University of Alberta

Neuro-fuzzy Qualitative and Quantitative Classification for Real-time Webserver Workloads via DTrace Instrumentation

by ©

Jarret Dyrbye

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2008

Your file Votre référence
ISBN: 978-0-494-45804-4
Our file Notre référence
ISBN: 978-0-494-45804-4

University of Alberta

Library Release Form

Name of Author: Jarret Dyrbye

Title of Thesis: Neuro-fuzzy Qualitative and Quantitative Classification for Real-time Webserver Workloads via DTrace Instrumentation

Degree: Master of Science

Year this Degree Granted: 2008

[illegible]

Signature

Abstract

New computerized systems are constantly being deployed to solve many of our problems at the expense of increased complexity for the IT professionals who manage them. The field of Autonomic Computing has been defined to address this complexity by encouraging engineering of self-managing systems. This thesis contributes a prototype system for making qualitative and quantitative classifications of a webserver's real-time workload. The system includes an extensible sentinel autonomic element and an intelligence layer in the form of an aggregator autonomic element. The sentinel element has been architected to instrument system readings through the use of Solaris DTrace and preprocess into a real-time datastream. The aggregator element uses neuro-fuzzy networks to perform the real-time classification. The investigative procedure for the construction of the prototype system, including the adaptation of methods for this particular task and training procedure with validation, results from the development of the aggregator models is presented.

Contents

List of Figures

1 Introduction

1.1 Complexity of Software Systems

In our modern society, new technologies are being applied to help us with our problems at an unprecedented rate. Tasks which once took large amounts of effort and time and were prone to errors can now be done quickly and efficiently by computerized systems. An unforeseen consequence of this pattern is the increased computerization of society leading to new problems of complexity. Issues such as reliability, maintenance, security, configuration and reconfiguration are a massive cost and an ongoing problem for engineers and technicians of these complex computerized systems. The next frontier of progress is to discover ways to reduce the cost and impact of dealing with these issues. One such example is the definition of the term 'Autonomic Computing.'

On this new frontier, a personal opinion of the author that momentum gathering has progressed very slowly. Problems are simple to identify, but massively difficulty to address. The necessary tasks are complex problems which are currently handled by highly trained knowledge workers. These are processes of higher intelligence which have historically known to be difficult to translate into computer actions. Furthermore, nearly all software and system design up until this point has been done with the assumption that a human will be present to troubleshoot and tweak.

The instrumentation facilities used as a basis for troubleshooting are built to be human-readable and rely on human intuition and observational skills for diagnosis. This is an unfortunate reality which has made the systems of today hostile towards such computerized examination. Until this barrier can be overcome, progress in the field will continue to move slowly.

1.2 Motivation

At the time of the beginning of this research, a new operating system monitoring tool, DTrace, became available for public use. DTrace marks a turning point where operating systems have become sufficiently sophisticated that to increase the attractiveness of a platform, focus has shifted to building in qualities of openness and observability. Instead of having the system architects as gatekeepers to the monitoring information available (naturally catering to a human administrator), nearly all conceivable information is available for the administrator (human or otherwise) to obtain.

This was viewed as an opportunity for a new approach at getting at the inner-workings of an operating system from the perspective of a software agent. DTrace was seen as having massive amount of potential for providing a strong machine-readable basis for making judgments analogous to higher-intelligence that human administrators posses. This strategy is realized in this research in the form of a framework which aggregates the DTrace data in a useful manner and the application of a machine learning technique to extract and apply knowledge. The system constructed serves as a prototype for an agent capable of making judgments about its own state in a self-aware fashion, which is a basis for the desired self-managing systems of the future.

1.3 Goal and Approach

From the outlook at the beginning of this research, there was the hope that the new advent of DTrace could have something significant to offer the domain of Autonomic Computing. The goal of this work was to find an application and to demonstrate its potential. Both these elements, DTrace and Autonomic Computing, are extremely young and have not had the chance be studied in combination by the academic community . As a result, this work has the chance to pioneer a venture into new technologies.

Two major components are needed to realize the intelligence layer. First, a machine learning technique to convert data into applicable knowledge, neuro-fuzzy networks in this case. Second, the component which can aggregate and process DTrace data into a form that can be interpreted by the machine learning technique which, for this work, is the Sysmonkey Open-Architected Sentinel Element (SOASE).

The approach of this research is first to construct the base component, SOASE, which will provide a stream of data which can be obtained from a running computer system. Second, the implementation of the neural-fuzzy networks to extract knowledge from the data is constructed. To address the goal of creating a system with autonomic elements, these parts need to be assembled and shown to be workable in the context of a realistic computing system. The context used for this work is the SPECweb2005 benchmark which is representative of the computerized task of serving web pages. By designing a set of experiments using these parameters to demonstrate a system which can provide useful knowledge to a decision making agent would be a fulfillment of the goal.

1.4 Contribution

To the knowledges of the author, this research attempts a problem which has not been examined before. Therefore, the first contribution of this work is the experimental setup which has been developed. The combination of elements, namely DTrace, Autonomic Computing and the SPECweb2005 benchmarks have a defined a problem domain and a set of criteria for being successful in it. The engineering that when into the open architecture of SOASE was carefully chosen considering the possibility for it to be extended and adapted as part of future work, in light of knowledge of this initial attempt.

The second and third contribution of this work is the construction and validation of two autonomic elements. The first being SOASE, the sentinel autonomic element, which monitors the system. The second being an aggregator element which takes the monitoring data and extracts meaning from it. The work that went into these two elements required specific consideration of and adaptation to fit a real-time environment. The working prototype system of these two elements working in conjunction can be presented as a contribution for consideration in future engineering of autonomic systems.

1.5 Thesis Overview

The body of this document begins in Section 2 by providing some necessary background knowledge for the starting elements of this work. First, the context of Autonomic Computing is set for this work. Second, the utilized elements of DTrace, Neuro-Fuzzy Modeling and the SPECweb2005 benchmark are discussed.

Section 3 discusses past previous work done in a similar area to what is investigated in this thesis. The nature of this present thesis work is not an evolutionary progression of existing work, but rather a venture into a new problem. The effect is that there is relatively little pre-existing work that can be directly compared. What is overviewed in the section is other pre-existing work that this research shares overlapping concerns and inspiration with.

Section 4 defines the scope of this particular research and the procedure that it follows.

Section 5 discusses the work that was done to construct the apparatus necessary for facilitating the investigations presented later on. This includes the hardware and pre-packaged software setup for the test system, the monitoring framework that was constructed to instrument the system (SOASE) and the adaptation of a neuro-fuzzy learning technique.

Section 6 discusses the approach of three investigations into the construction of three components of the aggregator element. This section sets down the purpose and design of the procedure for the training of these components which proceeds in the next section.

Section 7 presents the development and validation of the three aggregator element components as was planned in the previous section. It presents aggregated and visualized data which pertains to the aptitude of the system that has been constructed for this research.

Finally, Section 8 wraps up this research by recapping the results and discussing the perceived level of success and knowledge gained. Also, shortcomings of this work and potential future endeavors into related work are discussed.

2 Background

2.1 Autonomic Computing

The manpower cost of maintaining and setting up IT infrastructure in businesses continues to grow as computer systems become increasingly complex and networks increase in size. In order to help deal with these rising costs, IBM introduced the term “Autonomic Computing” in 2001 [34]. The word ‘autonomic’ is borrowed from biology to describe the automated regulation and adjustment of processes within organisms, such as heart rates speeding up during increased physical activity and the immune system which organizes to fight harmful elements. The vision is to design similar behavior into the computerized systems of the future.

For the inventions that are being made in the present and near future, the engineering mindset needs to be shifted from the expectation of constant human supervision towards a paradigm of designing self-managing systems.

Autonomic Computing has four areas of focus [14]:

- Self-configuration - The ability of new components to integrate themselves into existing infrastructure guided by the surrounding system which is detected.
- Self-healing - The diagnosis of failures and/or deficiencies of internal components and automated repair or compensation when possible.

- Self-optimization - The diagnosis of performance or procedural bottlenecks and the automated adjustment of tunable parameters to increase efficiency.
- Self-protection - The detection of intrusions or vulnerabilities that threaten a systems security and the ability to take steps to avoid or mitigate them.

Despite this being a new field of research in the official sense, some elements of Autonomic Computing exist in implementation already. Out of necessity, Internet routing protocols developed a self-healing element to deal with the unreliability of individual nodes. This is a well-cited example of a self-healing system. In the personal computing domain, there has been an unpleasant history of manual hardware configuration to get peripherals to work. In more recent times, this difficulty has been greatly reduced by hot-pluggable USB devices which provide their own drivers and interfaces which the user does not need to be concerned with. This is an excellent example of the benefits of a self-configuring piece of hardware. Autonomic Computing wishes to borrow the essence of some of these specific technologies and apply it to more mainstream technological problems. This is no easy task. In the two mentioned examples, the ease of use that we enjoy today was the result of an evolutionary process that arose over several revisions of similar technology. Earlier schemes had flaws that stood in the way of the advancement and usefulness of the technology that were fixed over time. A hope of Autonomic Computing is to recognize the trend and respond pro-actively with our approach to engineering. Ideally, from the end-user perspective, an autonomic system should implement "computing that just works" [2].

Now that the concept has been formalized, focus is on replicating the processes that knowledge workers, notably system administrators, apply to manage these systems. Legacy systems are examined and re-imagined with these aspects applied and architectural frameworks are developed to reduce need for human intervention [13] [31]. The economic incentive is that companies will want to e-source costly human effort from IT departments into autonomic actions done by the systems themselves. In one example, Apache, the dominant web server application, can be abstracted to not focus on tunable parameters, but tuned to the desired amount of system resources to consume [5]. This is useful by reducing the human intervention needed to allow the webserver to maintain reasonable performance. The procedural knowledge the administrator exhibits can be recorded by observation. Techniques of knowledge representation from the field of Machine Learning can then be used to understand and perhaps synthesize human configuration and management actions [16].

2.2 DTrace

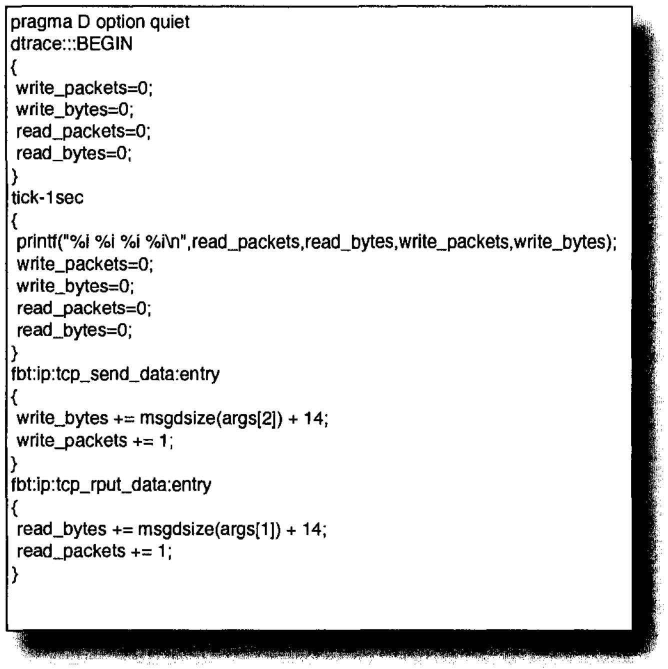
In January of 2005, Sun Microsystems released the latest version of its operating system, Solaris 10. This release included the Dynamic Tracing Framework, commonly referred to as DTrace [35]. This tool is highly regarded in the industry, as it is unprecedented and unique in the benefits it can provide. It won the Wall Street Journal's 2006 Technology Innovation Award [37] in the software category and John Siracusa of Ars Technica referred to it as "indistinguishable from magic" in his in-depth review of its port to Mac OS X 10.5 Leopard [28].

Using DTrace, an administrator can step in and view the internals of a running kernel in real-time. By flagging probes of interest, the administrator can collect statistics and develop views of how the operating system is interacting internally, with the hardware and with the running application.

The use of DTrace is analogous of how a programmer would use a debugger to insert breakpoints into his/her code to glimpse inside the program as the execution steps over those instructions. This is luxury that requires a special compilation of the code and to run it under only specially controlled test conditions. DTrace's technology allows this kind of observation to be made on a normally compiled program running in a production environment, notably the Solaris kernel. DTrace is designed in such a way that it has zero impact to system stability and near-zero impact on system performance. This is revolutionary because it allows observations to be made in real-world environment where kernel, hardware and applications interact. In the past, it was prohibitively awkward to get at such information where a special test-bed setup, possibly requiring a recompilation of the code and special apparatus was needed. DTrace allows the administrator to bypass the difficult steps and conduct an investigation on a live system in a real-world deployment [4].

The basic principal that DTrace works upon is the concept of a 'probe.' A probe is a location in the kernel code's execution which can be flagged to be active. As the code executes and steps over a flagged probe, DTrace is notified and is given the opportunity to access some direct info about the system state from the specific context of the exact moment the probe is stepped over. Along with the ability to record values when the probes fire, DTrace has the ability to aggregate and track statistics which can be derived from the firing of probes. This is done through the scripting language named 'D'. By crafting a short script, the administrator can make a report on any possible kernel function that he or she wishes instead of being limited to the baked-in monitoring tools which came with the operating system distribution. This flexibility allows a troubleshooter to drill down to exceptional granularity in a particular problem area.

Figure 1 is an example of a D script. The function of this script is to print the amount of bytes and packets sent and received every second. This is accomplished by invoking probes in the internals of Solaris 10's networking stack in a function involved in the handling of TCP connections. When the kernel function is called as the result of an application or otherwise, the probe will fire and pieces of the script will be executed.



```
pragma D option quiet
dtrace:::BEGIN
{
 write_packets=0;
 write_bytes=0;
 read_packets=0;
 read_bytes=0;
}
tick-1sec
{
 printf("%i %i %i %i\n",read_packets,read_bytes,write_packets,write_bytes);
 write_packets=0;
 write_bytes=0;
 read_packets=0;
 read_bytes=0;
}
fbt:ip:tcp_send_data:entry
{
 write_bytes += msgdsize(args[2]) + 14;
 write_packets += 1;
}
fbt:ip:tcp_rput_data:entry
{
 read_bytes += msgdsize(args[1]) + 14;
 read_packets += 1;
}
```

Figure 1: DTrace Example Script

In the example, when the network stack is called upon to send a piece of data to another machine over the network, the *tcp_send_data:entry* probe will fire. This probe corresponds to the *tcp_send_data* function of the source code file tcp.c which is a component of the Solaris 10 source tree[1]. The third argument of the *tcp_send_data* function as it was called at the time of the specific firing of the probe can be accessed by the D script by referencing *args[2]*. The argument is a *struct* type which contains the data to be passed by the packet. The *msgdsize* method is provided by the D

[1]Source code available under the CDDL and can be downloaded or viewed at `http://www.opensolaris.org`

language to count the size of the message in bytes. This script records the amount of bytes contained in the packet (adding 14 to account for the packet header) and also increments a counter to represent the number of packets that have been sent. The same calculation is done for incoming packets received with the *tcp_rput_data:entry probe.* The *tick-1sec* probe is provided by DTrace which will fire every second, on the second as determined by the system clock. This script simply displays the values counted and resets the counter. When run, this script will print the amount of bytes and packets sent and received every second.

This is a simple example to illustrate some basic functionality. There are many thousands of probes defined in Solaris. With proper knowledge of the internals of the kernel, much more elaborate and specialized scripts could be constructed to address a particular need. For the purposes of the implementations described in this research, this is an adequate level of understanding. The scripts that have been written are are a similar level of complexity. However, it should be noted that DTrace has many more features and capabilities beyond what is discussed here [35]. An excellent example of the full abilities available through DTrace is Brendan Gregg's extremely robust DTrace Toolkit [9].

2.3 Fuzzy Modeling

2.3.1 Fuzzy Neural Networks

The preferred method of machine learning used in this research is that of Fuzzy Neural Networks (FNN), also referred to as Neuro-Fuzzy Networks. This is a special type of Artificial Neural Network that incorporates the element of fuzzy logic into its structure [21] [22] [19] [24].

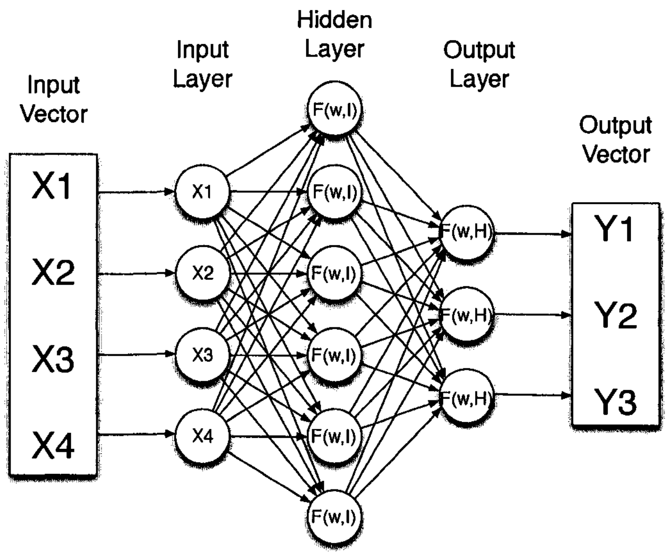


Figure 2: Basic Artificial Neural Network

A basic Artificial Neural Network (ANN), as illustrated in Figure 2, consists of a layer of input nodes, potentially multiple layers of hidden nodes and a layer of output nodes. Each layer of nodes will take inputs from the previous layer, combine them with weights of the synapses and calculate outputs via the node's transfer function.

Training the network is the process of adjusting the weights of the network to match the correct set of outputs to the input fed into the network. The method for doing this is referred to as backpropagation training. Weights are adjusted to minimize the error in the output proportional to how much the weight contributed to the error. Over many iterations of this process, the knowledge of patterns in the data can be trained into the structure of the network.

A FNN is a special case of a ANN. The nodes are of a special type referred to as fuzzy neurons [11]. All inputs, weights and outputs are fuzzy numbers and the transfer functions are T-Norms and S-Norms. A FNN is also limited to a single hidden layer between the input and the output. The motive for this design is that the network gains the ability to break down to a linguistic representation in the form of a set of fuzzy if-then rules. The transfer function at a hidden layer node is a S-norm which is a representation of the linguistic *OR* and the transfer function at a output layer node is a T-Norm and is a representation of the linguistic *AND*. This is illustrated in Figure 3.

Examining a fully trained normal artificial neural network can be daunting since the transfer functions can be difficult to impossible to articulate linguistically and the weights of the network take advantage of the entire range of real numbers. With a potential drawback of losing prediction accuracy, the network can become more interpretable by restricting it to linguistically representable elements [22]. FNNs do

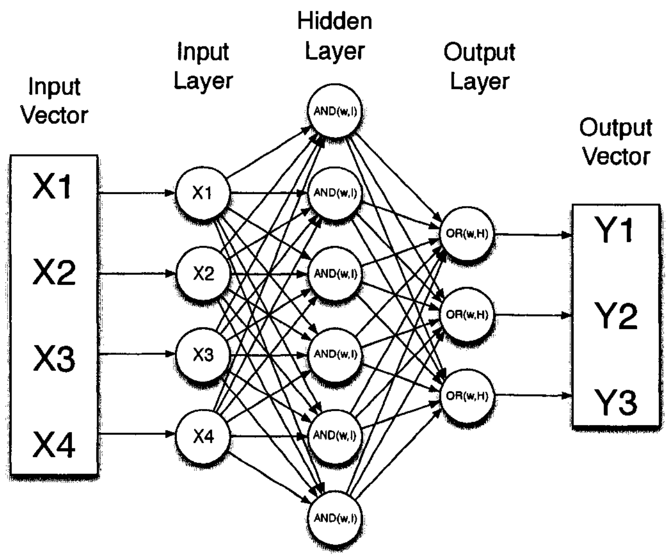


Figure 3: Fuzzy Neural Network with AND and OR transfer functions

this by using AND and OR operations which have both mathematical and linguistic meanings associated. Also, fuzzy set membership values have linguistic labels such as 'large,' 'medium,' and 'small' which linguistically represent indications of quantity in a similar dualistic fashion [7].

2.3.2 FNN Training

2.3.2.1 Overview The learning technique utilized in this work has a unique method of training FNNs. It was adapted from the evolutionary fuzzy modeling method developed by Pedrycz and Reformat [22]. This method defines three phases to the training process. First, a phase of generating a collection of discrete fuzzy sets from crisp values to serve as baseline building blocks for the later phases to utilize. Second, an evolutionary phase of genetic programming (GP) is used to develop the the structure of the network. The third and final phase is to apply parametric refinement to better train the best result from the GP phase.

The second and third phases are what is relevant to this work. For simplicity of discussion, all reference to these phases will be renumbered the first and second phase, respectively. The ignored first phase is defined for completeness in this background, but is not necessary for later explanation of concepts. The operation which it performs is done manually and described in detail on its own in Section 5.4.3. The GP and Backpropagation Training phases are described in some detail here.

2.3.2.2 Phase 1 - Genetic Programming The goal of this phase of the model development is to find the structure of a FNN which is the best fit as a model. The individual is a tree of operations and terminal nodes which represent a FNN. It is limited in size and depth to include the *AND* layer, the *OR* layer and the input layer. This phase is not concerned with the weights of the connections between the nodes on the FNN, only the parameter controlling the connection's existence or non-existence is modified. The crossovers and mutations occur on this tree to re-arrange the structure of the network. The fitness of a particular individual is determined by how well the output matches the desired output of the training data set. Another property of this GP scheme is that inputs are pruned down to a minimal set necessary. The connections that exist and are going into an *OR* node are regarded as having a weight of 1.0 for the evaluation and connections that do not exist are regarded as having a weight of 0.0 ($A \vee 0.0 = A$). This is reversed for the connections going into an *AND* node to be consistent with the exist/does not exist metaphor ($A \wedge 1.0 = A$). All inputs are not necessary to be taken advantage of in the prediction and the solutions that emerge only take into account the subset of inputs that prove useful [27]. This GP process is run for as many generations as is judged to be necessary to produce a suitable individual to advance and receive training in the next phase.

2.3.2.3 Phase 2 - Backpropagation Training Once an individual is selected from the GP phase, backpropagation training is applied to make the individual better fit the training dataset. Starting out, the weights of the network are set to either 1.0 or 0.0 depending on whether the connection exists or not. A better solution for the dataset most likely can be found by adjusting these extreme weights (0.0 or 1.0) to something intermediate in that domain. Adjustments are made according to the rules of backpropagation training. Each data point is run through the network for each iteration. For each point, the obtained output for the model is compared to the correct output at each of the output nodes. Then, traveling backwards through the network, the weights are adjusted proportional to first, the error of the result and second, how much the particular weight contributed to the result out of many in the same layer. The property of this training is that when many points are applied over many iterations, the network will converge to a equilibrium that reduces the total error in prediction across the training data set.

2.4 SPECweb2005 Benchmark

2.4.1 Overview

SPECweb2005 is the industry standard for evaluation of the performance of web servers. It was developed by the non-profit Standard Performance Evaluation Corporation among a suite of other benchmarks that measure system performance under a variety of criteria. Hardware manufactures can use this particular benchmark to compare and quantify the benefits of their offerings with those of other vendors based on how many simultaneous clients the system can support in a realistic web-serving environment. The benchmark is also, to a lesser extent, useful to operating system and web server software vendors who are enabled to compare how their products perform on a baseline set of hardware.

The scoring for this benchmark is handled analogous, to some extent, with competitive weightlifting. The benchmark will not automatically find the maximum effective level of workload that can be handled as would be generally expected of a benchmark. Rather, prior to running the benchmark, the benchmark administrator must challenge the system to handle a benchmark workload. The SPECweb2005 will run the benchmark at that setting and report back to the administrator whether or not the system met the standard of performance to pass the test. The benchmark administrator must keep raising the challenge until the upper limit of where the system can pass the test is reached.

This scheme is necessary for reasonable technical limitations. More commonly, benchmarks set up a certain amount of work for the system to perform and the score is based upon the time that it takes to complete the task or some other similar variation of the concept. Because of the client-server nature of the system where there is no simple contained workload, this particular benchmark model cannot be applied. Another imaginable benchmark scheme where the client would ramp up to a level where the performance can be maintained is also not feasible. This is because server performance in this application relies heavily on caching of often accessed content. To perform well, the server's caches needs time to saturate to an equilibrium to best meet the load. Therefore, the load must be stable for a period of time in order to get the maximum performance value. The scheme implemented for this benchmark is to accommodate this, but has the tradeoff of being quite time-consuming to run for each trial.

The SPECweb2005 package includes three different workloads to test the behavior of the system under loads of different characteristics. They are referred to as SPECweb_Banking, SPECweb_Ecommerce and SPECweb_Support. Here, they will be referred to more casually as Banking, Ecommerce and Support loads. The overall score of the system for the SPECweb2005 benchmark comes from a combination of the scores on the three workloads. The scores are combined as follows:

$$SPECweb2005_{score} = \sqrt[3]{\left(\frac{Bank_{score}}{Bank_{ref}}\right)\left(\frac{Ecom_{score}}{Ecom_{ref}}\right)\left(\frac{Supp_{score}}{Supp_{ref}}\right)} \cdot 100 \quad (1)$$

$Bank_{score}$, $Ecom_{score}$ and $Supp_{score}$ are the scores for the individual benchmarks. $Bank_{ref}$, $Ecom_{ref}$ and $Supp_{ref}$ are the scores of a reference system the benchmark administrator wishes to compare against. If there is no reference system and the benchmark administrator would just like a standardized raw score, a value of 100 should be used for each of the reference terms.

2.4.2 Workloads

In this section, there are brief descriptions of the three workloads available in the benchmark suite. The full design documentation and breakdown of the specifics are open to the public and available online for more in-depth detail, but they are briefly summarized here to include the relevant knowledge for this research [36]. Included in this discussion is the expected bottleneck that will limit the server's performance. The bottlenecks encountered may, in practice, be different depending on the specific hardware setup, but the description here reflect generalities that apply to most cases including all cases encountered in this research.

2.4.2.1 Banking Load The Banking load uses exclusively SSL connections for its traffic. The workload was modeled off of what was observed in the logs of a real-world Banking application where approximately 95% of the requests were HTTP GET requests and approximately 5% of the requests were HTTP POST. The intended bottlenecked resource is the CPU as it gets bogged down with the SSL encryption of traffic under a heavy load.

2.4.2.2 Ecommerce Load The Ecommerce load is designed to simulate a web application that allows client users to view, configure and purchase computer systems. It is modeled after statistics on average page size, image sizes and user behavior found on real Ecommerce sites. The webserver generates dynamic content and also serves out product information stored on the server's hard drive. The intended bottleneck on the system is the capacity of the system memory. The content being accessed is of great variety and the performance depends on how efficiently the content can be cached in RAM for quick access.

2.4.2.3 Support Load The Support load is modeled after a vendor's support website. The simulated users browse for a particular product and download a related file. This load was also developed from looking at real-world log files of major computer vendor's support sites. The expected stress on the benchmark system is the network and disk throughput as large files are read from the hard drive and sent over the network. These files are much bigger than is the case for the Ecommerce content, so their ability to be cached is limited.

2.4.3 Hardware Requirement

To run the SPECweb2005 benchmark, a minimum of three computer systems on a network are needed. The first is the System Under Test (SUT), which is the machine which is being scored by this benchmark on its ability to serve web clients. The second is the Backend Simulator (BeSim), which simulates a database backend for the SUT. Finally, at least one system must act as the client which generates requests to the server and collects statistics on how well the server responds to them. There are also options for software setup on the systems to provide flexibility. The specifics of how the hardware was set up for this work is detailed in Section 5.2.2.

2.4.4 Software Configurables

There are many configurable options which the benchmark administrator can adjust. In order to have a benchmark run meet the standards to get it recognized by SPEC, there are ranges of valid configurations. The default values are adequate for a valid benchmark run. There are many more configurables than are discussed here. Only the major ones that are relevant to the research work done are listed below:

2.4.4.1 SIMULTANIOUS_SESSIONS

The load that the benchmark is going to test the webserver's ability to handle. If the server passes the test, this value is the score that can be claimed as its score. This has to be manually adjusted to the maximum level that the system can handle as was described in Section 2.4.1. The range of the adjustments will vary greatly depending on amount of resources available in the hardware setup. By design of the benchmark, a 'client session' is roughly equivalent, but does not directly correspond to the activities a real-world human. The unit of a single session represents a unit of client workload which is agreed to be equivalent across the three workloads. Real-world client sessions controlled by a human tend to vary greatly in their activity over time, so a direct correspondence is not reasonable in a simulation. A session is a basic unit which formulates the final score of a system on a benchmark. The neutrality and balance of this unit, which is provided by SPEC, is essential to not bias the final score towards systems that are more adept in one workload over another in the score calculation.

2.4.4.2 THREAD_RAMPUP_SECONDS

This is the amount of time that the benchmark spends ramping up the load before reaching the SIMULTANIOUS_SESSIONS level. The default value is 180 seconds, or 3 minutes.

2.4.4.3 WARMUP_SECONDS

This is the period that the benchmark runs at full client load before the measurements begin. The default value is 300 seconds, or 5 minutes.

2.4.4.4 RUN_SECONDS This is the period over which the benchmark measures the request response times of the server. This period has to be sufficiently long to gather statistically significant values of the servers ability. The default value is 1800 seconds, or 30 minutes.

2.4.4.5 THREAD_RAMPDOWN_SECONDS Once the statistics-gathering period had been run, this is the amount of time that the benchmark spends ramping down to zero. The default value is 180 seconds, or 3 minutes.

2.4.4.6 ITERATIONS This is the amount of cycles of measurement are performed. The cycles proceeds as described below in Section 2.4.5. The default value is three iterations.

2.4.5 Benchmark Procedure

The benchmark proceeds, started by the client machine(s) by simulating many HTTP client sessions connecting to the server which include HTTP GET and HTTP POST requests at realistically randomized intervals modeled in a pattern that a client driven by a real human would likely proceed. As the benchmark starts, the amount of simulated client sessions interacting with the server slowly ramps up to the SIMULTANIOUS_SESSIONS parameter over THREAD_RAMPUP_SECONDS seconds. Before statistics are recorded, a warmup period of WARMUP_SECONDS seconds is run which allows the server's caching to reach a stable level so that bias from preexisting conditions in the server's state can be eliminated. The amount of client sessions then remains at that level for RUN_SECONDS, so that a confident statistical representation of how well the server is responding can be generated. After the period of stat collecting, the amount of requesting sessions ramps down over THREAD_RAMPDOWN_SECONDS seconds. They client side now pauses for 300 seconds (5 minutes) before the cycle repeats ITERATIONS times to make for that many RUN_SECONDS long stat-collecting sessions. This process is illustrated in Figure 4. The results are calculated by the client software and if TIME_TOLERABLE is above 98% and TIME_GOOD is above 95% the benchmark passes for that level of SIMULTANIOUS_SESSIONS. TIME_TOLERABLE is defined as the percentage of requests that were completed within four seconds and TIME_GOOD is defined as the percentage of requests completed within two seconds.

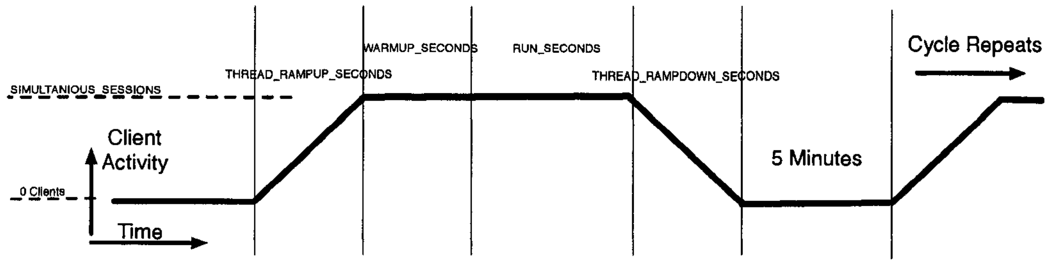


Figure 4: Ramp-up and Ramp-down Characteristic of Benchmark

3 Related Work

3.1 TUNEX by Samadi - 1989

TUNEX was a system developed by Behrokh Samadi and published in 1989 [26]. It was a system developed with the same inspiration as this project. It took data from UNIX tools such as the System Activity Reporter (sar), Activity Disk Profile (sadp) and Accounting Commands (acctcom) and applied heuristics to tune the UNIX system.

These heuristics are organized into three levels. Level 1 is the basic set of if-then heuristics that lay out the condition-response sequence for tuning a system. These are the simple expert system rules that a system administrator could easily apply without intensive analysis. They are directly based off the rules described in the tuning and system administration manuals. For example, "if CPU is highly utilized and wait for I/O is high and the disk buffer cache hit ratios are low, then recommend increasing the number of disk buffers." TUNEX tests the conditions by using a simple crisp-value comparison method which trigger the heuristics to recommend a response. This is similar to the autonomic agent concept which is related to, but external to the present work.

Level 2 is a set of modules that analyze the effects of adjusting certain parameters on the system. For example, a shadow disk buffer would be allocated and the kernel would be modified to keep track of the utilization of the normal buffer and the shadow buffer and then try to quantify the gained benefit of permanently allocating the shadow buffer as an addition to the normal buffer. This is also closely related to the work that would have to be done to develop a functional autonomic agent module that would put the results of the predictive models built here to use.

Level 3 was a planned extension to the existing TUNEX base which was not completed. The intention for the Level 3 heuristics to learn from historical data. As was found in this work, it is a non-trivial problem to tackle. Full exploration of this topic would most likely require a great amount of work dedicated to it on its own.

This work was done before the concept of Autonomic Computing was formalized, but it shares the same mindset of design. From examining the published paper is that the UNIX system circa 1989 being described is much more primitive than the Solaris 10 being used for the present set of work. Modern operating systems have much more sophisticated resource management default algorithms and tuning in the areas being described are no longer necessary [17]. An autonomic system like the one being done in the present work applied to a system like is being described in the 1989 work could likely be much more effective since the problem domain is much more straightforward and the default OS behavior is grossly inefficient by modern standards. While the work is quite analogous, the problems it solves are outdated and largely unnecessary due to modern operating system engineering.

The present work here would possibly be able to interface with a modern implementation of a system akin to TUNEX. This would work by providing premises which the heuristics could use. TUNEX would then be functioning as the policy application entity of the autonomic agent.

3.2 Apache Autotuning Agent by Diao, Hellerstien, Parekh and Bigus - 2003

A related system was constructed by Diao, Hellerstein, Parekh and Bigus which controlled the CPU and and memory utilization of the Apache web server by dynamically tuning application parameters [5]. This work was published in 2003 as a venture into Autonomic Computing. It represents a simple case of the task of an autonomic agent to self regulate based on resource consumption.

Using this system, the system administrator would set the desired CPU and Memory resources for Apache to consume in the agent instead of manually tuning Apache's parameters to hit a target. The agent would exploit the relationships between the tunable parameters and the resource consumption to keep the consumption aligned with the administrator's specification. The agent would accomplish this by periodically re-examining the system state and adjusting the *MaxClients* and *KeepAlive* parameters of Apache to keep the system near the target. Apache is designed to allow dynamic adjustment of these parameter to avoid server downtime which would occur otherwise in an application restart with new parameters. The mechanism for control is that of a state feedback controller. It works by applying heuristics to the observation of system state to control the system.

In this work, construction of the agent was facilitated by ABLE (Agent Building and Learning Environment) [3], a Java-Based framework for constructing and deploying hybrid intelligent agent applications. Time series data was collected which illustrated relationships between the system resource consumption and the KeepAlive and MaxClients parameters. This data was used to develop a linear quadratic regulation controller to function as the repository for the control heuristics of the system.

This is an excellent example of engineering a system to be administrated at a different layer of abstraction. One that doesn't leave the raw parameters exposed for a system administrator to have to ponder and spend time and effort to tweak. It is representative of a design perspective in a relatively simple implementation which is the exact definition of the Self-Configuration and Self-Optimizing autonomic properties as it is interpreted for the present work.

3.3 An Architectural Approach To Autonomic Computing by White, Hanson, Whalley, Chess and Kephart - 2004

This particular work was an attempt to invent design patterns and common understanding which can facilitate the design of autonomic systems [32]. It belongs to an initial set of literature generated by IBM to start the movement for Autonomic Computing. It addresses challenges involved for the construction and interaction between different elements of a self-managing system and lays down a rhetoric which can be used to build future discussion.

The first definition this piece makes is the that of the Autonomic Element. This is a self-contained piece of the infrastructure that has a responsibility in a particular area and exists decoupled from the rest of the autonomic apparatus. These elements must have a defined set of behaviors which govern how they behave and interact with the other elements. Examples are given of several infrastructure elements such as a registry for allowing elements to find each other, a sentinel which provides monitoring services, an aggregator which combines information in a value-added manner, a broker that facilitates interaction between elements and a negotiator which reasons and assists elements with coordinated effort.

The second definition is that of the policies of the autonomic system. Policies describe a desire for a certain behavior of the system. These exist in the form of action polices which are a heuristic for simple if-then actions, goal policies which describe conditions that must be obtained without explicitly describing the ways of obtaining them and utility function policies which are responsible for ranking the value of several goals when needs for trade-offs exist.

This piece also suggests strategies for architectural design to address the four schools of autonomic design. These being self-configuration, self-healing, self-protecting and self-optimization. This is a fairly easy exercise for someone who understands the problem domain, but it is done as an exercise under the newly defined rhetoric. The paper references two real-world prototype systems that were built, one for resource allocation and the other for data center management as examples of the developed design patterns.

3.4 Supporting Autonomic Computing Functionality via Dynamic Operating System Kernel Aspects by Engel and Freisleben - 2005

This paper presents the TOSKANA toolkit (Toolkit for Operating System Kernel Aspects with Nice Applications). This is an attempt to implement dynamic aspect oriented programming into an operating system.

It is a major observation of this paper that aspect-oriented programming is a necessary component of Autonomic Computing. In the case where an autonomic agent wishes to modify a software system such as an OS, an AOP (Aspect-Oriented Programming) model is an excellent abstraction for the agent to interact with. A few examples are provided for aspects which address the self-configuration, self-healing, self-optimizing and self-protection qualities necessary for Autonomic Computing.

Dynamic aspects in this context is the ability for an entity, such as an agent, to apply the aspect without otherwise interrupting the execution of the kernel. The TOSKANA toolkit is capable of splicing instructions into a compiled binary undergoing execution in an almost identical fashion as DTrace [35]. By adding functionality into the code dynamically, the system can be modified to suit the autonomic goals.

This system was implemented using the NetBSD kernel where a method was found to access the kernel memory space to facilitate the splicing in of code. Dynamic aspects are able to be 'woven' into the kernel through the TOSKANA toolkit which inserts the behaviors into the binary code by adding jump instructions to the new code.

The paper was published just after DTrace was released to the public. A section near the end acknowledges DTrace as a system that is interesting in conjunction with this particular work. It is clear that the author did not do a full analysis of the full capabilities of DTrace in the autonomic domain but it was worth mentioning because it uses a similar mechanism for inserting dynamic behavior into real-time running code. A second interesting point, not explicitly mentioned but indirectly implied (likely because it is out of the scope of this particular paper), is that DTrace would be perfectly wedded with TOSKANA in an autonomic setting to serve as the kernel instrumentation by which the agent can base decisions off of in order to invoke these dynamic kernel aspects. Together, it would allow the autonomic agent to make its round-trip from system monitoring to actions re-arranging the system without breaking out of this dynamic execution.

4 Research Scope

4.1 Autonomic Scope

In literature on Autonomic Computing, there is reference to the fundamental algorithm of an autonomic system. That algorithm is thus:

1. Examine system (check log files, etc.)
2. Diagnose problems
3. Fix problem
4. Repeat

First, the system is monitored for data relating to the internal operations. In the initial Autonomic Computing literature generated by IBM [14], this consists of monitoring log files and system statistics provided in the system to develop metrics of activity. The second step is to diagnose any problems or issues occurring in the ongoing deliberation of system functionalities. Once an actionable problem is diagnosed, the problem can be fixed or compensated for by the autonomic agent. The fourth step as shown above is the indication that this is a ongoing iterative process. Since the autonomic agent cannot become tired or neglectful as a human administrator can, the agent can be applying this process at all times.

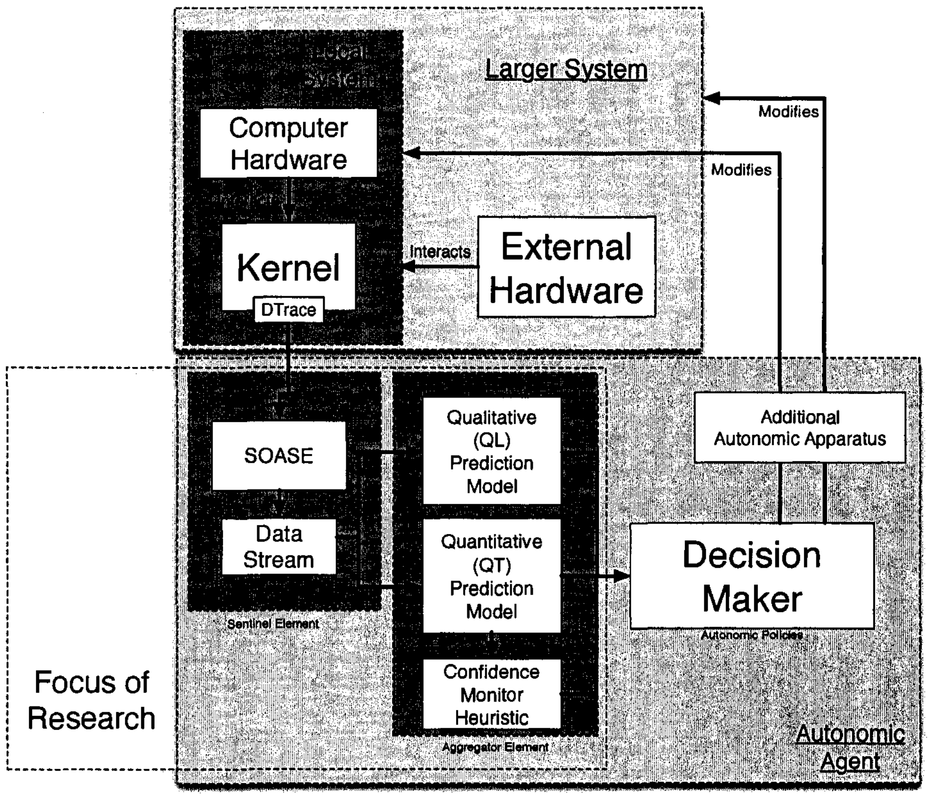


Figure 5: Diagram of Autonomic System

The core of this particular research is to address the first step of the algorithm and to provide information which is a partial component of the second step. The autonomic agent and system as imagined for this particular work are illustrated in Figure 5. The autonomic agent is a layer built on top of the hardware which makes up the computer system. While it is composed of software, it can be envisioned as a separate entity which communicates and interacts with the larger system. In Figure 5 the focus of this research is shown as a part of the autonomic agent. By utilizing a datastream obtained from DTrace, this system will process the data to construct premises which the decision maker of the autonomic agent can uses to make its choices. This component is comprised of two significant subcomponents. The first being SOASE (detailed in Section 5.3) and the second being the predictive models which take data from SOASE and generate the premises for the decision maker to reason upon.

Invoking the terminology of White [32] reviewed in Section 3.3, this work is examining the construction of sentinel autonomic elements to provide system monitoring and also aggregator autonomic elements to combine the collected data into higher-level meaning.

4.2 Road Map

SOASE took a significant amount of design and effort to construct. Its purpose is to extract and process DTrace information from a Solaris 10 system to turn into a datastream for the higher-level functioning of an autonomic agent. The internals and construction of SOASE are detailed in Section 5.3

The next phase in the construction of this system is the lower-level intelligence layer of the system. Specifically, the mechanism for providing premises for the higher-level intelligence of the autonomic agent to reason upon. In this work, it takes the form of two components, the quantitative and the qualitative prediction models. These two components are supplemented by a third component which gives an assessment of the confidence in the answers in the context of the current situation. The qualitative and quantitative models employ trained Fuzzy Neural Networks (detailed in Section 5.4) to process the datastream into classifications of the real-time workload characteristic. The experimental segment of this research focuses on the ability for these components to perform their task with the developed datastream from SOASE.

Figure 6 represents the architecture of the system as implemented at the conclusion of this work. This is a more targeted view of the system in Figure 5 drilled down into the context of the specific work being done. Represented in both figures are the qualitative and quantitative prediction models which are designed to answer two specific questions of the autonomic agent's decision making component and additionally, a third component which addresses the confidence of the qualitative model's prediction.

The analytical segment of this research is focused on the proof-of-concept of these quantitative and qualitative prediction components. The form that this investigation takes is a set of experiments which are outlined in Section 6. They take the form of three investigations. The first is regarding the qualitative prediction model, which is the more complex, is referred to as the QL model and is discussed in Section 6.1. The quantitative prediction model is referred to as the QT model and is discussed in Section 6.2. The component for determining the confidence of the the qualitative model is discussed in Section 6.3 and is referred to as a confidence monitor. The obtained results of the investigation are presented and discussed in Section 7.

This research goes no further than this question-answer interaction within the agent. While it was initially hoped to develop an entire control system much like the Apache Autotune Agent [5], the effort involved proved to be well beyond the scope for this project. It is therefore, left for future work to explore the possibilities of constructing a full autonomic agent to regulate the system. The discussion of this future work and other extensions to the work being done is contained in Section 8.2.

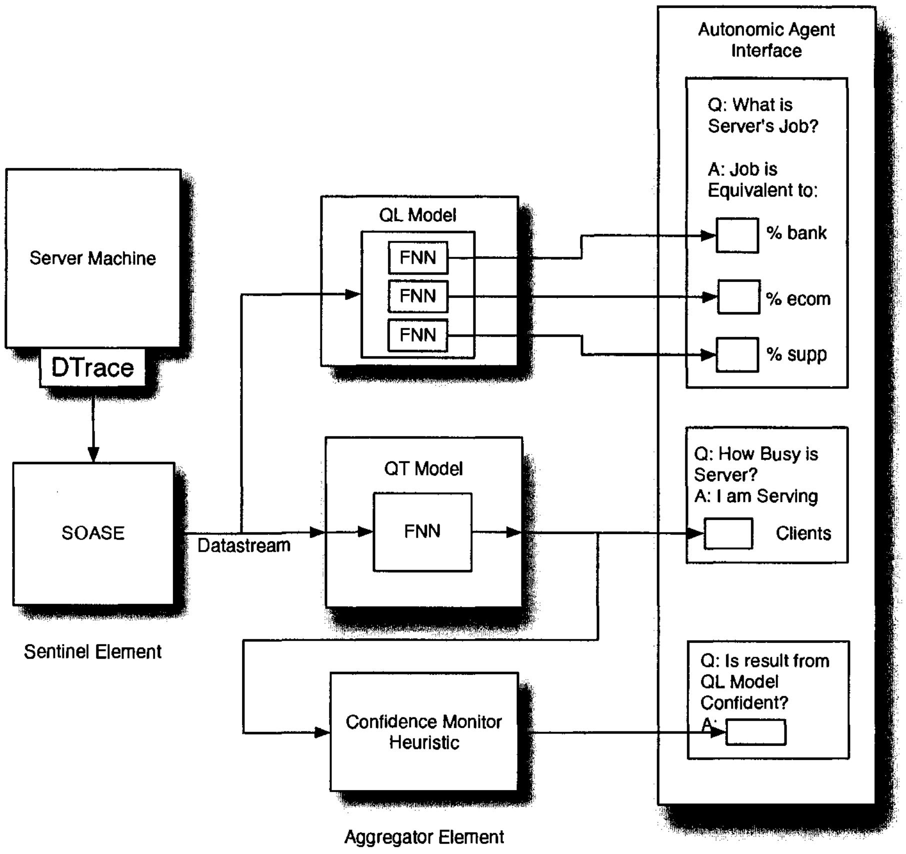


Figure 6: The Proposed Architecture for an Autonomic System Subcomponent

4.3 Procedure Framework

This section presents a procedure of steps which were followed to construct this prototype system. It is written as a guide for constructing a similar system as an aid to future work. It references the specific work done to construct the apparatus such as the construction of SOASE (which is detailed in Section 5.3) and the neuro-fuzzy learning technique (detailed in Section 5.4). In a extension of this work, these components could plausibly be replaced with some other analogous construct as an alternative.

1. Determine Target System – Computer hardware that comprises an autonomic system is likely to be quite varying. The work here is not to discover principals that can be applied to a wide range of different equipment. The analysis done in this work is directly tied to the specific behavior of a particular system. Also, a critical component relied upon is DTrace which limits the choice of operating system platform.[2] To embark on this work, that particular testbed has to be decided on as the hardware platform to collect data for. The data is assumed to be only relevant to that specific configuration of hardware. A SunBlade 1000 system and a particular suite of software detailed in Section

[2]At the time this work was started, DTrace was exclusively available in Solaris 10, however it has since been ported to Mac OS X 10.5 Leopard and an effort has been announced to port DTrace to a future version of FreeBSD

5.2.1 was the choice made for this work.

2. Write Plugins for SOASE – SOASE is the instrumenting tool that was constructed for this particular work. It provides the data on which the analysis is performed. At this stage, the scope of the system monitoring (specifics regarding daemons) must be determined. This may be taken in conjunction with the next step in that the areas of system activity to monitor may be targeted to provide coverage in the area that is needed.

3. Determine a Set of Comparison Points – Here, a decision needs to be made about the objective of the development of an autonomic system. As was discussed earlier, the target for this autonomic system is the webserver environment and the desire is to make qualitative and quantitative assessments about the load being experienced. To make the assessment, the system needs reference points to compare against. This is a semantic framework which is used to frame an answer to the inquiry. For this work, the three types of SPECweb2005 workload and the unit of one client sessions as designed by SPEC serves that purpose. In a different problem domain, outside of webservers and with different areas of assessment by the autonomic agent, a different choice would likely be necessary.

4. Design Training Data – Since this work uses a trained neuro-fuzzy method, an existing dataset is needed to train the model. For good results, this dataset

must be representative of the data that the model would encounter in a production system in a real-world scenario. Along the spectrum of output the model is to produce, sufficient coverage by the training data's points must be represented. In this work, special care was taken in the design of the training datasets to reflect the aspects of the necessary prediction output space.

5. Collect Data – The purity of the training data must be ensured. The collecting system should have minimal impact on the system resources to not influence the reading. In this work, low system impact was architected into SOASE and precautions were taken in the testbed to make sure there was no unaccounted factor influencing the datastream. Also, the training values associated with the different workloads being run must be as accurate as possible. In this work, the SPECweb2005 benchmarks are allowed to stabilize at the specified level before data is recorded to make sure the datapoints from that run assigned a training value are as uniform as possible.

6. Train Models – The next step is to extract the contained knowledge from the dataset. Since the major component of this research is that of qualitative classification, a soft computing approach is well-suited. In this work, an application was written which applies the two-phase neuro-fuzzy learning technique and outputs the model to a file which can be used later.

7. Validate Models with Independent Data -- To ensure the validity of the trained models, a new set of data must be run through it. This dataset must be crafted carefully to be independent from the training set which ensures the actual knowledge of the model is tested.

Following these steps will produce a set of models capable of making the classifications desired in the design. If the goal is to obtain the predictions in real time, the simple step of connecting the models to the real-time datastream remains. This task is not explored in this research. For the purposes of this exploration, this skipped step is of no value. The validation datasets serve as a simulation of this real-time deployment, where analysis can be conducted to evaluate the aptitude of the solution.

5 Experimental Platform

5.1 Overview

This section overviews the elements that had to be assembled to provide a testbed which allowed experimentation to be performed. That includes the physical hardware and prepackaged software setup (Section 5.2.1), SOASE (Section 5.3) and the Neuro-Fuzzy learning method that was adapted (Section 5.4).

5.2 Prepackaged Hardware and Software Setup

The hardware and software setup contain no components custom-built for this work. The discussion here details the assembly of pre-packaged components which were necessary to facilitate the further construction of apparatus. Here, they should be considered as an explanation of the setup mainly for completeness, but their construction is not representative of any gained value specific to this work.

5.2.1 SunBlade 1000 Systems and Network

The test hardware used in this work is three identical SunBlade1000 towers and a Linksys WRT54GS (V.6.0) router modified with DD-WRT firmware. The tower systems have dual 750MHz UltraSparcIII with 1GB of RAM and 37.5GB 10000 RPM SCSI drives running Solaris 10. One runs the client software (Client), another runs the BeSim (Backend Simulator) and the third is the server to be tested (System Under Test, or SUT). The webservers on the BeSim and SUT are running Apache (V. 2.2.3) using a PHP preprocessor (V. 5.1.6). The BeSim additionally requires mod_fastCGI (V. 2.4.0 is used in this case). These machines connect to each other over 100Mbit Ethernet through the switch in the Linksys router. Figure 7 shows a diagram of the network setup. The router provides a firewall against all outside Internet traffic which could make its way in and influence the test.

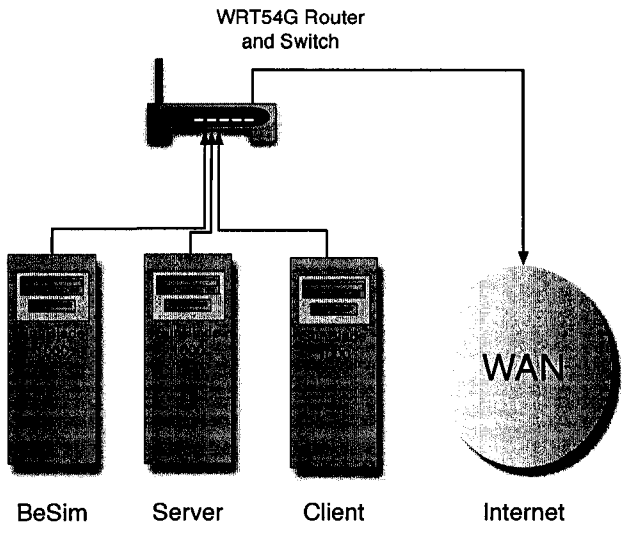


Figure 7: Benchmark Platform

5.2.2 Hardware Setup

Identical SunBlade 1000 systems are used as the three necessary machines. Here, the specifics of the setup are detailed.

5.2.2.1 Client The client is the machine that initiates the HTTP requests to the server and times responses. The SPECweb2005 client software can optionally be run on multiple client machines which make requests to the server. This is because the client simulation software is somewhat demanding of system resources. For all cases explored in this research, one system was able to handle the client simulation. For a larger simulation it may be necessary to spread the load over multiple machines so that the simulation is not bottlenecked on the client side.

5.2.2.2 Server This is the machine that is under the scrutiny of the benchmark, sometimes referred to as the System Under Test or SUT. It is set up with a web server, a choice of PHP or JSP and SSL (more description to come in Section 5.2.3). The PHP or JSP script execution by client HTTP or HTTPS requests is what drives the hardware and the efficiency that the requests can be completed is what the benchmark measures. The system is designed in such a way that this is the only software running on the system. It is simply a web server serving a special web application. All measurement is done on the client so there is no benchmark overhead cost incurred on the server.

5.2.2.3 BeSim This machine runs a simulation of the backend services that a webserver would likely access. This includes application and database services that would likely be present in a real system. On this machine, there is no actual database software running. Instead, for simplicity, a simulation of the delays that would be experienced by interacting with a real backend. This machine has to be set up with a web server, PHP or JSP and fastCGI modules (again, see Section 5.2.3). The server interacts with the BeSim system by accessing PHP or JSP scripts through HTTP interactions over the network.

5.2.3 Software Setup

There is a requirement of software dependencies to facilitate the benchmark. A few choices need to be made by the benchmark administrator while setting up the testbed. The software for the necessary three hardware systems is discussed independently below.

5.2.3.1 Client The client setup is relatively simple. The SPECweb2005 client software is written entirely in Java so, the only requirement is that each of the clients in the benchmark have a JVM installed (V. 1.5.0_06 in this case).

5.2.3.2 Server The server system needs to be set up with a web server package. The choice is given to the benchmark administrator which package to use. All are valid choices as long as they handle all standard HTTP requests and are able to preprocess the SPECweb2005 scripts. A version of the SPECweb2005 scripts are written is available in PHP and also JSP. The administrator will have to choose which one he or she wants and set up the testbed accordingly. Algorithmically, they are identical and the choice should not have any significant effect on the benchmark. Before the benchmark is run, the server needs the SPECweb2005 file resources installed on the filesystem for the server to access and serve to the client when requested. These files are the static pages, images, product descriptions and support files needed to serve to clients.

5.2.3.3 BeSim The BeSim requires a similar setup to the one done for the Server. A webserver software package which can process the PHP or JSP scripts is a requirement. If Apache is used, as it is for this setup, mod_fastCGI is a requirement. There are no file resources required by the BeSim. All interaction with the server is through the web server software and the SPECweb2005 scripts.

5.3 Sysmonkey Open-Architected Sentinel Element (SOASE)

5.3.1 Overview

DTrace's D language is quite complex and robust. It allows the administrator target very specific elements of the system and obtain very specific data regarding that element's operation. This is somewhat contrary to the informational needs of an autonomic agent. The agent requires information that is quite general in describing aspects of how the system is operating. As a result, relatively simple scripts were written which monitor general are in the prototype phase of the system. To collect and process this information, the Sysmonkey Open-Architected Sentinel Element (SOASE) was constructed. Full exploitation of the abilities of DTrace is not taken advantage of. In a limited capacity for this initial exploratory research, its unique abilities are leveraged to great benefit. It is left to future work to re-examine the scope of DTrace's usage in this way. The open-architecture design of SOASE was chosen to take the desire for future expandability into deliberate accommodation. It does this by decreasing coupling between the data collecting modules and the main aggregating component, allowing for different or additional data collecting modules to be added.

The SOASE was created to facilitate the monitoring of a Solaris 10 system. It monitors, preprocesses and presents data gathered from a live system in a sentinel element role of an autonomic agent. SOASE is primarily designed to handle monitoring data collected by DTrace which has access to extremely granular data from the use of custom written scripts. In addition to DTrace, it utilizes some facilities such as *vmstat* and *kstat* to gather some higher-level figures to present along with the other calculated figures.

The output of SOASE is a datastream of values for pre-defined measures of several aspects of the system activity. This listing is intended to be read programmatically by the aggregator element component of the autonomic agent, but is also interpretable by human eyes and could plausibly be used by a system administrator to diagnose. In this scenario, the human could take advantage of the preprocessing done on the raw data to obtain a high-level advisement. However, the goal of this research is to examine the needs of a software autonomic agent for managing the system.

A difficult challenge to be addressed by this design is to sort through and summarize the complexity without 'dumbing down' the result for human analysis. This can be illustrated better with an example. The load average statistic in a UNIX system is the average count of processes waiting in the ready-to-run state over a set amount of time. This measure is always provided over time intervals of one, five and fifteen minutes. An autonomic agent may be interested in this value over much shorter, or even much longer timeframes and should not be limited in this fashion. A primary function of SOASE is to perform data collecting and bookkeeping in order to provide proper flexibility to the agent.

5.3.2 Open Architecture Design

5.3.2.1 Design Overview There is no expectation of coming up with an all-inclusive monitoring scope, which is why the possibility for future expansion must be left open. Therefore, the requirement is that it be designed in a way which accommodates for new modules to be easily inserted. Also, given the exploratory and experimental nature of this work, this design is preferable to allow the easy addition of modules which monitor new statistics or families of statistics.

There is also design consideration for each of the modules to offer some flexibility in the statistics that are provided. Each module has an associated daemon which monitors the applicable DTrace probes and collects the data in memory structures. The main SOASE process is given a choice and will request certain stats the daemon is capable of generating. The stored data will then be processed into the desired result. This late-binding approach ensures the most flexibility by allowing variable parameters to be passed to the daemon.

5.3.2.2 Architecture As can be seen in Figure 8, the system takes on a layered architecture. The lowest level is the DTrace Layer containing D scripts which collect the raw data from the system. The data is passed upwards to the Daemon Layer process, which stores and processes it. Then the data, when requested is passed upwards to the main SOASE process through the specific module interface of the Collection and Processing Layer.

5.3.2.3 Modules Each module, as was seen in Figure 8 has three elements:

1. A DTrace D script process in the DTrace Layer
2. A daemon process in the Daemon Layer

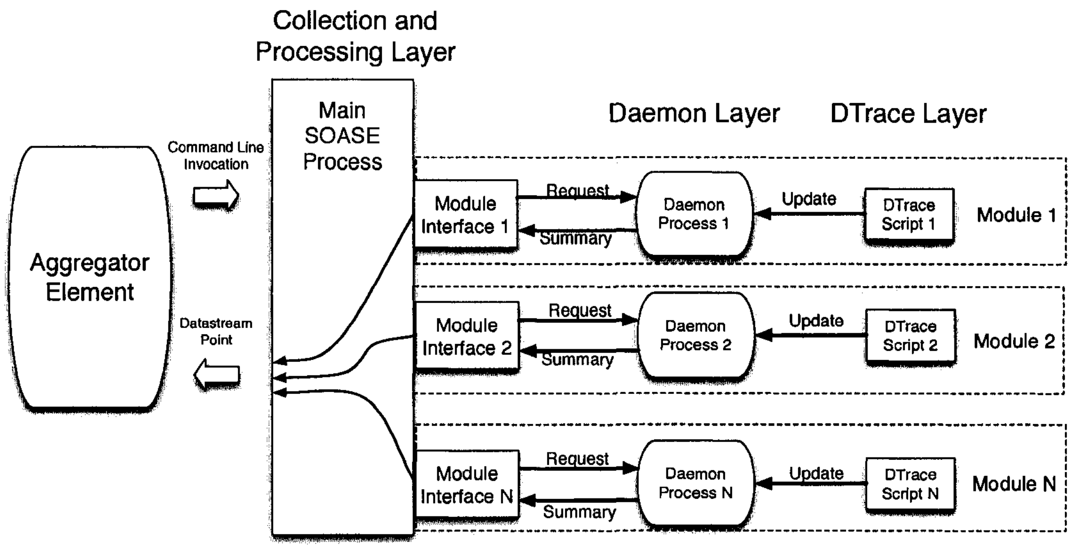


Figure 8: SOASE Architecture

3. A module interface in the main process in the Collection and Processing Layer

The DTrace script process is an executed instance of the console *dtrace* command running a script custom written for this module. It is invoked as a coprocess of the daemon process. All interaction is one-way, the script dumps its traces to standard out and they are read in by the daemon.

In the final version used for the analysis in this document, there are two daemon processes that are not tied to a DTrace script. One is invoking *vmstat* command running at a frequency of one second to monitor the free memory, swap and CPU activity. The second is querying the *kstat* API to monitor the levels of the DNLC cache. These were left this way as opposed to an identical DTrace-based solution because this was the most direct and simplest way of obtaining these particular values and no extra benefit was perceived to come from porting their operation to DTrace.

5.3.2.4 Main SOASE Process The main SOASE process is responsible for collecting all information from the daemons. It parses the optional command line parameters and passes correct messages along to the individual daemons, collects the responses and displays the results.

To communicate with the daemons, the main SOASE process opens a new domain socket to each and follows a protocol to request the information. This protocol follows a similar form for all daemons which eases the process for the addition of new modules, however the limitation exists that specific module interface code must be compiled in the main process codebase to handle the specific daemon. Less coupled architecture could be implemented which loads the interface binary in at runtime, but this was rejected to save time in implementation. Recompilation of code is not a prohibitive barrier for when new functionality is needed in the prototype phase.

5.3.2.5 Inside The Daemons Linked list data structures are employed to record events over time for a quite large time duration. This allows the data to be processed dynamically when a request from the parent process arrives at the daemon. This was chosen, as opposed to the destructive aggregation of data, to meet the flexibility for later additional functionality requirement as discussed earlier. The UNIX domain socket *accept()* call is used to maintain concurrency within the daemon, since it has to be constantly handling both new data arriving from DTrace and requests from the main SOASE process at random intervals. These requests will be handled in the order that they arrive, with their serialization guaranteed by the *accept()* call. This ensures that that the main SOASE process requests are met with the data current up to the exact point in time that the request is made, since the requests have to wait in the same queue as the incoming data.

5.3.3 Daemons

5.3.3.1 Overview This is a description of the six daemons of SOASE which were implemented and are used for the system analysis presented afterward. They were designed to cover an adequate subset of each of the four major subsystems in a computer system: CPU, Hard Disk, Memory and Network. The first four daemons were created to address those four in order and the last two were created to provide some other supplemental data. This is not a comprehensive set of system monitoring that could potentially be useful for this application. At this point, it is intended to establish a reasonable amount of coverage which could be built upon. Future work could include expanding this set to examine a greater cross-section of system activity.

The first four daemons utilize DTrace, whereas the last two get their data from the *kstat* kernel utility. *kstat* is able to provide similar data as DTrace, however it has shortcomings. Most prominently, obtaining data from *kstat* must be done on request, rather than DTrace's interrupt style of monitoring. That is, when a probe fires, the Daemon process can be notified immediately and they are received in order that events occur with an associated timestamp of the event. With *kstat*, the process has to poll the API at a regular interval for counter values kept track of inside the *kstat* kernel component. As a result, the data that can be obtained from *kstat* is of lower quality (it is aggregated and preprocessed) and the task of obtaining it is much more awkward. The values that SOASE obtains from *kstat* in this case is only supplemental to the more prominent DTrace monitoring. More description of this follows in the descriptions of the individual daemons.

5.3.3.2 ***procd*** This daemon provides a monitoring statistic which is quite similar to the Load Average statistic which is available on all Unix systems. The Load Average of a system is an average count of runnable processes that are waiting in the run queue or currently running over 1, 5 and 15 minute intervals. By consulting this statistic, an administrator can find out how busy the processors in a system are. If the load average is less than the amount of CPUs in the system, this indicates that the system is idle at least some of the time. A load average greater than the amount of CPUs in the system indicates that the system is past its maximum capacity proportional to the magnitude of the reading and there are likely slowdowns in process response times and throughput. The calculation of this statistic is very dependent on the sample rate of the polling of the process run queues. In Linux, and traditionally in Unix systems, this sample rate is every five seconds. [10] This is a problem with processes that are very short-lived and processes that wake up at a fixed interval which is less than or near the sample rate which cause this statistic to be quite inaccurate at times. SAR, the system activity reporter samples the run queues at a rate of once per second which is what gets provided to the Load Average statistic in Solaris 10, so it is easy to see how this can produce inaccurate numbers at times.

Here, with this daemon, we can increase the accuracy by increasing the sample rate to 1000Hz. DTrace also is able to minimize the performance impact of this measurement. With an increased sample rate, we can get an extremely accurate view of the processes queued up for the CPUs, which is more exact for the purposes needed in this research. The data is obtained by invoking the *profile-1000Hz* probe and accessing the data structure provided by DTrace of the current thread pointer and checking the length of the runqueue (*currthread->t_cpu->cpu_disp->disp_nrunnable*).

5.3.3.3 ***iod*** Hard drive access is often the biggest cause of slowdown in a computer system because it is the only part of the machine that has physical moving parts. A tool, *iostat* is provided in Solaris 10, which give measurements of the amount of data written or read from the drive. Similar tools are quite common in other operating systems. This is a high-level abstraction of the underlying issues that are present intended for human interpreters and is an excellent example of the problem described in the introduction of this document. Missing is the notion of sequential vs. random data access to a physical disk. Sequential access can provide a large data transfer rate and still have capacity to spare simply because the physical drive head does not need to spend time moving to provide I/O. Random data access, on the other had, can saturate a drive without indicating a high data throughput. This is because the drive head has to spend time moving back and forth across the cylinders. One would think that this could be addressed by also tracking the other factors which can lead to hard drive access such as system calls and page ins/outs and track the diversity of the locations involved to find out how far the drive heads are moving. While this option is available, it is sub-optimal. To improve hard drive performance, operating systems often cache content in memory and perform reads and writes in more efficient bursts to cut down on the head movement. Any meaning obtained from a measurement of drive access obtained from the non-DTrace tools the system becomes obfuscated in the internals of the operating system because the of these optimization.

This daemon cuts through all the abstraction and monitors right down at the I/O driver. It utilizes the *io:::done* probe from the kernel's hard drive interface driver to measure actual physical movements of the hard drive by recording the block number specified in the parameter of the kernel function associated with the probe (*args[0]->b_blkno*). This probe also provides stats on how much data was read and written to the drive (*args[0]->b_flags* to determine whether operation is a read or write and *args[0]->b_bcount* for the size of the data). Collected along with these drive transactions is the timestamps, and by implication, the serialized order that they occurred. This is especially useful for Autonomic Computing applications, since these raw statistics are exactly what are needed for recommending reconfiguration of the hardware in the disk subsystem.

5.3.3.4 ***paged*** The memory management subsystem is the next major system in need of monitoring. This system is responsible for bringing data into memory from the hard drive as needed and also providing memory to running processes. The daemon tracks the page-ins, page-outs and page-frees of the three types of memory pages in Solaris 10 which are executable pages, anonymous pages and filesystem pages. The probes:

- *vminfo:genunix:pvn_write_done:execpgout*
- *vminfo:genunix:pvn_write_done:execfree*
- *vminfo:genunix:pagio_setup:execpgin*
- *vminfo:genunix:pvn_write_done:anonpgout*

- *vminfo:genunix:pvn_write_done:anonfree*
- *vminfo:genunix:pageio_setup:anonpgin*
- *vminfo:genunix:pvn_write_done:fspgout*
- *vminfo:genunix:pvn_write_done:fsfree*
- *vminfo:genunix:pageio_setup:fspgin*

are the ones utilized for this daemon. When fired, they indicate a respective page in/out/free operation has taken place. The occurrence of these events are recorded and aggregated to pass upwards to the main SOASE process.

Executable pages are those which are binary executable programs that, to be run, need to reside in main memory for the CPU and faster caches to access in order to run. Anonymous pages are memory which is allocated by the processes for their own purposes. They generally are not a cache of content on the hard drive, since they are created on the fly by the processes, but in situations where memory is low, these are the pages that will be paged out and written to the swap space for later retrieval. File system pages are pages of data which are brought in from the hard drive at the request of a process for reading or modification. In Solaris, all memory which is not take up by Executable or Anonymous pages is often taken up by file pages which cache content in the chance that they are needed. The exact meanings of page-ins, outs and frees vary slightly for each type of page, but in general, page-ins means that the page has been allocated in memory and populated with the intended data, a page out means that the page was moved from memory to its backing store on the hard drive and a page free means that the page was destroyed without being written out (writing out not necessary when identical data already exists on the drive) and its space is returned to the free pool.

Solaris 10 has the *vmstat* tool which will give counts of these statistics sampled at a specified rate in seconds to display for the administrator. This has shortcomings in that it is also formatted for the human administrator to read and has a minimum sample rate of one second. By utilizing DTrace probes defined at each paging event, as each page action happens, the daemon records it specific to the millisecond in the order in which they occur. The end result, the produced data format is the same as *vmstat*, however the DTrace solution is much more accurate by not being constrained to one second sample rates.

5.3.3.5 ***tcpd*** The last major area of the system that needs a monitoring element is the network activity. This daemon monitors packets and data coming in and out through TCP connections. The existing facilities to monitor the traffic available are the *netstat* tool and also the *truss* tool which can monitor system calls to a network socket. Again, the problem with these tools is that they are made for human reading for diagnostic purposes, so they are a difficult fit to our needs. This daemon takes a similar approach as *iod* by going deep into the kernel to get the rawest data possible. This daemon uses probes from the networking stack to record the number and size of packets incoming and outgoing from the network and is able to provide counts for data throughput, average packet size and counts for the total packets sent and received. *fbt:ip:tcp_send_data:entry* fires when a TCP packet is written out and *fbt:ip:tcp_rput_data:entry* fires when a TCP packet arrives and the size can be determined by the associated kernel function's parameters (*args[2]*).

Examination of other transport-layer protocols are possible through DTrace in the same way that TCP connections are instrumented here. They were overlooked because it was known at the time of the design that monitoring of SPECweb2005 benchmarks were to be the focus of this work. What is generally though of as web servers (serving World Wide Web requests) typically stay within the domain of TCP connections. This is the case for the three types of SPECweb2005 behaviors. In the case that SOASE were to be extended for other applications, similar monitoring could be constructed for UDP or other network stack behavior.

5.3.3.6 *kstatd* Unlike the previous four daemons, *kstatd* does not utilize DTrace to obtain its data. Rather, it uses the *kstat* API which is provided by the kernel. It is accessed through a written program that uses the *kstat.h* interface and calls the *kstat_data_lookup()* function to obtain the structures which contain the data. The merits of this are discussed above in the overview. This daemon was developed as an attempt to monitor the filesystem from the perspective of the kernel's interaction with the filesystem modules. This proved difficult, since Solaris 10 supports many different filesystem types (UFS, NFS and ZFS being the major ones among many other minor ones) and each has its own segment of the kernel which applies. This daemon has the single task of monitoring the hit rate of the Directory Name Lookup Cache (DNLC) which is a cache in memory which maps file names of the mounted filesystems to inodes in the kernel. The DNLC is the only kernel component relating to the filesystem which is common to all. Initially, for the purposes of this research, the filesystem was treated as the fifth major component of the OS which needs to be monitored. Upon further exploration, it became more apparent that it has many overlaps in scope with other subsystems and cannot be deemed as important. This daemon, while less important, is still useful and was kept as part of the suite.

As is often the case in modern computer systems, the size of the filesystems are quite large. This can potentially become a rather large table which cannot be fully kept in memory at all times. The DNLC hit rate is the percentage of time that a kernel looks up a filename and doesn't have to page in part of the DNLC from the disk to make the mapping. This daemon goes into the *kstat* facility at a regular interval to obtain the DNLC lookups and DNLC misses over the last period and stores it for processing by request from the main SOASE process.

5.3.3.7 ***vmstatd*** This daemon also does not utilize DTrace to obtain its data. It instead uses the console *vmstat* tool to obtain some supplementary data. The stats generated by this daemon are the CPU utilization and memory utilization.

These are some basic stats that are easily available to the administrator in the system from many different locations. They are the most basic and straightforward measurements available and they were included to add information into the prediction model that would always be immediately available to any system administrator. They are not erratic and context-specific enough to benefit from the increased precision of DTrace. The daemon invokes *vmstat* every second and records the instantaneous measure to provide to the main SOASE process when requested.

It should be noted that the *vmstat* tool obtains its data from the *kstat* API as well and it would be possible to incorporate this data collection into *kstatd*. The reasons for their separation trace back to legacy factors when this framework was being developed. A future refactor of the SOASE codebase could result in merging these two daemons for increased logical cohesion.

5.3.4 Datastream

The output set of data from SOASE will be referred to as the datastream. This datastream is what is passed up the chain to the next components. Figure 9 illustrates the measurements being passed into the stream broken down from the individual daemons. All individual stats in a datapoint are a rolling time-averaged value of the statistic's reading over the last 10 seconds in the system. A datapoint is added to the datastream represents one second of system activity. For example, a page-in event recorded by DTrace and processed by SOASE would be represented in the average of ten different sequential datapoints from the second it occurred until 10 seconds later, when it it falls out of the rolling window.

The next elements in the chain to receive this stream of data are the neuro-fuzzy models which comprise the aggregator element. Ahead in Section 5.4.3 there is a discussion of the necessary fuzzification of this data. As a quirk of the system architecture, the fuzzification is actually performed within SOASE, therefore in actuality, the datastream is composed of the fuzzified sets of 3 or 5 values for each of the 23 statistics listed in Figure 9. SOASE was constructed this way to decouple the logic of the inner workings of the framework and the preprocessing of data from the machine learning components which implements the network training and evaluation.

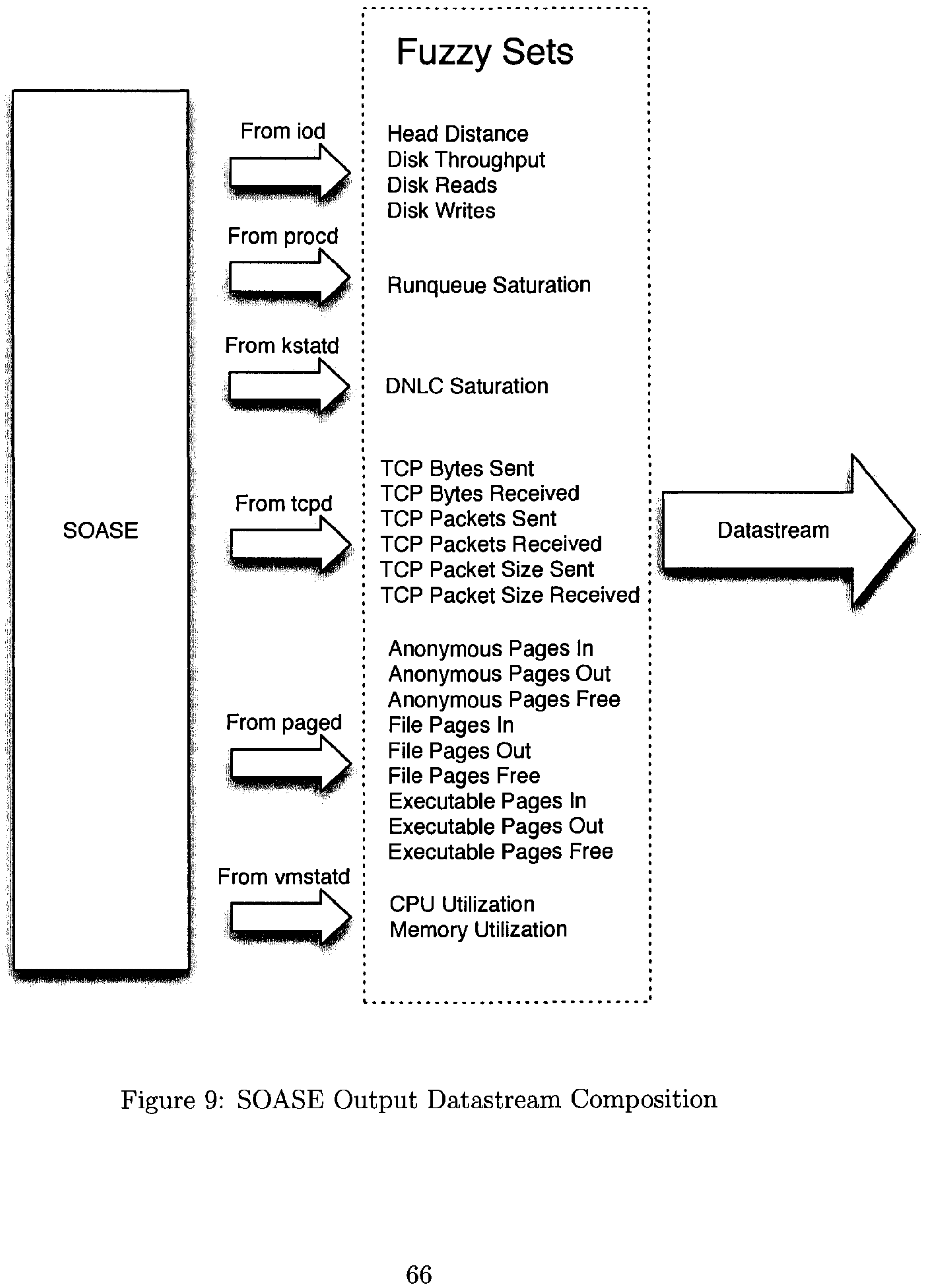


Figure 9: SOASE Output Datastream Composition

5.4 Fuzzy Neural Network Training

5.4.1 Specific Application of Two-Phase Learning Process

The learning method described in Section 2.3.2 is used in this work to train models from collected datasets. As a starting point, an implementation of the two-phase training process was obtained from some previously unpublished work done within the department. This was in the form of a program written in C for some specific exploration of the training process. This program was hard-coded to work in a different application with a different type of dataset, but with some modification, it was made to work in the domain of this particular problem. Since this application works with real-world crisp values, in order to plug the numbers into a FNN, fuzzification and defuzzification transformation had to be done. This is covered in Sections 5.4.3 and 5.4.4.

The backtraining method inherited from the code worked to train for accuracy of the multiple fuzzy outputs and not final crisp values. As a result, it had to be re-evaluated and modified to provide more accuracy in the training to meet the crisp-value target. The specifics of the adaptation to the backtraining phase are discussed in Section 5.4.5.

Also, a part of the inherited code is a scheme for cross-validation of the training data. This is expanded to a 10-fold cross validation process which is required for proper scrutiny of the knowledge which is gained by the model. The testing and validation partitions made by these schemes are utilized differently in the two phases which has the potential for causing confusion. Clarifying this procedure, as is done in Section 5.4.2, is helpful for understanding the analysis of the trained models that follows in Section 7.

5.4.2 Cross-Validation

Cross-validation of the models against the training data is not semantically identical between the two phases of the training process. Inherited from the original code was a scheme of holdout validation. The original dataset is split into two partitions, one for training and the other for testing. The testing partition is used to test the knowledge of the model trained with the training partition, however for this work, it is only preliminary validation. Proper validation will be done with fully independent data (which will be referred to as validation data sets) after the training process is complete. This ensures that the validation and analysis is done on datapoints which are independent from the points that are used to build up knowledge in the model.

In the revised implementation, the training is done in a k-fold manner with k equal to 10. In essence, the new procedure is identical to the holdout scheme, but it is conducted ten times with a different split of the partitions to provide the benefit of multiple models to obtain a better cross-section of the learned characteristics. The data is split into ten equal partitions and the two-phase process is run ten times to train ten networks. Each run uses a different partition as the testing partition and the remaining nine as the training partition. Roughly the same proportion of datapoints should be contained in each due to the randomized order. The ten models are preliminarily analyzed and discussed in Section 7 using their respective testing partition. In the validation component of this research in Section 7 where completely new data is used, all ten models are again evaluated to provide a better representation of the ability for the knowledge to be learned by aggregating the results from ten different examples.

In the GP phase, the randomly generated and combined models of a generation are scored for fitness based on how well the model performs (validates) on the training partition. Since the scoring for selection is based entirely on this partition, there is the risk of overtraining in this phase. It was observed that if the first phase was run until the fitness levels out to an apparent minima, overtraining was occurring. It was then observed that this trained model was unsuited to make any gains from the second phase and resulted in a sub-optimal final result. By stopping the GP while the fitness was still in motion towards the minima, the problem of overtraining is avoided. That model, taken through the second phase results in a model with a performance which is better than at the apparent minima in the first phase.

The backpropagation phase works by feeding the points of the training partition into the network one-by-one and making backpropagation adjustments to the weights in the network. Here, the testing partition is not used for the backpropagation, but it is essential as a metric of the training's progression. So, in contrast to the first phase, the testing partition is used as a metric of fitness instead of the training partition. Again, overtraining is a possibility and was observed during the investigation. An indication of this is when the performance of the model on the testing partition becomes gradually worse as training progresses. The only way to combat this phenomenon is to stop the training when the measured performance on the dataset reaches a minima and it can be certain that a better minima will not be obtained in a reasonable amount of time if the process is allowed to continue. The final cross-validated score during the second phase is the basis for reaching a stopping point for the training process.

To recap, the training partition is used to evaluate the fitness of evolved models in the first phase and is use as the basis for the backpropagation training phase. The testing partition is only used to determine the stopping criterion in the backpropagation training phase and is reserved for preliminary analysis of the final resulting model.

5.4.3 Fuzzification

To work as inputs to the FNN, the crisp values have to be fuzzified. This is necessary for a linguistic representation that will eventually be possible to be derived from the model. A library routine was programmed to split a crisp value into any number of Gaussian fuzzy sets.

The library routine works by taking the range of values which are the domain of the crisp value and splits that region into multiple Gaussian fuzzy sets.

Given the domain of the crisp values $[A_{min}, A_{max}]$, the height of function's crossover $B_c, 0 \leq B_c \leq 1$ and the cardinality of the fuzzy set, N, the resulting fuzzy transformation $F_i(x)$ is calculated as:

$$F_i(x) = e^{\frac{-(x-b_i)^2}{c_i^2}} \tag{2}$$

b_i governs the horizontal offset in the domain and c_i is a control parameter the 'width' of the curve. They are calculated from the provided parameters as follows:

$$b_i = A_{min} + i(\frac{A_{max} - A_{min}}{N-1}) \tag{3}$$

$$x_i = A_{min} + (\frac{A_{max} - A_{min}}{2(N-1)}) + i(\frac{A_{max} - A_{min}}{N-1}) \tag{4}$$

$$c_i = \sqrt{\frac{-(x_i - b_i)^2}{ln(B_c)}} \tag{5}$$

Two examples are illustrated in Figure 10.

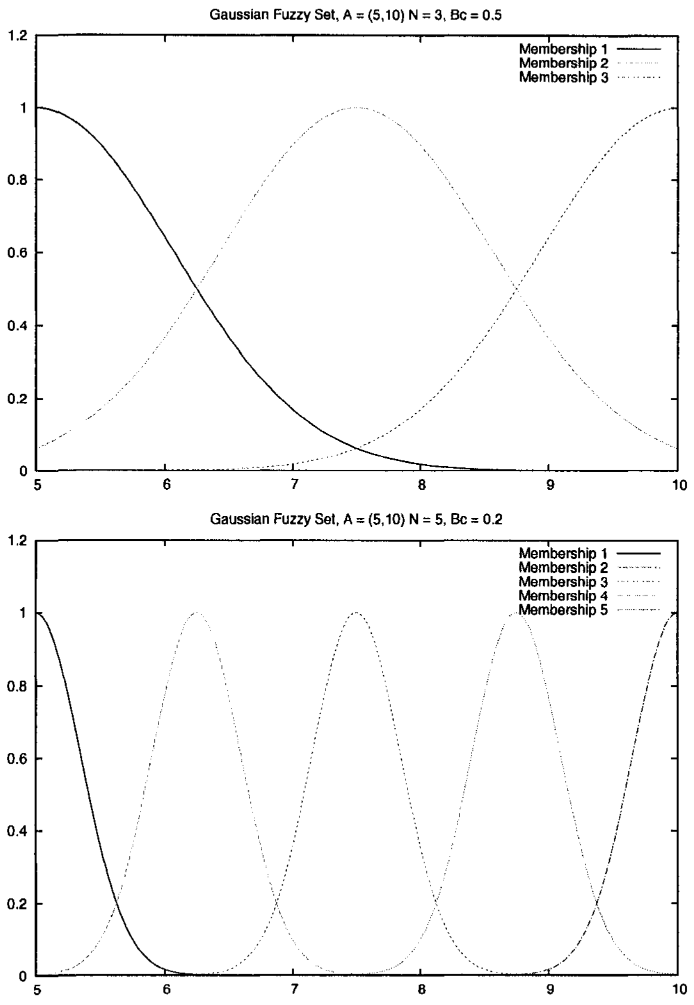


Figure 10: Gaussian Fuzzy Set Parameters Illustrated

Each monitoring statistic is split into a subjectively chosen amount of fuzzy partitions. The number is chosen on a case-by-case basis by examining visualizations of the result from the datasets. Values with clear 'small', 'medium' and 'large' spreads were split into three fuzzy partitions $N = 3$ and the new points were labeled as (original stat)_few, (original stat)_some and (original stat)_lots. This is chosen because, that is likely how one would refer to the values linguistically when describing. This was done as a 'cheat' to help the training phase of the models pick up on these trends easier by designing these particular fuzzified inputs in an intelligent manner. Values which have a much more ambiguous spread were split into five different fuzzy partitions $N = 5$, using suffixes 'not_very_much','a_few', 'a_bunch', 'a_lot' and 'tons'.

For simplicity, the crossover point, B_c was chosen to be 0.5, since there was no reasonable motivation from making any adjustment at the time which this was designed.

The domain parameters, A_{min} and A_{max}, were also chosen on a case-by-basis for the statistics obtained from the SOASE daemons. The practical range of values that occurred in in the running of the likely target workloads of this particular research was determined through observation. In the case a point is observed out of the bounds, the value is clipped to fit the maximum or minimum level of the domain. By nature of many of the measured statistics, for example packets received, a reading of zero is the minimum observed, often in the case of no system activity. In these cases, a negative reading is nonsensical so the lower bound was chosen as zero.

5.4.4 Defuzzification

On the output end of the network, three values comprises the fuzzy set which represent a crisp value. To obtain that crisp value, a defuzzification operation needs to be done. A routine is used which defuzzifies the set by performing a center of mass calculation (COM) on a shape composed of superimposed alpha-cuts of the three Gaussian functions. This step, in practice, is quite time-consuming during the training process and consumes the majority of the CPU cycles. The center of mass calculation is done by splitting the surface into n differential areas and proceeding with the center of mass for a n-particle system. For the GP and backpropagation process, this routine needs to be executed many times and it proves to be a large bottleneck if a large n is used. To speed up the training process, a reasonably accurate approximation was used for the purposes of this work with a $n = 60$. A middle Riemann sum approximation calculation is done to minimize the error for low n calculations. The y-coordinate of the center of mass is irrelevant for the defuzzification. The x-coordinate of the center of mass is what the routine produces.

First, the function of which we are approximating the contained area of, $F(x)$ has to be defined. It is the superposition of the α-cuts of the membership function. In this example, we will assume the amount of fuzzy values we are defuzzifying is three. $f_1(x)$, $f_2(x)$ and $f_3(x)$ are the α-cuts of the Gaussian functions which are cut off at the level of the corresponding membership values. The parameters of the Gaussian function, b_i and c_i were calculated using the method described earlier with a domain of $(0.0, 1.0)$, a cardinality of $N = 3$ and a crossover point of $B_c = 0.5$

$$\begin{aligned}
f_1(x) &= MIN(e^{\frac{-(x-b_1)^2}{c_1^2}}, x) && (6)\\
f_2(x) &= MIN(e^{\frac{-(x-b_2)^2}{c_2^2}}, x) && (7)\\
f_3(x) &= MIN(e^{\frac{-(x-b_3)^2}{c_3^2}}, x) && (8)\\
F(x) &= MAX\Big(MAX\big(f_1(x), f_2(x)\big), f_3(x)\Big) && (9)
\end{aligned}$$

Now, to obtain the center of mass of the area under the curve, we start with the generalized formula:

The center of mass R of a system of N particles is defined as an average of their positions r_i, weighted by their masses m_i.

$$R = \frac{1}{M}\sum i = 0Nm_i \cdot r_i \tag{10}$$

Once that is defined, the center of mass x-coordinate (R_x) is then derived to be for $n = 60$ particles is:

$$R_x = \frac{\sum_{i=0}^{59} F(\frac{i}{20} - \frac{39}{40}) \cdot (\frac{i}{20} - \frac{39}{40})}{\sum_{i=0}^{59} F(\frac{i}{20} - \frac{39}{40})} \tag{11}$$

Equation 11 is what is used for all defuzzification in this research and it was optimized into a library routine.

5.4.5 Backpropagation Training Adaptation

In the evolutionary phased of the FNN's construction, selection was based on how the defuzzified values matched the values the training set. The fitness function scored the results by passing the outputs of the network through the defuzzification process to compare to the crisp training value, rather than a fuzzified representative of it.

When a crisp value is fuzzified, it produces a fuzzy set with perfect alignment, as illustrated in Figure 11. Perfect alignment cannot be relied upon from the outputs of a neural network since the architecture is not bound by such constraints. The center of mass defuzzification compensated for this fact by still being able to reconstitute an imperfectly aligned set, as is also illustrated in the figure, into a crisp value.

An example of this is a network providing an output that is 1.0 membership to the 'small' category and paradoxically, also a 1.0 membership to the 'large' category. By COM defuzzification, this result can still be made sense of as somewhere in the middle ground. Alternatively, a moderate value fuzzified into a set would have a large membership in the 'medium' category and very little membership in the 'small' and 'large' category.

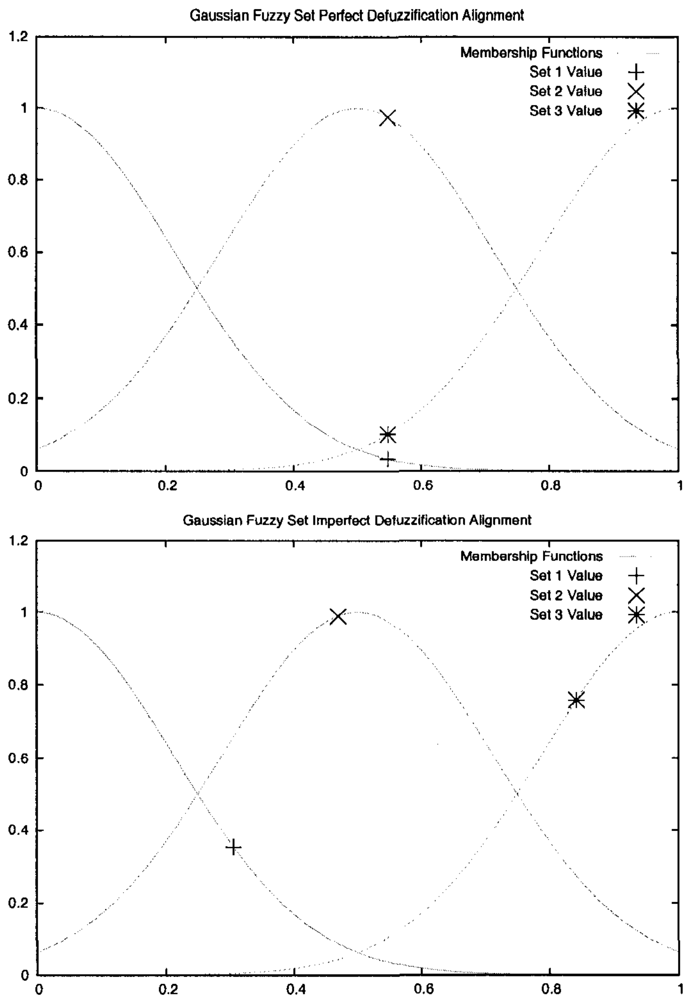


Figure 11: Gaussian Fuzzy Set Perfect and Imperfect Alignment

This required step of defuzzification poses no problem to the selection in the GP phase of training. This however, does have implications in the backpropagation training phase. Traditionally, for FNN training, the backpropagation is started at each of the outputs that each represent a fuzzy membership value. The error term is calculated from how closely the output (likely having imperfect alignment) matches the same membership value from a fuzzified crisp training value (having perfect alignment). This has the downside of reducing the output domain space of the network. This occurs because the network is only trained towards definite, crisp answers. Ambiguous answers as described in the previously mentioned example are trained out of the network, when perhaps they have some validity and their apparent ambiguity carries meaning.

A way to preserve the complete output space is to train the network towards how well the set of fuzzy membership outputs defuzzify to match a crisp training value rather than training to match a value of perfect alignment. For this to occur, a method had to be invented and applied to the backpropagation algorithm. This method functions by making adjustments of weights backwards through the network proportional to first, how much the weight contributed to a wrong answer and second, how wrong that answer actually was. A good explanatory analogy for this concept is 'blame.' If a node shares some blame for a bad answer, it is adjusted proportional to its blame in the right direction to make the answer more correct. If these adjustments are repeated many times over many points, the weights in the network will converge to a state which minimizes error in the defuzzified output as in normal backpropagation training where error is minimized in the individual node's output.

To factor the error in the defuzzified value into the training process, a blame value has to be assigned to each membership function node based on how much it contributes to the error of the derived defuzzified value. This process is not as straightforward as in normal training where the analog for blame is simply the error in the output of the node. Here, the set of three node's outputs have to be taken in account in order to assign a blame to each node. For purposes of description and for all applications in this research, only three output nodes are taken into account at a time for all calculations. That is not to say that this couldn't be adapted for defuzzification of larger cardinality fuzzy sets. In that case, this scheme would have to be mathematically extended to accommodate which is not done in this work.

The first step in this calculation is to decide which direction the node's output should be adjusted in order to improve the defuzzification. Starting with the three values of the outputted fuzzy set, the one corresponding to the node in question is set to a value of 1.0 and the defuzzification is performed. Another defuzzification is performed with the same node's value set to 0.0. The errors of the two defuzzifications are then compared to the training value and the better prediction is determined. It can be now be decided that the prediction would be improved by adjusting the node's output in the winning direction.

The second step is determining the magnitude of the blame. For convergence of the network, the adjustments need to be proportional to the amount the nodes are in error. There are three factors in the calculation of magnitude: the *output error*, the *improvement factor* and the *blame factor*. The *output error* is simply the error between the result defuzzified value and the training value for the data point. The *improvement factor* is the error between the defuzzification of the winning adjusted value set and the training value. The *blame factor* is a constant which is set at the beginning of the training that ensures that the adjustments are made in a proper order of magnitude to allow the network to reach convergence. For the training done in this research, a *blame factor* of 10.0 was found to be an acceptable value.

A pseudocode representation of the implementation described above is presented in Figure 12. This method obtains the blame for a particular output node (belonging to a three node set that produces a fuzzy set) which is used in the backpropagation process. *defuzzify()* evaluates Equation 11 as was described in Section 5.4.4. *output* is the resulting three-value vector obtained from the network. *correct* is the crisp training value. *node_no* is the node number which we desire the blame for.

Once this blame value has been obtained for the node, the backpropagation proceed normally using this value as a substitute for what normally would be the error of that nodes's output. This method was invented out of necessity for the application of

```
begin get_blame

    estimate = defuzzify(output[1],output[2],output[3])
    output_error = abs_value(correct - estimate)

    if node_no is 1 do
        estimate_zero = defuzzify(0.0, output[2], output[3])
        estimate_one =  defuzzify(1.0, output[2], output[3])
    else if node_no is 2 do
        estimate_zero = defuzzify(output[1], 0.0, output[3])
        estimate_one =  defuzzify(output[1], 1.0, output[3])
    else if node_no is 3 do
        estimate_zero = defuzzify(output[1], output[2], 0.0)
        estimate_one =  defuzzify(output[1], output[2], 1.0)
    end

    delta_zero = abs_value(estimate_zero - correct)
    delta_one =  abs_value(estimate_one - correct)

    if delta_zero is less than delta_one do
        direction = -1
        improvement_factor = abs_value(estimate_zero - estimate)
    else
        direction = 1
        improvement_factor = abs_value(estimate_one - estimate)
    end

    blame = direction * blame_factor * output_error * improvement_factor

    return blame
end
```

Figure 12: Pseudocode of Blame Calculation

this particular neuro-fuzzy learning technique in a real-world scenario. Its behavior perhaps deserves deeper study, which it will not receive in this work. Its validity can only be argued in this case with real-world performance-based evidence of its contributions to this problem.

6 Model Overview

6.1 QL Model Investigation

6.1.1 Purpose

This particular investigation is the central part of this work. It is designed to test whether a neuro-fuzzy model can be trained to classify the current experienced load on a machine in terms of the three types of load provided in the SPECweb2005 suite. The Banking, Ecommerce and Support loads of the benchmark are designed to stress and bottleneck different hardware subsystems. It is easily hypothesized that the differences from the cross-section of data collected from DTrace as described in Section 5.3.3 will be adequate to distinguish between the three types of activity. This hypothesis was proven to be true by a simple preliminary experiment. In this experiment, samples of DTrace data were trained into a simple multilayer perceptron model plugin using the Weka[3] [33]. This model's task was to take a datapoint as input and classify it into one of three discrete classes, one corresponding to each of Bank, Ecommerce and Support. The perceptron model was able to provide a recall and precision of greater than 99% which was quite promising for futher exploration of the data for more complex classification. This is a simple reflection of the fact that the benchmarks were designed to stress different areas of the system and this

[3]Available for free under the GPL at `http://www.cs.waikato.ac.nz/ml/weka/`

shows up quite recognizably in the monitoring datastream.

This preliminary result is not particular interesting since this task is far from what an administrator would encounter in a real-life situation. The questions such as varying degrees of composition and amount of load would be much more difficult for a human to answer, but a machine learning approach may have a chance. In the real world, a particular workload is not going to exactly match one of the three types and the model will have to make a prediction of a value along a continuum.

The purpose of this investigation is to determine if a model can be constructed which can classify an arbitrary web server load in a three-axis qualitative space defined by the three types of SPECweb2005 benchmark runs.

One can imagine an autonomic agent as an intelligent entity. One philosophical measure of computer intelligence is the ability to engage in a conversation with it. Being able to ask a machine questions and get meaningful answers back from it is a highly useful ability [20] [3]. First, is could be useful to a system administrator who is engaged in the task of gathering information in their system. Second, it could be useful for other computerized systems or autonomic elements since it represents knowledge in a digital form that can be reasoned and extended upon. The question that we are developing a model to answer in this case, albeit in a limited capacity, is "What is your job?" By classifying a machine's real-time load on these axis provides knowledge that can be processed into an answer to the question.

6.1.2 Design

A set of test data needs to be collected to use for training a model. In this specific experiment, the client load (net SIMULTANIOUS_SESSIONS) remains constant for the period of collection for the model's training data. This is to isolate out all other factors in the experiment and focus on the one critical variable that we are interested in for this particular investigation. The varying parameter is the composition of the load which the client system is requesting from the server system. The goal is to prove that meaningful predictions can be made regarding the load composition when all other parameters remain equal. The problem of dealing with situation where the client load is outside of the parameters which a model is trained under is addressed in the confidence monitor investigation in Section 6.3.

Creating a variable makeup of the load presents a problem. This runs against a assumption that was made when the SPECweb2005 benchmark was designed. The assumption is that the administrator of this benchmark will be using it to test the performance of the system and not to provide custom test workloads for an alternate purpose. Fortunately, an element of the benchmark's design is able to be exploited to provide what is needed. Since a separate set of software does not need to be installed on the server machine for each type of load, the scripts for each of the three workloads are world-readable for access and execution at all times on the server. This allows three separate client processes of different types to make requests to the same server over the network, each unaware of the existence of the other two. All requests will be handled by the server. The type of load and amount of SIMULTANIOUS_SESSIONS is configurable at each client and the three sessions can then be executed at the same time. The server then receives a load that is a mixture of three different types of traffic which constitutes what is needed to meets our requirements. This modification is illustrated in Figure 13.

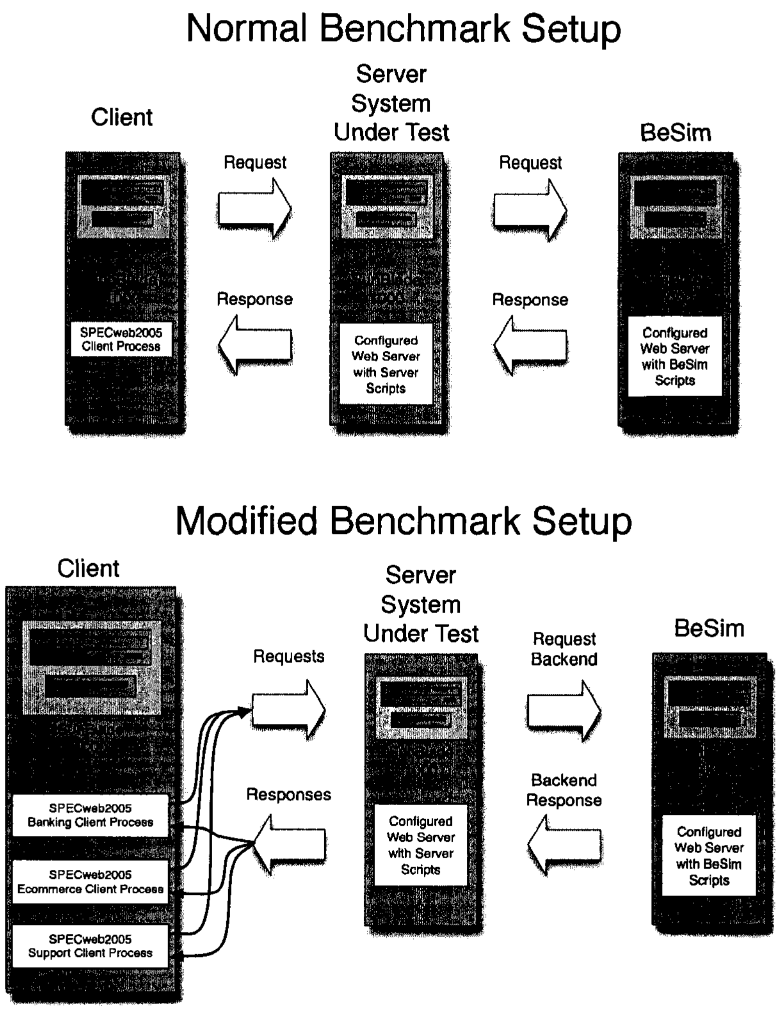

Normal Benchmark Setup
Client
Server System Under Test
BeSim
Request
Request
Response
Response
SPECweb2005 Client Process
Configured Web Server with Server Scripts
Configured Web Server with BeSim Scripts
Modified Benchmark Setup
Client
Server System Under Test
BeSim
Requests
Request Backend
Responses
Backend Response
SPECweb2005 Banking Client Process
SPECweb2005 Ecommerce Client Process
SPECweb2005 Support Client Process
Configured Web Server with Server Scripts
Configured Web Server with BeSim Scripts

Figure 13: Normal and Modified Benchmark Setup

To complete this investigation, a validation dataset will be collected and run through the obtained model. This dataset will be composed of datapoints with different proportions of client activity than the conditions that the network was trained under. This has the specific design intent of testing the generalized knowledge gained by the model and will validate ability of the model to perform in a real-time environment.

6.1.3 Visualization

The model being trained will give, as output, the similarity of a load to the three different extremes. This output point is able to be visualized on special plot called a ternary diagram. A ternary diagram is the mapping of a three-axis space into a two-dimensional diagram. It is only applicable when the three-dimensional data conforms to a special property. If the sum of the three datapoints is constrained to always equal to 1.0 it is able to be plotted on the ternary diagram, as is the case in our problem domain. Figure 14 gives an example of a ternary diagram, as it will be used in this work with a few sample points for illustration.

Activity in the system will be classified against three points of reference which are defined by the three workloads of the SPECweb2005 benchmark. Visualizing a plane where these points are three corners of an equilateral triangle, a point which is within the bounds represents a workload that is characteristically similar to the three corners of the triangle by how close the point is to one of the triangle's corners in Cartesian distance. A triple of variables a, b and c where the sum is constrained to equal to 1.0 can be mapped on a ternary diagram. The ternary plot can be represented in a two dimensional space by applying a transform to the triple of points down to a two dimensional Cartesian coordinate. The three extremes corresponding to $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ map to the equilateral triangle vertexes of $(0,0)$, $(1,0)$ and $(\frac{1}{2}, \frac{\sqrt{3}}{2})$ respectively. The intermediate points, generalize by (a, b, c), can be mapped onto the plot as $(b + \frac{c}{2}, \frac{c\sqrt{3}}{2})$. This provides an easily interpretable visualization of the data upon which understanding can be had and reasoning can be made upon.

In this work, ternary diagrams are used as a visual aid and also as a descriptive aid. Concepts can often be more easily described by involving the topology of a ternary diagram.

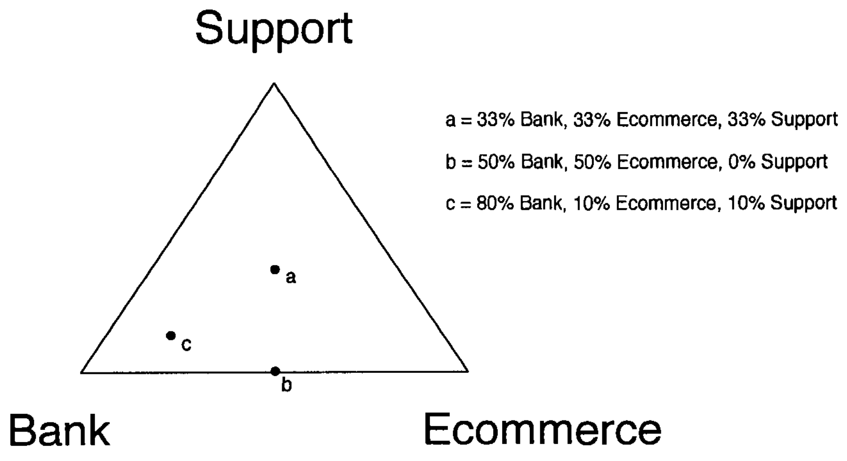

Support
a = 33% Bank, 33% Ecommerce, 33% Support
b = 50% Bank, 50% Ecommerce, 0% Support
c = 80% Bank, 10% Ecommerce, 10% Support
a
c
b
Bank
Ecommerce

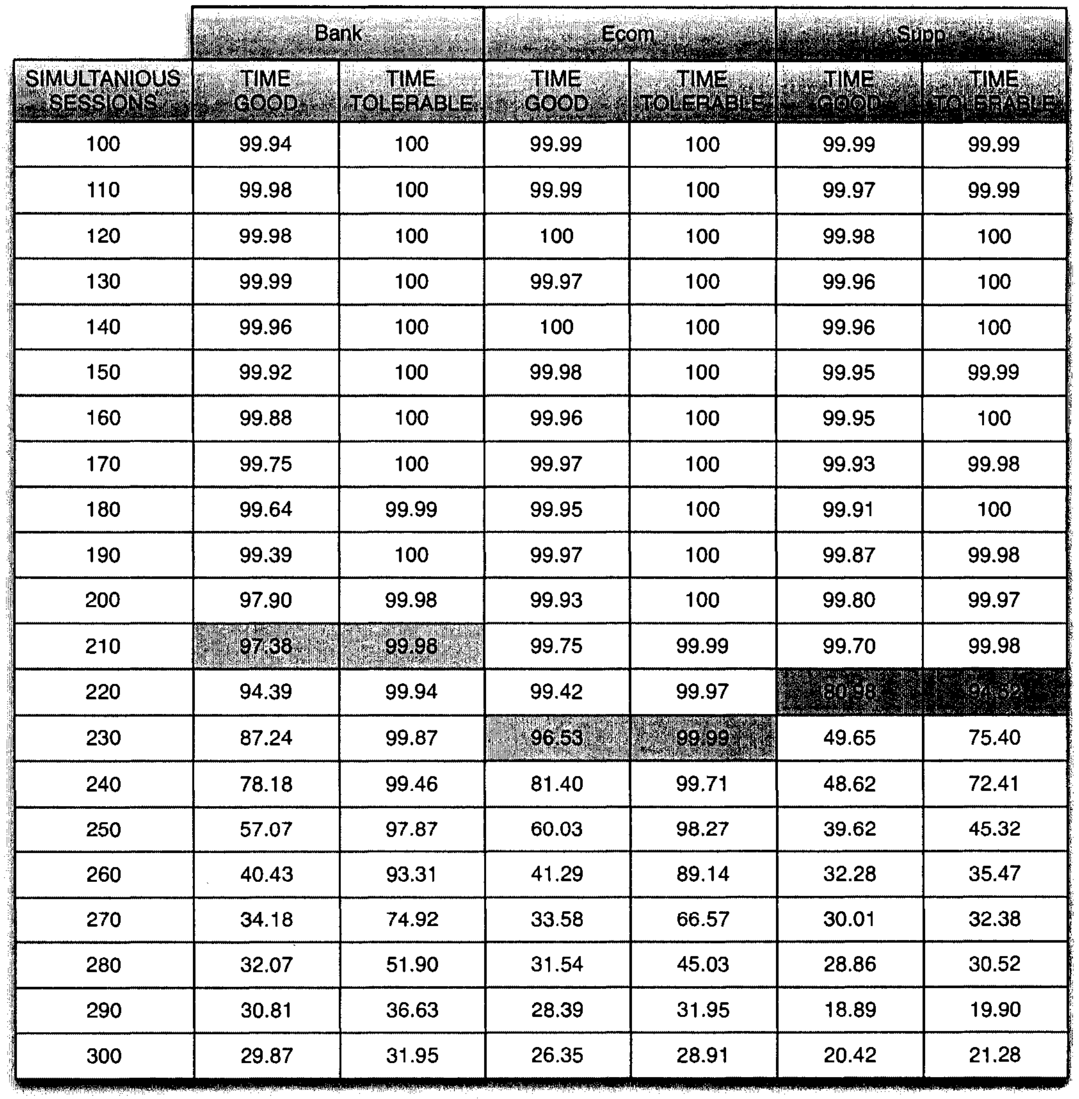
Figure 14: Ternary Diagram Example of Qualitative Classification

In situations where action may be necessary for configuring and optimizing a system, the machine needs a sense of the nature of the load that it is experiencing. Space on this diagram can be partitioned to correspond to a certain behavior or action for the autonomic agent to consider. Using the developed model in a real-time environment by an autonomic agent, conclusions can be made upon which area a real-time datapoint falls in this two-dimensional ternary diagram space.

In addition to the ternary diagram, in this investigation results from the models will be shown on histograms. This is useful for seeing relative differences between the results broken down by category and group of results. It is more intuitive for spotting differences in absolute values of error and standard deviation and is used to give a more exact analysis.

6.1.4 Training Data Set

Figure 15 illustrates the performance capabilities of the SunBlade 1000 server system as measured by the SPECWeb2005 benchmark under normal circumstances. At every setting of the SIMULTANIOUS_SESSIONS variable, three benchmark runs are done; one for each workload type. For each run, two statistics are displayed, TIME_GOOD and TIME_TOLERABLE. TIME_GOOD is the percentage of requests handled in under 2 seconds and TIME_TOLERABLE is the percentage of requests handled in under 4 seconds. As would be expected, the ability of the system to handle requests in a timely manner as the amount of simultaneous clients goes up is decreased.



	Bank		Ecom		Supp	
SIMULTANIOUS SESSIONS	TIME GOOD	TIME TOLERABLE	TIME GOOD	TIME TOLERABLE	TIME GOOD	TIME TOLERABLE
100	99.94	100	99.99	100	99.99	99.99
110	99.98	100	99.99	100	99.97	99.99
120	99.98	100	100	100	99.98	100
130	99.99	100	99.97	100	99.96	100
140	99.96	100	100	100	99.96	100
150	99.92	100	99.98	100	99.95	99.99
160	99.88	100	99.96	100	99.95	100
170	99.75	100	99.97	100	99.93	99.98
180	99.64	99.99	99.95	100	99.91	100
190	99.39	100	99.97	100	99.87	99.98
200	97.90	99.98	99.93	100	99.80	99.97
210	97.38	99.98	99.75	99.99	99.70	99.98
220	94.39	99.94	99.42	99.97	80.98	94.52
230	87.24	99.87	96.53	99.99	49.65	75.40
240	78.18	99.46	81.40	99.71	48.62	72.41
250	57.07	97.87	60.03	98.27	39.62	45.32
260	40.43	93.31	41.29	89.14	32.28	35.47
270	34.18	74.92	33.58	66.57	30.01	32.38
280	32.07	51.90	31.54	45.03	28.86	30.52
290	30.81	36.63	28.39	31.95	18.89	19.90
300	29.87	31.95	26.35	28.91	20.42	21.28

Figure 15: SunBlade 1000 Benchmark Performance Levels

Here, a standard level of client load is needed for the design of the dataset to eliminate it as a variable. By looking at Figure 15, it can be seen that after 210 SIMULTANIOUS_SESSIONS, the performance on the Banking workload starts to fall off. The same happens for about 230 sessions for the Ecommerce workload and 220 sessions for the Support workload. By design in the SPECweb2005 benchmark, the load corresponding to a single client session is equal across the three types of activities to not bias results to favor one hardware configuration over another. These differences can be attributed to the specifics of the hardware in the tested SunBlade systems. In an autonomic setting, about 200 sessions would likely be a situation where a change of configuration would be considered for this set of hardware. In all cases for this trial the net SIMULTANIOUS_SESSIONS will be constant at 200 and a single session of each of the three types are treated as being exactly equivalent in terms of load quantity for the purposes of comparison. If future work were to be done involving different hardware, these assumptions for performance may have to be re-evaluated and the investigative procedure shifted to match.

Following the decision to keep the client load constant at 200 sessions, a dataset for training the QL model was collected. Six thousand data points were collected from SOASE for this investigation, referred to as the QL_Train dataset. This dataset's design is broken down in Figure 16. Each data point corresponds to a snapshot taken every second from SOASE's buffers which aggregated the previous ten seconds of data (from the time of the snapshot). Twenty-four different mixtures of Bank, Ecommerce and Support clients are included in the data, which makes for 250 data points of each and a total data set size of 6000 data points. These mixtures were chosen to be well spread out on the ternary diagram as is illustrated in Figure 17.

The breakdown of the client sessions for the benchmark runs are presented in Figure 16 as well as the transformation to Bank, Ecommerce and Support membership values used for the training process. The group column will be used later for analysis to mark similar runs (e.g. similar or same proportions of clients) to be compared to runs belonging to other groups. This format will also be used for other datasets later in this document.

The dataset was collected by starting the three benchmark clients and waiting until they reached their statistic collecting period. In this period, the timestamps were watched carefully and the datapoints corresponding to the 250 point window after all three clients were running full capacity. This is illustrated in Figure 18.

Run	Bank Clients	Ecom Clients	Support Clients	Bank Fuzz	Ecom Fuzz	Supp Fuzz	Group
1	68	66	66	0.34	0.33	0.33	1
2	66	68	66	0.33	0.34	0.33	1
3	66	66	68	0.33	0.33	0.34	1
4	80	80	40	0.4	0.4	0.2	2
5	80	40	80	0.4	0.2	0.4	2
6	40	80	80	0.2	0.4	0.4	2
7	120	40	40	0.6	0.2	0.2	3
8	40	120	40	0.2	0.6	0.2	3
9	40	40	120	0.2	0.2	0.6	3
10	100	100	0	0.5	0.5	0	4
11	100	0	100	0.5	0	0.5	4
12	0	100	100	0	0.5	0.5	4
13	150	50	0	0.75	0.25	0	5
14	50	150	0	0.25	0.75	0	5
15	50	0	150	0.25	0	0.75	5
16	150	0	50	0.75	0	0.25	5
17	0	150	50	0	0.75	0.25	5
18	0	50	150	0	0.25	0.75	5
19	160	20	20	0.8	0.1	0.1	6
20	20	160	20	0.1	0.8	0.1	6
21	20	20	160	0.1	0.1	0.8	6
22	200	0	0	1.0	0	0	7
23	0	200	0	0	1.0	0	7
24	0	0	200	0	0	1.0	7

Figure 16: Table of QL_Train Dataset Composition

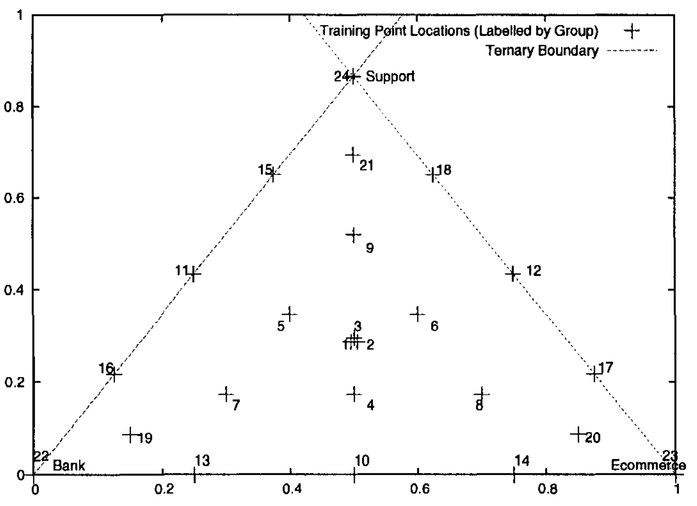


Figure 17: Ternary Diagram of QL_Train Dataset Composition by Run

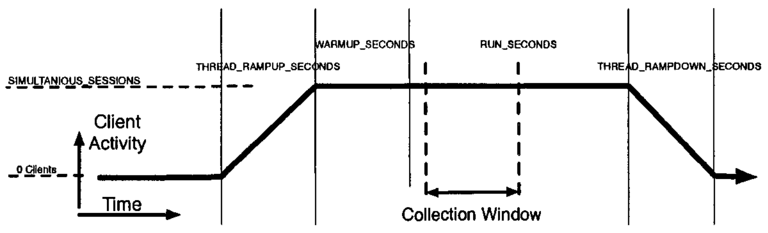


Figure 18: Collection Window in Benchmark for Datasets

6.1.5 Exploration

For addressing the problem of qualitative classification, this research takes the approach of dividing the output space and focusing on segments of that space with separate models. This is a 'divide and conquer' strategy where the three-value output domain is split up and a separate neuro-fuzzy network is trained to handle each.

The motive for this choice that the search space during the evolutionary phase for each of the models will be reduced which will result in a faster converging solution.

A second approach to the training of the QL model was an intimate part of the later developments of this research. This approach utilized a single unified neuro-fuzzy model to provide the three necessary outputs. The intention was to compare and contrast the two approaches and select the superior approach. The findings of this phase was that a larger network was able to perform adequately when compared to the divided smaller networks, but required roughly an order of magnitude more CPU time to train. Even with this much larger training time, the performance could not quite match or exceed the level that was obtained rather quickly from the divided approach. Also, no identifiable benefit of the larger network, such as efficiency of design, could be found in the investigation. The second approach was decided to be impractical and left out of this document for conciseness.

The specifics of the training process and results and statistics pertaining to its validation will be examined in greater depth in Section 7.1.

6.2 QT Model Investigation

6.2.1 Purpose

For the QT model, the goal is to qualitatively differentiate between system activity. This is a useful for informing an autonomic agent which type of activity is currently being experienced assuming a known level of client activity. For the agent to make a decision that results in action, more information is likely needed. In a situation where the system load is well below the capacity, the autonomic agent has less motivation to make any adjustments to the system state. If the system is reaching capacity, the agent will then likely start to consider some sort of action. If the system is past capacity, then action is almost certainly necessary. Conversely, in a network environment with multiple hosts, if a system is detected as being not busy and other systems on the network are saturated, an autonomic agent could take the knowledge that a system has extra capacity to spare to make an action which balances the workload.

Returning to the conversational analogy of computer intelligence, assuming we have addressed the first question posed for the development of the QL model, the most useful extension that we can make is the followup "How busy are you right now?"

The purpose of this investigation is to determine whether a model can be constructed which can classify the currently running load on the system on a single axis of system *activity level independent of the composition of the load.*

By building a model to answer this question, in conjunction with a facility to answer to the question in the QL model investigation, we can obtain a cross-section of the system state that is quite useful for autonomic decision making. The neuro-fuzzy approach applies particularly well to this problem since the answer domain falls into the three categories as described above. A model which provides three outputs corresponding to the fuzzy values which describe the load as 'small', 'nearly saturated' and 'over saturated' is quite descriptive and interpretable. For purposes of evaluation and discussion here, defuzzification is applied on all answers. This greatly cuts the complexity of the analysis of the validity of the real-world results, which takes priority in this work.

6.2.2 Design

Since the type and composition of the load is not the subject of study and furthermore the model is expected to function under a variety of loads, there will have to be a reasonable cross-section of these values developed as a standard of load diversity. The variable parameter is the client load inflicted on the system.

Another difficulty is the criteria of quantitative comparison. The only reasonable measure of quantity of load on our system is the SIMULTANIOUS_SESSIONS parameter from the benchmarks. This is the same unit as SPEC defines to be neutral and unbiased and is the obvious choice.

The goal of for the trained network will be to take a datapoint as input and be able to classify the amount of workload the system is experiencing. The network output will belong to a continuum of values along a pre-defined range of client load activity. To train in this ability, the spectrum has to be defined and represented in the training dataset. Again, the dataset will be collected by running three SPECweb2005 client processes to interact with the server to provide datapoints where the client load is a known quantity to go into the training dataset.

To complete this investigation, as was the case for the QL investigation, a validation dataset will be collected and run through the obtained model. This dataset will be composed of datapoints with different levels of client session activity the conditions that the network was trained under. This will test and validate the obtained model's ability to apply generalized knowledge in a situation which is a simulation of a real-time environment.

6.2.3 Visualization

For the QL model, ternary diagrams are the natural fit because they perfectly represented the output space in a way that matched the problem. Here, it is not as straightforward and the analysis will have to rely, firstly, on histograms. The histograms are capable of illustrating relative differences between categories being scrutinized. Secondly, a confusion matrix is used to illustrate the location of predictions relative to the scale established by the constituent points. This solution, while not as graphically interesting and descriptively powerful as a ternary diagram, is the right choice for the task at hand.

6.2.4 Training Data Set

As was mentioned above, the dataset must be representative of a spectrum of client load that the model is expected to classify under. This implies that the lower and upper bound of the spectrum. Using the data of Figure 15 on page 90 to justify, a good choice for the upper bound of the system activity is 300 sessions. This value is well past the point where a system administrator would consider overloaded and a reasonable range of everyday activity would likely be no greater than 250. The lower bound of the spectrum is chosen to be zero, naturally.

For this part of the research, SOASE was used to collect data at various levels of client load. 9600 data points compose what will be referred to as the QT_Train dataset. The design is displayed in tabular form in Figures 19 and 20.

Run	Cb	Ce	Cs	Sessions	Training Value	Group
1	6	8	11	25	0.083333	1
2	3	2	20	25	0.083333	1
3	10	12	3	25	0.083333	1
4	4	11	10	25	0.083333	1
5	3	29	18	50	0.166666	2
6	5	9	36	50	0.166666	2
7	2	35	13	50	0.166666	2
8	23	23	4	50	0.166666	2
9	38	33	4	75	0.250000	3
10	38	17	20	75	0.250000	3
11	30	13	32	75	0.250000	3
12	59	0	16	75	0.250000	3
13	0	4	96	100	0.333333	4
14	38	30	32	100	0.333333	4
15	41	53	6	100	0.333333	4
16	17	28	55	100	0.333333	4
17	20	103	2	125	0.416667	5
18	34	19	72	125	0.416667	5
19	2	26	97	125	0.416667	5
20	7	36	82	125	0.416667	5
21	19	97	34	150	0.500000	6
22	48	63	39	150	0.500000	6
23	67	60	23	150	0.500000	6
24	69	7	74	150	0.500000	6

Figure 19: Table of QT_Train Dataset Composition - Part 1

Run	Bank Sessions	Ecom Sessions	Supp Sessions	Total Sessions	Training Value	Group
25	122	39	14	175	0.583333	7
26	34	50	91	175	0.583333	7
27	60	95	20	175	0.583333	7
28	41	126	8	175	0.583333	7
29	99	65	36	200	0.666667	8
30	122	6	72	200	0.666667	8
31	6	72	122	200	0.666667	8
32	54	116	30	200	0.666667	8
33	135	77	13	225	0.750000	9
34	179	0	46	225	0.750000	9
35	169	52	4	225	0.750000	9
36	122	54	49	225	0.750000	9
37	197	53	0	250	0.833333	10
38	79	32	139	250	0.833333	10
39	28	138	84	250	0.833333	10
40	78	82	90	250	0.833333	10
41	71	98	106	275	0.916667	11
42	64	40	171	275	0.916667	11
43	97	95	83	275	0.916667	11
44	121	107	47	275	0.916667	11
45	6	196	98	300	1.000000	12
46	35	73	192	300	1.000000	12
47	139	133	28	300	1.000000	12
48	74	134	92	300	1.000000	12

Figure 20: Table of QT_Train Dataset Composition - Part 2

This dataset is designed to isolate the factor that we wish to separate out. In this case, the factor is the client load. The data is made up of forty-eight different benchmark runs (each adding 200 data points to the dataset) which is split into twelve groups, each composed of four runs. Each of these groups is selected to represent a different amount of client load or SIMULTANIOUS_SESSIONS directed at the server and each of the runs has a different load composition.

To isolate the client load, a problem arises from the nature of the SPECweb2005 benchmark which requires a composition of the client sessions for Banking, Ecommerce and Support be decided ahead of time. To truly separate out the targeted factor, the composition of the load would be randomly variable. This would eliminate any sort of assumptions about the load composition trained into the model. To minimize any such bias that could occur from the load composition, values of composition were randomly chosen. This was done by randomly generating a pair of floating-point numbers between 0.0 and 1.0 and plotting it using one value as an x-coordinate and the other as a y-coordinate. If this point is determined to be within the bounds of a ternary diagram plotted in the same space, the point is accepted. This process was repeated until enough points inside the ternary diagram were generated for the amount of data collection trials. This ensured that there was no human-influenced bias in the load composition which could be raised as an objection to the results. The locations of the randomly generated load compositions on a ternary diagram is further illustrated in Figure 21.

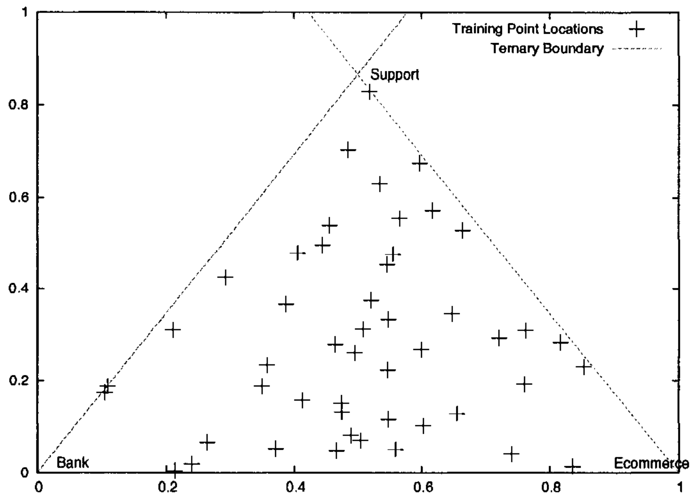


Figure 21: Ternary Diagram of QT_Train Dataset Random Composition

For each level of client activity, four benchmark runs are used to collect data, each with a different client session ratio randomly picked as described above. This is short of optimal. Ideally, a dataset which covers the entire spectrum with many variations of each at every point on the spectrum, but the great amount of time required to collect the dataset imposes this restriction. Twelve different levels of client load was deemed an acceptable amount in the trade-off for effort spent collecting data versus the quality of model. If the results of the validation are good enough, this can hopefully be overlooked. A production system implementation would definitely need to examine and address this shortcoming likely by collecting a much larger training set.

6.2.5 Exploration

Much like the QL model, a single approach is considered for building a model to meet the task at hand. This approach is a single fuzzy neural network which takes the same input stream as the previously discussed networks. The output of this network is a single crisp value which indicates the model's answer to the level of client activity. Behind this crisp value is the three outputs of the network which are the fuzzy representation of the client load. Because this prediction model is much less complex, spending effort on two different approaches is not necessary. Again, as was the case in the QL investigation, a 10-fold protocol will be used to train ten different models to provide a basis for generalization. In Section 7.2 the abilities of the trained model will be explored.

6.3 Confidence Monitor Investigation

6.3.1 Purpose

With the QL model, for purposes of isolating the element of the load composition, the factor of the quantity of load that the system is currently experiencing was lost. Training data for the QL model was collected at a client load of 200 SIMULTANIOUS_SESSIONS and resource constraints prevented investigation into the problem using a different load. Under the assumption that a model can be trained that will accurately predict the composition of the load at 200 SIMULTANIOUS_SESSIONS, it would be unreasonable to expect that the same model can then operate with the same accuracy at 100 or 300 sessions because it is outside of the parameters that it was trained for.

It is conceivable, with more resources, that the procedure for the training of the QL model could be repeated at multiple levels of SIMULTANIOUS_SESSIONS. That would result in multiple different trained networks, each adept within a certain range of parameters of the client load. The QT also provides the system with a sense of what the current activity is, so the knowledge could be utilized to select the right model out of many to give the most accurate prediction. Unfortunately, the time and resources are not available for this project. Instead, a indicator will be constructed which informs the autonomic decision maker on the confidence of the results from the model which was trained at 200 SIMULTANIOUS_SESSIONS in light of the knowledge of the likely amount of load sessions that the server is currently experiencing.

The purpose of this investigation is to develop a heuristic which can indicate the confidence of a prediction made by a qualitative classification model based on knowledge from the quantitative predictive model.

6.3.2 Design

This investigation, by design, benefits from the work done in the previous two investigations. By using the trained QL model to perform classification on data that was collected at different client loads, the aptitude of the model for performing under those circumstances can be predicted. The results from that evaluation will be examined and a simple heuristic will be designed to segment off the range of client load which produces confident results from the range that does not.

This investigation will take a slightly different course than the previous two. The heuristic will be intelligently designed rather than evolved and trained by a machine learning process. To inform the process for the design of the heuristic, the behavior of the models has to be examined in-depth to ensure an effective heuristic is found. This analysis will be performed by running data through the model and examining the behavior through visualization and numeric methods.

The format of the heuristic will be of the 'If-Then' type. More specifically, *"IF <condition> THEN Confident ELSE Not Confident."* Since the heuristic operates off of the QT model's prediction, the condition will be a range of values. If the QT prediction is inside the range then the conditions are met. If it is outside, they are not.

This heuristic is the analog of a neuro-fuzzy model in the previous two investigations. It is a container of obtained knowledge and must then be validated following the same format of the previous two investigations by using independent data to test the aptitude in a real-time setting.

6.3.3 Visualization

In this investigation, the visualization of the data has a special purpose. The visualization is necessary for informing the design of the heuristic rather than just illustrating the performance of models.

To design an effective heuristic, an in-depth exploration of the data is necessary. Using histograms, as was done in the previous investigations, is a simple way of doing providing this. However, it does not provide enough accuracy to get down to the necessary detail. Histograms will be used for an initial overview of the data, but for more in-depth visualization a more sophisticated plot will be used.

The task of designing the heuristic is to determine the upper and lower bound for the QT prediction for the result to be considered confident. All possible heuristics will be evaluated and compared. The best tool for this job is a 3D plot. The upper and lower bound of the heuristic lay on the two horizontal axis. The performance of the heuristic using those parameters will be plotted on the vertical axis. Visualizing the resulting surface can then display the most adept set of upper and lower bounds by the minimas or maximas (depending on the performance measure) of the surface.

6.3.4 Training Data Set

For this investigation to be performed, a dataset which meets a few requirements must be obtained. First, the dataset must span many incremental levels of client activity. Second, the dataset must have several different proportions of load composition at each level. Third, to reduce noise in the error plots, the choices of load composition must be evenly balanced to avoid the case that one group has an unusual proportion of points that are difficult to classify. An ideal dataset would require an extremely large amount of collection runs of the SPECweb2005 benchmarks in order to satisfy these three criteria. Again, this exceeds the resources available to this project, so a compromise must be made.

Luckily, the QT_Train dataset collected for the QT investigation (Figures 19, 20 and 21) does a reasonable job of satisfying these three criteria. The QT_Train dataset was not used in the QL investigation and therefore it is completely independent of the data used to train the QL model. By design, it covers the spectrum of client load from very little, to past the capacity of the server system and is intentionally not biased by the load composition.

It must be noted that for each level of client activity represented in the QT_Train dataset, there are only four random proportions of client load ($n = 4$). To eliminate noise in the data, the ideal case would be to have a large number of randomly selected proportions ($n \gg 4$). Unfortunately, this is impractical and it must be accepted that there will likely be a visible amount of noise in the results and that must be taken into consideration for the evaluation.

The design of the heuristic also involves the use of the QT model which, unfortunately, the QT_Train dataset cannot be considered totally independent of. This presents a methodological problem which can be mitigated by careful evaluation of the dataset. This mitigating step is a result of the fact that the models were trained in a 10-fold cross-validation manner. For the model of each fold, there is a training partition which is independent data which did not contribute to the particular model's training. When evaluating the QT_Train dataset, it must be ensured that the QT model from the correct fold is used, or else the result cannot be considered legitimate. There will be more discussion of this problem in Section 7.3.2.2.

6.3.5 Exploration

Unlike the QL and QT investigations, there is no component here in which a neuro-fuzzy network is trained. Performing this investigation is a matter of taking the obtained QL model, running the QT_Train dataset through and examining the results to design a heuristic. In Section 7.3, the results which are used for designing the heuristic, the rationale behind the heuristic and the validation of the heuristic will be displayed and discussed.

7 Results and Statistics

7.1 QL Model

7.1.1 Overview

The this approach, as discussed in Section 6.1.5, is that of utilizing three specialized fuzzy neural networks to segment the classification of the three categories, Banking, Ecommerce and Support to each model. The same input datastream is used to train three separate models. Each has the specific responsibility of providing a rating for the membership of the current system workload to the class corresponding to the models. These three models work in tandem as illustrated in Figure 22 to classify each datapoint with the three output values.

7.1.2 Training

Ten sets of networks to make up a model, one for each fold of the training data, were developed using the parameters displayed in Figure 23. The training process for each individual network took roughly 12 hours on a single 3.4Ghz Pentium D (Presler) core. Fortunately, an entire lab of these machines were available to train the required 30 networks. The great amount of time required left very little opportunity for trial and error to tune the training parameters. The parameters were chosen based on the results for a single fold of the data and then used to train the 10 folds.

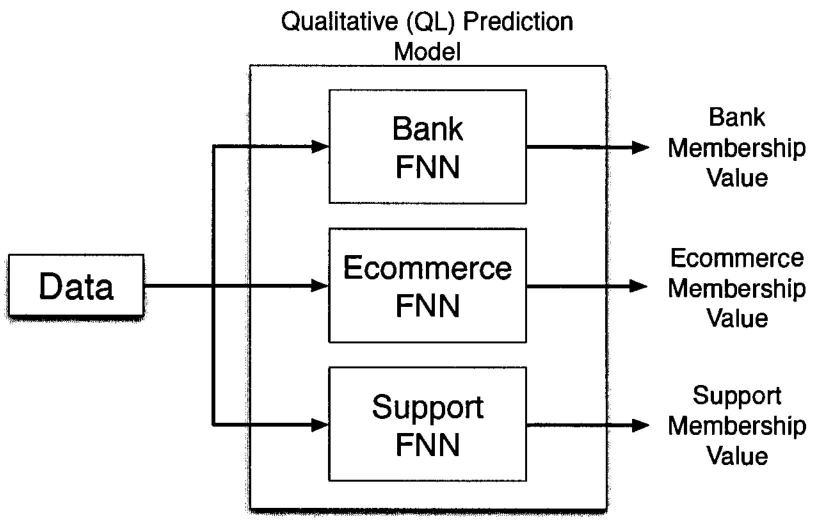
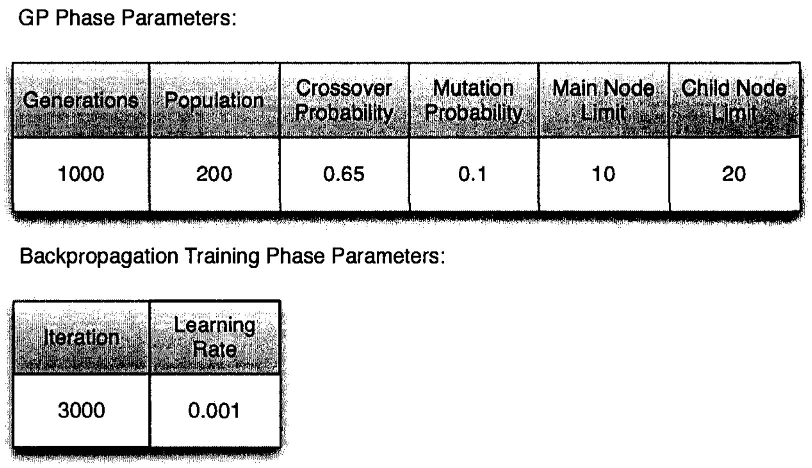


Figure 22: Structural Diagram of QL Model



GP Phase Parameters:

Generations	Population	Crossover Probability	Mutation Probability	Main Node Limit	Child Node Limit
1000	200	0.65	0.1	10	20

Backpropagation Training Phase Parameters:

Iteration	Learning Rate
3000	0.001

Figure 23: QL Model Training Parameters

7.1.3 Performance

As outlined in Section 5.4.2, for proper scientific scrutiny, all evaluation must be done on an independent dataset which was not involved in the training phase of the model. For each model, the test partition for its particular fold is used. The combination of the testing partition results from the ten models, which encompass all 6000 points of the QL_Train dataset are presented in this section. While the testing partitions of the QL_Train dataset were used for deciding on the stopping criteria for the second phase of the backpropagation training, they were not involved in the training. This could be used as a minor objection to the results presented here. For this reason, a second evaluation and analysis on the QL_Valid dataset, which is wholly independent both in the time of its collection and in load composition, is conducted later in Section 7.1.4.1.

Figure 24 presents the overall performance obtained from feeding the models the datapoints of the test partition of the QL_Train dataset. Given the sets of outputs from the network (b, e and s), the sets of actual quantitative training values for each point from the dataset (B, E and S) and the amount of points in the dataset (N), the mean absolute error for each of the three workloads, $\overline{b}$, $\overline{e}$ and $\overline{s}$ are calculated as follows:

$$\begin{aligned}\overline{b} &= \frac{1}{N}\sum_{i=1}^{N}|b_i - B_i| \\ \overline{e} &= \frac{1}{N}\sum_{i=1}^{N}|e_i - E_i| \\ \overline{s} &= \frac{1}{N}\sum_{i=1}^{N}|s_i - S_i|\end{aligned}$$

The standard deviations, σ_b, σ_e and σ_s are further calculated as:

$$\begin{aligned}
\sigma_b &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(|b_i - B_i| - \overline{b})^2} \\
\sigma_e &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(|e_i - E_i| - \overline{e})^2} \\
\sigma_s &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(|s_i - S_i| - \overline{s})^2}
\end{aligned}$$

		Bank	Ecommerce	Support
Model 1	Mean Abs. Error	0.114389	0.167150	0.134438
Model 2	Mean Abs. Error	0.106798	0.161450	0.104826
Model 3	Mean Abs. Error	0.113819	0.156488	0.104166
Model 4	Mean Abs. Error	0.126136	0.176010	0.116362
Model 5	Mean Abs. Error	0.115818	0.161007	0.111098
Model 6	Mean Abs. Error	0.102275	0.147970	0.106218
Model 7	Mean Abs. Error	0.126780	0.166722	0.114566
Model 8	Mean Abs. Error	0.110348	0.161733	0.105305
Model 9	Mean Abs. Error	0.104528	0.161848	0.096781
Model 10	Mean Abs. Error	0.117972	0.150830	0.116329
Total	Mean Abs. Error	0.113886	0.161121	[illegible]
Model 1	Std. Deviation	0.084798	0.115396	0.092952
Model 2	Std. Deviation	0.082762	0.116818	0.073391
Model 3	Std. Deviation	0.084640	0.111519	0.074501
Model 4	Std. Deviation	0.094624	0.120576	0.084563
Model 5	Std. Deviation	0.080567	0.111246	0.078744
Model 6	Std. Deviation	0.076584	0.107912	0.080278
Model 7	Std. Deviation	0.090297	0.114343	0.084674
Model 8	Std. Deviation	0.083644	0.108593	0.080568
Model 9	Std. Deviation	0.080130	0.107671	0.069700
Model 10	Std. Deviation	0.088308	0.108628	0.083811
Total	Std. Deviation	0.085160	[illegible]	[illegible]

Figure 24: QL Model/QL_Train Dataset Overall Performance

By comparing the results across the ten folds, the results are very similar. Particular folds have slightly easier time predicting some categories, but that can be considered a random variation. The total score averaged is more representative of the model's ability. In general, Ecommerce is somewhat more difficult for the model to accurately classify. The standard deviation is also slightly higher for the Ecommerce class. These results here will be compared in light of the later results, but it can be taken away that this represents quite reasonable performance of the model.

Figure 25 plots the same performance of the models broken down by the group of the datapoints. The reference to the points which make up each group is listed in Figure 16. A general trend that is immediately apparent when examining the breakdown is that there is much greater accuracy for points in the center of the ternary diagram (Groups 1, 2 and 3). Points on the corners and outer extremes tend to have more inaccuracy. It is observed that this model has a bias to produce results which tend towards the middle region. The reason for this is likely a useful adaptation generated in the evolutionary phase. In the QL_Train training set, there is a quite evenly balance proportion of interior points and exterior points, so it can be concluded that this isn't a flaw in the design of the dataset. This can be most easily attributed to the nature of the problem domain where it is beneficial for the model to default to moderate predictions when faced with ambiguity. Erring towards the moderate in this manner produces less net error which was selected for in the evolutionary process.

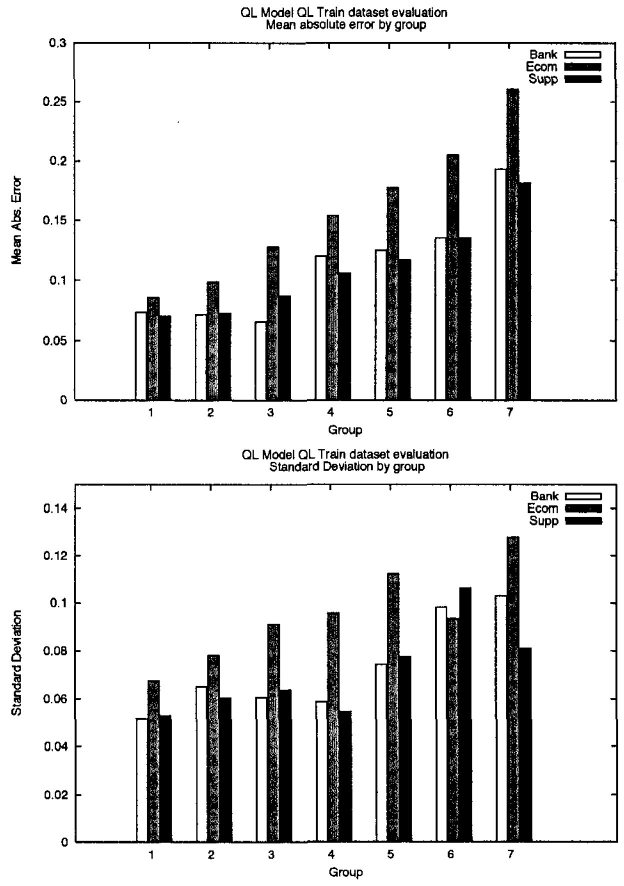


Figure 25: QL Model/QL_Train Dataset Mean Absolute Error and Standard Deviation by Group

The standard deviation also correlates with the mean absolute error, broken down by group and also by the category. However, there is less spread between the interior and exterior points. Both measurements appear to be in correlation to the intrinsic difficulty of the particular group to be classified. One anomaly is observable for the standard deviation in Group 6 where the Ecommerce value is not as pronounced and in line with the other categories. This is a phenomenon which has no obvious and immediate explanation, since the points of Group 6 lie in the interior of the ternary diagram. By drilling into the data, it was found that this trend for Group 6 holds across all folds, so its cause is likely an intrinsic characteristic of the data.

Figure 26 is a set of plots of predicted points on multiple ternary diagrams. The location of the predicted point on the diagram is calculated by taking the three outputs of the model and normalizing them to sum to 1.0 to meet the criteria for plotting them on the ternary diagram. The normalization is necessary because the output will not normally evaluate to sum to exactly one. The reason is the same as was discussed earlier in Section 5.4.5 with the output rarely having a perfect alignment. The plots are split up by the group of points in the QL_Train dataset (reference Figures 16 and 17 on pages 93 and 94). Each group is also further divided and marked by the individual benchmark run. Only a random sampling of available results are plotted to reduce visual congestion, as the purpose of this visualization is to identify trends. One thing to note is that Group 5, which is twice the size of the other groups (having 6 collection points rather than 3), is split into two separate plots for visual simplicity.

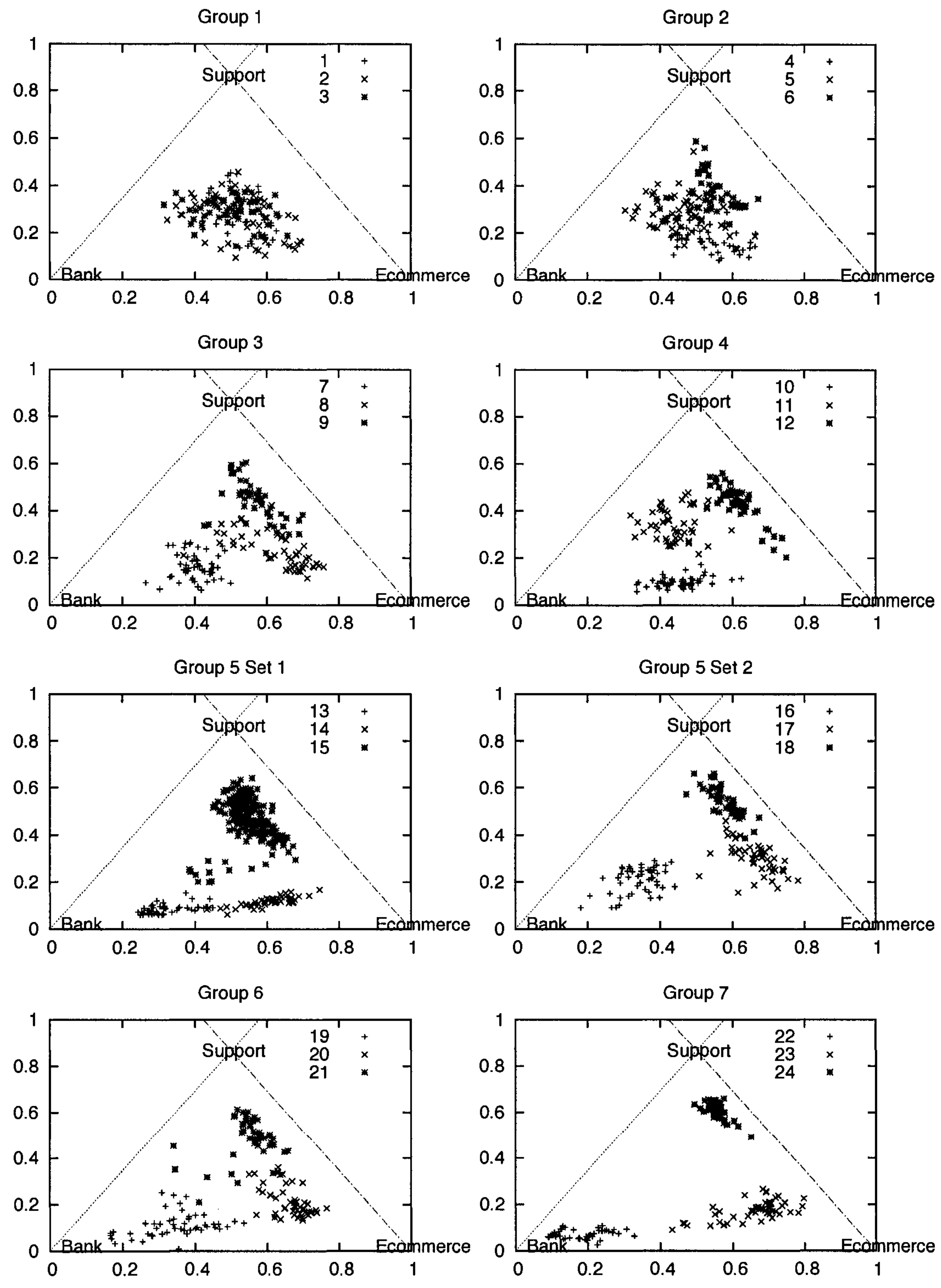


Figure 26: QL Model/QL_Train Dataset Ternary Diagrams by Group

The greater difficulty in predicting the Ecommerce component of the client load is extremely apparent in this visualization. The clusters of points are elongated in the direction which points towards the Ecommerce corner of the diagram. Also, the points which are composed of primarily Ecommerce have clusters which are spread much wider. This is in agreement with mean error and standard deviation is much higher in the Ecommerce category as is displayed in the histograms. Also, the previously observed bias towards moderate prediction is clearly visible in these plots. Group 7's points are almost entirely classified in the territory of Group 6. Group 6 points are spread around the territory of Group 3 and other groups show a clear inward bias.

7.1.4 Validation

The best test of the validity of a constructed model for a real-world task is how well it performs in a different environment from which it has been trained. The attempt here is to measure the gained intelligence in the model for dealing with data it has not seen before. To be useful, the model must be able to apply principals picked up in the training process to intelligently classify input of arbitrary workloads. A comprehensive evaluation of this ability must include both an objective and subjective evaluation. A subjective evaluation, while may be considered as inexact and perhaps unscientific, is included because objective measurement does not give a complete picture. The categories of classification (Bank, Ecommerce and Support) were chosen to be as objective as possible backed up by the benchmark design, but still, there is a subjective element since the three types are still relatively arbitrary orientation posts to base all measurements against. The subjective evaluation done here gives an indication of the aptitude of the method of using SPECweb2005 as orientation posts when considering an arbitrary workload that is different in nature from serving web pages.

7.1.4.1 Objective The models were trained to pick up knowledge from the QL_Train dataset (Figure 16 and 17 on pages 93 and 94) so, the first test is the ability for the models to classify a workload made up of differing proportions of the SPECweb2005 benchmarks. The QL_Valid dataset of 2400 points was collected in the same way as the QL_Train dataset, but the workloads were chosen to be different in their composition. The design of the QL_Valid dataset is shown in tabular form in Figure 27 and plotted on the ternary diagram in Figure 28. A view of Figure 17 and 28 superimposed would show that the location of the points in the 2D space of the ternary diagram do not overlap which meets the requirements of independence among the two datasets. By testing the model's accuracy on the QL_Valid dataset, we can see how the intelligence of the model holds up in the regions of the ternary diagram which are not covered by the set that was used to train the model.

Run	Bank Clients	Ecom Clients	Supp Clients	Bank Fuzz	Ecom Fuzz	Supp Fuzz	Group
1	60	70	70	0.30	0.35	0.35	1
2	70	60	70	0.35	0.30	0.35	1
3	70	70	60	0.35	0.35	0.30	1
4	75	75	50	0.375	0.375	0.25	2
5	75	50	75	0.375	0.25	0.375	2
6	50	75	75	0.25	0.375	0.375	2
7	100	50	50	0.50	0.25	0.25	3
8	50	100	50	0.25	0.50	0.25	3
9	50	50	100	0.25	0.25	0.50	3
10	140	50	10	0.70	0.25	0.05	4
11	140	10	50	0.70	0.05	0.25	4
12	50	140	10	0.25	0.70	0.05	4
13	50	10	140	0.25	0.05	0.70	4
14	10	140	50	0.05	0.70	0.25	4
15	10	50	140	0.05	0.25	0.70	4
16	180	10	10	0.90	0.05	0.05	5
17	10	180	10	0.05	0.90	0.05	5
18	10	10	180	0.05	0.05	0.90	5

Figure 27: Table of QL_Valid Dataset Composition

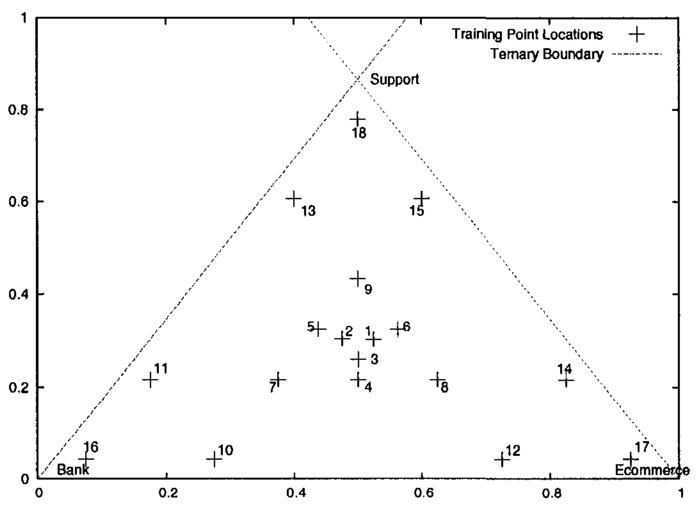
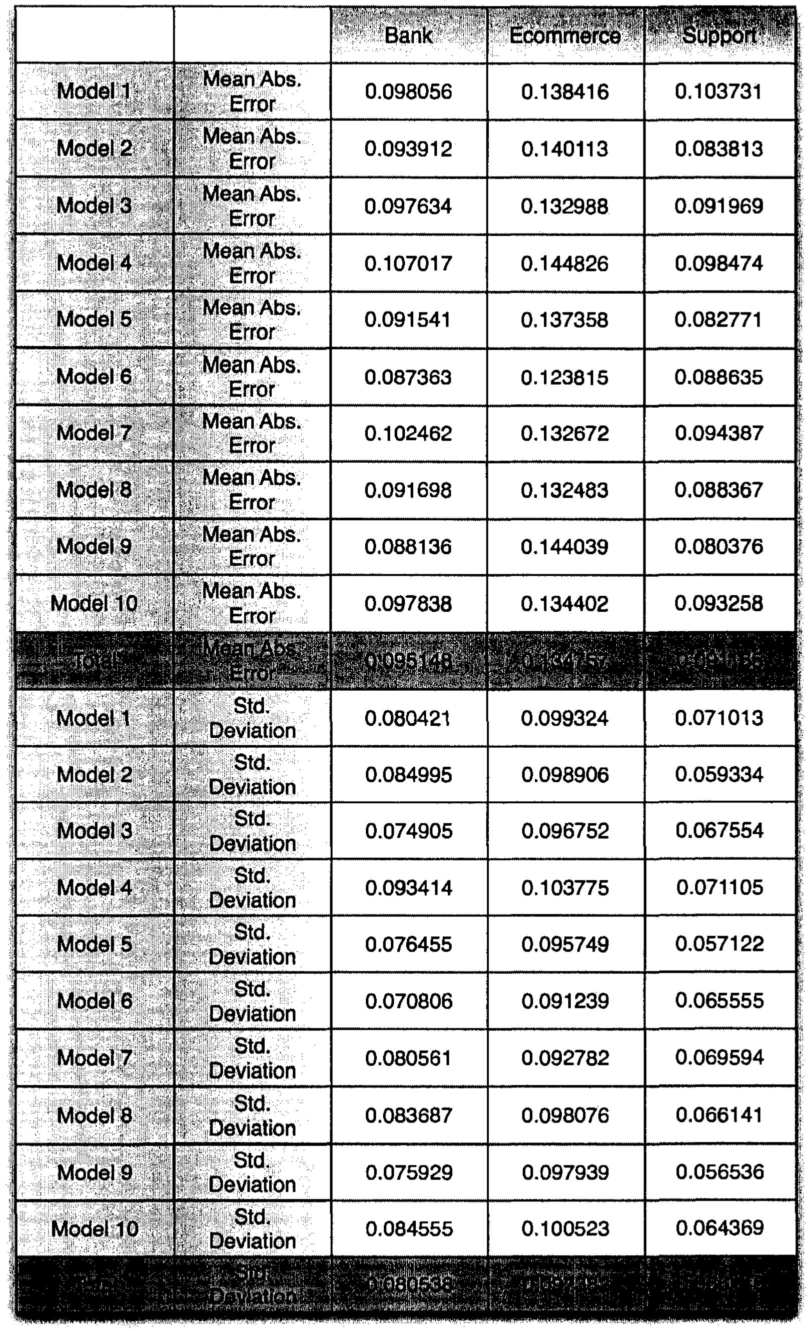


Figure 28: Ternary Diagram of QL_Valid Dataset Composition by Run

The set of models obtained in the training process, when fed the QL_Valid dataset gave the results presented in Figures 29, 30 and 31. Again, the calculations for mean absolute error and standard deviation described in Section 7.1.3 were applied here. Since there is only a single dataset and no notion of associated testing partitions associated with the model of each fold, the 2400 points of this dataset is re-used for each. The QL_Valid dataset is run through the model for each fold, to produce 24000 results which are used to generate the figures and plots in this section.

The most striking observation that can be made from a quick examination is that the results are significantly better than the results with QL_Train's testing partitions. This can be largely explained by the fact that the QL_Valid dataset does not contain points which are as far from the center as Group 7 from the QL_Train dataset which provided the most error in that evaluation. Here, the absence of such points reduce the average error. This centrally-biased trend is also observable here, with Group 5 of the QL_Valid dataset being the furthest from the center and having the greatest mean error and standard deviation.



		Bank	Ecommerce	Support
Model 1	Mean Abs. Error	0.098056	0.138416	0.103731
Model 2	Mean Abs. Error	0.093912	0.140113	0.083813
Model 3	Mean Abs. Error	0.097634	0.132988	0.091969
Model 4	Mean Abs. Error	0.107017	0.144826	0.098474
Model 5	Mean Abs. Error	0.091541	0.137358	0.082771
Model 6	Mean Abs. Error	0.087363	0.123815	0.088635
Model 7	Mean Abs. Error	0.102462	0.132672	0.094387
Model 8	Mean Abs. Error	0.091698	0.132483	0.088367
Model 9	Mean Abs. Error	0.088136	0.144039	0.080376
Model 10	Mean Abs. Error	0.097838	0.134402	0.093258
Total	Mean Abs. Error	0.095148	0.134757	[illegible]
Model 1	Std. Deviation	0.080421	0.099324	0.071013
Model 2	Std. Deviation	0.084995	0.098906	0.059334
Model 3	Std. Deviation	0.074905	0.096752	0.067554
Model 4	Std. Deviation	0.093414	0.103775	0.071105
Model 5	Std. Deviation	0.076455	0.095749	0.057122
Model 6	Std. Deviation	0.070806	0.091239	0.065555
Model 7	Std. Deviation	0.080561	0.092782	0.069594
Model 8	Std. Deviation	0.083687	0.098076	0.066141
Model 9	Std. Deviation	0.075929	0.097939	0.056536
Model 10	Std. Deviation	0.084555	0.100523	0.064369
[illegible]	Std. Deviation	0.080538	[illegible]	[illegible]

Figure 29: QL Model/QL_Valid Dataset Overall Performance

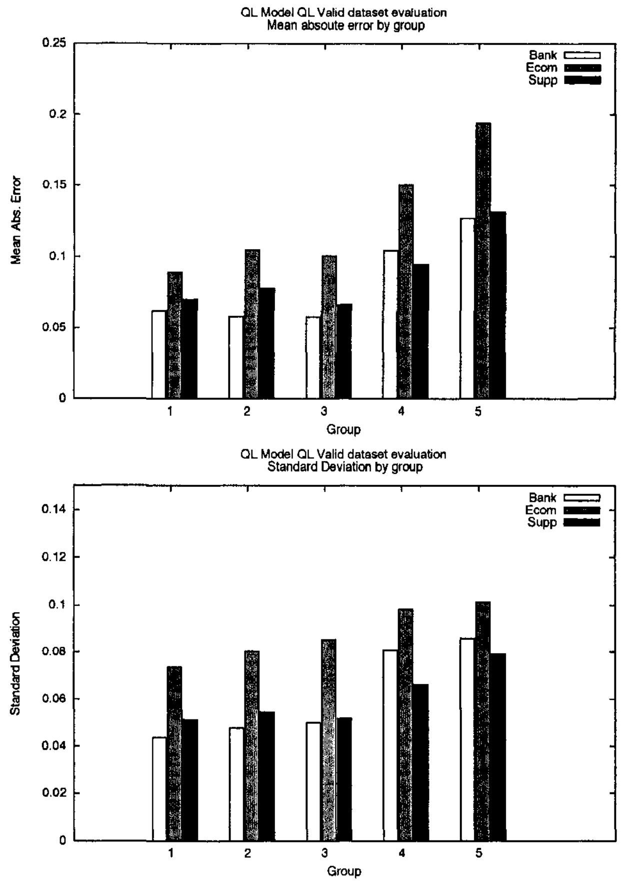


Figure 30: QL Model/QL_Valid Dataset Mean Abs. Error and Standard Deviation by Group

Visible in the ternary plots for this evaluation is the QL_Valid dataset being processed by the model. Group 4 is spread between two different plots to reduce visual congestion. Here the results are quite similar with what was witnessed for the QL_Train dataset and there is nothing new to discuss.

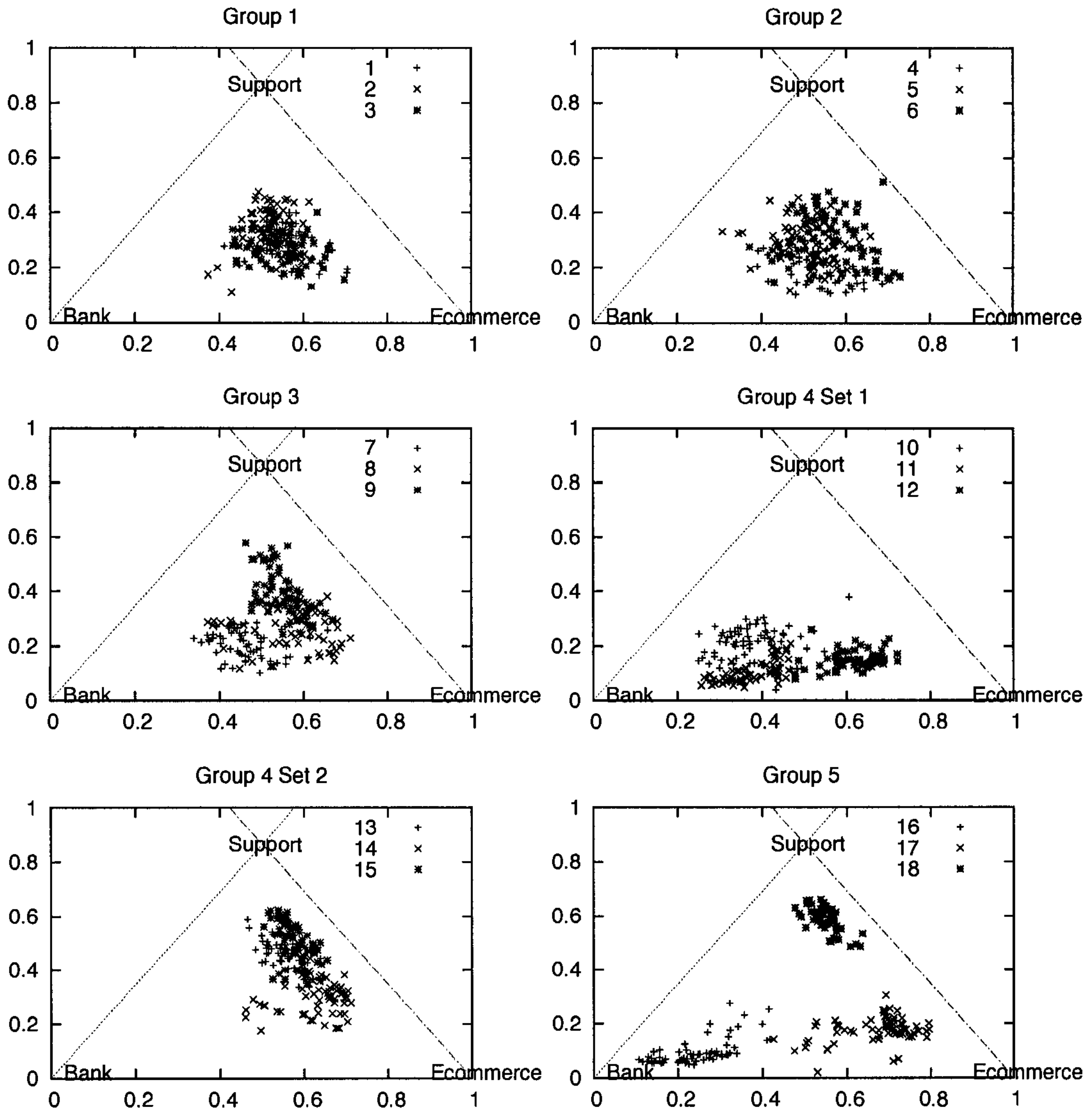


Figure 31: QQL_Valid Ternary Diagrams By Group

7.1.4.2 Subjective

For the subjective evaluation component of this investigation, three workloads were run which are clearly external to the realm of web serving. The objective of this test is to determine whether the knowledge contained inside the trained models can be applied to intelligently classify non-webserving workloads.

To accomplish this, three workloads were designed to be subjectively similar to the three benchmark types. The criteria of on which the model will be evaluated is how the classification appears to match up with pre-conceived ideas of what the results should be. As mentioned earlier in this document, this is arguably unscientific. However, in defense of it, good subjective results can show promise for more elaborate future work in extension of that which is presented here.

The first workload that is tested is that of a *scp* operation to transfer portions of a filesystem over the network to an external machine (labeled 'File Copy'). This is designed to be similar in characteristic to the Support workload which primarily is concerned with accessing a mixture of large and small files off of the hard disk and sending the data over the network.

The second workload is that of neural network training (labeled 'Network Train'). Specifically, this is running instances of the program that was constructed to train the FNNs for the QL and QT models. This involves maximum of CPU usage and periods of hard disk access to access the training data. This is modeled to be somewhat similar to the Bank workload in the dependence on CPU usage. Because the

SunBlade 1000 system is of dual-processor architecture, two simultaneous sessions of this program were run in order to fully saturate the CPU resource as in the case of the Banking workload.

The final workload is a custom-written application to provide a subjectively similar characteristic of the Ecommerce workload (labeled 'Memory Hog'). This simple program, written in Java is designed to consume the entirety of the memory resources of the system. It allocates the majority of the system memory and for each megabyte of allocated space, it spins off an independent thread to make random reads and writes to that particular chunk. By keeping the allocated memory active, it forces the kernel's paging algorithm to manage the pages that have to be brought in and sent out of main memory. This has the effect of saturating and stressing the system's memory subsystem, as well as keeping a large pool of threads constantly running.

Figure 32 presents the ternary diagrams of the points collected from SOASE during the workload's run and run through the QL models. The plot is also broken down by the models of the ten folds to make it possible to generalize. This is needed because random differences exist in each individual model and each will likely classify the subjective workload slightly different than other models with other random differences. The outputs from the network were also, as was the case for the objective evaluation, normalized to fit on the diagram.

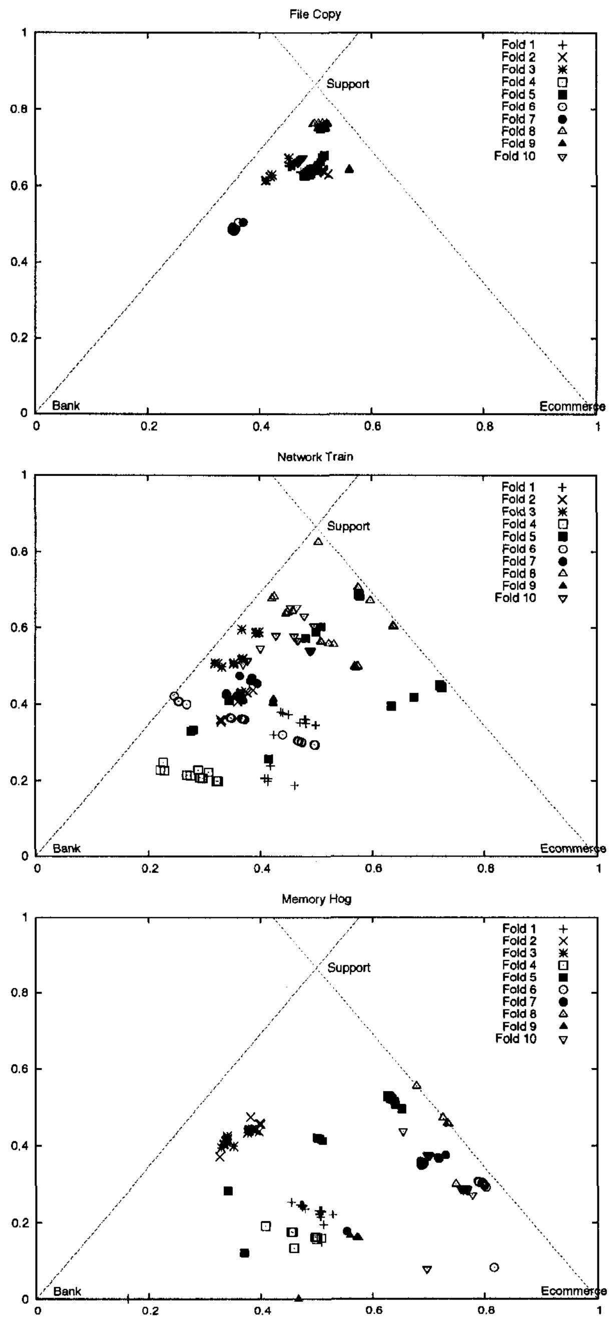


Figure 32: QL Model Subjective Evaluation

As is seen in this visualizations, there are definite indications of the intelligence contained in the model, though it is far from perfect. All three of the workload's classifications are biased towards Support and there is large differences between predictions of the models. Despite this, the file copy workload is uniformly classified heavily as Support which counts as a subjective 'hit' in this case. The 'Memory Hog' workload is classified mainly as Support, but it tends towards Ecommerce for which it was designed. The 'Network Train' workload similarly tends towards the Banking corner.

In the 'Network Train' and 'Memory Hog' workload, there is a noticeable level of disagreement between the models from the ten training folds. This indicates that there is not a clear principal on which the networks apply to make these classifications. Random structural differences between the models of ten folds lead to widely different results and the system activity during these new workloads work to confuse the models as they were not actively selected and trained for this specific system behavior. This could suggest that a model constructed specially for a different workload domain such as this one could pick up on different cross-sections of data to refine such predictions and be more successful at that task. This is brought up again in Section 8.2 as a potential future investigation.

The differences witnessed between the three workloads are the result of the model picking up on characteristics of the datastream which have distinct differences. In this case, it would be unreasonable to expect results that fit exactly with the hypothesis since there are clearly flaws in the design of these workloads. They are clearly not directly similar to the three workloads that they are modeled after and they should be classified differently. One plausible explanation for the accuracy of the 'File Copy' workload and the inaccuracy of the other two is the network activity. Network activity is an essential part of web serving and it is no doubt that the models use the monitoring of the network as a large component of the prediction. The scattered results for 'Memory Hog' and the 'Network Train' workloads are likely due to them not generating any sort of network traffic to provide a basis for prediction.

When evaluating this data, one also has to take into account the possibility that these results appear to fit based on chance and the decisions that went into the classification are not inherently intelligent. This is representative of the methodological flaws of subjective testing and further objective study is the only way to address the questions raised. As a result, the results obtained in this section should be considered very tentative in light of further study.

7.2 QT Model

7.2.1 Overview

As discussed in Section 6.2, this investigation is to test the feasibility of constructing a model which can output, in real time, a prediction of the quantity of load the system is experiencing in terms of client sessions. The model performing the prediction is a FNN which takes the DTrace-derived data from SOASE as input and produces a three fuzzy-number set which defuzzifies to an output. This model shares the same structure as a single one of the three models from the QL model. In terms of complexity, this model's task is simpler than the qualitative prediction, having only a single crisp output parameter as opposed to three. For this reason, the single FNN is expected to be perfectly adequate for the task that it is applied to. Again, the 10-fold cross-validation procedure is used, which produces ten such models for evaluation.

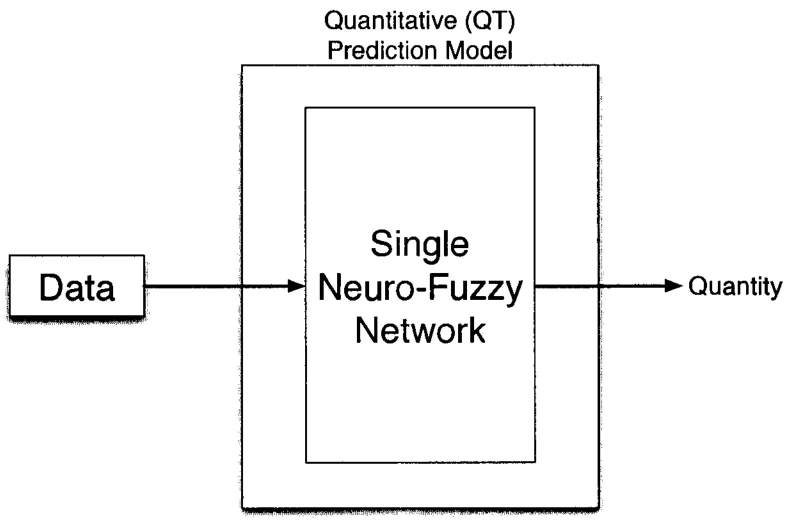


Figure 33: Structural Diagram of QT Model

7.2.2 Training

The parameters used for the training are displayed in Figure 34. Each of the 10 model's training took approximately 18 hours to run on a 3.4Ghz Pentium D core (Presler). As was done for training the QL models, the GP phase was stopped well before the model converged to its best solution. The backpropagation phase was utilized to bring the model to its final convergence. Initially, 3500 iterations of the backpropagation training were applied. It was observed that overtraining was occurring. By reducing the number of iterations back to 2850, the testing partition's evaluation was improved in 8 of the 10 folds.

GP Phase Parameters:

Generations	Population	Crossover Probability	Mutation Probability	Main Node Limit	Child Node Limit
900	300	0.65	0.1	10	20

Backpropagation Training Phase Parameters:

Iteration	Learning Rate
2850	0.001

Figure 34: QL Model Training Parameters

7.2.3 Performance

Following the same pattern as QL model, the first analysis of performance is performed on the testing partition of the QT_Train dataset. It must be noted that the results presented here are in terms of a range of values between0.0 and 1.0 with 0.0 corresponding to 0 clients and 1.0 corresponding to 300 clients. As an example of the conversion, 0.0657 is equal to approximately 20 simultaneous client sessions $(0.0657 * 300)$.

Figure 35 shows the overall mean absolute error and standard deviation. These values are calculated in a very similar way as was done for the evaluation of the QL model. Mean absolute error $(\overline{x})$ of the prediction given the set of quantitative predictions (x), the set of actual quantitative values (X) and the amount of points in the set (N) is calculated as:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N}|x_i - X_i| \tag{12}$$

The standard deviation (σ_x) is calculated as:

$$\sigma_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(|x_i - X_i| - \overline{x})^2} \tag{13}$$

The results of the evaluation on the testing partition are presented in Figures 35, 36 and 36. As can be seen in the group-by-group breakdown (Figure 36), the mean absolute error is greatly hurt by Group 12 which is made up of datapoints from runs of 300 simultaneous client sessions. The extreme inaccuracy of this group drags down the average. The cause is likely the fact that the system is well past the point of client load saturation and system activity can no longer scale to provide a cross-section which is meaningful to the model. This indicates that in the system activity, there is very little that can be used to differentiate during higher client loads. The model has adapted to show bias by classifying these more ambiguous data points belonging to Group 12 as in the territory of slightly lower client activity.

	Mean Abs. Error	Standard Deviation
Model 1	0.069117	0.063237
Model 2	0.064688	0.062937
Model 3	0.084037	0.079399
Model 4	0.070615	0.058412
Model 5	0.058122	0.047810
Model 6	0.058681	0.056630
Model 7	0.061517	0.066424
Model 8	0.058733	0.055337
Model 9	0.064584	0.057666
Model 10	0.066697	0.061923
Total	0.065678	0.061928

Figure 35: QT Model/QT_Train Dataset Overall Performance

As the client loads ramp up through Groups 2 to 8, the error ramps up. This cor-

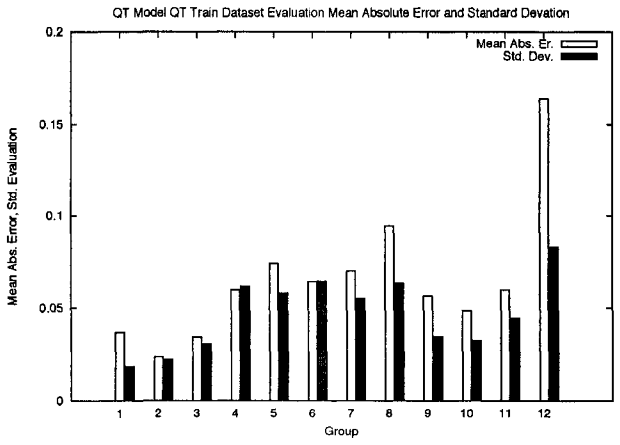


Figure 36: QT Model/QT_Train Dataset Mean Abs. Error and Standard Deviation By Group

responds to a client load between 25 and 200, which is below the saturation point. From this, it seems that the characteristics which are used to make distinctions becomes more ambiguous as the activity increases. Once the system reaches the saturation point, it can be seen that the mean error becomes reduced. The hypothesized cause is that once the system crosses the saturation point, other factors of the system like memory swapping to disk come into play, giving the model additional traction for classification. In Figure 15 on page 90, the performance of the benchmark systems was observed to start dropping off past 200 sessions, which supports this conjecture.

The confusion matrix for the performance on the QT_Train dataset is in Figure 37. Hits for a particular group to populate the matrix were determined by looking at the closest group (in absolute difference) to the predicted quantity. As can be seen the model is quite adept at determining the system load and can make definite distinctions between different quantities of client load. All are relatively near the correct value, so there is large amount of confidence in the predictions made by this model. This is a reflection of the relatively low standard deviation seen in Figures 35 and 36.

Closest Predicted

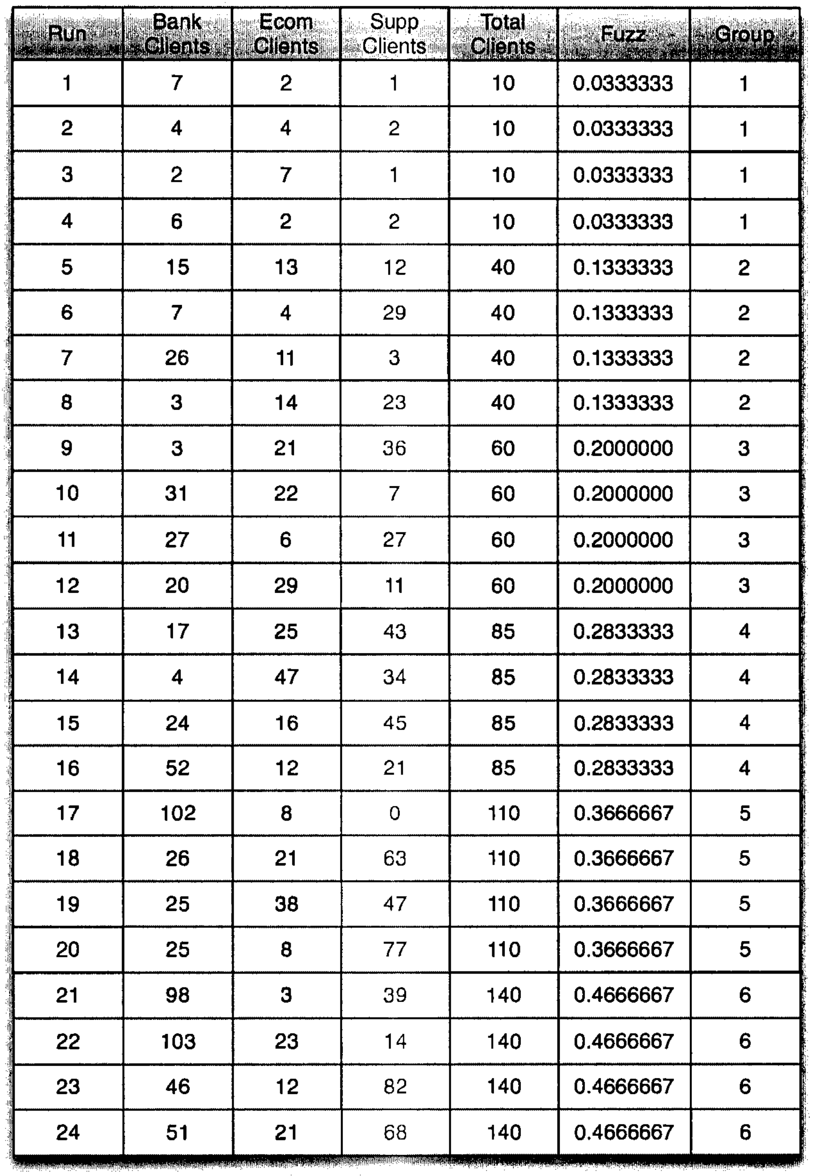
		1	2	3	4	5	6	7	8	9	10	11	12
Actual	1	519	281	0	0	0	0	0	0	0	0	0	0
	2	22	678	98	2	0	0	0	0	0	0	0	0
	3	0	77	558	151	11	3	0	0	0	0	0	0
	4	5	15	151	397	156	51	18	3	3	1	0	0
	5	0	2	12	215	275	178	86	31	1	0	0	0
	6	0	0	0	11	151	382	153	62	27	13	1	0
	7	0	0	0	0	14	190	304	178	92	21	1	0
	8	0	0	0	0	1	36	104	219	222	185	32	1
	9	0	0	0	0	0	0	2	14	322	435	27	0
	10	0	0	0	0	0	1	6	6	172	343	272	0
	11	0	0	0	0	0	0	2	9	46	415	325	3
	12	0	0	0	0	0	5	19	43	114	312	303	4

Figure 37: QT Model/QT_Train Dataset Confusion Matrix by Group

As was hypothesized earlier, it is illustrated in the matrix that the model is unable to differentiate between various large quantities of client load. Groups 8 through 11 have a disproportionally high amount of hits from points that are actually belonging to Group 12. The model seems to be having trouble with Group 12 and it finds the points to be similar to those of lower client load. Discussion of precision and recall is not entirely relevant to the evaluation of the model, since inaccurate prediction that are still close are still very useful for an autonomic agent.

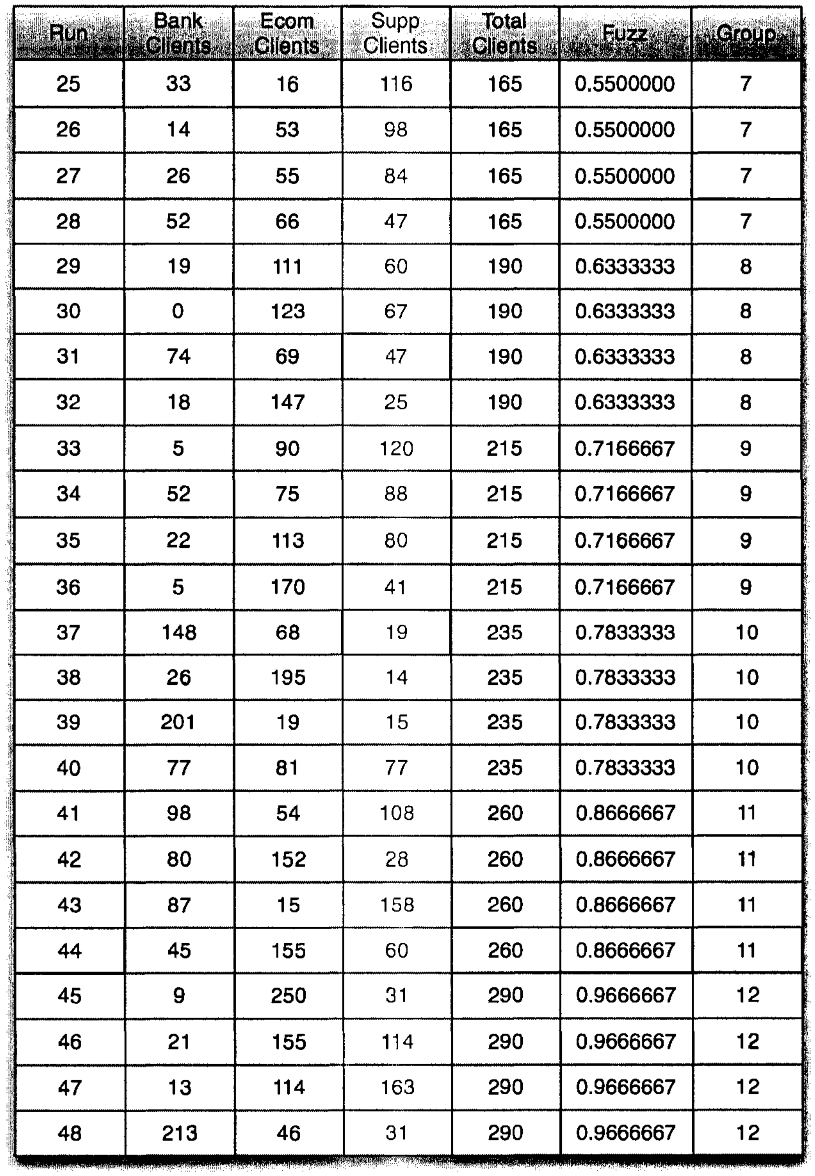
7.2.4 Validation

For objective measurement of the quality of the developed model, the same approach was taken as for the QL model. A completely independent dataset was collected which differs in its choice for load. This dataset, named the QT_Valid dataset, is outlined in tabular form in Figures 38 and 39. It is composed of 200 data points from each of forty-eight benchmark runs which makes for a total of 9600 points. This dataset was designed to have client load levels that are in the gaps of the QT_Train dataset. The composition of the load was chosen to be random as was the case for the QT_Train dataset. Again, the random selection of composition is illustrated in Figure 40.



Run	Bank Clients	Ecom Clients	Supp Clients	Total Clients	Fuzz	Group
1	7	2	1	10	0.0333333	1
2	4	4	2	10	0.0333333	1
3	2	7	1	10	0.0333333	1
4	6	2	2	10	0.0333333	1
5	15	13	12	40	0.1333333	2
6	7	4	29	40	0.1333333	2
7	26	11	3	40	0.1333333	2
8	3	14	23	40	0.1333333	2
9	3	21	36	60	0.2000000	3
10	31	22	7	60	0.2000000	3
11	27	6	27	60	0.2000000	3
12	20	29	11	60	0.2000000	3
13	17	25	43	85	0.2833333	4
14	4	47	34	85	0.2833333	4
15	24	16	45	85	0.2833333	4
16	52	12	21	85	0.2833333	4
17	102	8	0	110	0.3666667	5
18	26	21	63	110	0.3666667	5
19	25	38	47	110	0.3666667	5
20	25	8	77	110	0.3666667	5
21	98	3	39	140	0.4666667	6
22	103	23	14	140	0.4666667	6
23	46	12	82	140	0.4666667	6
24	51	21	68	140	0.4666667	6

Figure 38: Table of QT_Valid Dataset Composition - Part 1



Run	Bank Clients	Ecom Clients	Supp Clients	Total Clients	Fuzz	Group
25	33	16	116	165	0.5500000	7
26	14	53	98	165	0.5500000	7
27	26	55	84	165	0.5500000	7
28	52	66	47	165	0.5500000	7
29	19	111	60	190	0.6333333	8
30	0	123	67	190	0.6333333	8
31	74	69	47	190	0.6333333	8
32	18	147	25	190	0.6333333	8
33	5	90	120	215	0.7166667	9
34	52	75	88	215	0.7166667	9
35	22	113	80	215	0.7166667	9
36	5	170	41	215	0.7166667	9
37	148	68	19	235	0.7833333	10
38	26	195	14	235	0.7833333	10
39	201	19	15	235	0.7833333	10
40	77	81	77	235	0.7833333	10
41	98	54	108	260	0.8666667	11
42	80	152	28	260	0.8666667	11
43	87	15	158	260	0.8666667	11
44	45	155	60	260	0.8666667	11
45	9	250	31	290	0.9666667	12
46	21	155	114	290	0.9666667	12
47	13	114	163	290	0.9666667	12
48	213	46	31	290	0.9666667	12

Figure 39: Table of QT_Valid Dataset Composition - Part 2

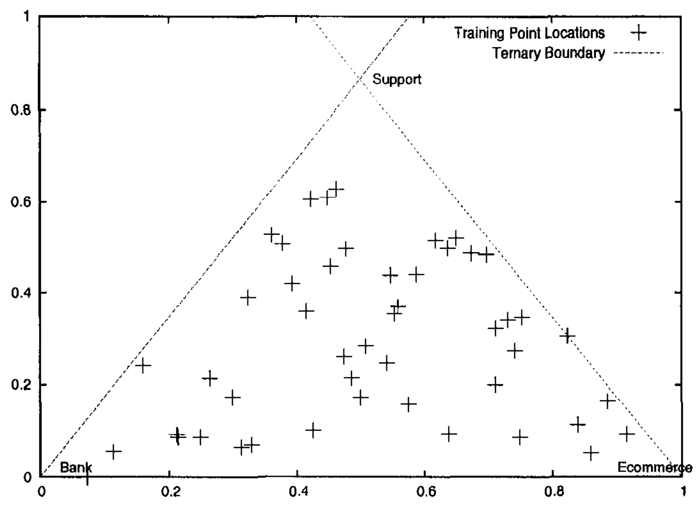
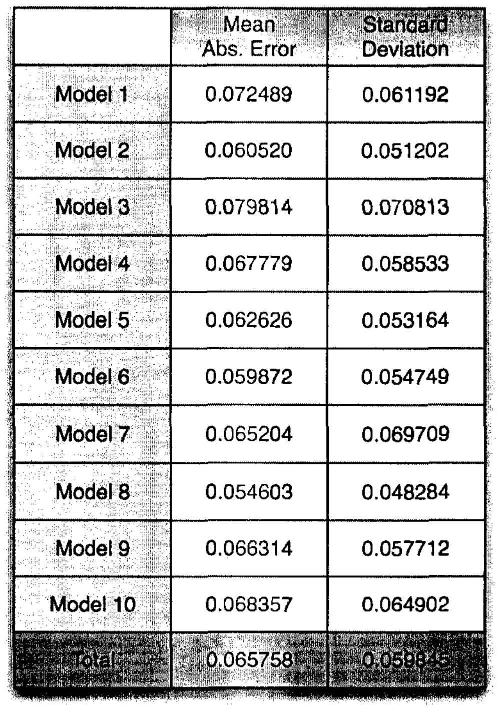


Figure 40: Ternary Diagram of QT_Valid Dataset Random Composition

As was the case in the validation of the QL model using the QL_Valid dataset, for the validation of the ten models (of the ten folds), the QT_Valid dataset is run through each. This makes for 96000 datapoints to be used for the result set which generated the figures and statistics here displayed in Figures 41 and 42. The results here are very similar and are quite supportive of the results in the previous section. This is a very strong indication of the strength of the prediction capability of the trained network. Since the result set is 10 times the size of the QT_Train validation, the level of noise, particularly in Figure 42 is reduced and much smoother trends are visible.

The general trend of an increasing error through the middle region is confirmed here. The trend of lower error in the region above 200 sessions is also present as



	Mean Abs. Error	Standard Deviation
Model 1	0.072489	0.061192
Model 2	0.060520	0.051202
Model 3	0.079814	0.070813
Model 4	0.067779	0.058533
Model 5	0.062626	0.053164
Model 6	0.059872	0.054749
Model 7	0.065204	0.069709
Model 8	0.054603	0.048284
Model 9	0.066314	0.057712
Model 10	0.068357	0.064902
Total	0.065758	0.059845

Figure 41: QT Model/QT_Valid Dataset Overall Performance

was witnessed from the QT_Train evaluation and the same increase in error once the system becomes completely oversaturated is is also present.

When comparing this confusion matrix (Figure 43) to the matrix for the QT_Train validation partition (Figure 37 on page 137), the characteristics are nearly identical. It can be concluded that the model was able to absorb generalized knowledge from the QT_Train dataset. This is a particularly good result since the composition of the load across the three benchmark types were both randomly composed and this demonstrates the lack of dependence on factors of load composition (qualitative

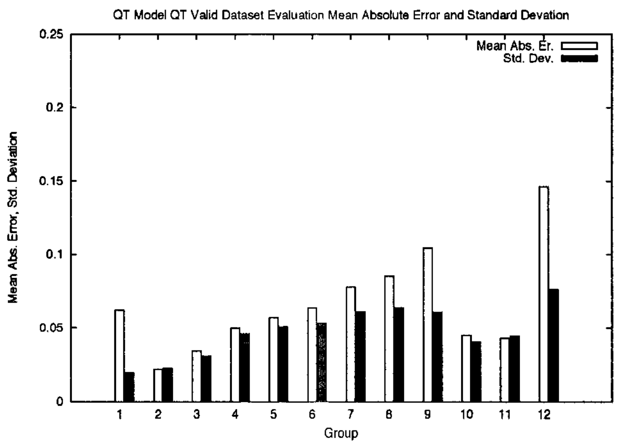


Figure 42: QT Model/QT_Valid Dataset Mean Abs. Error and Standard Dev. by Group

factors) when making a quantitative prediction.

		Closest Predicted											
		1	2	3	4	5	6	7	8	9	10	11	12
	1	2454	5546	0	0	0	0	0	0	0	0	0	0
	2	0	6215	1732	50	3	0	0	0	0	0	0	0
	3	3	879	5284	1693	121	20	0	0	0	0	0	0
	4	45	129	1876	4329	1304	255	62	0	0	0	0	0
	5	0	29	115	1206	4394	1585	635	36	0	0	0	0
Actual	6	0	0	0	22	1717	3745	1744	561	173	38	0	0
	7	0	0	0	0	241	2325	2639	1682	665	374	74	0
	8	0	0	0	0	1	277	1862	2313	1896	1024	567	60
	9	0	0	0	3	22	58	319	743	1315	2181	3119	240
	10	0	0	0	0	0	5	82	284	800	4420	2384	25
	11	0	0	0	0	5	11	34	77	198	2063	5063	549
	12	0	0	0	0	0	38	86	246	776	2616	3815	423

Figure 43: QT Model/QT_Valid Dataset Confusion Matrix by Group

7.3 Confidence Monitor

7.3.1 Overview

We have already shown for the QL model that a reasonable standard of performance is available at 200 SIMULTANIOUS_SESSIONS. The confidence monitor investigation is to determine the range over which a reasonably level of performance can still be expected. The definition of 'reasonable' itself is rather vague. To address this, several different standards of what is desired from the heuristic will be developed and compared.

As was discussed earlier, this heuristic will be intelligently designed. The first task in this section is to inform the design process. The data displayed here are generated by running the QT_Train dataset (collected for the QT investigation) through the QL model and later, both the QL and QT model. By examining the data, the heuristic will be designed to meet several different criteria. Finally, these results will be validated by independent data and compared.

7.3.2 QL Model Performance when Varying Client Session Level

7.3.2.1 Initial Data Overview Figure 44 presents the mean absolute error and standard deviation of the prediction of the QL model when evaluated with the QT_Train dataset. The entire QT_Train dataset is used for the ten models trained for each fold, which makes the result set 96000 points large. In Section 6.3 it was disclaimed that due to a shortcoming of the QT_Train dataset, there would be a certain level of noise in the dataset. The existence of such noise is detectable by looking at these histograms, but it must be overlooked due to the barrier of necessary time and effort not being available to address this problem.

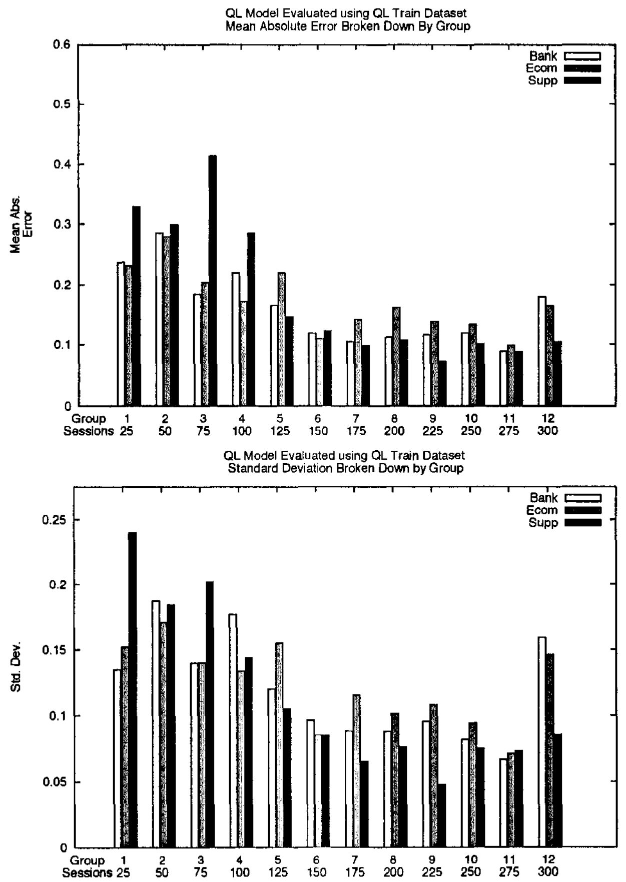


Figure 44: QL Model/QT_Train Dataset Mean Abs. Error and Standard Deviation by Group

The client load level of 200 SIMULTANIOUS_SESSIONS is represented on the histogram as Group 8. Group 8's values match quite well with the results in the QL model's evaluation in Figure 24 on page 114, as one would expect. Keeping in mind the noise that is present in this data, it can be clearly seen that the error and standard deviation remains roughly constant in Groups 6 through 11. The predictions then become quite unstable at the upper and lower bounds of that range. This range corresponds to 150 to 275 SIMULTANIOUS_SESSIONS. This implies that the QL model trained at 200 sessions produces reasonable results over a range of 125 sessions. This is a much greater range that was initially expected, an this is a quite pleasing result.

It is tempting to declare the heuristic as *IF <QT prediction is between 150 and 275 sessions> THEN <QL prediction> is 'Confident' ELSE <QL prediction> is 'Not Confident'*. This would be the most simple and straightforward approach, but it discounts a large amount of uncertainty in the system. First, the boundary between groups are 25 sessions large and finer refinement may be needed. Second, the known client activity level in a real-time environment also depends on the QT model which is also prone to uncertainty. It is clear that a more thorough approach is needed.

This investigation will proceed by evaluating all combinations of upper and lower bounds of the heuristic to discover which range is superior.

7.3.2.2 Design of Heuristic Comparison Method All combinations of upper and lower bounds need to be tested using the QT model's prediction. The performance of the QL model must be compared between predictions that fall inside the heuristic's range and those which do not. For each set of bounds, the ability for the heuristic to separate the datapoints into groups which are 'Confident' and 'Not Confident' is evaluated. To determine which is the 'best' heuristic, criteria for comparison need to be defined and the method to obtaining the criteria data must be developed.

To used the QT_Train dataset to develop this heuristic, each datapoint needs to be evaluated by both models in the manner which was illustrated in Figure 6 on page 40. Since we have models for each of the 10-training folds for both the QL and QT parts, the QT_Train dataset will be run through all ten QL models to make a result set of 96000 points. A corresponding result set is also developed by running the dataset through the QT models. As was mentioned in Section 6.3.4 the QT_Train was involved in the training of the QT model and this poses a problem to this investigation. To address this, it must be ensured that the QT prediction for each point is made by the model for which the point belongs to the testing partition. The QT_Valid dataset is a possible alternative, however, it is reserved for the validation of the heuristic in Section 7.3.4. Ideally, a new dataset similar to the QT_Train dataset would be collected for this purpose (which would take a significant amount of time). For practical purposes, this is not necessary. The comparison of the validation of the QT_Train and QT_Valid datasets indicates that the results from a newly-collected dataset would be quite similar to those obtained from the evaluation which proves that this is a very minor issue.

Once the QL predictions (b, e and s) and corresponding QT predictions (x) have been made (all sets of size N), all combinations of the bounds of the heuristic $H_{j,k}$ must be applied to each pair of QL and QT predictions.

Since the QL model was trained at 200 sessions, the cases of lower bound that are tested are 1 through 200 ($1 \leq j \leq 200$) and similarly, the cases of upper bound that are tested are 201 through 300 ($201 \leq k \leq 300$) to satisfy all possibilities.[4]

For each possible heuristic $H_{j,k}()$, it is applied to the QT predictions. The result sorts the set of corresponding points of QL predictions into two sets, one where the heuristic evaluates 'Confident' (b, e and s of size N^c) and the other where it evaluates 'Not Confident' (b', e' and s' of size N'). Along with the correct training values (B, E, S, B', E' and S') for the dataset, this provides a basis for calculating measures of a heuristic's aptitude.

Four criteria will be defined for evaluating each heuristic:

[4]In practicality, the results from evaluating a lower bound of 40 or less are ignored. This is because very few points actually evaluated lower in the QT predictions and further, they are obviously a bad choice for a lower bound which can be seen from examining Figure 44.

1. Base Mean Abs. Error (Δ^b) – This is the mean absolute error of the points which evaluate 'Confident.' Values which are lower constitute a 'better' heuristic. A separate measure for each of Bank, Ecommerce and Support categories (Δ^b_b, Δ^b_e and Δ^b_s) are calculated as:

$$\Delta^b_b = \frac{1}{N^c}\sum_{i=1}^{N^c}|b_i - B_i| \tag{14}$$
$$\Delta^b_e = \frac{1}{N^c}\sum_{i=1}^{N^c}|e_i - E_i| \tag{15}$$
$$\Delta^b_s = \frac{1}{N^c}\sum_{i=1}^{N^c}|s_i - S_i| \tag{16}$$

2. Mean Abs. Error Gain (Δ^g) – This is the improvement in mean absolute error of those which evaluate 'Confident' over those which evaluate 'Not Confident.' Values which are higher constitute a 'better' heuristic. A separate measure for each of Bank, Ecommerce and Support categories (Δ^g_b, Δ^g_e and Δ^g_s) are calculated as:

$$\Delta^g_b = \frac{1}{N'}\sum_{i=1}^{N'}|b'_i - B'_i| - \frac{1}{N^c}\sum_{i=1}^{N^c}|b_i - B_i| \tag{17}$$
$$\Delta^g_e = \frac{1}{N'}\sum_{i=1}^{N'}|e'_i - E'_i| - \frac{1}{N^c}\sum_{i=1}^{N^c}|e_i - E_i| \tag{18}$$
$$\Delta^g_s = \frac{1}{N'}\sum_{i=1}^{N'}|s'_i - S'_i| - \frac{1}{N^c}\sum_{i=1}^{N^c}|s_i - S_i| \tag{19}$$

3. Base Standard Deviation (Θ^b) – This is the standard deviation of the points which evaluate 'Confident.' Values which are lower constitute a 'better' heuristic. A separate measure for each of Bank, Ecommerce and Support categories

(Θ_b^b, Θ_e^b and Θ_s^b) are calculated as:

$$\Theta_b^b = \sqrt{\frac{1}{N^c}\sum_{i=1}^{N^c}(|b_i - B_i| - \overline{b})^2} \tag{20}$$

$$\Theta_e^b = \sqrt{\frac{1}{N^c}\sum_{i=1}^{N^c}(|e_i - E_i| - \overline{e})^2} \tag{21}$$

$$\Theta_s^b = \sqrt{\frac{1}{N^c}\sum_{i=1}^{N^c}(|s_i - S_i| - \overline{s})^2} \tag{22}$$

Where

$$\overline{b} = \frac{1}{N^c}\sum_{i=1}^{N^c}|b_i - B_i| \tag{23}$$

$$\overline{e} = \frac{1}{N^c}\sum_{i=1}^{N^c}|e_i - E_i| \tag{24}$$

$$\overline{s} = \frac{1}{N^c}\sum_{i=1}^{N^c}|s_i - S_i| \tag{25}$$

4. Standard Deviation Gain Θ^g – This is the improvement of standard deviation of the error in the points between the points which evaluate 'Confident' vs. those which evaluate 'Not Confident.' Values which are higher constitute a 'better' heuristic. A separate measure for each of Bank, Ecommerce and Support categories (Θ_b^g, Θ_e^g and Θ_s^g) are calculated as:

$$\Theta_b^b = \sqrt{\frac{1}{N'}\sum_{i=1}^{N'}(|b_i' - B_i| - \overline{b'})^2} - \sqrt{\frac{1}{N^c}\sum_{i=1}^{N^c}(|b_i - B_i| - \overline{b})^2} \tag{26}$$

$$\Theta_e^b = \sqrt{\frac{1}{N'}\sum_{i=1}^{N'}(|e_i' - E_i| - \overline{e'})^2} - \sqrt{\frac{1}{N^c}\sum_{i=1}^{N^c}(|e_i - E_i| - \overline{e})^2} \tag{27}$$

$$\Theta_s^b = \sqrt{\frac{1}{N'}\sum_{i=1}^{N'}(|s_i' - S_i| - \overline{s'})^2} - \sqrt{\frac{1}{N^c}\sum_{i=1}^{N^c}(|s_i - S_i| - \overline{s})^2} \tag{28}$$

Where $\overline{b}$, $\overline{e}$ and $\overline{s}$ are as above and

$$\begin{aligned}
\overline{b'} &= \frac{1}{N'}\sum_{i=1}^{N'}|b'_i - B'_i| &(29)\\
\overline{e'} &= \frac{1}{N'}\sum_{i=1}^{N'}|e'_i - E'_i| &(30)\\
\overline{s'} &= \frac{1}{N'}\sum_{i=1}^{N'}|s'_i - S'_i| &(31)
\end{aligned}$$

This is not an all-inclusive set of criteria one could use to evaluate the heuristic. For simplicity of discussion here, the evaluation is limited to these criteria. Future work may be interested in improving some element of this process and may better address this issue based on what criteria is desired at the time.

The values of these criteria were calculated for each combination of lower and upper bound on the heuristic and the results are plotted in Figures 45 and 46 in the next section.

7.3.3 Design of Heuristic

Figures 45 and 46 illustrate the evaluation of the heuristic by the criteria defined in the previous section.

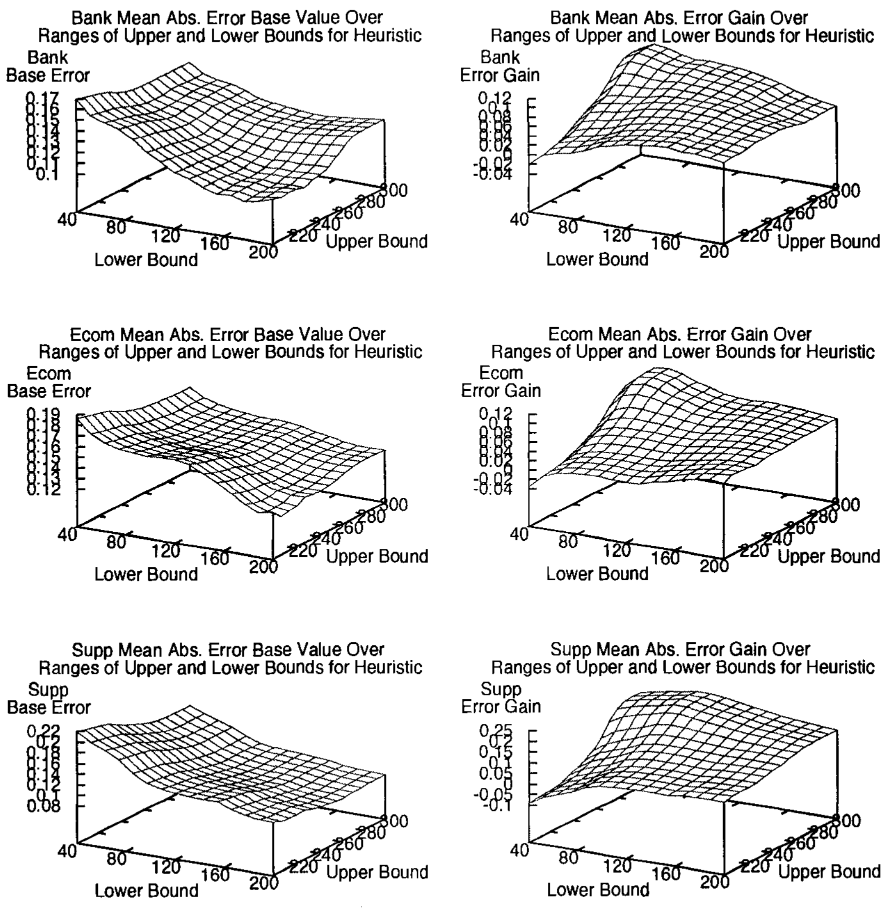


Figure 45: Plot of Base and Gain Mean Abs. Error for all Heuristics by Category

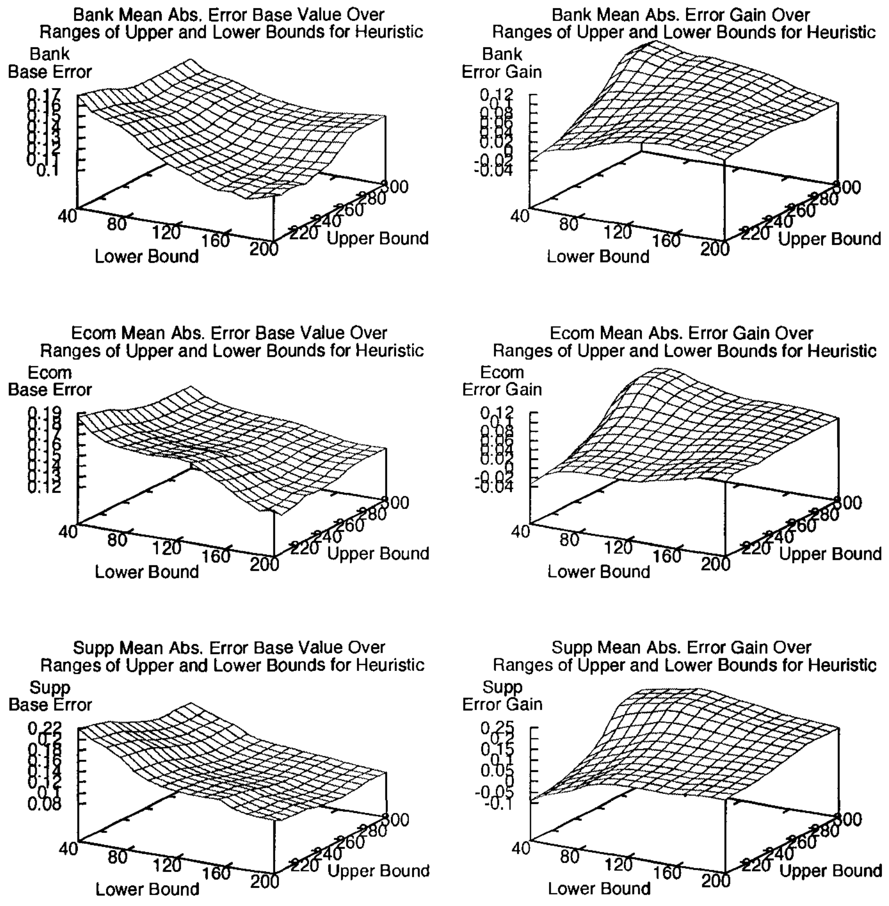


Figure 46: Plot of Base and Gain Std. Deviation for all Heuristics by Category

In the case of the base mean and standard deviation plots, the minima represents the optimal heuristic for that criteria. For the gain plots, the maxima represents the optimal heuristic. To make a good choice of heuristic, all criteria need to be balance in the decision. As can be seen in the base value plots in Figures 45 and 46, the minimas and maximas do not correspond to the same choice of heuristic. A compromise has to be reached between the criteria to find the best overall choice.

Here, three strategies of the design will be taken to develop three heuristics which will then be carried on to validation. These are representative of what could be interpreted from the rather ambiguous term 'Confident.' The motivation of the strategies are:

- Strategy 1 – Minimize Base Mean Absolute Error and Standard Deviation
- Strategy 2 – Maximize Mean Absolute Error and Standard Deviation Gain
- Strategy 3 – A Compromise Between Minimizing Base Values and Maximizing Gain Values

These strategies will be executed by determining a function for the 'fitness' of each heuristic, plotting the fitness to discover the maxima and taking the corresponding heuristic as the 'best.' The function chosen for each is just one arbitrary choice of many that could be used to fulfill the objective. The choices here were deemed adequate to fill the current need, but future work could include a more sophisticated selection process to fit a perhaps different need.

For Strategy 1, the fitness (F_1) is calculated as:

$$F_1 = \frac{1}{\Delta_b^b + \Delta_e^b + \Delta_s^b} + \frac{1}{\Theta_b^b + \Theta_e^b + \Theta_s^b} \tag{32}$$

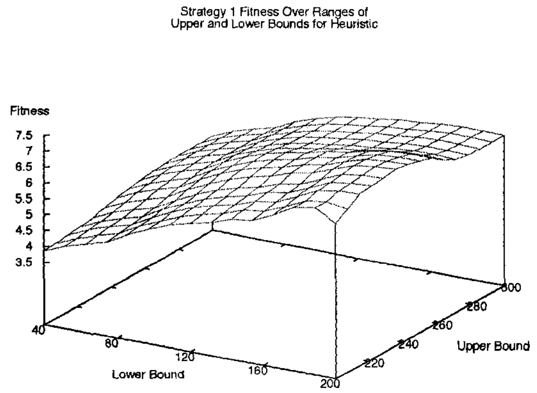
For Strategy 2, the fitness (F_2) is calculated as:

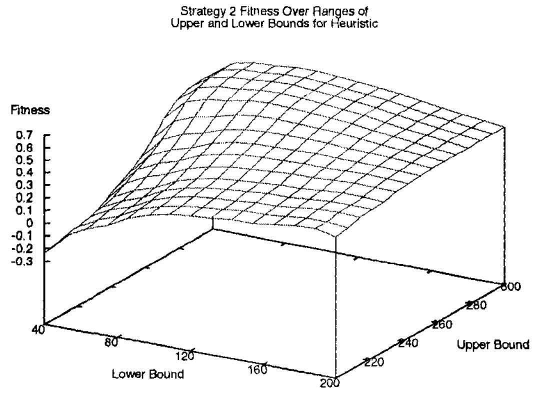
$$F_2 = \Delta_b^g + \Delta_e^g + \Delta_s^g + \Theta_b^g + \Theta_e^g + \Theta_s^g \tag{33}$$

For Strategy 3, the fitness (F_3) is calculated as:

$$F_3 = \frac{\Delta_b^g + \Delta_e^g + \Delta_s^g}{\Delta_b^b + \Delta_e^b + \Delta_s^b} + \frac{\Theta_b^g + \Theta_e^g + \Theta_s^g}{\Theta_b^b + \Theta_e^b + \Theta_s^b} \tag{34}$$

The fitness of the heuristics by the three strategies are plotted in Figure 47.





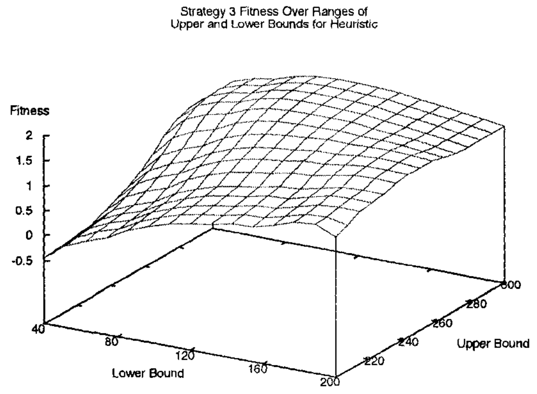


Figure 47: Heuristic Fitness by Strategy

As is illustrated in the plot, the heuristic for the maxima for Strategy 1 is *IF <QT prediction is between 195 sessions and 238 sessions> THEN <QL prediction> is 'Confident' ELSE <QL prediction> is 'Not Confident'*

The heuristic for the maxima for Strategy 2 is *IF <QT prediction is between 91 sessions and 291 sessions> THEN <QL prediction> is 'Confident' ELSE <QL prediction> is 'Not Confident'*

The heuristic for the maxima for Strategy 3 is *IF <QT prediction is between 108 sessions and 291 sessions> THEN <QL prediction> is 'Confident' ELSE <QL prediction> is 'Not Confident'*

The associated base and gain values of the three heuristics are displayed in tabular form in Figure 48. They will also be validated using independent data in the next section.

		Mean Abs. Error Base Value	Mean Abs. Error Gain	Std. Deviation Base Value	Std. Deviation Gain
Strategy 1 195-238 Sessions	Bank	0.106648	0.061469	0.081851	0.061291
	Ecom	0.123475	0.053145	0.097382	0.0451700
	Supp	0.093704	0.095394	0.067405	0.108305
Strategy 2 91-291 Sessions	Bank	0.131272	0.104762	0.111362	0.058847
	Ecom	0.145980	0.085967	0.117900	0.039533
	Supp	0.118088	0.214952	0.099955	0.111124
Strategy 3 108-291 Sessions	Bank	0.126275	0.102429	0.105475	0.063030
	Ecom	0.143852	0.077869	0.115339	0.041393
	Supp	0.110842	0.200742	0.092001	0.114369

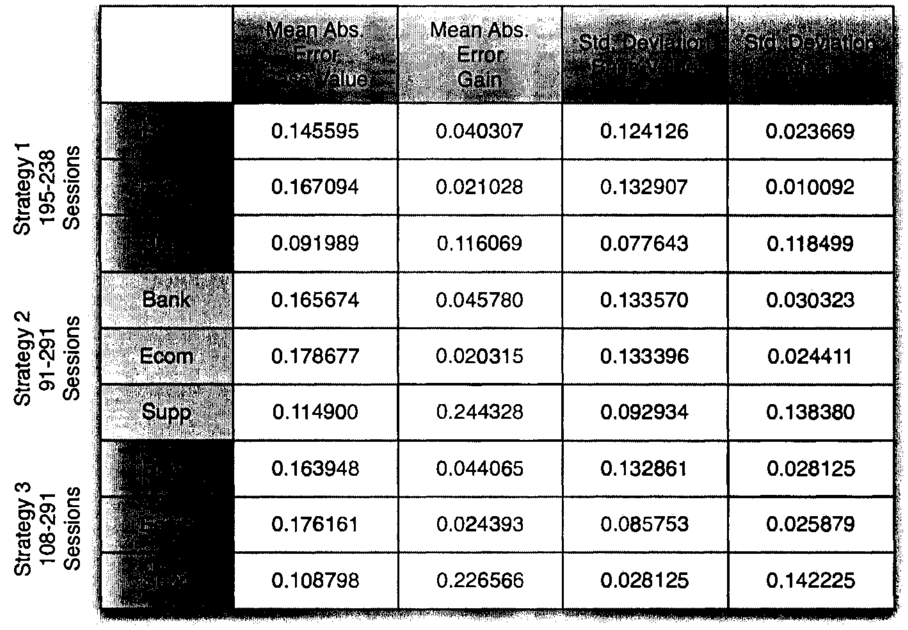
Figure 48: Confidence Monitor Heuristics' Performance on QT_Train Dataset

7.3.4 Validation

Since the above heuristics were 'trained' using the QT_Train dataset. It follows that the validation should be performed with an independent dataset. Conveniently, the QT_Valid dataset fits the requirement, as it was designed for this purpose in the QT model's validation and it will be re-used here. The independence of this set from the training sets which trained the QL and QT models make this a fair simulation of a real-world situation. The heuristics are tested by putting each datapoint from the dataset through both models from the earlier two investigations, applying the heuristics and breaking down the error by the result of the heuristic (as was done in the above section). One would expect, if the heuristic is working, the values to be similar to those in Figure 48. Figure 49 presents obtained results from the QT_Valid dataset.

It can be seen in the figure that the strategies that went into the heuristics hold in the validation. Strategy 1 does indeed have the least base mean absolute error and standard deviation, Strategy 2 does indeed have the most gain and Strategy 3 is a compromise. This can be considered a successful result.

One thing that is immediately apparent is the drastically different base levels than in the previous evaluation in Figure 48. The 'Confident' base mean absolute error value in the Bank category is significantly higher than was witnessed in the evaluation of the QL model and the relative levels of error between the three categories are



		Mean Abs. Error [illegible] Value	Mean Abs. Error Gain	Std. Deviation [illegible]	Std. Deviation [illegible]
Strategy 1 195-238 Sessions	[illegible]	0.145595	0.040307	0.124126	0.023669
	[illegible]	0.167094	0.021028	0.132907	0.010092
	[illegible]	0.091989	0.116069	0.077643	0.118499
Strategy 2 91-291 Sessions	Bank	0.165674	0.045780	0.133570	0.030323
	Ecom	0.178677	0.020315	0.133396	0.024411
	Supp	0.114900	0.244328	0.092934	0.138380
Strategy 3 108-291 Sessions	[illegible]	0.163948	0.044065	0.132861	0.028125
	[illegible]	0.176161	0.024393	0.085753	0.025879
	[illegible]	0.108798	0.226566	0.028125	0.142225

Figure 49: Confidence Monitor Heuristics' Performance on QT_Valid Dataset

of different proportions as well. This can be explained by the random noise of a relatively small sample size of datapoints in the QT_Valid dataset. By looking at the random distribution of the load composition in Figure 40 on page 141, it can be seen that there are more points near the extreme of the Bank corner of the diagram and less near the Support corner. As was known from earlier evaluations, those points tend to have higher error and influence the average score. The Ecommerce corner is more evenly balanced and as a result, its base mean average error in the 'Confident' category is roughly in line with what was seen in the QL model.

Comparison with Figure 44 on page 146, is a good corroboration of the values seen in Figure 49. In that figure, the Support category varies wildly whereas, the Bank and Ecommerce categories change more smoothly from the 'Confident' region to the 'Not Confident' region.

The three heuristics that were developed were chosen using arbitrary criteria. They entirely depended on the working definition of 'Confident.' Further, the fitness functions that were chosen also may be considered less than optimal. The results show a noticeable bias towards the Support category because the gains were large and the function did not compensate towards the other categories. In an extension of this work, the criteria and fitness functions could be adjusted to meet another perception of what is desired from the heuristic. These possible perceived deficiencies will not be addressed here since this section. In its present form, this investigation has accomplished its goal of showing the process of designing an effective heuristic and presenting a prototype of an effective confidence monitor in a real-time setting which matches a desired design goal.

8 Conclusions and Future Work

8.1 Conclusions

This thesis presents a new architectural approach for the construction of some base-level autonomic elements. The monitoring framework SOASE, a sentinel element, was architected and implemented to instrument the internal operation of the web-server system. An intelligence layer consisting of neuro-fuzzy networks (which comprise an aggregator element) was created to process the obtained data into qualitative and quantitative information regarding the real-time operation of the system. The knowledge learned by the neuro-fuzzy models in the intelligence layer was tested on new independent data to examine how the performance holds up in a real-world situation objectively and also subjectively in the case of the qualitative model. Further, a simple heuristic approach was tested for determining the range of input parameters in which the qualitative prediction would be confident.

To complete a system that one could consider fully autonomic, one would need to extend the system to make responses to the real-time QL and QT outputs. Expert knowledge would need to be applied to the qualitative and quantitative values to make some modification to the system in a way to satisfy some self-configuring or self-optimizing quality. This would likely be an adjustment of the tunable parameters in the operating system or webserver software to better match the experienced workload, or else as is illustrated in Figure 5, it could also potentially include adjustment of a much larger system of many servers. The expert knowledge would need to be developed from a familiarity of how the modification of parameters will affect the throughput of the single webserver or larger system which could be developed by experimentation and/or informed knowledge.

This constructed prototype serves as a proof-of-concept of qualities for the engineering of future systems to exhibit to be useful towards the aforementioned completed autonomic system. The task of this prototype is to provide the required knowledge for the basis of an informed decision. It addresses a traditionally difficult step in the autonomic loop by bridging the gap between highly granular low-level monitoring and higher level concepts which can be easily reasoned upon. It has bestowed on the server system a limited sense of 'self-awareness' to know what it's own workload is without human intervention. For that, in the opinion of the author, this research has been successful.

The direct contributions of this thesis can be summarized as:

- The Experimental Setup – This research defined the parameters for its own investigations. It started with the idea of finding an application of Solaris DTrace in the field of Autonomic Computing. To accomplish this goal, the SPECweb2005 benchmark was chosen to represent a basis of system activity and the overall operating domain. The combination of these elements in this setting is the most unique contribution of this thesis and promise for future investigations has been established.

- SOASE – As a means to accomplish the goals established at the outset, SOASE was architected and constructed. An effective basic set of DTrace probes for instrumenting a system in real time was established. Furthermore, the architecture was left open to act as a baseline for future scrutiny and refinement of the work done here.

- QL and QT Model Prototypes – The neuro-fuzzy techniques chosen to constitute the intelligence layer of this system have been demonstrated as being effective for the task. The validation on independent data, which was a near-perfect simulation of a real-time environment, showed the neuro-fuzzy networks to contain generalized knowledge which could be applied effectively to this problem.

At the conclusion of analysis, it was found that the constructed system performed to a quite reasonable standard which exceeded the initial expectations. While the performance results were good, there is certainly room for improvement in the accuracy. Also, given this initial positive result, this system shows promise to be adapted to a slightly different or improved application in which it may be even more effective. These issues will be discussed in the following final section.

8.2 Future Work

Several areas of possible future research are presented below:

- SOASE Daemons DTrace Utilization – As was mentioned in the discussion of the design of SOASE, the developed daemons only cover a small subset of the data available from DTrace. Examination of a larger set of data could give the aggregator elements more ability to distinguish characteristic in the system behavior. The probes that were chosen were chosen because they were quite likely to exhibit differences for the web serving task that the system would be undergoing. Further refinement in the objectives of future investigation could utilize DTrace probes which are more targeted to the specifics of those task. Due to the complexity and openness of the Solaris kernel, there is no shortage of potential in this area.

- Intelligence Layers inside SOASE – Many probes defined by DTrace have a tendency to be very verbose as the system executes. This leads to an overload of data available for processing. A method of aggregation could be developed to leverage these probes effectively. DTrace has facilities for aggregation of fired probes, but an opportunity exists for a different machine learning approach to be effective. The techniques of Sequential Pattern Mining [1] may have something to offer by exploiting the very deterministic and sequential nature of the kernel execution through different paths. Also, a soft computing approach could discover ways to exploit a verbose datastream in such a way to collect the data more effectively and efficiently. This approach would introduce an intelligence layer in the into the sentinel element. This layer compounded with the intelligence layer adds a great deal of uncertainty and complexity which would require a great amount of examination to address which is why it was left out of the scope of this project.

- Creation of Autonomic Policies and Actions – To extend this work into a complete autonomic system, a few things must be developed. First, expert knowledge (autonomic policies) must be developed to give the system a sense of which actions can improve the system given a knowledge of an existing condition detected by the sentinel and aggregator elements. The development of this knowledge has been explored to some extent by Thereska [29] by designing facilities to address resource-specific "What if?" questions to assist system administrators optimization. This knowledge could be leveraged in the construction of autonomic policies.

  Second, autonomic apparatus must be implemented to make the necessary adjustments to carry out the recommendations of the policies. Finally, a control system (likely in the form of a registry element) must be developed to ensure that changes which are carried out that affect the sentinel and aggregator elements are compensated for given the knowledge that the parameters in the underlying system have changed. These are difficult tasks that require the lower-level apparatus as developed in this work are already in place. However, to build a fully-realized autonomic system, this type of work is necessary.

- Examining a Different Problem Domain – The subset of the web serving environment was explored in this work. Some examination of the system's behavior in a separate domain was examined in the subjective evaluation of the QL model. It is suspected that for a different range of tasks the system could be performing, the best results will come from constructing new elements tuned towards that task. Orientation points for comparison could be chosen to be more relevant to the task than the three SPECweb2005 workloads. These could perhaps be other benchmarks or other representations of 'extreme' types of system activity.

- Improved Training Dataset – The SPECweb2005 benchmark imposed restrictions on the quality of data which could be used in the model training process. With more time available for data collection, this could have been alleviated. In the absence of such practical limits, a better dataset could have been collected and trained into the neuro-fuzzy network. This would produce models which are more accurate and more suitable for deployment in a production environment.

- Interpretation of Obtained Models to Refine the Training Process – One benefit of the utilization of Fuzzy Neural Networks is the interpretability property of the obtained model. A step which was not taken in this work is the white-box examination of the obtained networks. By looking at the criteria which goes into the output of the network, insight could be obtained for improvements in the data collection in the sentinel elements. There is a large potential for improvement in the accuracy and efficiency of the developed system if this study is conducted.

- Exploration of Other Machine Learning Techniques – A choice to use a different, perhaps non-connectionist, technique for developing the intelligence layer. The neuro-fuzzy technique had a set of benefits in this work, but another could be just as and perhaps more, effective at the task. A set of directly engineered heuristics could be more direct and intuitive as a method or perhaps a decision tree or other model obtained through data-mining algorithms could seek the important patterns. Exploring these possibilities and how they could be used in the construction of other autonomic apparatus would be a worthwhile examination to conduct.

References

[1] Agrawal, R. & R. Srikant, "Mining Sequential Patterns" *Proceedings of the Eleventh International Conference on Data Engineering*, pp. 3-14, March 6-10, 1995.

[2] Bantz, D.F & C. Bisdikian & D. Challender & J.P. Karidis & S. Mastrianni & A. Mohindra & D. G. Shea & M. Vanover "Autonomic Personal Computing" *IBM Systems Journal*, Vol. 54, No. 1, 2003

[3] Bigus, J.P. & D.A. Schlosnagel & J.R. Pilgrim & W.N. Mills & Y. Diao "ABLE: A Toolkit for Building Multiagent Autonomic Systems" *IBM Systems Journal* Vol. 41, No. 3, pp. 350-372, 2002

[4] Cantrill, B. M.& M. W. Shapiro & A. H. Leventhal "Dynamic Instrumentation of Production Systems" *Proceedings of the 2004 USENIX Annual Technical Conference* available at `http://www.usenix.org/event/usenix04/tech/general/full_papers/cantrill/cantrill_html/` Accessed Dec. 2007

[5] Diao, Y. & J.L. Hellerstein & S. Parekh & J.P. Bigus, "Managing Web Server Performance with AutoTune Agents" *IBM Systems Journal*, Vol. 42, No. 1, 2003.

[6] Engel, M. & B. Freisleben "Supporting Autonomic Computing Functionality via Dynamic Operating Systems Kernel Aspects" *Proceedings of the 4th International Conference on Aspect-Oriented Software Development*, ACM Press, March 2005.

[7] Furuhashi, T. "On Interpretability of Fuzzy Models" *Advances in Soft Computing - Proceedings of the 2002 AFSS International Conference on Fuzzy Systems*, Springer-Verlag Berlin Heidelberg, pp. 12-19, 2002

[8] Goebel, M. & L. Gruenwald "A survey of Data Mining and Knowledge Discovery Software Tools" *SIGKDD Explorations*, Vol. 1, Issue 1, pp. 20-33, June 1999

[9] Gregg, Brendan "The DTrace Toolkit" Source Cod, Description and Documentation Available At `http://www.brendangregg.com/dtrace.html#DTraceToolkit` Accessed Dec. 2007

[10] Gunther, N. "UNIX Load Average Part 1: How it Works" Performance Dynamics Company, Feb. 2003, Available at `http://www.teamquest.com/resources/gunther/display/5/index.htm` Accessed Dec. 2007

[11] Gupta, M. M. & J. Qi "On Fuzzy Neuron Models" *IJCNN-91 Seattle International Joint Conference on Neural Networks* Vol. 2, pp. 431-436, Jul. 1991

[12] Ishibuchi, H. "Development of Fuzzy Neural Networks" *Fuzzy Modeling: Paradigms and Practice*, edited by Wiltold Pedrycz, Kluwer Academic Publisher, 1996, pp. 185-202

[13] Kaiser, G. & P. Gross & G. Kc & J. Parekh & G. Valetto "An Approach to Autonomizing Legacy Systems" *Workshop on Self-Healing, Adaptive and Self-Managed Systems*, June, 2002.

[14] Kephart, J. O. & D. M. Chess "The Vision of Autonomic Computing" *IEEE Computer Magazine*, Vol. 36, No. 1, pp. 41-50, Jan. 2003.

[15] Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection* MIT Press, 1992

[16] Lau, T. & D. Oblinger & L. Bergman & V. Castelli & C. Anderson "Learning Procedures for Autonomic Computing" *Workshop on AI and Autonomic Computing*, Acapulco, Mexico 2003.

[17] Mauro, J. & R. McDougall *Solaris Internals: Core Kernel Architecture* 1st Edition, Prentice Hall PTR, October 2005.

[18] Mauro, J. & R. McDougall & B. Gregg *Solaris Performance and Tools* 1st Edition, Prentice Hall PTR, October 2005.

[19] Nguyen, H. T. & N. R. Prasad & C. L. Walker & E. A. Walker *A First Course in Fuzzy and Neural Control* Chapman & Hall/CRC, 2003

[20] Norman, D.A. & A. Ortony, & D. Russel "Affect and Machine Design: Lessons for the Development of Autonomous Machines" *IBM Systems Journal*, Vol. 42 Issue 1, pp. 29-38, 2003

[21] Pedrycz, W. "Heterogeneous Fuzzy Logic Networks: Fundamentals and Development Studies" *IEEE Transactions on Neural Networks* Vol. 15, No. 6, pp. 1466-1481, Nov. 2004

[22] Pedrycz, W. & M. Reformat "Evolutionary Fuzzy Modeling" *IEEE Transactions on Fuzzy Systems*, Vol. 11, Issue 5, pp. 652-665, Oct. 2003

[23] Pedrycz, W. & M. Reformat "Genetically Optimized Logic Models" *Fuzzy Sets and Systems*, Vol. 150, No. 2, 2005, pp. 351-371

[24] Rutkowska, D. *Neuro-Fuzzy Architectures and Hybrid Learning* Physica-Verlag Heidelberg, 2002

[25] Salehie, M. & L. Tahvildari, "Autonomic Computing: Emerging Trends and Open Problems" *Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, St. Louis Missouri ACM Press, pp. 1-7, 2005

[26] Samadi, Behrokh "TUNEX: A Knowledge-Based System for Performance Tuning of the UNIX Operating System" *IEEE Transactions on Software Engineering* Vol. 15, No. 7, July 1989, pp. 861-874

[27] Siedlecki, W. & J. Sklansky "A Note on Genetic Algorithms for Large-Scale Feature Selection" *Pattern Recognition Letters*, Vol. 5, Issue 5, November 1989.

[28] Siracusa, J. "Mac OS X 10.5 Leopard: The Ars Technica Review" *Ars Technica* Available at `http://arstechnica.com/reviews/os/mac-os-x-10-5.ars` Accessed Dec. 2007

[29] Thereska, E. & M. Abd-El-Malek & J. J. Wylie & D. Narayanan & G. R. Ganger "Informed Data Distribution Selection in a Self-predicting Storage System" *Carnegie Mellon University, Tech. Rep.*, January 2006

[30] Tianfield, Huaglory "Multi-Agent Based Autonomic Architecture for Network Management" *Proceedings of the 2003 IEEE International Conference of Industrial Informatics*, pp. 462-469, 2003

[31] Want, R. & T. Pering & D. Tennenhouse, "Comparing Autonomic & Proactive Computing", *IBM Systems Journal*, Vol. 42, No. 1, 2003

[32] White, S.R. & J.E. Hanson & I. Whalley & D.M. Chess & J.O. Kephart "An Architectural Approach To Autonomic Computing" *Proceedings of the International*

Conference on Autonomic Computing 2004, pp. 2-9. May 2004

[33] Witten, I. H. & E. Frank *Data Mining: Practical Machine Learning Tools and Techniques* 2nd Edition, Morgan Kaufmann, San Fransisco, 2005.

[34] IBM, "The Autonomic Computing Manifesto" October 2001, available at `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf` Accessed Dec. 2007

[35] Sun Microsystems, *Solaris Dynamic Tracing Guide* 2005, Sun Microsystems Inc. Available at `http://docs.sun.com/app/docs/doc/817-6223` Accessed Dec. 2007

[36] Standard Performance Evaluation Corporation *SPECweb2005 Online Documentation* Available at `http://www.spec.org/web2005` Accessed Dec. 2007

[37] Wall Street Journal Technology Innovation Awards, Available at `http://online.wsj.com/public/article/SB115755300770755096-Puh3Kr2L9dGEhvkWy094UivIRwA_20070910.html` Accessed Dec. 2007