

USING NEURAL NETWORKS FOR  
WEB PROXY CACHE REPLACEMENT

by

JAKE COBB

Advisor  
HALA ELAARAG

A senior research proposal submitted in partial fulfillment of the requirements  
for the degree of Bachelor of Science  
in the Department of Mathematics and Computer Science  
in the College of Arts and Science  
at Stetson University  
DeLand, Florida

Fall Term  
2005

## TABLE OF CONTENTS

TABLE OF CONTENTS.....	ii
ABSTRACT.....	1
1. INTRODUCTION .....	2
2. BACKGROUND .....	4
3. RELATED WORK.....	10
4. NEURAL NETWORK PROXY CACHE REPLACEMENT .....	12
5. SIMULATION.....	16
6. SUMMARY .....	18
7. FUTURE WORK.....	20
REFERENCES .....	22

## **ABSTRACT**

Web traffic has continued to increase at a significant rate and is resulting in considerable strain on web servers and bandwidth providers, such as Internet Service Providers (ISP). Proxy caches are often used to reduce this burden. In this research, a novel approach to web proxy cache replacement is developed. Unlike previous approaches, this research utilizes neural networks for replacement decisions. The neural network is given information about objects in the cache known to be important in web proxy caching as input and the network's output is used to guide cache object replacement.

## 1. INTRODUCTION

The world wide web accounts for a major portion of contemporary Internet traffic. Popular web sites easily exceed the number of requests, in terms of computational time and bandwidth, that a single server is able to handle. Furthermore, competition amongst web sites makes response time critical. Proxy servers are one method of distributing the load without maintaining multiple exact copies of individual web sites. They are also useful for Internet Service Providers (ISP) who want to make efficient use of limited bandwidth by minimizing unnecessary traffic. The proxy server wants to serve as many objects from the cache as possible, serve as much data from the cache as possible, or both. Optimizing both is ideal, but many practical algorithms optimize for one over the other. There are a number of static algorithms, such as least recently used (LRU) and least frequently used (LFU), which currently address the question of deciding which objects to cache and which objects to remove from the cache in order to accommodate new objects.

Neural networks have gained considerable popularity in recent years. They have been employed in a number of applications, particularly in the area of pattern recognition. Neural networks are able to learn by experience and hold an internal representation of meaning in data. An appropriately structured neural network will be able to generalize the knowledge acquired from training to data that lies outside the training set. This property makes neural networks useful for solving problems that contain uncertainty or have a problem space which is too large for an exhaustive search.

Although a small amount of work that applies neural networks to caching problems exists, until the writing of this paper, there is no work that applies neural networks to web proxy caching specifically. In this research, we present an approach to web proxy cache replacement that uses neural networks for decision making. The rest of the paper is organized as follows. Section 2 presents background information on web proxy cache replacement algorithms and neural networks. Section 3 reviews some related work in web proxy cache replacement and in the application of neural networks to caching problems. Section 4 discusses a novel proxy cache replacement scheme that incorporates neural networks. Section 5 explains the methods and metrics used for simulation and results analysis. Section 6 presents a summary of the paper. Section 7 outlines the implementation, simulation and results analysis phase of the research which will be conducted next semester.

## **2. BACKGROUND**

This section reviews web proxy caching and discusses existing cache replacement schemes. Artificial neural networks will then be reviewed with emphasis on multi-layer feedforward neural networks, which is the organizational method that will be used in this research.

### **2.1 Web Proxy Caching**

Web proxy caching is a paging problem. Strategies applied to other paging problems, such as main memory management, have been adapted to address web proxy caching. Web proxy caching has two main factors that must be addressed: cache replacement and cache consistency. Cache replacement refers to deciding what should be removed from the cache to make room for new data. Cache consistency refers to ensuring that items in the cache are still the same as items on the original server. This research will address the former.

In [1], the authors conduct an extensive survey of web cache replacement techniques. Cache replacement strategies are classified into recency-based, frequency-based, recency/frequency-based, function-based and randomized strategies. Recency-based strategies are modeled after Least Recently Used (LRU). Recency-based strategies center around the idea of temporal locality, meaning that a reference to an item is likely to be followed by additional references. They are simple, quick and somewhat adaptive but many do not handle size information well. Frequency-based strategies are modeled

after Least Frequently Used (LFU). These strategies are best for static environments. They are used less often in commercial applications than recency-based strategies because they tend to be more complex. Furthermore, they tend to suffer from cache pollution if an appropriate aging factor is not used. Recency/frequency-based strategies simply consider both recency and frequency in decision-making. Function-based strategies apply a function to candidate items and select for replacement based on that value. The functions have weights or parameters that determine behavior and are thus able to be tuned to specific workloads. Randomized strategies are those that are non-deterministic. Randomized strategies can be fully random, but the vast majority cull choices down to a smaller group and then select an item for replacement at random from the remaining list. Podlipnig et al. [1] list the most commonly considered factors for web proxy cache replacement strategies as a whole as recency, frequency, size, cost to fetch, modification time and expiration time.

## **2.2 Neural Networks**

Neural networks are comprised of many inter-connected simple processing units (neurons) which (when combined) are able to learn from data sets [2]. Neural networks typically mimic biological neurons by using a set of weights, one for each connection, which is similar to the exciting and inhibiting properties of actual neurons [2-5]. By adjusting these weights such that the network is able to provide correct outputs for most of (ideally all of) the inputs, the network is said to gain knowledge about the problem. This is particularly useful for problems where a definite and/or optimal algorithm is

unknown. Neural networks are also valuable for developing heuristics for problems where the data set is too large for a comprehensive search [3].

A neuron with an associated weight for each input is known as a McCulloch and Pitts (MCP) neuron [2]. A neural network comprised of MCP neurons is considerably more powerful than a neural network with un-weighted neurons. The weights allow the network to develop its own representation of knowledge [3]. The output of each neuron is determined by applying a function to the sum of every output multiplied by the weight of the associated connection [2-6]. The MCP model does not suggest a specific rule for the translating neuron input to output, so one must be chosen according to the properties of the problem the network is designed to solve.

### *2.2.1 Firing Rule / Squashing Function*

The firing rule determines whether or not the neuron will “fire” based on the inputs. Neuron firing rules are generally linear (proportional output), threshold (binary output), or sigmoid (non-linear proportional output) [2]. Although “firing” is a simple on or off for threshold networks, it is more typically used to mean a rule for determining the amount of output. Neural networks that use non-linear proportional output sometimes refer to the firing rule as the squashing function because they are typically chosen to constrain extreme positive and negative values. Common squashing functions include the sigmoid, tanh and step functions [6].

### *2.2.2 Multilayer Perceptrons*

All neural networks have a layer of inputs and a layer of outputs. Neural networks which also feature one or more additional “hidden” layers are called multilayer perceptrons (MLP) [4,6]. In this research, we will employ a feed-forward network. Feed-forward networks are fully connected, meaning all of the neurons in a given layer, proceeding from input to output, are connected to each neuron in the next layer. The output  $y_i$  of node  $i$  is  $f(a_i)$  where  $f$  is the squashing function and  $a_i$  is the activity on node  $i$ . The activity is

$$a_i = \sum_{j < i} w_{ij} y_j, \quad (1)$$

where  $w_{ij}$  is the weight of the connection to node  $i$  from node  $j$ . The name is derived from the fact that signals always proceed forward (from input to output) in the network [2,4,6]. Feedback networks also exist, but are more complicated and less intuitive than feed-forward networks. In a feedback network, neurons may be connected to other neurons in the same layer, a previous layer, or even to itself. These networks, unlike feed-forward networks, fluctuate upon receiving new input until reaching an equilibrium point [2]. A feed-forward MLP with two hidden layers is able to classify regions of any shape [6,7] and approximate any continuous bounded function [6,8].

### *2.2.3 Supervised Learning*

The MLP model described earlier requires the weights to be tuned to the specific problem the network is designed to address. Supervised learning is one method of determining the proper weights. The MLP is trained on inputs where the target output is known. The weights are then adjusted according to the correctness of the produced output. This process is repeated until the network is able to provide the correct output for

each of the training inputs [2-6]. Since the MLP approximates a function that matches the training data, this approach aims to achieve weights that allow the network to generalize to input/output combinations not included in the training set. In practical applications, the MLP needs two methods: one to measure how closely a training output matches the target output and a second to adjust the weights such that the error level is reduced [2,6].

#### *2.2.4 Objective Function*

The objective function, also called the error function, is used to quantify the level of correctness of training output. According to [6], the sum of squared errors is the most commonly used objective function. However, all objective functions make some assumptions about the distribution of errors and perform accordingly. Some applications require objective functions that lack the benefit of easy differentiability and/or independence found in the most common error functions [6].

#### *2.2.5 Back-propagation*

Back-propagation is the most commonly used method of adjusting the weights in a MLP. It calculates the error derivative of the weights by calculating the rate of change of the error as the activity for each unit changes. The values are first calculated for the output layer. The same calculations are then performed for each hidden layer based on information from the layer “in front” of that layer. Once the error derivative of the weights is known, they can be adjusted to reduce the error. Back-propagation is so

named because the error derivatives are calculated in the opposite direction of signal propagation. The delta value  $\delta_i$  of node  $i$  for hidden nodes is calculated as

$$\delta_i = f_i' \sum_k w_{ki} \delta_k , \quad (2)$$

where  $i < k$ . This process is repeated until the error is less than a threshold determined by the application of the network [2-6].

Reed and Marks [6] point out that back-propagation actually refers to both the derivative calculation and a weight change algorithm. The basic weight update algorithm changes each weight by negative the derivative of the error with respect to the weights multiplied by a learning rate  $\eta$ , as shown in equation 3.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \quad (3)$$

Two common variations of this algorithm are batch-mode and on-line learning. Batch-mode runs all patterns in a training set. The error derivative with respect to the weight for each pattern is summed to obtain the total error derivative with respect to the weights. All weights are then adjusted accordingly. On-line training, on the other hand, runs a pattern at random from the training set. The weights are updated after each single pattern using the error derivative with respect to the weights for the current training pattern.

### **3. RELATED WORK**

This section discusses some current web proxy cache replacement algorithms. Next, work using neural networks for caching is reviewed.

#### **3.1 Existing Algorithms**

There are a wide variety of web proxy cache replacement algorithms in the literature. According to [1], Pyramidal Selection Scheme (PSS) [9], Greedy Dual Size Frequency (GDSF) [10], Least-Unified Value (LUV) [11] and other algorithms developed from them are considered “good enough” for current web caching needs. The metrics used to determine this designation are discussed later. PSS is a recency-based algorithm. It uses multiple LRU caches and chooses an item for replacement from the selections of the individual LRU lists. GDSF and LUV are both function-based strategies. GDSF extends GD-Size [12] to take account for frequency in addition to size, cost to fetch and an aging factor. LUV uses a sliding window of request times to gather parameters for an undefined function  $F(x)$  which is used to calculate the probability  $p(i)$  that an object  $i$  will be referenced in the future. This probability is used along with cost to fetch and size information to select an object for replacement.

#### **3.2 Neural Network Caching**

Khalid proposed a neural network-based cache replacement scheme [13] called KORA. KORA is based upon earlier work by Pomerene et al. [14] which uses shadow

lines. Shadow lines refer to cache lines which are not currently active but have been active in the past. KORA and KORA-2 [15] use neural networks to designate cache lines as shadow lines which then receive preferential treatment. Outside of this distinction, the KORA algorithms use LRU. The KORA algorithms use feed-forward back-propagation neural networks, as will this research. However, KORA addresses traditional cache replacement. Algorithms used in traditional cache replacement form the basis for replacement algorithms used in web caching, but have received extensive modification. Web proxy cache workloads differ from traditional caching due to variable object size and bursty request sequences, amongst other things [1,9-12,17-19].

## 4. NEURAL NETWORK PROXY CACHE REPLACEMENT

In this section we present our Neural Network Proxy Cache Replacement (NNPCR) technique. The main idea behind NNPCR is to construct and train a multi-layer feed-forward artificial neural network to handle web proxy cache replacement decisions. The weights of the network are adjusted using back-propagation. Activation and error functions will be selected experimentally from those found in the literature. NNPCR uses a two hidden layer structure since networks of this class are known to have a universal approximation property. The overall size of the network is determined experimentally but aims to be as small as possible. Networks which are too large often learn the training set well but are unable to generalize [6]. The hidden layers will remain relatively the same size, if possible, to ease training [6,16]. The cache functions according to the behavior that Podlipnig et al [1] suggest is typical of practical implementations; it will fill until it reaches a high mark  $H$  and then objects will be selected for replacement until it reaches a low mark  $L$ . Network inputs are decided experimentally but includes information about the recency, frequency and size of the object at the very minimum. Network output is interpreted as a score rating the choice of the object under consideration for replacement. Objects are selected for replacement by simply comparing the scores given by the neural network. This approach is a function-based replacement strategy. Neural networks are often used to approximate functions from a sample of the data set [6]. The network is trained against shorter segments of data from the trace data sets which are selected at random.

NNPCR creates a neural network with small random weights. The training cycle is repeated until the network converges for the training sets or the number of iterations allowed for convergence is exceeded. For online training, the order of the training sets is randomized before each cycle. For each training set, NNPCR simulates cache activity for each request in the set. When cache replacement is necessary, cache object information is given to the neural network as input and the output determines which object is replaced. After each training set, the replacement decisions determined by the neural network are compared to the optimal decisions and error derivatives with respect to the weights are calculated. For online training, the weights are adjusted immediately using the current set of error derivatives. For batch training, the current set of error derivatives are added to the set of total error derivatives. Batch training updates the weights based on the set of total error derivatives at the end of each training cycle. The following pseudo-code demonstrates how NNPCR constructs and trains the neural network employed in proxy cache replacement decisions assuming the previously mentioned high and low mark method is used.

```

neural_network* nnpchr_gen(int hidden_layers, int[] nodes,
                          double error_threshold, int update_mode,
                          trace_set[] trace_sets)
{
    neural_net = neural_network(hidden_layers + 2, nodes);
    initialize neural_net weights with small random values;
    training_sets[] = randomly selected subsets of trace_sets;
    iterations = 0;

    /*
    / total_edw is a 3-dimensional array, indexed by layer and
    / sending and receiving nodes, in that order. For example,
    / total_edw[1][2][0] refers to the weighted connection from
    / the third node in the first hidden layer (second actual layer)
    / to the first node in the second hidden layer.
    */

    do

```

```

{
  if(iterations >= MAX_ITERATIONS)
    abort with error: did not converge within allotted iterations;

  if(update_mode == ONLINE)
    randomize order of training_sets;
  else //update_mode == BATCH
    set all values of total_edw array to 0;

  for(i = 0; i < length(training_sets); i++)
  {
    training_set = training_sets[i];
    training_cache = copy of training_set.initial_cache;
    optimal_cache = training_set.optimal_final_cache;

    for(j = 0; j < length(training_set.requests); j++)
    {
      request = training_set.requests[j];

      if(training_cache.contains(request.object))
      {
        update object's info in training_cache;
        update hit counts for training_cache;
      }
      else
      {
        update miss counts for training_cache;

        if(training_cache.fill + request.object.size
           >= training_cache.high_mark)
        {
          do
          {
            for(k = 0; k <= length(training_cache.objects); k++)
            {
              cache_object = training_cache.objects[k];
              neural_net.apply_input(cache_object.properties);
              cache_object.score = neural_net.output;
            }

            training_cache.remove(object with lowest score);

          } while(training_cache.fill + request.object.size
                 > training_cache.low_mark);
        }

        training_cache.insert(request.object);
      }
    }

    edw = find_error_derivatives(training_cache, optimal_cache);

    if(update_mode == ONLINE)
      neural_net.update_weights(edw);
    else //update_mode == BATCH
    {
      for(j = 0; j < edw.depth; j++)

```

```

        for(k = 0; k < edw.width; k++)
            for(l = 0; l < edw.height; l++)
                total_edw[j][k][l] += edw[j][k][l];
    }

    iterations++;
}

if(update_mode == BATCH)
    neural_net.update_weights(total_edw);

} while( abs(total_error) > abs(error_threshold) );

return neural_net;
}

```

NNPCR does not dictate exactly when replacement should occur, although training set simulation follows the high/low mark method from [1]. Once the cache is ready to replace one or more objects, NNPCR rates each candidate replacement object by applying each object's characteristics, such as number of previous accesses, across the inputs of the neural network. NNPCR is flexible about the choice of characteristics but requires the characteristics used in application to match those used in training in terms of both format and semantics. The neural network's output is replacement "score" for the object the input is derived from. A lower score represents a better choice for replacement; one or more objects are selected for replacement by simply comparing these scores.

## 5. SIMULATION

Simulation will consist of a simple iterative analysis procedure. Since this research is investigating the web proxy cache replacement problem as it exists independent of specific hardware, low-level cache simulation, such as that performed by DiskSIM [20], is not necessary. The difference in time required to retrieve an object from the disk versus main memory is assumed to be trivial compared to the difference in time required to send an object from the cache versus retrieving and retransmitting a fresh copy of the object. The simulation will run with trace data from IRCache [21]. Unique documents will be identified by size and Uniform Resource Identifier (URI).

Rhea et al [17] propose value-based web caching to address concerns about the effects of resource modification and aliasing. Although the point is valid, it will not be considered by NNPCR because the actual data transmitted will not be available. Furthermore, the trace data is sanitized before being made publicly available so that dynamic information is not available. For example, form data passed by appending the parameters to the URI is removed during sanitation. Finally, such approaches are designed with cache consistency in mind, which is beyond the scope of NNPCR.

### 5.1 Metrics

Hit-rate and byte-hit-rate are the two most commonly used metrics to evaluate the performance of cache replacement techniques [1,10,18,19]. Podlipnig et al [1] also mention the delay-savings-ratio but claims it is unstable and thus not generally used. Hit-

rate refers to the ratio of objects served from the proxy cache versus those retrieved from the original server. This metric targets user perceived delay and availability. Byte-hit-rate is the ratio of number of bytes served from the proxy cache versus retrieved from the original server. Optimizing for this metric reduces the amount of network traffic and thus eases link congestion. Algorithms that favor many small objects in the cache optimize for hit-rate whereas those that prefer fewer large objects optimize for byte-hit-rate. In [1], an algorithm is considered “good enough” for current web proxy caching needs if it performs well for more than one metric. Therefore, both hit rate and byte hit rate will be considered in evaluating NNPCR.

## **5.2 Performance Evaluation**

In order to judge how well the neural network performs, GDSF, PSS and LUV will be implemented and run on the same sets of trace data that the neural network is. Since these algorithms meet current web proxy caching needs, they provide a more realistic comparison than using basic algorithms such as LRU or LFU. The trace data will also be analyzed to determine the maximum hit rate and byte hit rate possible; each strategy will be compared with the best possible outcome.

Some trace data sets will not be used for training segment selection. The neural network’s performance on trace data sets used for training segments and data sets not used in training will be compared; the level of difference found will be indicative of the neural network’s ability to generalize.

## 6. SUMMARY

Web proxy cache replacement has received considerable attention in the literature. Strategies used in local disk caching, such as LRU and LFU, comprise the base of most strategies proposed for proxy cache replacement. However, algorithms have been developed for web proxy caches specifically, because the traffic patterns seen by a web proxy server vary from those seen in local caches on significant characteristics such as variation in object size and locality of references. The respective performances of the replacement algorithms are heavily dependant on metric and workload. These algorithms tend to either have assumptions about workload characteristics built in or include tune-able parameters to control which assumption(s) the algorithm favors.

Artificial neural networks are a model with a structure consisting of many nodes, often arranged in layers, and weighted connections between the nodes which seek to imitate the processing procedure of actual neural networks. MLP model neural networks are appropriate for web proxy caching because they are able to learn by example and generalize knowledge gained through training. The weights can be set to appropriate values through the process of supervised learning, wherein the network is trained against known input/output pairs and the weights are adjusted accordingly until the network converges to presenting correct output for all patterns in the training set. If the network is not over-trained, it should then be able to generalize reasonably to patterns outside the training set.

NNPCR is a novel web proxy cache replacement scheme which incorporates a neural network. The network is a two-hidden layer feed-forward artificial neural network which falls under the MLP model. NNPCR sets the connection weights through supervised learning with back-propagation. NNPCR trains the network using random subsets of the trace data sets. It calculates the optimal replacement choices for each training set and uses this information for target output when calculating error. Once the neural network has been created and properly trained, NNPCR is ready to handle cache replacement decisions. All candidate replacement objects are rated by applying information, such as number of accesses or size, about that object across the inputs of the network. An object is selected for replacement based on the rating returned by the neural network. NNPCR aims to take advantage of neural networks' universal approximation and generalization properties. The neural network created by NNPCR will, ideally, develop generalized knowledge about web proxy work loads from training which can be effectively applied for diverse, real-world workloads. Thus, NNPCR derives workload assumptions from the training data (partial workloads) rather than holding a static set of assumptions or requiring the user to set parameters that dictate which assumptions behavior will reflect.

## 7. FUTURE WORK

NNPCR will be implemented and evaluated in the Spring of 2006. Several aspects of the construction and training of the neural network must be determined experimentally. Properties such as universal approximation belong to neural networks as a group, so not all networks will be able to converge on every problem. The first choice is the number of nodes to use in each of the two hidden layers. The ratio of nodes in the first layer to those in the second layer will be kept close to one unless the network is unable to converge for values fitting this ratio. Second, a squashing function must be selected. A common squashing function, such as sigmoid or tanh, that works for a variety of applications is a desirable choice. However, a modified version of one of these functions or an entirely new function may be necessary to properly train the neural network. Third, an error function must be selected. Common error functions, such as the sum of squared errors, are nice to use because they tend to be differentiable and independent. If NNPCR uses one of these functions, a method will be devised for quantifying output such that it can be used in direct arithmetic comparison. Finally, an appropriate back-propagation variation and values for associated parameters must be determined. For example, some variations include a momentum factor so that weight changes are influenced by previous changes. Learning rate is one such parameter that is shared by virtually all back-propagation algorithms.

In addition to NNPCR itself, the PSS, GDSF and LUV algorithms will be implemented. These implementations will be used for comparison against the

performance of NNPCR. Optimal replacement choices will be calculated for the trace data sets and stored prior to simulation. Each algorithm will be run through a simulation covering all the trace sets. When simulation is complete, the performance of NNPCR will be discussed relative to the other algorithms, relative to the optimal choices and in terms of the effect of workload.

## REFERENCES

- [1] S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374-398, 2003.
- [2] C. Stergiou and D. Siganos, "Neural Networks," [Online document] [cited Sept. 9, 2005] Available WWW: [http://www.doc.ic.ac.uk/~nd/surprise\\_96/journal/vol4/cs11/report.html](http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html).
- [3] L. Fu, "Knowledge discovery based on neural networks," *Communications of the ACM Archive*, vol. 42, no. 11, pp. 47-50, 1999.
- [4] P. P. van der Smagt, "A comparative study of neural network algorithms applied to optical character recognition," in *Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (ACM Press, 1990, vol. 2, pp. 1037-1044)
- [5] B. Dasgupta, H.T. Siegelmann, and E. Sontag, "On a learnability question associated to neural networks with continuous activations," *Annual Workshop on Computational Learning Theory*, pp. 47-56, 1994.
- [6] R. D. Reed and R. J. Marks II, *Neural smithing: supervised learning in feedforward artificial neural networks*. The MIT Press, 1999.
- [7] R. P. Lippmann, "An introduction to computing with neural nets," *ASSP Magazine*, pp. 4-22, April 1987.

- [8] A. Lapedes and R. Farber, "How neural nets work," in *Neural Information Processing Systems*. (American Institute of Physics, 1988, pp. 442-456)
- [9] C. C. Aggarwal, J. L. Wolf and P. S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowl. Data Eng.* 11, pp. 94-107, Jan. 1999.
- [10] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, "Evaluating content management techniques for Web proxy caches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 4, pp. 3-11, 2000.
- [11] H. Bahn, K. Koh, S. L. Min and S. H. Noh, "Efficient replacement of nonuniform objects in Web caches," *IEEE Comput.* 35, pp. 65 – 73, June 2002
- [12] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," in *Proceedings of USENIX Symposium on Internet Technologies and Systems*. (1997, pp. 193-206)
- [13] H. Khalid, "A new cache replacement scheme based on backpropagation neural networks," *ACM SIGARCH Computer Architecture News*, vol. 25, no. 1, pp. 27-33, 1997.
- [14] J. Pomerene, T.R. Puzak, R. Rechtschaffen and F. Sporacio, "Prefetching Mechanism For a High-Speed Buffer Store," US Patent, 1984.
- [15] H. Khalid, "Performance of the KORA-2 cache replacement scheme," *ACM SIGARCH Computer Architecture News*, vol. 25, no. 4, pp. 17 – 21, 1997.
- [16] J. de Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *IEEE Transactions on Neural Networks*, 4(1), pp. 136-141, 1993.

- [17] S. C. Rhea, K. Liang and E. Brewer, "Value-based Web caching," in *Proceedings of the 12th International Conference on World Wide Web*. (ACM Press, 2003, pp. 619-628)
- [18] R. Caceres, F. Douglis, A. Feldmann, G. Glass and M. Rabinovich, "Web proxy caching: the devil is in the details," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 11 – 15, 1998.
- [19] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 158 – 170, 2000.
- [20] G. Ganger, B. Worthington, Y. Patt and J. Bucy, "The DiskSim Simulation Environment," [Online document] Sept. 2005 [cited Sept. 16, 2005] Available WWW: <http://www.pml.cdu.edu/DiskSim/>
- [21] "IRCache Home," [Online document] [cited Sept. 5, 2005] Available WWW: <http://ircache.net/>