching threshold, past which, the subsampling procedure begins to improve accuracy. Nevertheless, from examining the results of subsampling, one can determine that for datasets of several weeks, subsampling must occur; and to what degree will depend on classification technique and domain.

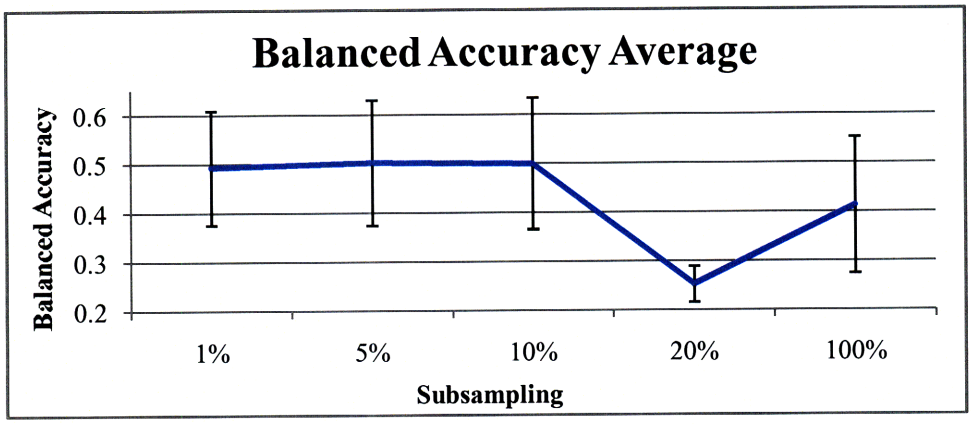


Figure 9: Subsampling within the Review Board dataset causes an increase in balanced accuracy. That is, by reducing the overwhelming number of level 0 labeled data points, more emphasis can be added to other key classifications points.

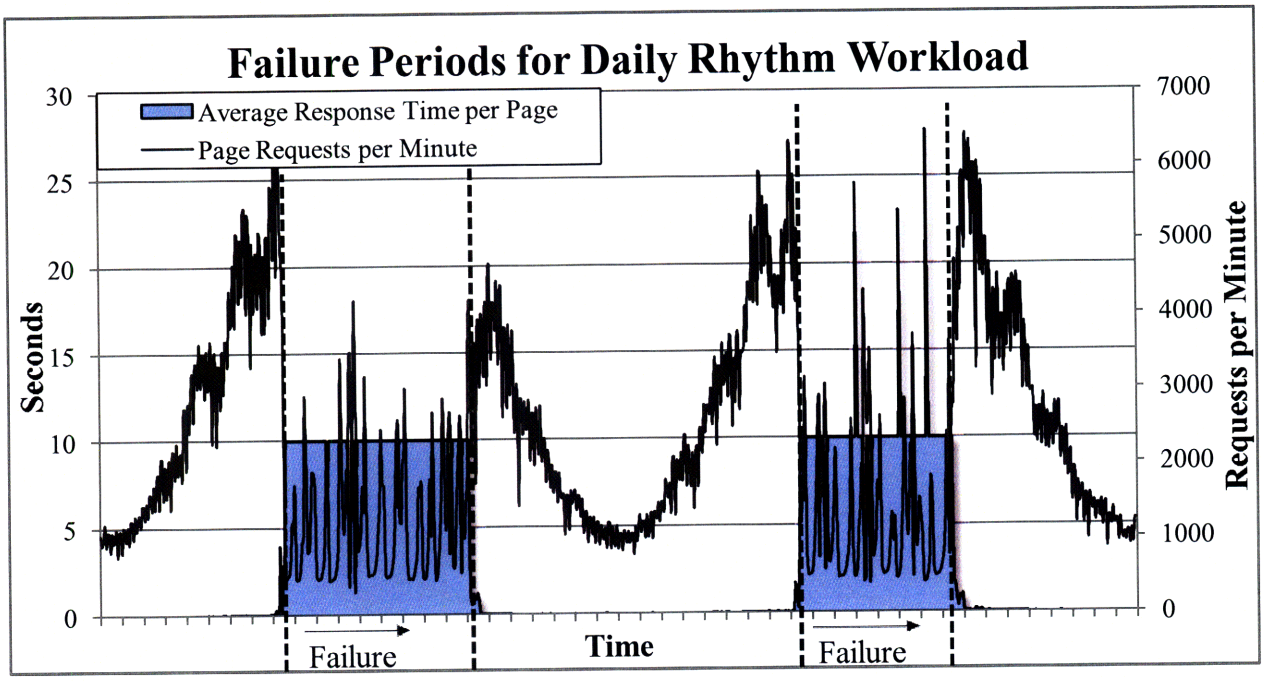


Figure 10: Daily rhythm workload experiment results showing the average response time per page in the shaded region and the load induced on the system in the background. The dashed lines represent the beginnings and ends of the failure periods where the average response time per page has crossed the 10 second threshold.

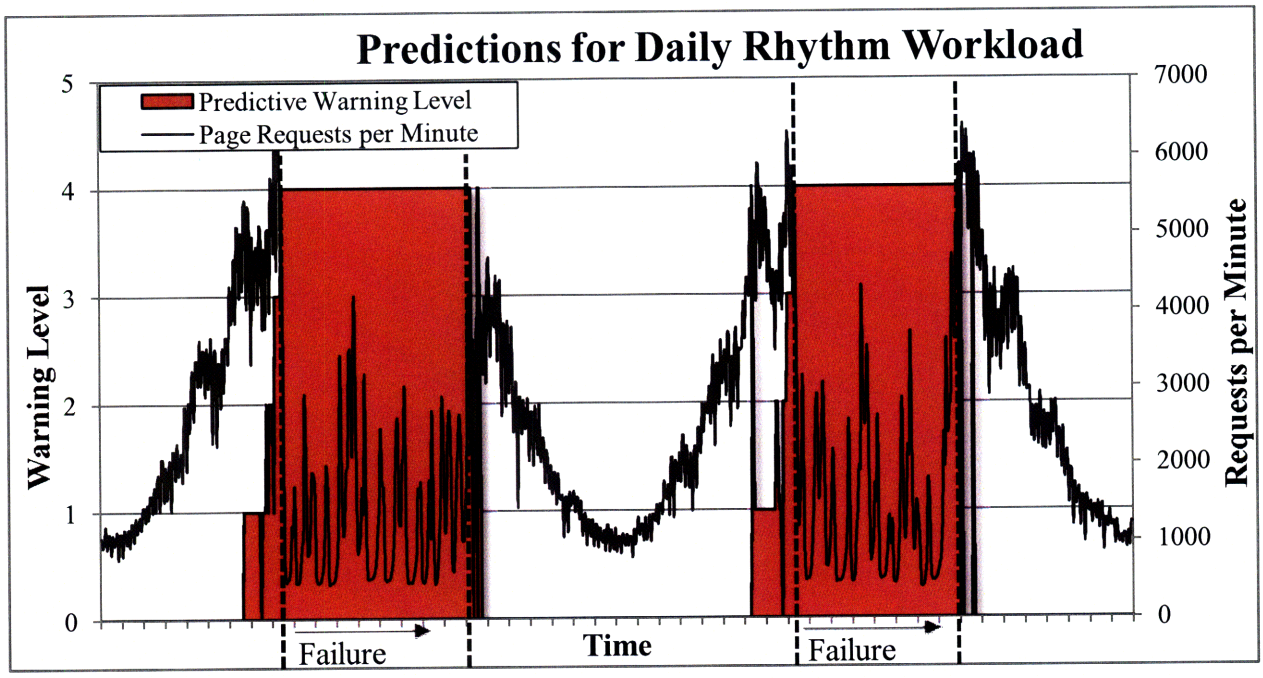


Figure 11: Daily rhythm workload classification results showing the predictive warning level in the shaded region and the load induced on the system in the background. Notice as we move along time to a failure period, the warning system escalates. It begins with a level 1 warning approximately 60 minutes before the first failure and continues to increase until it has reached a level 4 warning, which corresponds with the failure depicted in the shaded region of Figure 10.

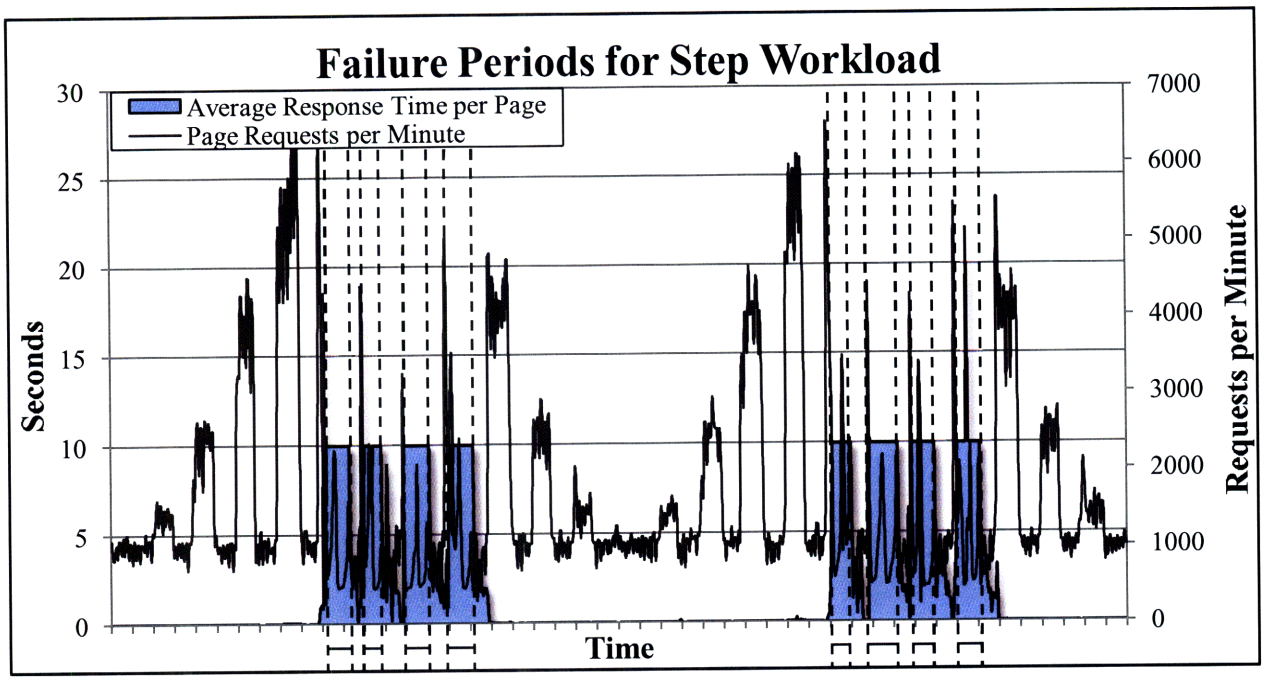


Figure 12: Step workload experiment results showing the average response time per page in the shaded region and the load induced on the system in the background. Each set of dashed lines with corresponding horizontal bar represents a failure period. Notice the sharp transitions to and from failures.

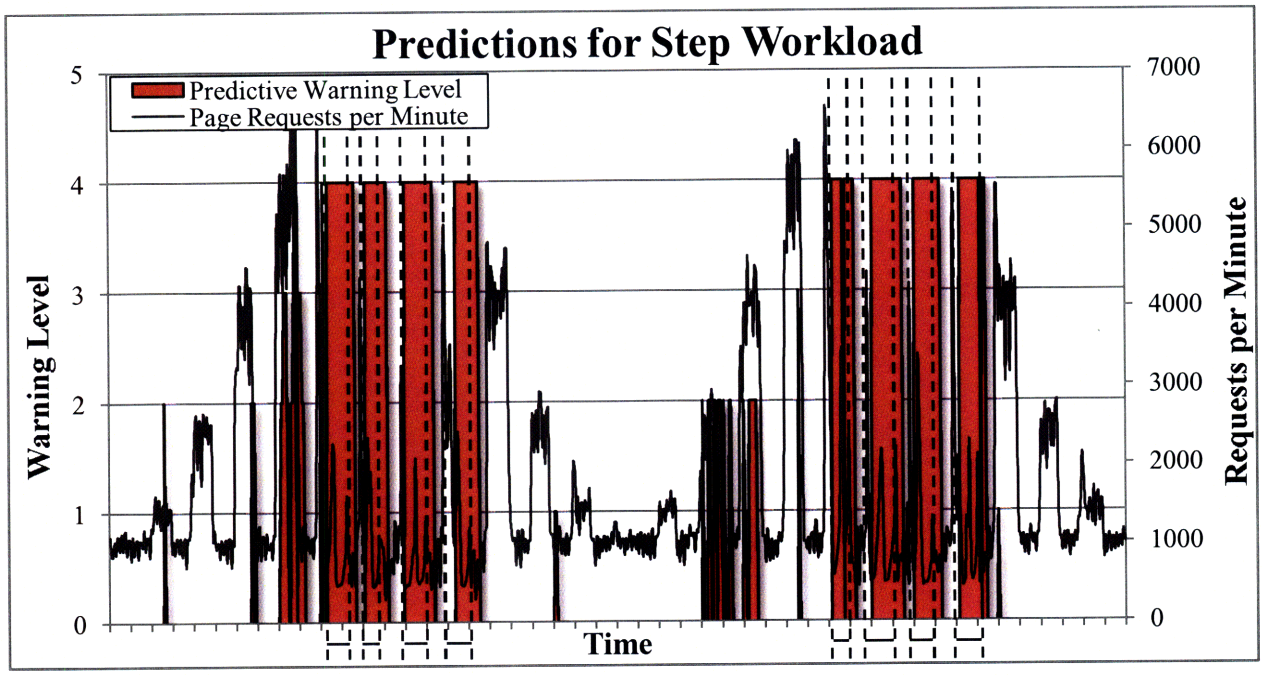


Figure 13: Step workload classification results showing the predictive warning level in the shaded region and the load induced on the system in the background. Although many of the failure periods are correctly classified, many false positives occur. The quick transitions most likely worked against any correlation made using first degree changes in metrics.

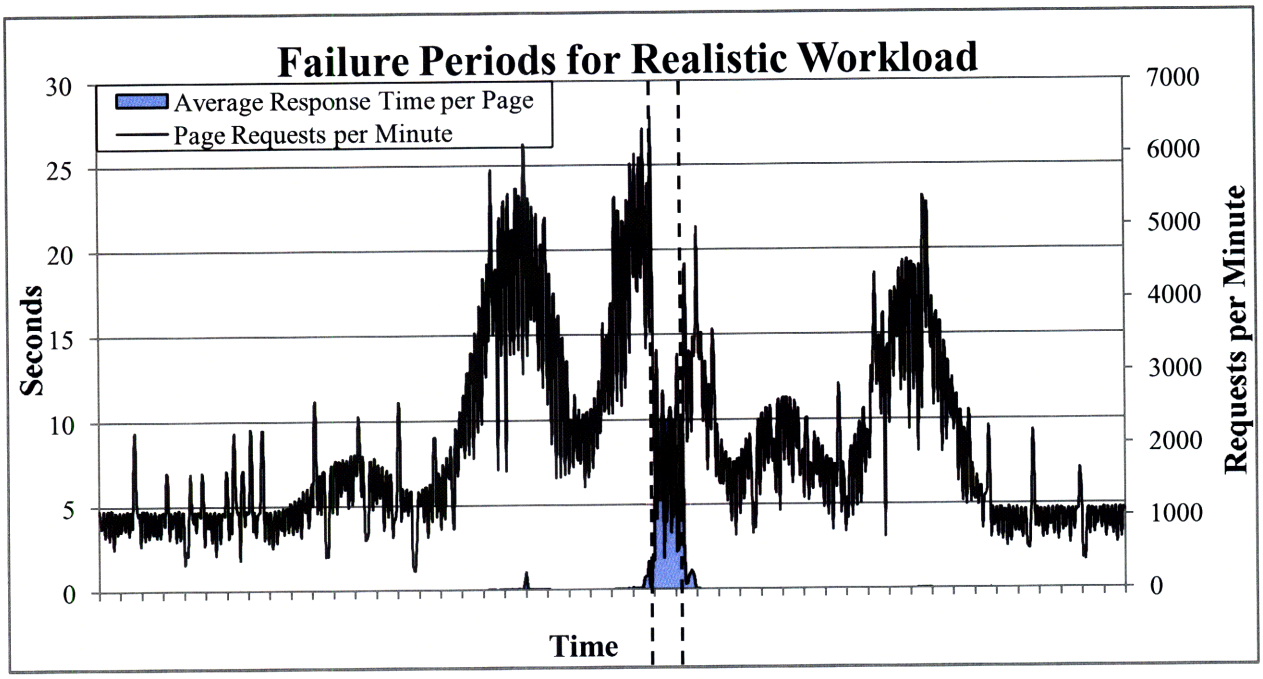


Figure 14: Realistic workload experiment results showing the average response time per page in the shaded region and the load induced on the system in the background. Although there are many dips above and below the 10 second threshold in our failure period marked by the dashed lines, we shall consider this a single period.

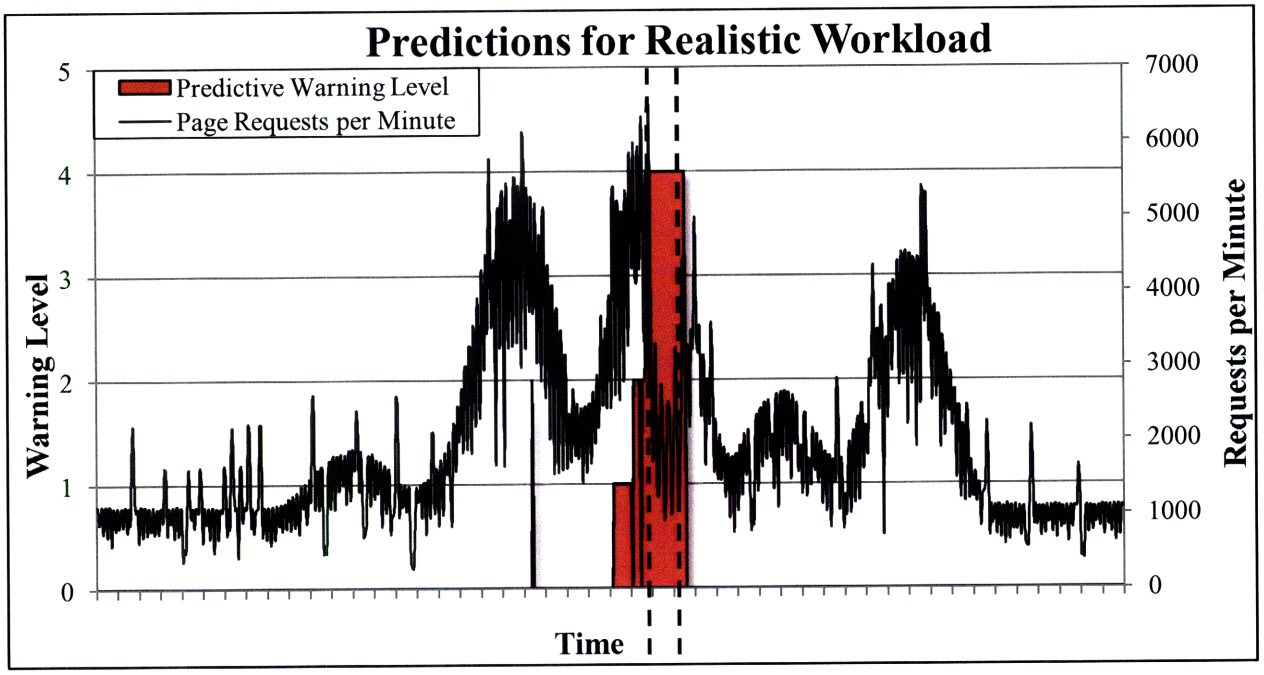


Figure 15: Realistic workload classification results showing the predictive warning level in the shaded region and the load induced on the system in the background. Notice that approximately and hour before the first failure, we correctly classify the level 1 failure and proceed to escalate until a level 4 error is reached during the actual fail period. Furthermore, the false positive in the first hump is not a strict misclassification if we few it as a warning that given the current state, it would have been likely the system would have entered a fail period had it not died down.

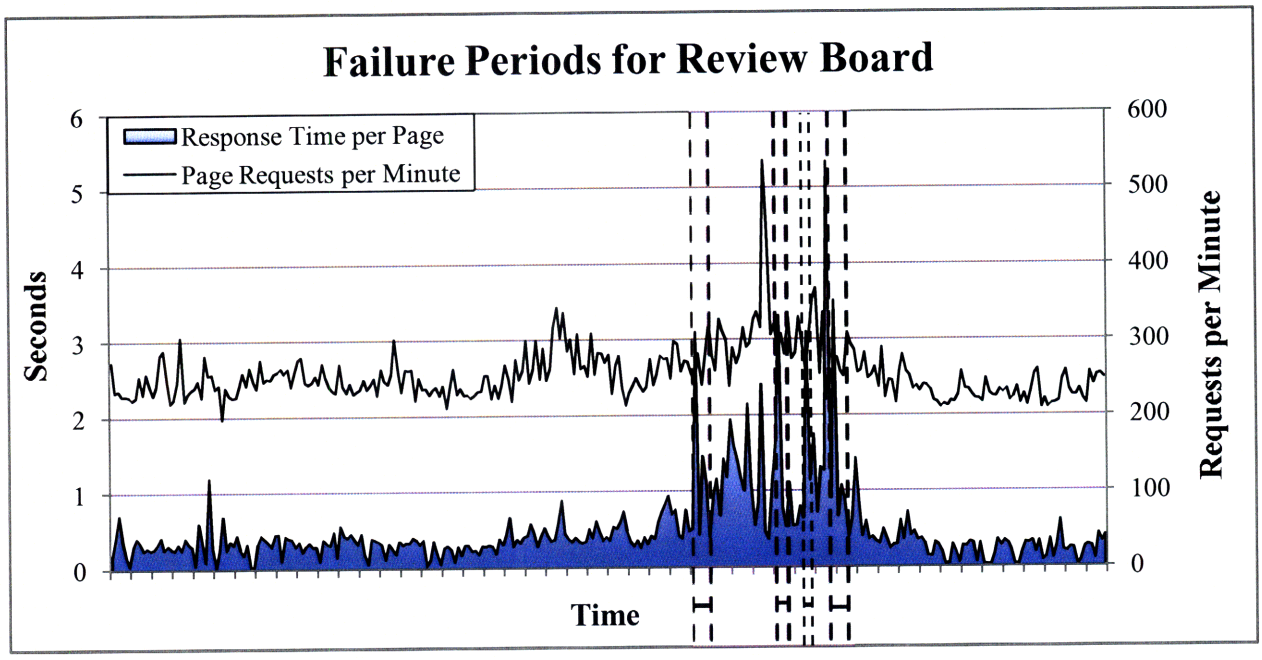


Figure 16: Review Board experiment results showing the average response time per page in the shaded region and the load induced on the system in the background for a single day. Notices how there are no prolonged periods of failure. This sporadic behavior is hard to classify perfectly. Thus, total cost may shed a better light on these results.

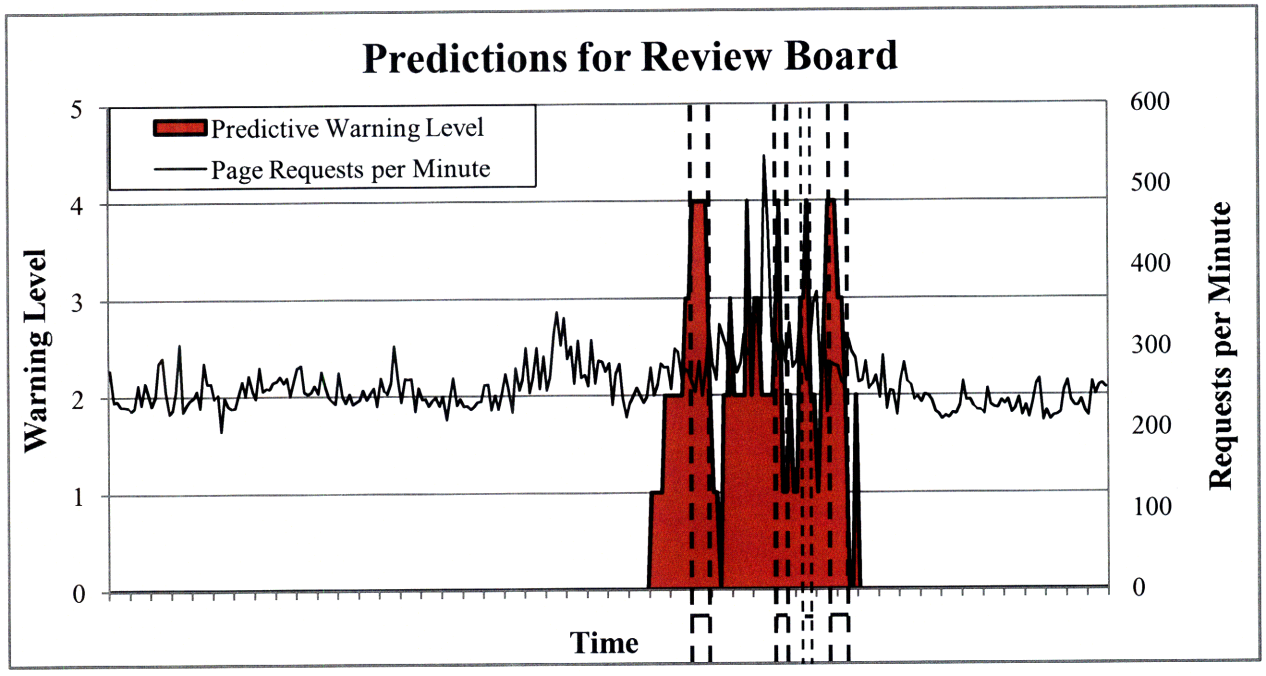


Figure 17: Review Board classification results showing the predictive warning level in the shaded region and the load induced on the system in the background. Notice that because of the sporadic behavior, not directly due to load on the system, the warning level was more difficult to classify correctly. Although SLO violations were accurately detected, there are many false positives up to those points.

5.3 Feature Selection

Varying the parameters of feature selection method and the optimal number of features, i.e. metrics, to select, showed that although feature selection was domain specific, there did exist general trends that promote a higher average balanced accuracy.

Number of metrics

In stable workloads with a large training dataset, we find that increasing the number of metrics used in classification increases the overall accuracy of the system. For a full data cross validation of the daily rhythm workload, shown in Figures 18 and 19, we see drastic improvements by increasing the number of metrics up to 10, at which point we no longer see any significant gains. For a smaller datasets, such as training and testing a daily rhythm workload with only one day of data shown in Figure 20, however, we find that effects of the number metrics to be positive but not as significant.

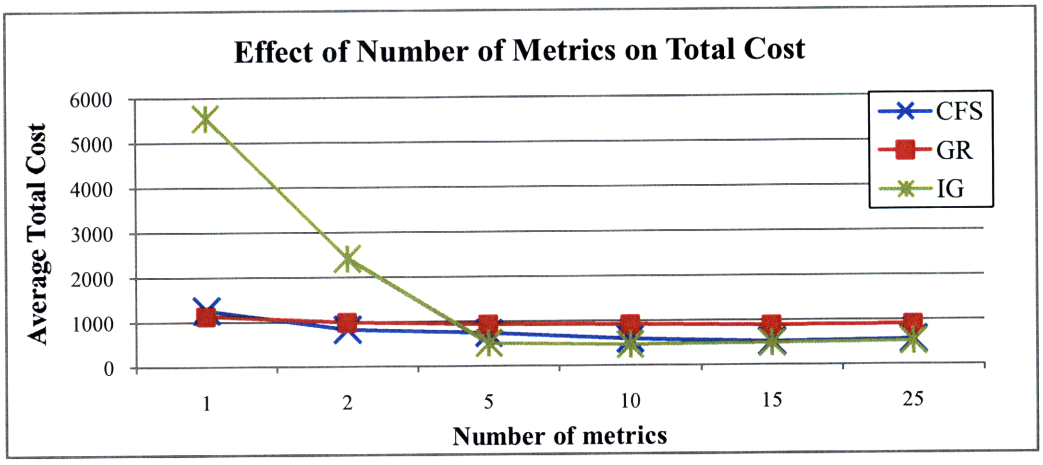


Figure 18: Effect of metric number on total cost for daily rhythm cross validation. As the number metrics increases, the total cost of misclassification goes down, indicating an increase in accuracy. After 10 metrics, there are no longer any significant gains being made.

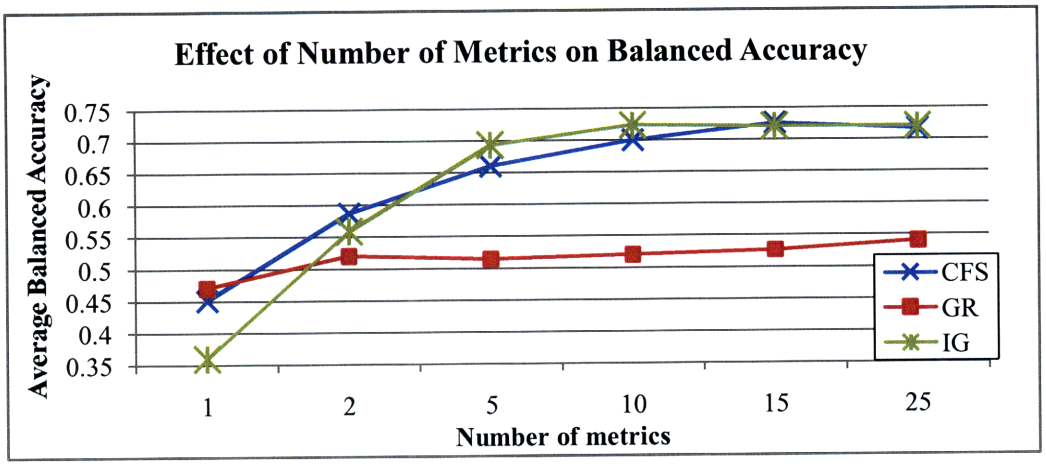


Figure 19: Effect of metric number on balanced accuracy for daily rhythm cross validation. As the number of metrics goes up, the IG and CFS feature selection algorithms perform better, but do not increase much after 10 metrics. The GR feature selection method is unaffected by the number metrics selected.

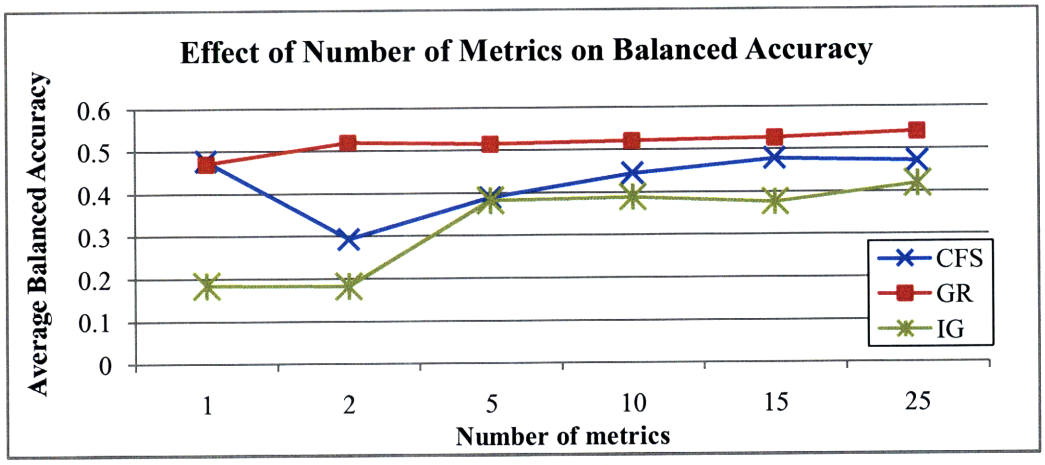


Figure 20: Effect of metric number on balanced accuracy for a daily rhythm day-to-day evaluation. We notice that in testing smaller datasets, there is a loss of stability of the general pattern seen in Figure 19. We also notice that IG and CFS perform poorly in this experiment, leading us to believe that CFS and IG will perform better on larger datasets.

We also find that in domains that have a tendency to be unpredictable, such as the step workload, increasing the number of metrics will reveal performance limitations specific to that domain. In one such experiment of training PreCog with a daily rhythm workload and testing with a step, shown in Figure 21, an increase of the number metrics causes a convergence of the

balanced accuracy, indicating that choosing optimal metrics only plays a single part in the classification system as a whole.

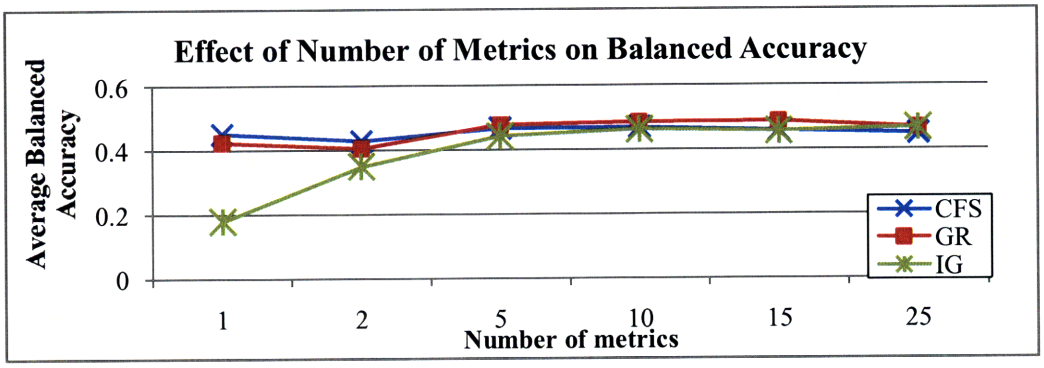


Figure 21: Effects of number of metrics on balanced accuracy for a step to daily rhythm day-to-day evaluation. Notice that a onvergence of balanced accuracy indicates that choosing optimal metrics only plays a single part in the classification system as a whole.

Furthermore, when examining a domain that is truly unpredictable, such as training a system with a real workload and testing with the same workload, but with a foreign VM competing for resources, shown in Figure 22, we again reach an upper-bound on how well metric selection will improve accuracy. Although we cannot predict the future in this environment, the correct choice of metric number and selection method ensures that the potential accuracy will be maximized.

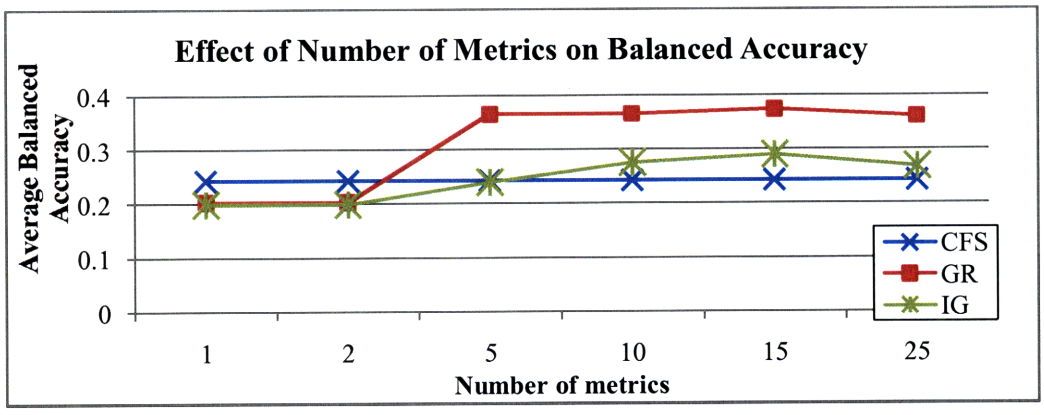


Figure 22: Effects of number of metrics on balanced accuracy for a realistic workload tested with a foreign VM in a day-to-day evaluation. Although there is a slight increase in accuracy for 5 metrics, we are limited by the lack of information inherent in the unpredictable domain.

5.4 Feature Selection Method

Through analysis of the average total costs and balanced accuracies in all experiments, we found that no single feature selection method proved best for all domains. The GR selection method, shown in Figures 23 and 24, ranked lowest among domains when cross validation evaluation was used. The IG and CFS selection methods ranked highest in cross validation experiments, with CFS ranking just slightly higher than IG in most domains. In examining all results, it appears that CFS is the preferred metric selection method.

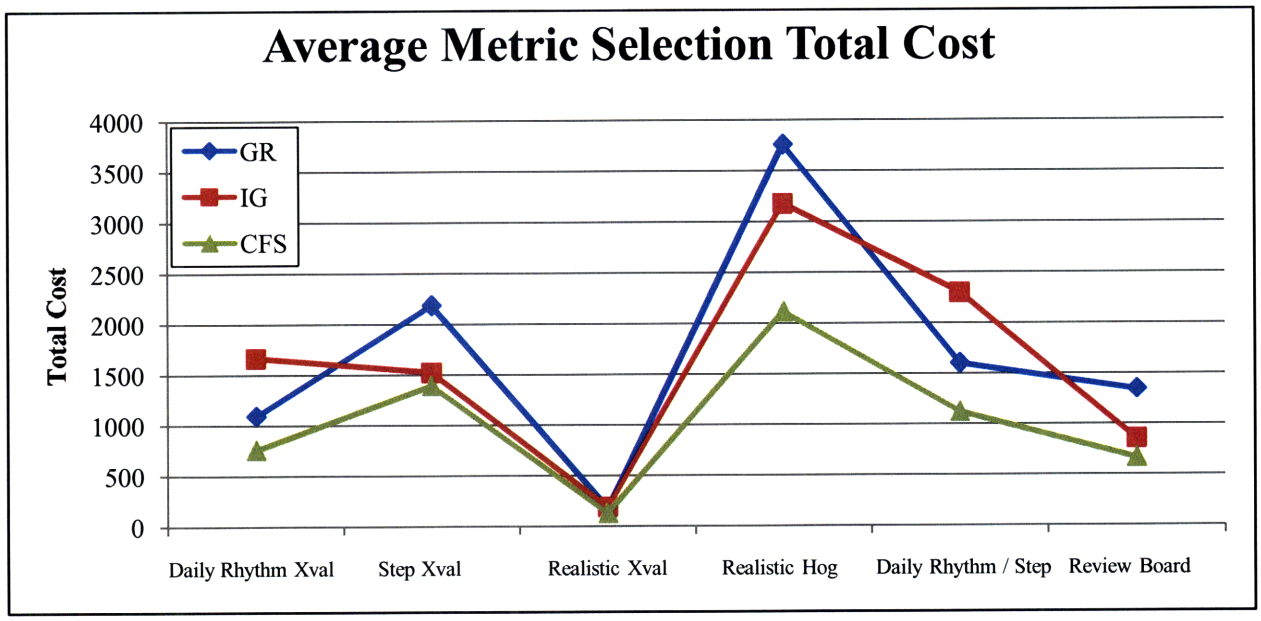


Figure 23: The average metric selection method total cost per domain. CFS across all domains has a smaller total cost when used for feature selection for a classifier. Given a random domain, CFS would be the safest choice to use as a feature selection method.

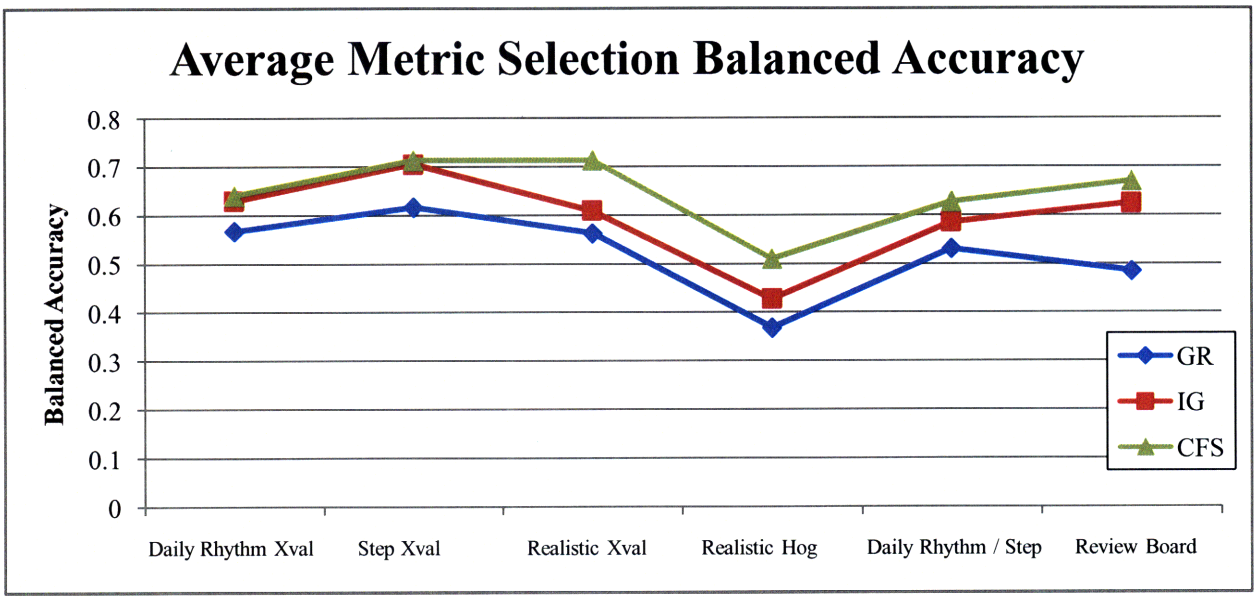


Figure 24: The average balanced accuracy of metric selection per domain. We see here that CFS again has a higher accuracy than IG and GR. We also notice however that as where GR overtook IG in some domains within Figure 23, there is now an ordering of CFS, IG and GR. That is, in some domains, GR was able to bring more missclassifications closer to their true values than IG was.

5.5 Classifier Selection

Through analysis of each classifier over all permutations of parameters in each domain, we see two significant results: AdaBoost generates a higher balanced accuracy and lower total cost over almost all domains and since NB performs poorly in most domains, we may deduce that the metrics are not independent. From examining Figure 25, we see that in all experiments but the train with daily rhythm and test on step, AdaBoost generates a higher balanced accuracy than all other classifiers. Most likely, AdaBoost's tendency to focus on boosting misclassifications worked against itself for a domain in which there is a high variance of data. The NB classifier on the other hand performed poorly on all but the daily rhythm/step experiment. As the NB classifier assumes that all features are independent, it follows that in an environment where features are likely to have dependencies (a sharp spike in CPU usage will see a large spike in network usage on a web server) NB would perform poorly, such as we see in Figure 26. Furthermore, the assumption that system metrics were independent proved to help in the daily

rhythm/step experiment where it was likely that large variances in system metric values prevented any dependence being induced.

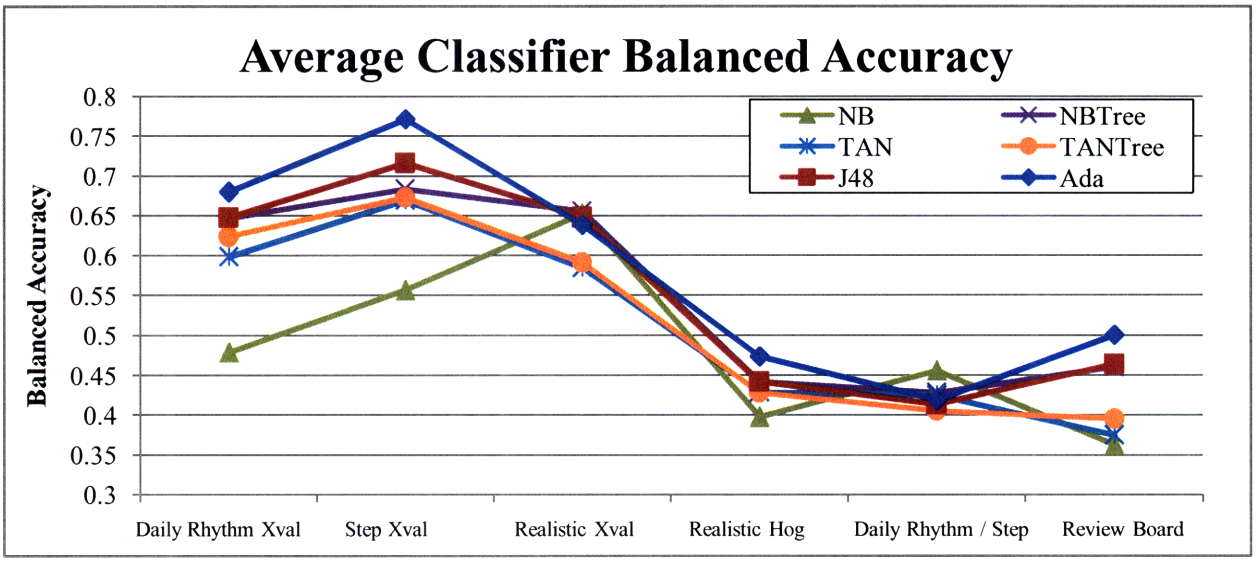


Figure 25: The average classifier balanced accuracy per domain. In all domains but the train on daily rhythm and test on step, AdaBoost does significantly better. Furthermore, NB's poor accuracy in the daily rhythm cross validation domain validates the claim that there exist dependencies between metrics.

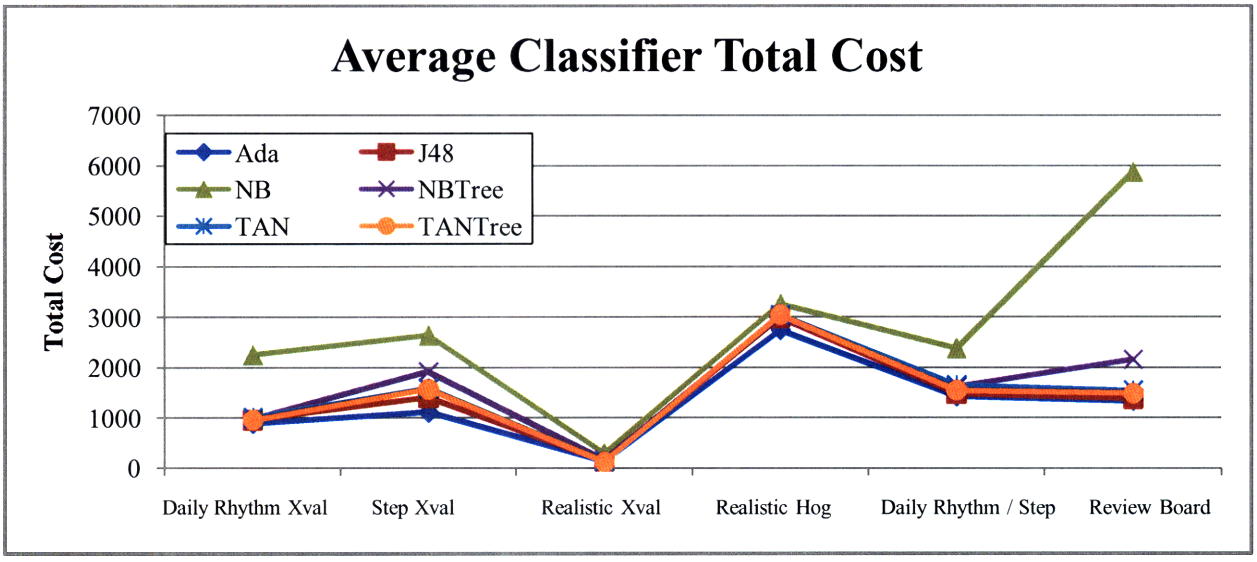


Figure 26: The average classifier total cost per domain. In all domains, the NB classifier has a higher average total cost. NB especially does poorly in the Review Board domain where there would be significant dependencies since there exists only one VM and host.

6 CONCLUSIONS

In this work, we have shown that a reliable and accurate system may be developed by correlating system metrics with application failures using statistical induction. Using statistical induction, namely AdaBoost with CFS metric selection, PreCog is able to induce a behavioral model of the system without any prior knowledge of the domain. Furthermore, by leveraging a virtual environment, any domain can be adapted to the PreCog system given a proper SLO.

6.1 OVERALL PERFORMANCE

We have shown that, although PreCog system is reliable and efficient at determining future application failure, it is very much dependent on domain. That is, we often find that we are not limited by the classifier, metric selection method, or number of metrics we choose, but by the information that the domain can provide. The daily rhythm workload on the JPetStore testbed, for example, had many clearly labeled instances of failure from which an accurate classifier could be induced. When applying this classifier to a separate workload however, in this case the step workload, accuracy drastically diminished.

Although each domain presented its own difficulties in classification, AdaBoost tended to outperform all other classifiers in terms of balanced accuracy and total cost. Coupled with the IG or CFS metric selection methods, one would expect to achieve 85% balanced accuracy in workloads similar to daily rhythm and realistic. Furthermore, even in systems that are highly unpredictable, such as the realistic/hog, AdaBoost maintains accuracies well above 50%, mostly due to maintaining accuracy of at-time failure.

6.2 Autonomic Computing

Although PreCog has proven itself useful as a strong foundation, there are still many directions in which to explore. For one, there exists the problem of maintain adaptable system models. If PreCog successfully models system behavior such that an autonomic computing agent, an agent that aims to diagnose and repair failures autonomously [17], may act on this failure so as to avoid it, then fewer failures will occur. If, however, the workload to the system changes, and a new failure arises, it is unclear how the system should adapt to this change. Using the system we have developed in this work, a new statistical model may be induced using the data that produced the SLO violations. Following this training session, PreCog would now rely on this statistical model to predict future application failure. Although we have now successfully replaced the model, our old failure may go undetected. If there are stable workloads such as Review Board, then this should not be an issue as training sessions may be rare. If PreCog is implemented, however, in an environment that has a high amount of VM movement, as seen in the realistic/hog experiment, accuracies and reliability will likely decrease.

PreCog shows a large potential to be utilized as part of a universal autonomic computing environment. It can provide the much needed preparation time for avoiding or mitigating failure, while also providing an accurate assessment of the current system health. Although PreCog has shown itself to be reliable when faced with high levels of noise, highly unpredictable domains still prove to be a problem for this system. Finding reason within the randomness may one day help PreCog, and predictive systems in general, achieve higher levels of accuracy and usefulness in the ever-changing environments in which they are deployed. PreCog is an important step in that direction.

BIBLIOGRAPHY

1. *Combining statistical monitoring and predictable recovery for self-management.* **Fox, Armando, Kiciman, Emre and Patterson, David.** New York, NY, USA : ACM Press, 2004. WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems. pp. 49-53.

2. *A decision-theoretic generalization of on-line learning and an application to boosting.* **Freund, Yoav and Schapire, Robert E.** 1995. European Conference on Computational Learning Theory. pp. 23-37.

3. *Correlating instrumentation data to system states: a building block for automated diagnosis and control.* **Cohen, Ira, et al.** Berkeley, CA, USA : USENIX Association, 2004. OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation. pp. 16-16.

4. *Bayesian Network Classifiers.* **Friedman, Nir, Geiger, Dan and Goldszmidt, Moises.** 2-3, Hingham, MA, USA : Kluwer Academic Publishers, 1997, Mach. Learn., Vol. 29, pp. 131-163.

5. *Short term performance forecasting in enterprise systems.* **Powers, Rob, Goldszmidt, Moises and Cohen, Ira.** New York, NY, USA : ACM Press, 2005. KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 801-807.

6. **Witten, Ian H and Frank, Eibe.** *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems).* s.l. : Morgan Kaufmann, 2005.

7. *Three research challenges at the intersection of machine learning, statistical induction, and systems.* **Goldszmidt, Moises, et al.** Berkeley, CA, USA : USENIX Association, 2005. HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems. pp. 10-10.

8. **Duda, R O and Hart, P E.** *Pattern classification and scene analysis.* s.l. : A Wiley-Interscience Publication, New York: Wiley, 1973, 1973.

9. **Quinlan, J Ross.** *C4.5: programs for machine learning.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1993.

10. **Mitchell, Thomas M.** *Machine Learning.* s.l. : McGraw-Hill Higher Education, 1997.

11. *Estimating Continuous Distributions in Bayesian Classifiers.* **John, George H and Langley, Pat.** pp. 338-345.

12. *Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid.* **Kohavi, Ron.** 1996. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. pp. 202-207.

13. **Ayan, Necip Fazil.** *Using Information Gain as Feature Weight.*

14. **Hall, M and Smith, L.** *Practical feature subset selection for Machine Learning.* 1996.

15. **Hall, M.** *Correlation-based Feature Selection for Machine Learning.* 1998.

16. *httperf: A tool for measuring web server performance.* **Mosberger, David and Jin, Tai.** 3, New York, NY, USA : ACM Press, 1998, SIGMETRICS Perform. Eval. Rev., Vol. 26, pp. 31-37.

17. *Failure Diagnosis Using Decision Trees.* **Zheng, Alice X, Lloyd, Jim and Brewer, Eric.** Washington, DC, USA : IEEE Computer Society, 2004. ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04). pp. 36-43.

18. *Ensembles of Models for Automated Diagnosis of System Performance Problems.* **Zhang, Steve, et al.** Washington, DC, USA : IEEE Computer Society, 2005. DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05). pp. 644-653.

19. *On the Use of Fuzzy Modeling in Virtualized Data Center Management.* **Xu, Jing, et al.** s.l. : IEEE Computer Society, 2007. ICAC. p. 25.

20. *Critical event prediction for proactive management in large-scale computer clusters.* **Sahoo, R K, et al.** New York, NY, USA : ACM Press, 2003. KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 426-435.

21. *Using computers to diagnose computer problems.* **Redstone, Joshua A, Swift, Michael M and Bershad, Brian N.** Berkeley, CA, USA : USENIX Association, 2003. HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems. pp. 16-16.

22. *The Vision of Autonomic Computing.* **Kephart, Jeffrey O and Chess, David M.** 1, Los Alamitos, CA, USA : IEEE Computer Society Press, 2003, Computer, Vol. 36, pp. 41-50.

23. *Detecting performance anomalies in global applications.* **Kelly, Terence.** Berkeley, CA, USA : USENIX Association, 2005. WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems. pp. 8-8.

24. *Eigenspace-based anomaly detection in computer systems.* **IDÉ, Tsuyoshi and KASHIMA, Hisashi.** New York, NY, USA : ACM Press, 2004. KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 440-449.

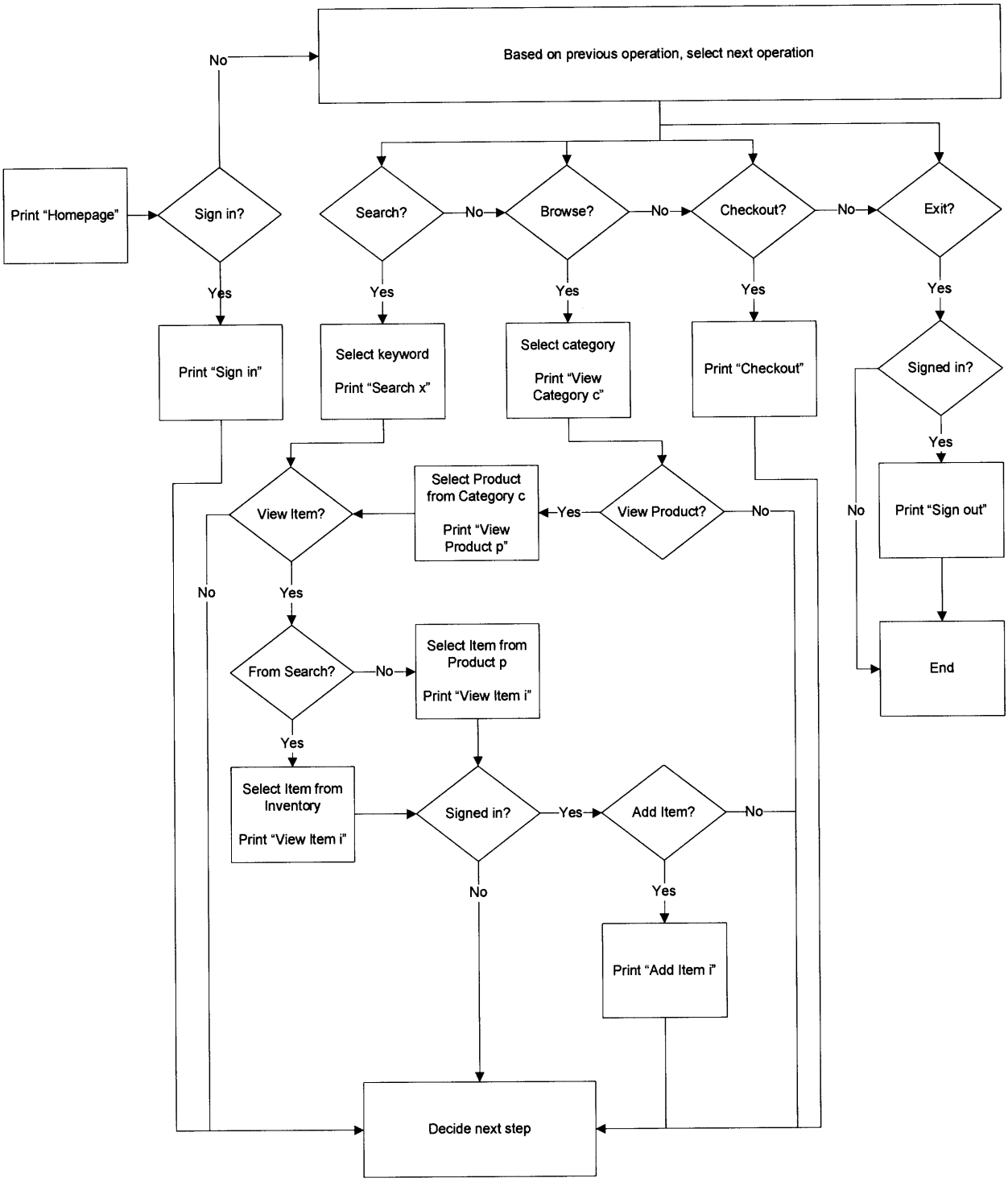
25. *Using runtime paths for macroanalysis.* **Chen, Mike, et al.** Berkeley, CA, USA : USENIX Association, 2003. HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems. pp. 14-14.

26. *Pinpoint: Problem Determination in Large, Dynamic Internet Services.* **Chen, Mike Y, et al.** Washington, DC, USA : IEEE Computer Society, 2002. DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks. pp. 595-604.

27. *Path-based faliure and evolution management.* **Chen, Mike Y, et al.** Berkeley, CA, USA : USENIX Association, 2004. NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation. pp. 23-23.

28. *Magpie: online modelling and performance-aware systems.* **Barham, Paul, et al.** Berkeley, CA, USA : USENIX Association, 2003. HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems. pp. 15-15.

APPENDIX A


Based on previous operation, select next operation
No
Print "Homepage"
Sign in?
Search?
No
Browse?
No
Checkout?
No
Exit?
Yes
Yes
Yes
Yes
Yes
Print "Sign in"
Select keyword
Print "Search x"
Select category
Print "View Category c"
Print "Checkout"
Signed in?
Yes
Select Product from Category c
Print "View Product p"
View Item?
Yes
View Product?
No
No
Print "Sign out"
No
Yes
From Search?
No
Select Item from Product p
Print "View Item i"
End
Yes
Select Item from Inventory
Print "View Item i"
Signed in?
Yes
Add Item?
No
No
Yes
Print "Add Item i"
Decide next step

APPENDIX B

Table 1: All 214 metrics scraped from the JPetStore environment from the performance monitor API of VMware Virtual Center.

ESX_Host cpu usage average rate	ESX_Host cpu usagemhz average rate
ESX_Host mem usage average absolute	ESX_Host mem granted average absolute
ESX_Host mem active average absolute	ESX_Host mem shared average absolute
ESX_Host mem zero average absolute	ESX_Host mem unreserved average absolute
ESX_Host mem swapused average absolute	ESX_Host mem sharedcommon average absolute
ESX_Host mem heap average absolute	ESX_Host mem heapfree average absolute
ESX_Host mem state latest absolute	ESX_Host mem swapin average absolute
ESX_Host mem swapout average absolute	ESX_Host mem vmmemctl average absolute
ESX_Host mem overhead average absolute	ESX_Host disk usage average rate
ESX_Host disk numberRead summation delta	ESX_Host disk numberWrite summation delta
ESX_Host disk read average rate	ESX_Host disk write average rate
ESX_Host net usage average rate	ESX_Host net packetsRx summation delta
ESX_Host net packetsTx summation delta	ESX_Host net received average rate
ESX_Host net transmitted average rate	ESX_Host sys uptime latest absolute
ESX_Host cpu reservedCapacity average absolute	ESX_Host cpu used summation delta
ESX_Host cpu idle summation delta	ESX_Host disk commands summation delta
ESX_Host disk commandsAborted summation delta	ESX_Host disk busResets summation delta
ESX_Host sys resourceCpuUsage average rate	ESX_Host rescpu actav1 latest absolute
ESX_Host rescpu actpk1 latest absolute	ESX_Host rescpu runav1 latest absolute
ESX_Host rescpu actav5 latest absolute	ESX_Host mem reservedCapacity average absolute
ESX_Host rescpu actpk5 latest absolute	ESX_Host rescpu runav5 latest absolute
ESX_Host rescpu actav15 latest absolute	ESX_Host rescpu actpk15 latest absolute
ESX_Host rescpu runav15 latest absolute	ESX_Host rescpu runpk1 latest absolute
ESX_Host rescpu maxLimited1 latest absolute	ESX_Host rescpu runpk5 latest absolute
ESX_Host rescpu maxLimited5 latest absolute	ESX_Host rescpu runpk15 latest absolute
ESX_Host rescpu maxLimited15 latest absolute	ESX_Host rescpu sampleCount latest absolute
ESX_Host rescpu samplePeriod latest absolute	ESX_Host mem consumed average absolute
ESX_Host mem sysUsage average absolute	Tomcat cpu usage average rate
Tomcat cpu usagemhz average rate	Tomcat cpu system summation delta
Tomcat cpu wait summation delta	Tomcat cpu ready summation delta
Tomcat cpu extra summation delta	Tomcat cpu guaranteed latest absolute
Tomcat mem usage average absolute	Tomcat mem granted average absolute
Tomcat mem active average absolute	Tomcat mem shared average absolute
Tomcat mem zero average absolute	Tomcat mem swapped average absolute
Tomcat mem swaptarget average absolute	Tomcat mem swapin average absolute
Tomcat mem swapout average absolute	Tomcat mem vmmemctl average absolute
Tomcat mem vmmemctltarget average absolute	Tomcat mem overhead average absolute

Tomcat disk usage average rate	Tomcat disk numberRead summation delta
Tomcat disk numberWrite summation delta	Tomcat disk read average rate
Tomcat disk write average rate	Tomcat net usage average rate
Tomcat net packetsRx summation delta	Tomcat net packetsTx summation delta
Tomcat net received average rate	Tomcat net transmitted average rate
Tomcat sys uptime latest absolute	Tomcat sys heartbeat summation delta
Tomcat cpu used summation delta	Tomcat disk commands summation delta
Tomcat disk commandsAborted summation delta	Tomcat disk busResets summation delta
Tomcat rescpu actav1 latest absolute	Tomcat rescpu actpk1 latest absolute
Tomcat rescpu runav1 latest absolute	Tomcat rescpu actav5 latest absolute
Tomcat rescpu actpk5 latest absolute	Tomcat rescpu runav5 latest absolute
Tomcat rescpu actav15 latest absolute	Tomcat rescpu actpk15 latest absolute
Tomcat rescpu runav15 latest absolute	Tomcat rescpu runpk1 latest absolute
Tomcat rescpu maxLimited1 latest absolute	Tomcat rescpu runpk5 latest absolute
Tomcat rescpu maxLimited5 latest absolute	Tomcat rescpu runpk15 latest absolute
Tomcat rescpu maxLimited15 latest absolute	Tomcat rescpu sampleCount latest absolute
Tomcat rescpu samplePeriod latest absolute	Tomcat mem consumed average absolute
Apache Ubuntu cpu usage average rate	Apache Ubuntu cpu usagemhz average rate
Apache Ubuntu cpu system summation delta	Apache Ubuntu cpu wait summation delta
Apache Ubuntu cpu ready summation delta	Apache Ubuntu cpu extra summation delta
Apache Ubuntu cpu guaranteed latest absolute	Apache Ubuntu mem usage average absolute
Apache Ubuntu mem granted average absolute	Apache Ubuntu mem active average absolute
Apache Ubuntu mem shared average absolute	Apache Ubuntu mem zero average absolute
Apache Ubuntu mem swapped average absolute	Apache Ubuntu mem swaptarget average absolute
Apache Ubuntu mem swapin average absolute	Apache Ubuntu mem swapout average absolute
Apache Ubuntu mem vmmemctl average absolute	Apache Ubuntu mem vmmemctltarget average absolute
Apache Ubuntu mem overhead average absolute	Apache Ubuntu disk usage average rate
Apache Ubuntu disk numberRead summation delta	Apache Ubuntu disk numberWrite summation delta
Apache Ubuntu disk read average rate	Apache Ubuntu disk write average rate
Apache Ubuntu net usage average rate	Apache Ubuntu net packetsRx summation delta
Apache Ubuntu net packetsTx summation delta	Apache Ubuntu net received average rate
Apache Ubuntu net transmitted average rate	Apache Ubuntu sys uptime latest absolute
Apache Ubuntu sys heartbeat summation delta	Apache Ubuntu cpu used summation delta
Apache Ubuntu disk commands summation delta	Apache Ubuntu disk commandsAborted summation delta
Apache Ubuntu disk busResets summation delta	Apache Ubuntu rescpu actav1 latest absolute
Apache Ubuntu rescpu actpk1 latest absolute	Apache Ubuntu rescpu runav1 latest absolute
Apache Ubuntu rescpu actav5 latest absolute	Apache Ubuntu rescpu actpk5 latest absolute
Apache Ubuntu rescpu runav5 latest absolute	Apache Ubuntu rescpu actav15 latest absolute
Apache Ubuntu rescpu actpk15 latest absolute	Apache Ubuntu rescpu runav15 latest absolute
Apache Ubuntu rescpu runpk1 latest absolute	Apache Ubuntu rescpu maxLimited1 latest

	absolute
Apache Ubuntu rescpu runpk5 latest absolute	Apache Ubuntu rescpu maxLimited5 latest absolute
Apache Ubuntu rescpu runpk15 latest absolute	Apache Ubuntu rescpu maxLimited15 latest absolute
Apache Ubuntu rescpu sampleCount latest absolute	Apache Ubuntu rescpu samplePeriod latest absolute
Apache Ubuntu mem consumed average absolute	MySQL cpu usage average rate
MySQL cpu usagemhz average rate	MySQL cpu system summation delta
MySQL cpu wait summation delta	MySQL cpu ready summation delta
MySQL cpu extra summation delta	MySQL cpu guaranteed latest absolute
MySQL mem usage average absolute	MySQL mem granted average absolute
MySQL mem active average absolute	MySQL mem shared average absolute
MySQL mem zero average absolute	MySQL mem swapped average absolute
MySQL mem swaptarget average absolute	MySQL mem swapin average absolute
MySQL mem swapout average absolute	MySQL mem vmmemctl average absolute
MySQL mem vmmemctltarget average absolute	MySQL mem overhead average absolute
MySQL disk usage average rate	MySQL disk numberRead summation delta
MySQL disk numberWrite summation delta	MySQL disk read average rate
MySQL disk write average rate	MySQL net usage average rate
MySQL net packetsRx summation delta	MySQL net packetsTx summation delta
MySQL net received average rate	MySQL net transmitted average rate
MySQL sys uptime latest absolute	MySQL sys heartbeat summation delta
MySQL cpu used summation delta	MySQL disk commands summation delta
MySQL disk commandsAborted summation delta	MySQL disk busResets summation delta
MySQL rescpu actav1 latest absolute	MySQL rescpu actpk1 latest absolute
MySQL rescpu runav1 latest absolute	MySQL rescpu actav5 latest absolute
MySQL rescpu actpk5 latest absolute	MySQL rescpu runav5 latest absolute
MySQL rescpu actav15 latest absolute	MySQL rescpu actpk15 latest absolute
MySQL rescpu runav15 latest absolute	MySQL rescpu runpk1 latest absolute
MySQL rescpu maxLimited1 latest absolute	MySQL rescpu runpk5 latest absolute
MySQL rescpu maxLimited5 latest absolute	MySQL rescpu runpk15 latest absolute
MySQL rescpu maxLimited15 latest absolute	MySQL rescpu sampleCount latest absolute
MySQL rescpu samplePeriod latest absolute	MySQL mem consumed average absolute

APPENDIX C

J48

binarySplits – Whether to use binary splits on nominal attributes when building the trees.

confidenceFactor – The confidence factor used for pruning (smaller values incur more pruning).

debug – If set to true, classifier may output additional info to the console.

minNumObj – The minimum number of instances per leaf.

numFolds – Determines the amount of data used for reduced-error pruning. One fold is used for rruning, the rest for growing the tree.

reducedErrorPruning – Whether reduced-error pruning is used instead of C.4.5 pruning.

saveInstanceData – Whether to save the training data for visualization.

seed – The seed used for randomizing the data when reduced-error pruning is used.

subtreeRaising – Whether to consider the subtree raising operation when pruning.

unpruned – Whether pruning is performed.

useLaplace – Whether counts at leaves are smoothed based on Laplace.

PARAMETERS

binarySplits – *False*

confidenceFactor – 0.25

debug – *False*

minNumObj – 2

numFolds – 3

reducedErrorPruning – *False*

saveInstanceData – *False*

seed – 1

subtreeRaising – *True*

unpruned – *False*

useLaplace – *False*

NAÏVE BAYES

debug – If set to true, classifier may output additional info to the console.

useKernelEstimator – Use a kernel estimator for numeric attributes rather than a normal distribution.

useSupervisedDiscretization – Use supervised discretization to convert numeric attributes to nominal ones.

PARAMETERS

debug – *False*

useKernelEstimator – *False*

useSupervisedDiscretization – *False*

TREE AUGMENTED NAÏVE BAYES (TAN)

markovBlanketClassifier – When set to true (default is false), after a network structure is learned a Markov Blanket correction is applied to the network structure. This ensures that all nodes in the network are part of the Markov blanket of the classifier node.

scoreType – The score type determines the measure used to judge the quality of a network structure. It can be one of Bayes, BDeu, Minimum Description Length (MDL), Akaike Information Criterion (AIC), and Entropy.

PARAMETERS

markovBlanketClassifier – *False*

scoreType – *BAYES*

NAÏVE BAYES TREE (NBTREE)

debug – If set to true, classifier may output additional info to the console.

PARAMETERS

debug – *False*

TREE AUGMENTED NAÏVE BAYES TREE (TANTREE)

debug – If set to true, classifier may output additional info to the console.

PARAMETERS

debug – *False*

ADAPTIVE BOOSTING (ADABOOST)

classifier – The base classifier to be used.

debug – If set to true, classifier may output additional info to the console.

numIterations – The number of iterations to be performed.

seed – The random number seed to be used.

useResampling – Whether resampling is used instead of reweighting.

weightThreshold – Weight threshold for weight pruning.

PARAMETERS

classifier – *J48 – Decision Stump*

debug – *False*

numIterations – 10

seed – 1

useResampling – *False*

weightThreshold – 100

INFORMATION GAIN (IG)

binarizeNumericAttributes – Just binarize numeric attributes instead of properly discretizing them.

missingMerge – Distribute counts for missing values. Counts are distributed across other values in proportion to their frequency. Otherwise, missing is treated as a separate value.

PARAMETERS

binarizeNumericAttributes – *False*

missingMerge – *True*

GAIN RATIO (GR)

missingMerge – Distribute counts for missing values. Counts are distributed across other values in proportion to their frequency. Otherwise, missing is treated as a separate value.

PARAMETERS

missingMerge – *True*

CORRELATION-BASED FEATURE SELECTION (CFS)

locallyPredictive -- Identify locally predictive attributes. Iteratively adds attributes with the highest correlation with the class as long as there is not already an attribute in the subset that has a higher correlation with the attribute in question

missingSeperate – Treat missing as a separate value. Otherwise, counts for missing values are distributed across other values in proportion to their frequency.

PARAMETERS

locallyPredictive – *True*

missingSeperate – *False*