File Access Prediction Using Neural Networks

Prashanta Kumar Patra, Muktikanta Sahu, Subasish Mohapatra, and Ronak Kumar Samantray

***Abstract*—One of the most vexing issues in design of a high-speed computer is the wide gap of access times between the memory and the disk. To solve this problem, static file access predictors have been used. In this paper, we propose dynamic file access predictors using neural networks to significantly improve upon the accuracy, success-per-reference, and effective-success-rate-per-reference by using neural-network-based file access predictor with proper tuning. In particular, we verified that the incorrect prediction has been reduced from 53.11% to 43.63% for the proposed neural network prediction method with a standard configuration than the recent popularity (RP) method. With manual tuning for each trace, we are able to improve upon the misprediction rate and effective-success-rate-per-reference using a standard configuration. Simulations on distributed file system (DFS) traces reveal that exact fit radial basis function (RBF) gives better prediction in high end system whereas multilayer perceptron (MLP) trained with Levenberg–Marquardt (LM) backpropagation outperforms in system having good computational capability. Probabilistic and competitive predictors are the most suitable for work stations having limited resources to deal with and the former predictor is more efficient than the latter for servers having maximum system calls. Finally, we conclude that MLP with LM backpropagation algorithm has better success rate of file prediction than those of simple perceptron, last successor, stable successor, and best k out of m predictors.**

***Index Terms*—Competitive predictor, file prediction, Levenberg–Marquardt (LM) backpropagation, multilayer perceptron (MLP), probabilistic predictor, radial basis function (RBF) network, success rate.**

I. Introduction

While processor and main memory speeds have improved at an exponential rate, disk access times are only one-third to one-fourth of what they were 30 years ago. As a result, input from and output to secondary storage remains the primary bottleneck in any computer system. The problem is not new and is likely to worsen as memory access times continue to decrease at much faster rate than disk access times [28]. In addition, the exponential increase in hard drive capacity compounds this problem. To minimize the consequences of this bottleneck, computer systems *cache*, or store in main memory, data that are expected to be requested from secondary storage in the future. Traditional caching strategies insert the most recently accessed data into the cache while making use of a *cache replacement* algorithm [4] to remove older data when the cache is full. An enhancement to traditional caching is *prefetching* [24]. Prefetching actively loads into memory data that has been predicted to be useful, while traditional caching simply keeps data that has already been loaded into memory. Algorithms for cache replacement and prefetching both require a predictor to determine which data will most likely be accessed in the near future [11]–[13], [15]–[19].

A. Caching

1) Traditional Caching: The caching of data blocks that were recently accessed is the most commonly used strategy for minimizing the performance reduction due to disk input/output (I/O). This strategy operates on the premise that data which have been accessed recently is likely to be accessed again soon. Ideally, a cache containing N blocks holds the next N blocks to be requested. Caching algorithms cannot predict the future; however, they can apply various heuristics to approximate that goal. The main challenge of traditional caching is to determine which block to remove from the cache to make room for a new block. To address this challenge, a variety of cache replacement algorithms [1]–[4], [6], [7], [9], [10] have been developed. These algorithms predict which of the cached blocks will be accessed farthest in the future. Two such algorithms are least recently used (LRU) and least frequently used (LFU). LRU selects the block that was last accessed farthest in the past, while LFU selects the block that has been accessed the fewest times since its arrival in the cache. The more accurately the cache replacement algorithm can predict the best block for replacement, the more effective the cache will be. Nonetheless, even a relatively ineffective cache may be worth including in a system, as traditional caching does not involve a high-performance cost.

2) Prefetching: Traditional caching techniques are limited in that they restrict the blocks that may be present in a cache to those that have been recently accessed. Prefetching is an enhancement to traditional caching which retrieves data before they are requested and places them in the cache. Consequently, when the system requests the data, they are already in memory. Prefetching aids the optimization of I/O scheduling by retrieving data for future accesses during periods of central processing unit (CPU) activity or user think time. As with cache block replacement, effective prefetching requires accurate prediction of future data accesses. However, the consequences of poor predictions are especially high because poor predictions will both render the prefetching technique useless as well as force the system to perform unnecessary disk accesses. While a traditional cache that is not very effective may still be better than no cache at all, it is crucial that a prefetching technique be accurate in order to be worth the cost.

Manuscript received August 31, 2007; revised September 01, 2009, January 23, 2010, and February 10, 2010; accepted February 11, 2010. Date of publication April 22, 2010; date of current version June 03, 2010.

P. K. Patra is with the Department of Computer Science and Engineering, College of Engineering and Technology, Bhubaneswar 751003, India (e-mail: pkpatra@cet.edu.in).

M. Sahu is with the Department of Computer Science and Engineering, International Institute of Information Technology (IIIT), Bhubaneswar 751003, India (e-mail: muktikanta_sahu@yahoo.co.in).

S. Mohapatra is with the Computer Science and Engineering Department, Institute of Technical Education and Research (ITER), Bhubaneswar 751030, India (e-mail: subasish2003@yahoo.com).

R. K. Samantray is with Microsoft Corporation (I) Pvt. Ltd., Hyderabad 500033, India (e-mail: ronak_rks@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2010.2043683

B. File Access Prediction

While data are usually cached a block at a time, traditional caching as well as prefetching may be applied at either the block or file level. Block-level prefetching relies on the fact that blocks within a file are typically accessed sequentially. Therefore, it is often beneficial to prefetch the next block in the file some point before being evicted from the cache.

In contrast, there is no concept of a "next file"; therefore, file-level prefetching requires the use of more sophisticated prediction mechanisms known as *file access predictors*. While most file access predictors rely on a prediction heuristic that is *static* [5], [8], [13], [15], [19], [20], [22], [23], [27], [29], [31], [33], [35], [36] meaning it does not change as it is used, recent research has turned to *dynamic* [21], [25], [26], [32], and [34] algorithms that change behavior as patterns within the file request stream change.

1) Static File Access Prediction: As we will see, the earliest file access predictors were complex; however, simple static predictors have been found to be as effective. These predictors include *last successor*, *stable successor*, and *recent popularity*, all of which observe simple patterns in past file accesses in order to predict future accesses.

2) Dynamic File Access Prediction: Ari *et al.* [6] found that at different intervals during a file request stream, different cache replacement algorithms have the greatest success. In other words, which algorithm is "best" depends on the characteristics of the workload at a given point in the request stream. It is reasonable to expect that the relative success of file access predictors is similarly dynamic.

Researchers have recently proposed dynamic file access prediction using the *multiple expert* techniques. The multiple expert techniques use machine learning to determine how much confidence to put in each of a pool of *experts* based on the success of their past predictions. When this technique is applied to file access prediction, each expert is a file access predictor.

In this paper, we present a dynamic file access predictor that selects among simple static predictors using a neural network. We have extended our work further on dynamic prediction of files using exact-fit radial basis function (RBF)-based predictor, multilayer perceptron (MLP)-based predictor, competitive predictor, and probabilistic predictor.

II. Related Work

Our work in file access prediction builds upon that of previous researchers. Before presenting our prediction scheme, we review earlier research in the fields of file prefetching and file access prediction such that our work may be understood in the proper context.

A. Motivation

Griffioen and Appleton [13] present an analysis of the potential effects of predictive prefetching on system performance. Their analysis revealed that, on average, a file was opened once every 200 ms, which means that predicting a file even one request in advance may be enough for a performance gain. They also found that there was a correlation between successive pairs of file accesses over 94% of the time, even in a multiuser environment.

Patterson *et al.* [27] suggest that in addition to improving cache performance, file prefetching can also substantially reduce the impact of disk I/O on system performance by integrating with technologies designed to increase throughput such as disk arrays [i.e., redundant array of inexpensive disks (RAID)] and log structured file systems.

Cherniack *et al.* [10] observe that prefetching may also be useful in distributed and mobile computing environments where the possibility of disconnection from the data source is possible. The idea is that a system which prefetches data when a connection is active can then make it available for offline use when the connection goes down.

Kroeger and Long [15] point out that file access prediction can be used not only to prefetch files, but also to find related files which may be grouped in close proximity on disk in order to improve I/O performance.

B. Early File Access Predictors

Early predictors, developed during the 1990s, were complex, relying on the storage of a large amount of file access history data. For example, Palmer and Zdonik [23] proposed Fido, an associative-memory-based file access predictor, which alternates between training and prediction modes. During the training mode, Fido identifies patterns in an observed file access stream that are sufficiently different—based on a tunable parameter—from previously stored patterns.

Lei and Duchamp [19] propose a similar scheme which keeps track of file access patterns observed for each application in a system. Unlike Fido, this scheme stores file patterns in a structured *access tree* which encodes not only the order of file access, but also which program initiated each access request. Since every program is a file itself, this information can easily be encoded into the tree by making each file a child of the program which requested it.

Application-driven prefetching [13] would require rewriting—or at least recompiling—many applications on a system in order to realize its effectiveness on that system which is potentially more effective than prefetching based on observation of past requests.

Patterson *et al.* [27] introduce the concept of transparent informed prefetching (TIP). They state that the effectiveness of prefetching may be greatly enhanced by relying on *knowledge*, instead of guesses, of future requests. They propose a file access prediction scheme based on *hints* given by applications. For example, when the *grep* command is issued by a UNIX shell user, the files which will be searched are immediately known. Either *grep* or the shell could provide this information to the I/O system, which could then immediately fetch the files, before *grep* requests them.

In more recent work, Cherniack *et al.* [10] propose an application-driven prefetching scheme based on the concept of user *profiles*. Each profile provides detailed information about a particular user's data requirements. Profiles include more complex information than the hints of the TIP approach.

Kroeger and Long proposed [15] that finite multiorder context modeling (FMOC), a text compression technique, may be adapted for file access pattern tracking and prediction. They further show, with their partitioned context modeling (PCM) technique [17], [18], that modifying FMOC to limit the amount of information that may be tracked conserves space while slightly

improving accuracy. They then expand this approach with the extended PCM (EPCM) [15], which allows for a series of predictions at a time, resulting in prefetches several accesses in advance.

The majority of simple predictors are *successor predictors*. Amer [2] observed that most files accumulate few unique successors over time, and many simple successor predictors successfully rely on this fact by basing their predictions on successors observed in the past.

Recent work in file access prediction can be divided into two branches: static file access prediction and dynamic file access prediction.

C. Recent Static File Access Predictors

Static file access predictors rely on a heuristic that does not change as the request stream is observed. In order to be successful, these predictors must have a heuristic that can detect patterns in a variety of common scenarios.

A number of these very basic static predictors are discussed next.

1) First Successor (FS): The FS [3] predictor maintains a constant successor prediction after a file is first observed. For example, if the first observed successor of A is B, then B will be predicted as the successor every time A is observed, irrespective of future observed successors of A.

2) Last Successor (LS): The LS file prediction algorithm proposed by Lei and Duchamp [19] is the simplest of the useful file access predictors. LS maintains the last observed successor of each file. When a file is requested, this algorithm predicts that its last observed successor will be requested next. If this is the first observed request of the file, LS declines to predict.

3) Stable Successor (SS): Amer and Long [2], [3] found that there are times when FS, despite its lack of adaptability, outperforms LS. This is due to LS's sensitivity to noise in the file request stream. They introduced the notion of the *optimal pairing* algorithm. This algorithm maintains both the first and last observed successors for each file and always selects the correct one of the two when making a prediction. The optimal pairing algorithm cannot, of course, be implemented, as it requires future knowledge. The idea is that if the noise-filtering property of FS and the adaptability of LS could be combined, then the resulting algorithm would optimally have accuracy greater than that of both the FS and LS algorithms.

The SS predictor [3], also known as *Noah*, is an extension of the LS algorithm which attempts to filter out the noise in the file request stream in such a way that approximates, and occasionally even supercedes, the accuracy of the optimal pairing algorithm. SS predicts the LS which was observed at least N times in a row, where N is a tunable parameter. If no successor has been observed N times in a row, SS declines to predict.

Adjustment of N allows for a tradeoff between prediction accuracy, the percentage of predictions which are correct, and prediction coverage, the percentage of requests for which a prediction is made. If $N = 1$, SS reduces to LS. SS with $N = 2$ was found to produce the best performance.

4) Recent Popularity (RP): The RP, or j-out-of-k [4] predictor is an extension of the SS algorithm that uses an additional parameter to allow adjustment in prediction accuracy at the expense of prediction coverage and system resources. RP maintains the last k observed successors for each file. It predicts the most frequent successor to appear in at least j of the last k requests, with recent as a tiebreaker. If no such file is found, RP declines to issue a prediction.

An enhanced version of RP [35] retains the last predicted successor and adds a third parameter l. If no successor occurs in j of the last k requests, the last predicted successor is predicted again if it occurs in at least l of the last k requests. Adjustment of the j/k ratio modifies the preferred tradeoff between prediction coverage and prediction accuracy. If $j = k$ and $l = 0$, RP reduces to SS.

5) First Stable Successor (FSS): Shah *et al.* [29] proposed an extension to the FS algorithm known as FSS. FSS does not make any predictions until N consecutive occurrences of a successor to a file are observed, after which that successor is always predicted as a successor to that file.

6) Predecessor Position (PP) and Prepredecessor Position (PPP): PP [36] is a simple predictor which predicts successors to pairs of files. For example, if the sequence BCD is observed, then the next time the sequence BC is observed, D will be predicted as the next file to be requested.

PPP [36] is an extension of PP which predicts successors to sequences of three files.

7) Examples: Table I illustrates the differing behaviors of various simple static file access predictors over the course of an observed sequence of file accesses. The predictors are: FS, LS, SS with $N = 2$, RP with $j = 2, k = 3$, and $l = 0$, FSS with $m = 2$, PP, and PPP. Note that for each pair of predictors, there is at least one case in which they issue different predictions. PP, on the other hand, would predict C for the same sequence, not taking advantage of the additional context provided by the request stream.

8) Two-Expert Approach: Chen *et al.* [9] present the two-expert approach that takes predictions from two simple predictors and will not make a prediction unless they agree. This approach will reduce the number of inaccurate predictions, increasing prediction accuracy at the expense of coverage.

TABLE I
BEHAVIORS OF STATIC FILE ACCESS PREDICTORS

File Request Sequence	FS	LS	SS	RP	FSS	PP	PPP
A	-	-	-	-	-	-	-
AB	-	-	-	-	-	-	-
ABA	B	B	-	-	-	-	-
ABAC	-	-	-	-	-	-	-
ABACD	-	-	-	-	-	-	-
ABACDA	B	B	-	-	-	-	-
ABACDAC	D	D	-	-	-	D	-
ABACDACA	B	C	C	C	C	-	-
ABACDACAD	A	A	-	-	-	-	-
ABACDACADA	B	D	C	C	C	C	-
ABACDACADAD	A	A	A	-	A	A	-
ABACDACADADA	B	D	D	D	C	D	D
ABACDACADADAB	A	-	-	-	-	A	-
ABACDACADADABA	B	B	D	D	C	C	C
ABACDACADADABAD	A	A	A	A	A	A	-
ABACDACADADABADA	B	D	D	D	C	B	B
ABACDACADADABADAB	A	A	A	-	A	A	A
ABACDACADADABADABA	B	B	D	B	C	D	D
ABACDACADADABADABAC	D	A	-	-	-	A	D

D. Recent Dynamic File Access Predictors

The simple static predictors described in the preceding section offer a reasonable accuracy with low resource requirements, but they do not provide adaptability to changes in the request stream over time unlike simpler dynamic predictors, which use fewer resources, which are discussed in the next section.

1) Composite Predictor: The composite predictor [35] takes predictions from four different heuristics (SS, PP, PPP, and RP) and then selects the one that is most likely to be correct. Each prediction is given a weight that attempts to express the probability that the prediction is correct. This weight is based on the accuracy of the heuristic which generated the prediction, as determined from simulations using several file request stream traces. Predictions below of a certain probability are excluded with a *probability threshold*. A per-file *confidence measure* was used to prevent prefetching of files determined to be difficult to predict.

2) Multiple Experts: The *multiple-expert* approach has been utilized in various areas to create dynamic prediction algorithms. This approach relies on giving multiple-algorithm adjustable weights and making predictions based on the recommendations of each expert and its current weight.

3) Adaptive Caching Using Multiple Experts: Ari *et al.* [6] propose application of the multiple expert approach to the problem of cache block replacement. Their approach, adaptive caching using multiple experts (ACME), takes the recommendations of various simple algorithms such as LRU and LFU and weights them based on their success in the recent past. ACME then produces a composite.

4) File Access Prediction Using Multiple Experts: Brandt [7] applies the multiple expert strategies to the problem of file access prediction. For a given file, this technique considers each previously observed successor to be an expert that predicts itself as that file's next successor. In addition, he introduces the concept of a *null* predictor which predicts that all of the other experts are incorrect and, therefore, no prediction should be made.

The multiple expert schemes assign a weight to each successor and adjust it based on the accuracy of that successor over time. When a file is requested, the algorithm predicts that file's successor—or chooses not to predict—by following the recommendation of the most highly weighted expert.

5) Perceptron: Ravichandran and Pâris [28] assert that as the gap between computation and I/O access speed increases, the usefulness of file access predictors that only predict the immediate successor decreases, since the system may not have time to prefetch the predicted file before it is requested. As such, they propose that one solution is to predict files more than one request in advance of their access. They present a simple perceptron-based approach for making predictions several files ahead of time.

We can draw two major conclusions from this overview of previous work. First, techniques combining several predictors outperform any predictor relying on a single heuristic. Second, the potential of neural networks in predicting file successors remains barely exploited. We propose to combine these two approaches by using neural networks to select the most likely prediction among the outcomes of several predictors.

III. Proposed Neural Network Predictors

Here the work is carried out in neural networks to dynamically select the best file access prediction heuristic. The scheme for file access prediction uses a neural network to dynamically select among multiple simple file access prediction heuristics as the file request stream is observed. We will henceforth refer to this scheme as the *neural-network-based file access predictor* or simply the *NN-based predictor*.

We chose LS, SS with $N = 2$, and RP with $j = 3, k = 4$, and $l = 0$ as our simple heuristics, or base predictors.

After each file access request, our neural network takes as inputs numerical file identifiers representing the last M observed successors of the requested file and the predictions of the three base predictors.

If a base predictor declines to make a prediction, this is represented by a unique numerical input. The network provides as outputs the probabilities that each of the base predictors has made a correct recommendation. Our NN-based predictor selects the recommendation of the base predictor with the highest probability that exceeds a tunable *probability threshold*. If no probability is sufficiently high, our predictor makes no prediction. In the case of a tie, predictions are selected based on the following order of preference of the base predictors: RP, SS, LS. If the network, for example, assigns the same probability to all three predictions, and this probability exceeds the threshold, then our predictor will issue the recommendation of RP.

We organized our network in two layers with a tunable number of hidden units. Our network is trained using the backpropagation learning algorithm, which updates the weights in a network one layer at a time by propagating the error "backwards" from the output units to the input units. Each input unit is activated with the identity function, and each hidden and output unit is activated using the sigmoid function

$$\text{activation} = 1/(1 + e^{-x})$$

where $x =$ weighted sum of inputs.

Our predictor trains the neural network in regular intervals of a configurable size. During the first interval, while the network remains untrained, our predictor replicates the RP heuristic. Once the network has been trained on the first interval, we use it to predict successors for subsequent intervals. As each interval completes, we retrain the network with the previous interval's file access requests and associated base predictions. For this method to be the most effective, parallel processing should be utilized to allow prediction of current successors at the same time as training using previously observed successors.

IV. Experimental Results

The proposed work does two distinct things: using an NN to select the best heuristic, and using NN-based heuristic for direct prediction of files. Several experiments were carried out for the above two purposes. Sections IV-B–IV-D show the results for predicting the best heuristic, whereas Sections IV-F–IV-I show the results for direct prediction of files.

A. Simulation Methodology

We simulated our file prediction scheme with the aid of a neural network simulator, file traces collected by other researchers, and our own tools.

1) Platform Information: All the NN predictors i.e., exact-fit RBF-based predictor, MLP-based predictor, competitive predictor, and probabilistic predictor below were simulated using MATLAB 7.5 on a system having Intel Pentium 4, 2.80-GHz, 512-MB RAM, and 80-GB HD.

2) Homegrown Tools: We developed tools to generate base predictions from the file traces, train the network, predict file accesses using outputs from the network, and analyze the resulting predictions.

We have also developed our own simulation tools in "C" language for static predictions like LS and SS. For dynamic prediction, we have developed separate program in "C" language.

3) File Traces: We used file traces collected at Carnegie Mellon University (CMU, Pittsburgh, PA) to simulate processing of actual file request streams. These traces were generated using CMU's distributed file_system trace (DFST) [20] consisting of inodes. An *inode* is a *data structure* on a *file system* on Linux and other Unix-like operating systems that stores all the information about a file except its name and its actual data. When a file is created, it is assigned both a name and an *inode number*, which is an integer that is unique within the file system researchers logged all system calls issued on 33 systems over a period of approximately two years. The traces we used in our simulations were generated on *Mozart*, a personal workstation; *ives*, the system with the largest number of users; *dvorak*, the system with the largest proportion of write activity; and *barber*, the system with the highest number of system calls per second.

B. Evaluation Metrics

We have used a variety of metrics such as accuracy, coverage, succes-per-reference, and effective-miss-ratio to analyze the results of our simulations. Accuracy is the percentage of predictions that are correct. Coverage is the percentage of file requests for which a prediction is generated

$$\text{Accuracy} = A = N_{cp}/N_p$$

where

$$N_{cp} = \text{number of correct predictions}$$
$$N_p = \text{number of predictions}$$
$$\text{Coverage} = C = N_p/N_{fr}$$

where

$$N_{fr} = \text{number of file requests.}$$

The primary disadvantage of using accuracy as a lone measure of success is that it does not take the effect of coverage into account. An extremely accurate predictor which predicts only 2% of the time may not be as useful as a moderately accurate predictor that predicts 80% of the time. Therefore, accuracy is often combined with coverage via the *success-per-reference* metric. Success-per-reference is the percentage of file requests which are correctly predicted

$$\begin{aligned}\text{Success-per-reference} = S_{pr} &= A \times C \\ &= N_{cp}/N_{fr}.\end{aligned}$$

Unfortunately, the main weakness of success-per-reference as an evaluation metric is that it does not adequately incorporate accuracy. A predictor may have a higher success-per-reference than another while still making more incorrect predictions, which can be costly.

Effective-Miss-Ratio (EMR): EMR is a metric that incorporates the strengths of both accuracy and success-per-reference into a single metric. It uses an adjustable parameter α that indicates the expense of an incorrect prediction

$$\text{EMR} = (N_{fr} - N_{cp} + \alpha \times N_{ip})/N_{fr}$$

where

$$N_{ip} = \text{number of incorrect predictions}$$

and

$$0.0 \leq \alpha \leq 1.0.$$

An α of 0.0 indicates no expense, in which case EMR reduces to success-per reference, while an α of 1.0 indicates that an incorrect prediction costs as much as a complete file fetch, which is more reflective of real system behavior.

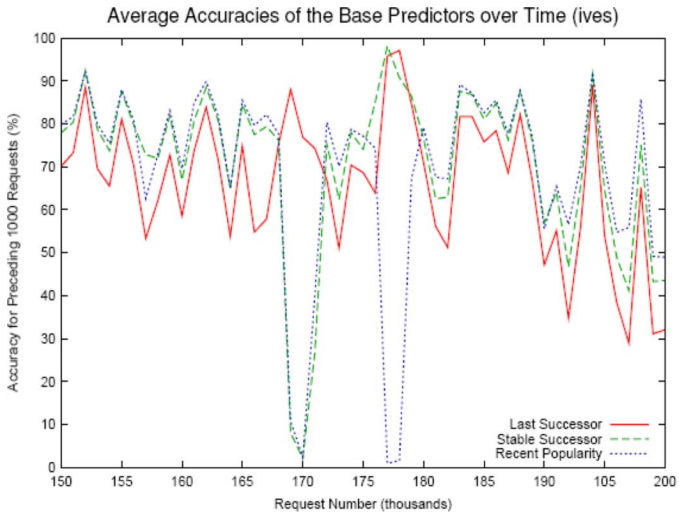

Fig. 1. Average accuracies of the base predictors over time (ives).

C. Base Predictor Analysis

To determine the viability of a predictor that dynamically selects among LS, SS, and RP, we compared the success of the base predictors over time. We found that while RP usually performed the best, as expected, there were numerous intervals in our file traces during which each of the other base predictors had the greatest success. One such interval is illustrated in Fig. 1, which compares the average accuracies of LS, SS, and RP during a short segment of the *ives* trace.

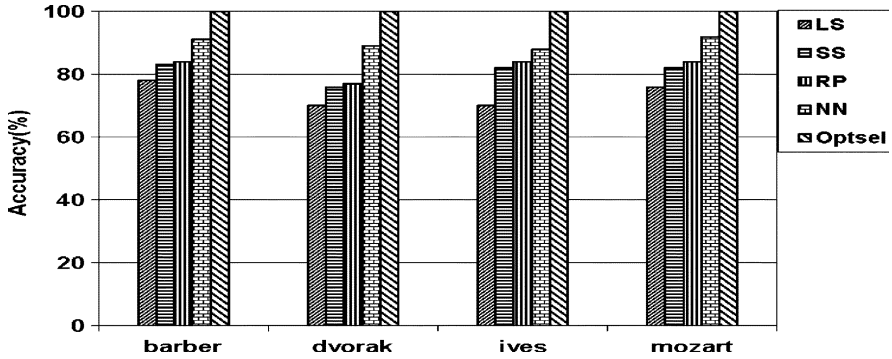

Fig. 2. Accuracy of NN-based predictor under standard configuration compared with other predictors.

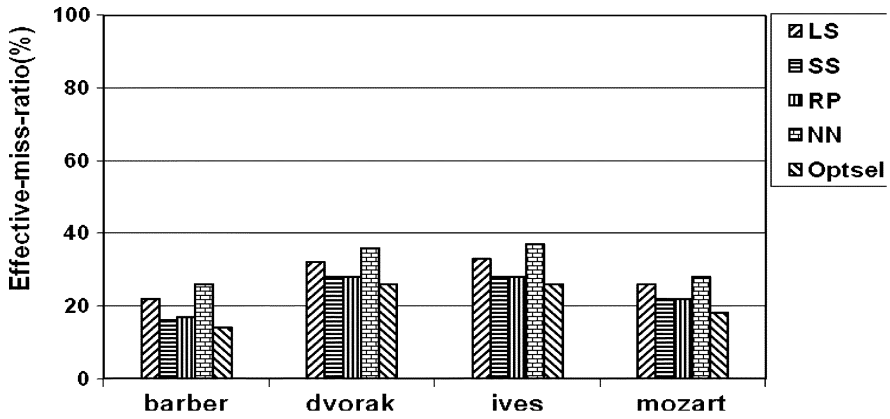

Fig. 3. EMR ($\alpha = 0.0$) under the standard configuration.

It can be seen that there are periods in which LS, SS, and RP all have the highest accuracy. As a result, we know that a predictor that dynamically selects among the three base predictors has the potential to improve upon the success of all three.

D. Simulation Results

In our simulations, we varied the parameters of our NN-based predictor in order to determine which combination of parameters produces the best results. By varying the length of the successor history, the size of the hidden layer, and the probability threshold, we found that the optimal configuration varied by file trace. However, selecting a reasonable set of parameter values and applying them with all the traces still produced good results.

We were not able to vary the training interval size in our simulations due to time constraints, as the interval size has a significant effect on the amount of time required to train the network. We experimented with the interval size using the monthlong *Mozart* trace and found no notable effect on prediction ability. Therefore, in all of our simulations, we use a fixed training interval size of 200.

To determine the maximum potential of our predictor, we implemented a simple *optimal selector* that generates a prediction for a file access only when at least one of our three base predictors is correct. In such a case, a correct prediction is issued. Of course, optimal selector cannot be implemented in a real system, as a real system lacks future knowledge.

E. Overall Evaluation

1) Untuned Results: After varying each parameter of predictor in turn, we found that a successor history of size 4, a hidden layer of size 20, and a probability threshold of 0.6 provide good results for all of the traces. We will henceforth refer to this set of parameter values as the *standard configuration*.

Figs. 2–5 compare our NN-based predictor under the standard configuration to the base predictors and optimal selector. In these and all remaining results presented, yearlong versions of the four traces were used.

As shown in Fig. 2, the proposed NN-based predictor is able to significantly improve upon the base heuristics, producing accuracies 8.14%–15.98% higher than those of RP. As a result, our predictor generated 43.63% incorrect predictions which are fewer than 53.11% of RP, which is beneficial in a system with a high misprediction cost. Unfortunately, our improved accuracy comes at the cost of reduced coverage. The resulting reduction in correct predictions is not adequately compensated by the reduction in incorrect predictions in a system in which there is no performance cost associated with a misprediction. In such a system, as Fig. 3 illustrates, any one of the base predictors would be a better choice. In most real-world systems, however, predicting the wrong file incurs a significant penalty. The effectiveness of our predictor improves as the cost associated with a bad prediction increases. Fig. 4 shows that in a system where a bad prediction has half the performance cost of a complete fetch, our predictor slightly outperforms the base predictors for *barber* and *dvorak*.

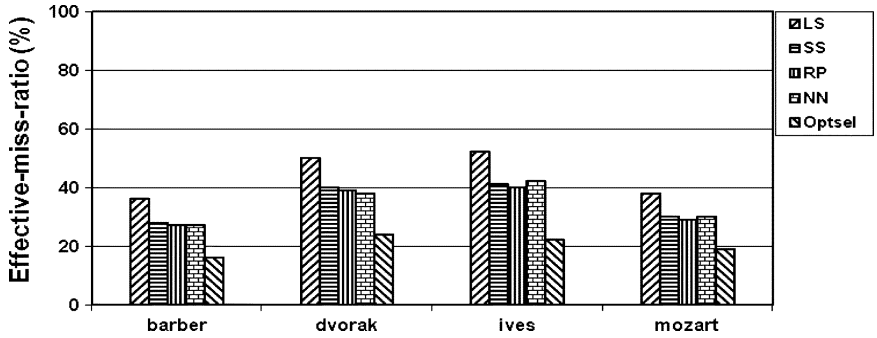

Fig. 4. EMR ($\alpha = 0.5$) under the standard configuration.

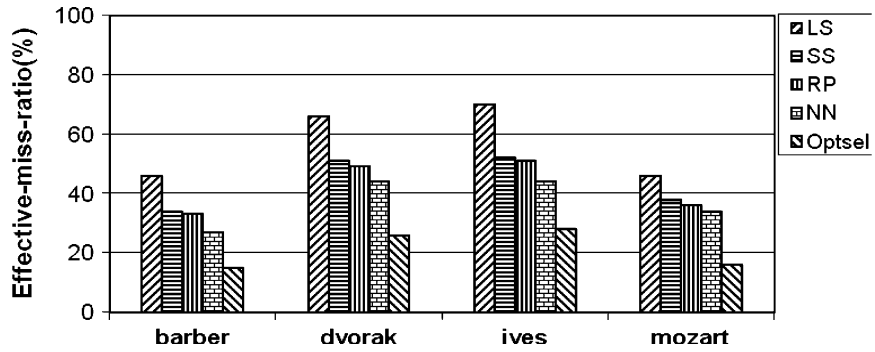

Fig. 5. EMR ($\alpha = 1.0$) under the standard configuration.

TABLE II
OPTIMAL CONFIGURATIONS TO MAXIMIZE ACCURACY

Trace Name	Success or History	Hidden Nodes	Thres hold	Accuracy
Barber	4	20	0.999	96.11%
Dvorak	4	20	0.999	94.05%
Ives	4	20	0.999	94.67%
Mozart	4	20	0.999	94.60%

TABLE III
OPTIMAL CONFIGURATIONS TO MAXIMIZE EMR ($\alpha = 0.0$)

Trace Name	Success or History	Hidden Nodes	Thres hold	Accuracy
Barber	4	32	none	81.18%
Dvorak	4	20	none	72.20%
Ives	4	36	none	71.40%
Mozart	4	24	none	78.71%

TABLE IV
OPTIMAL CONFIGURATIONS TO MAXIMIZE EMR ($\alpha = 0.5$)

Trace Name	Success or History	Hidden Nodes	Thresho ld	Accuracy
barber	4	20	0.3	74.56%
dvorak	4	20	0.3	62.40%
ives	4	20	0.2	59.66%
mozart	4	20	0.3	70.71%

TABLE V
OPTIMAL CONFIGURATIONS TO MAXIMIZE EMR ($\alpha = 1.0$)

Trace Name	Successo r History	Hidden Nodes	Thres hold	Accuracy
Barber	4	20	0.6	70.16%
Dvorak	4	20	0.6	57.48%
Ives	4	20	0.7	53.06%
Mozart	4	20	0.6	65.72%

Fig. 5 shows that our predictor improves upon the EMR of RP by 5.72%–14.71% for all of the traces when a bad prediction results in a complete fetch.

2) Tuned Results: We improved our results by modifying each of the parameters in turn and then selecting the configuration which produced the optimum results for each trace and metric. We henceforth refer to the product of this manual tuning as the *optimal configuration* for each trace and metric.

Tables II–V summarize the results of our manual tuning for each metric.

Figs. 6–9 compare the best results generated by our predictor for each metric to those of the base predictors and the optimal selector.

Our predictor, at its optimal configuration, is much more accurate than the base predictors for all four traces. As Fig. 6 shows, we improved upon the accuracy from 13.19% to 25.77%, resulting in a reduction in the number of mispredictions to 69.02% from 78.43%. Therefore, our NN-based predictor is much more beneficial than the base predictors in any system in which bad predictions are expensive. As Fig. 7 illustrates, even in a system in which a bad prediction results

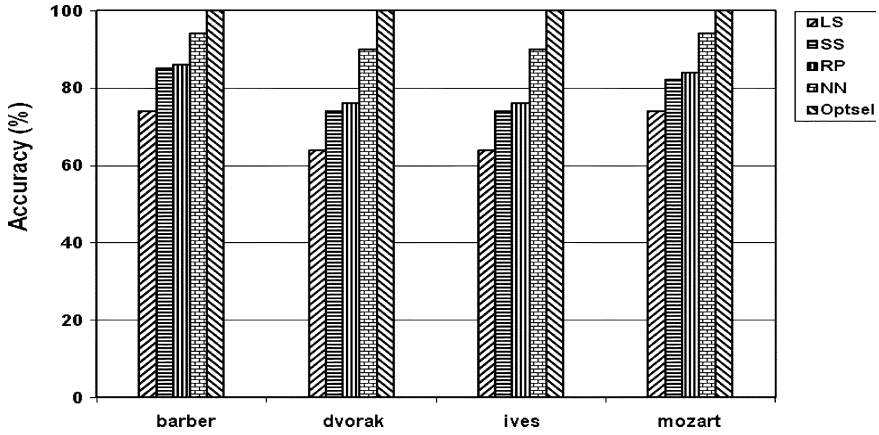

Fig. 6. Accuracy of the NN-based predictor under the optimal configuration compared with other predictors.

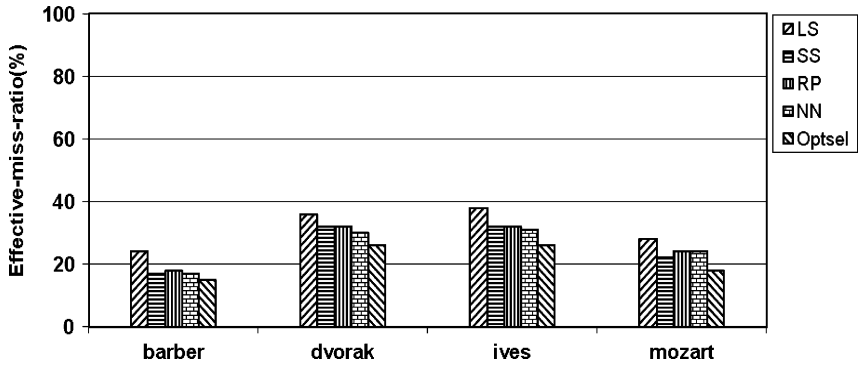

Fig. 7. EMR ($\alpha = 0.0$) under the optimal configuration.

TABLE VI
ACCURACY OF PREDICTION ON THE BASIS OF SPATIAL SPREAD

Spatial_spread	Prediction Accuracy
s = 0	52 %
s = 1	73 %
s = 2	80 %
s = 3	90 %
s = 4	90 %
s = 6	92 %
s = 13	95 %
s =14	96 %
s =15	97 %
s =17	99 %

in no performance loss, our predictor slightly outperforms the base predictors for all four traces. In the case of a system where a mispredicted file access results in a performance loss equivalent to half of a complete fetch, our predictor does marginally better, as indicated by Fig. 8. It is in a system with a very high misprediction cost; however, our predictor is most effective, as our predictor's high accuracy is its main strength. Fig. 9 shows that we are able to achieve an EMR 14.71% which is higher than 5.72% of RP for all four traces in such a case. Note that the EMR for the *barber*, *dvorak*, and *Mozart* traces under the optimal configuration is equal to those produced under the standard configuration, and that of *ives* is only slightly better. This indicates that it is possible to approximate the performance of our optimally configured predictor without manual tuning.

3) Effect of Network Parameters: We altered the successor history length, number of hidden units, and probability threshold as we ran our simulations. Here, we examine the effect of each on the performance of our NN-based predictor.

4) Number of Hidden Nodes: The size of the hidden layer in an NN can impact how effectively it learns. The size of the hidden layer is varied from 4 to 40 nodes while keeping the successor history length fixed at 4. No probability threshold is used for these simulations. Varying the number of hidden nodes has little effect on the performance of our predictor. The best and worst accuracy, success-per-reference, and EMR ($\alpha = 1$) for each trace are within 1% of each other. There is no discernible correlation between the size of the hidden layer and the performance of the predictor, as all three evaluation metrics fluctuate as the number of hidden nodes is increased.

5) Successor History Length: The successor history contains all of the information our base predictors need to make their predictions. It also is a part of the data our network uses to determine the probability that each base predictor is correct. We varied the successor history length from four to ten successors while keeping the size of the hidden layer fixed at 20. We did not use a probability threshold for these simulations. Varying

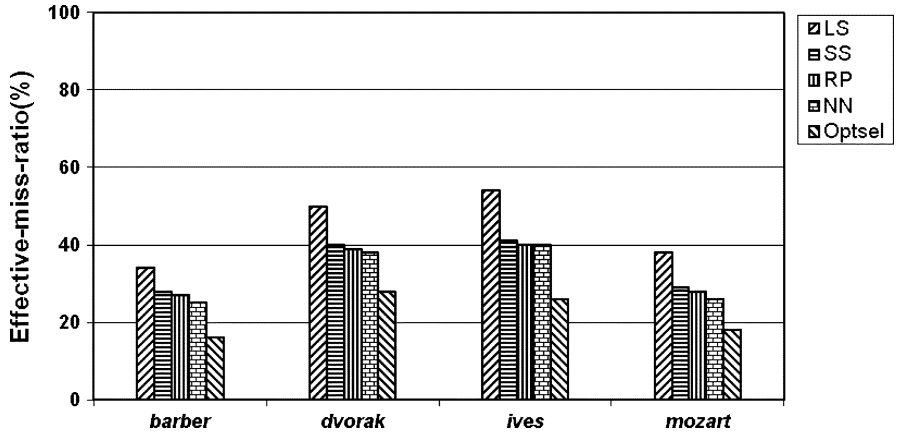

Fig. 8. EMR $(\alpha = 0.5)$ under the optimal configuration.

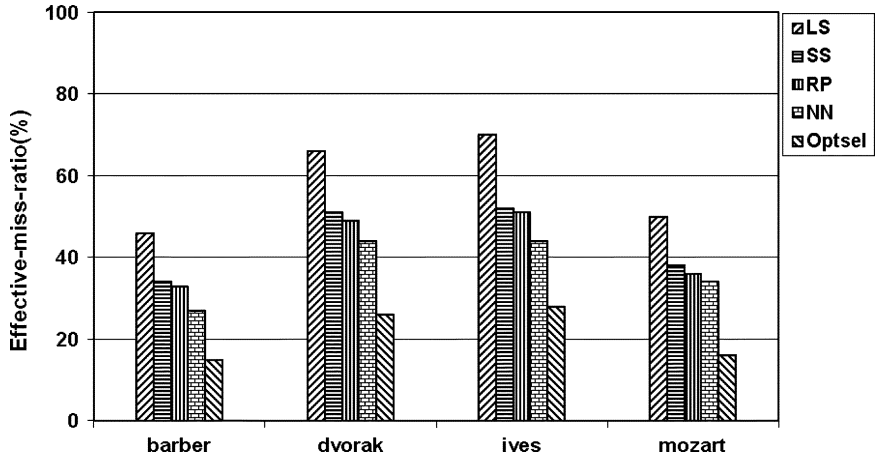

Fig. 9. EMR $(\alpha = 1.0)$ under the optimal configuration.

the number of successors maintained has little effect on the performance of our predictor. The best and worst accuracy, success-per-reference, and EMR $(\alpha = 1)$ for each trace are within 1% of each other. In addition, there is no discernible correlation between the successor history length and the performance of the predictor, as all three evaluation metrics fluctuate as the number of successors is increased.

6) Probability Threshold: The probability threshold determines how confident our network must be in a base heuristic before our NN-based predictor will select its prediction. If no probability is above the threshold, our predictor issues no prediction. We varied the probability threshold from 0.1 to 0.999 while keeping the successor history length and hidden layer size fixed at 4 and 20, respectively. As Figs. 10–12 illustrate, increasing the threshold has a significant effect on the performance of our predictor. As we increase the threshold, the accuracy increases linearly for all four traces, as shown in Fig. 10. We were able to improve the accuracy by 9.72%–18.52% by increasing the threshold from 0.1 to 0.999. This is not surprising, as we expect that predicting only when there is a high confidence in the prediction will lead to a higher percentage of correct predictions. This result provides confidence that our network is able to correctly determine the probability that each base predictor is correct at a given time.

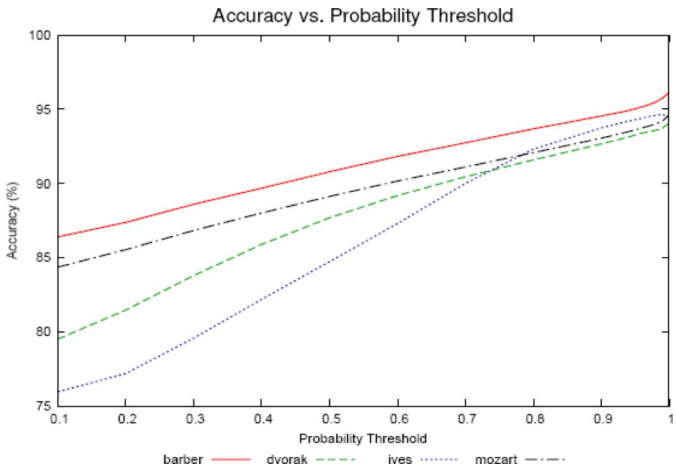

Fig. 10. Accuracy versus probability threshold.

Increasing the threshold has the secondary effect, however, of reducing coverage, and sometimes even good predictions are rejected because the network does not have enough confidence in the prediction. As a result, our success-per-reference, shown in Fig. 11, linearly decreases with an increase in the threshold up to approximately 0.85. After that, the decrease is much more

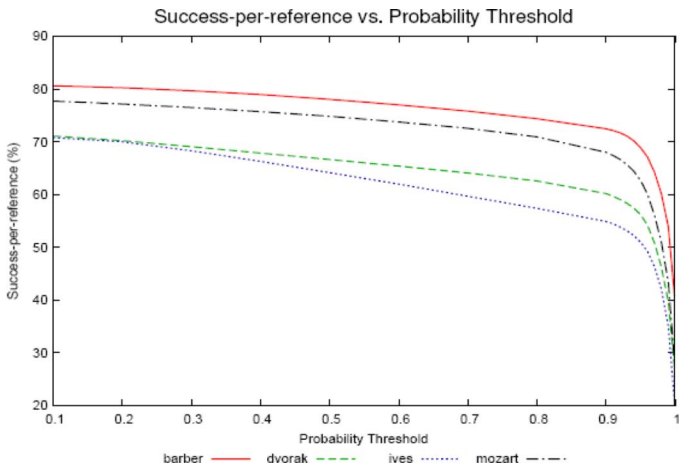

Fig. 11. Success-per-reference versus probability threshold.

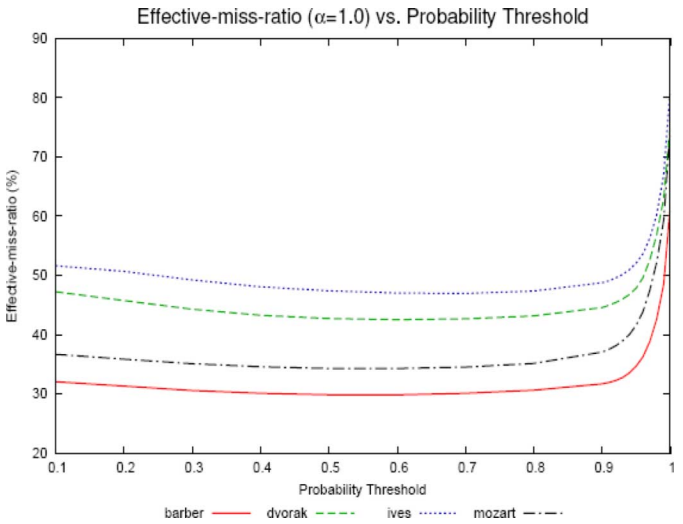

Fig. 12. EMR ($\alpha = 1.0$) versus probability threshold.

abrupt. This decrease is expected, as there is no way to ensure that only bad predictions are rejected. Therefore, using our predictor with a high threshold is optimum in an environment in which there is a high cost for misprediction, such that the reduction in incorrect predictions will be worth the corresponding loss of correct predictions. Fig. 12 shows that even in such a system, there is a threshold value above which the EMR will decrease due to the reduction in successful predictions. Therefore, it is optimum to tune our predictor to use a threshold value in the midrange between 0.0 and 1.0. For our traces, the 0.4–0.85 range is optimal as it results in very high accuracy without too much impact on success-per-reference.

In both Figs. 11 and 12, it can be seen that threshold has the most dramatic effect on the *ives* trace. We found that for *barber*, *dvorak*, and *Mozart*, a threshold of 0.6 did the best. For *ives*, a threshold of 0.7 provided the highest EMR.

By doing the above simulations, we were able to significantly improve upon the accuracy, success-per-reference, and effective-success-rate-per-reference of RP when using our NN-based predictor with the proper tuning.

We further extended our work to directly predict the next file rather than predicting the best heuristic with radial basis

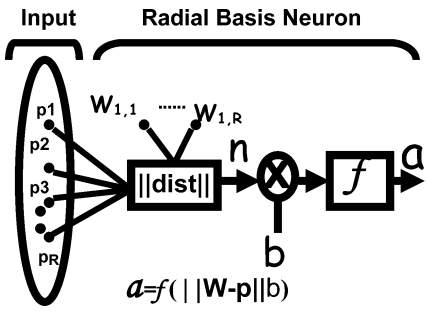

Fig. 13. Radial basis neuron.

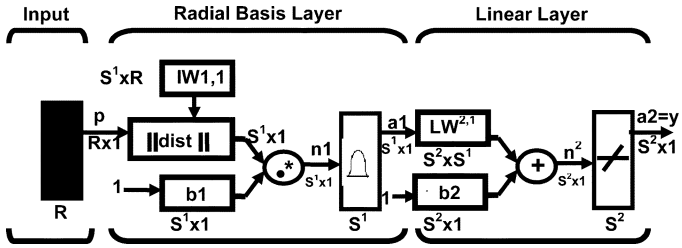

Fig. 14. RBFNN predictor.

function neural network (RBFNN), probabilistic neural network (PNN), competitive predictor, and MLP with LM backpropagation algorithm.

F. Exact-Fit RBF-Based Predictor

In this model, we used an exact-fit RBFNN as shown in Fig. 14. The advantages of such networks are twofold: the training procedures are substantially faster than the general backpropagation algorithm for training MLP networks and they do not encounter the local minima problem. The advantage of using an exact-fit RBFNN is that it generally gives us zero error on training vectors.

The transfer function for a radial basis neuron is given by

$$\text{radbas(x)} = e^{-x2}. \tag{1}$$

The RBF has a maximum of 1 when its input is 0. As the distance between w and p decreases, the output increases. Thus, a radial basis neuron (Fig. 13) acts as a detector that produces 1 whenever the input p is identical to its weight vector w. The bias b allows the sensitivity of the RBF neuron to be adjusted.

The RBF network (Fig. 14) consists of two layers: a hidden radial basis layer of S^1 neurons and an output linear layer of S^2 neurons [30], [14]. The box $||\text{dist}||$ in this figure accepts the input vector p and the input weight matrix $\text{IW}^{1,1}$, and produces the dot product of the two, giving a vector having S^1 elements. The elements are the distances between the input vector and vectors $\text{IW}^{1,1}$ formed from the rows of the input weight matrix. The bias vector b^1 and the output of are combined with the MATLAB operation.*, which does element-by-element multiplication. The output of the first layer for a feedforward network *net* is obtained with the following code:

$$a\{1\} = \text{radbas}(\text{netprod}(\text{dist}(\text{net} \cdot \text{IW}\{1,1\}, p), \text{net} \cdot b\{1\})). \tag{2}$$

The net input to the RBF transfer function is the vector distance between its weight vector w and the input vector p, multiplied by the bias b.

We have P input vectors (file traces from Mozart), T target vectors (set to 1), and spread constant as SPREAD. The RBF for a neuron has a center and a radius (also called a SPREAD). The radius is different for each neuron. With larger SPREAD, neurons at a distance from a point have a greater influence. Small SPREAD is very selective in nature where as large spread is not very selective.

Each bias in the first layer of this network is set to 0.8326/SPREAD. It gives RBFs that cross 0.5 at weighted inputs of $+/-$SPREAD. This determines the width of an area in the input space to which each neuron responds. If SPREAD is 4, then each radial-basis neuron will respond with 0.5 or more to any input vectors within a vector distance of 4 from their weight vector. SPREAD is made large enough so that neurons respond strongly to overlapping regions of the input space.

The second-layer weights $\mathrm{IW}_{2,1}$ and biases b^2 are found by simulating the first layer

$$[\mathrm{IW}_{2,1} b^2] X[a^1; \mathrm{ones}] = T \tag{3}$$

outputs a^1, and then solving the following linear expression.

In (3), we know the inputs to the second layer and the target (T), and the layer is linear.

We used the following code to calculate the weights and biases of the second layer to minimize the sum-squared error:

$$W_b = T/[P; \mathrm{ones}\,(1, Q)]. \tag{4}$$

Here, in (3), W_b contains both weights and biases, with the biases in the last column. We have a problem with C constraints (input/target pairs) and each neuron has $C+1$ variable (the C weights from the C radial-basis neurons, and a bias). A linear problem with C constraints and more than C variables has an infinite number of zero error solutions. Hence, we find that the sum-squared error will always be 0. The only condition we have to meet is to make sure that SPREAD is large enough so that the active input regions of the radial-basis neurons overlap enough so that several radial-basis neurons always have fairly large outputs at any given moment.

According to "spatial locality of reference" the likelihood of referencing a storage location is greater if a storage location near it has been recently referenced. On the basis of this principle, we set a spatial_spread value $(\boldsymbol{s})$. Spatial_spread value determines the total number of files to be fetched into the system along with the target file. Suppose the target file is x and $\boldsymbol{s} = 3$, then we have to fetch the $x-3, x-2, x-1, x, x+1, x+2, x+3$ files into the cache.

We know that the general size of a file in an ext3 file system is 128 B. If the spatial_spread value is set to be 3, then we need to fetch 896 B into the cache simultaneously. Hence, the performance of this model is determined by the bandwidth of the disk-to-memory communication channel. So depending on the system configuration, we can set the spatial_spread value for effective speedup.

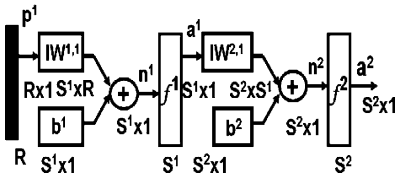

Fig. 15. MLP-based predictor. R = number of elements in input vector; S^1 = number of neurons in layer 1; S^2 = number of neurons in layer 2; $a_1 1$ = radbas $(\|\mathrm{IW}^{1,1} - p\| b_1 I)$; $a2$ = purelin $(\mathrm{LW}^{2,1} a1 + b2)$; $a_1 1$ is the ith element of $a1$, where $\mathrm{IW}^{1,1}$ is a vector made of the ith row of $\mathrm{IW}^{1,1}$

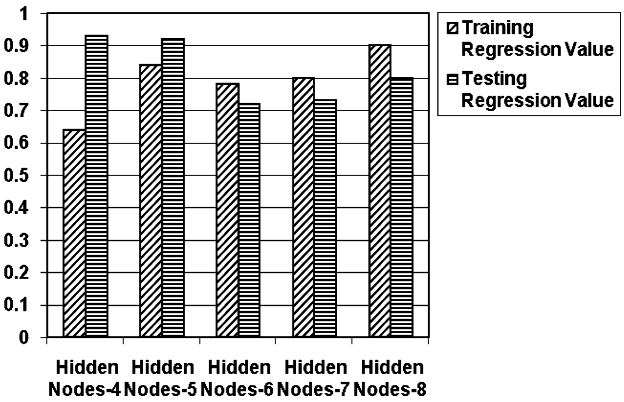

Fig. 16. Regression value versus hidden nodes in MLP.

G. MLP-Based Predictor

This model is an extension of the perceptron model discussed in [28]. In this model, we use multiple-layer feedforward networks instead of a perceptron for prediction which is shown in Fig. 15. Here the superscript 1 denotes hidden layer and superscript 2 denotes the output layer. P is the input vector. An output of a three-layer MLP network is defined by [14] (Fig. 15):

$$a_k^2 = f^2 \left(\sum_{j-1}^{S^1} w^2 jk f^1 \left(\sum_{j-1}^{R} w_{ij}^1 p_i + b_j^1 \right) + b_k^2 \right), \quad k = 1 \text{ to } S^2 \tag{5}$$

where superscript 1 denotes hidden layer and superscript 2 denotes output layer. R, S^1, and S^2 illustrate the numbers of the input, hidden, and output units, respectively. Also f, w, and b represent transfer function, synaptic weight parameter, and bias, respectively. In Fig. 15, a three-layer perceptron network with R input neurons, S^1 hidden neurons by tan-sigmoid transfer function, and one output neuron by linear transfer function are selected.

To train the network, we use the LM backpropagation algorithm. Considering the space complexity and the time complexity of the MLP, we have taken the history length as 10, i.e., we use the last ten file accesses to predict the file which would be up to five accesses ahead. These values have been taken completely on heuristic basis. Fig. 16 shows the results obtained during the experiment as we varied the number of hidden nodes for the network.

The regression value as specified in the plot shows the relation between the target output from 0 to 1 where 1 signifies exact match between them and 0 shows a random relation.

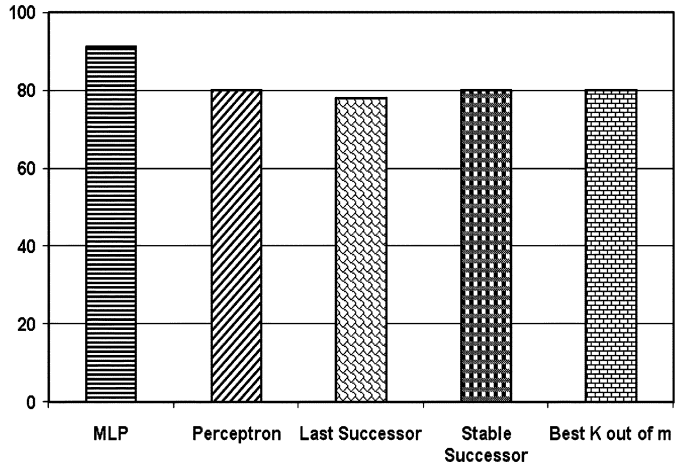

Fig. 17. Success rate of various predictors (with respect to the Mozart file trace).

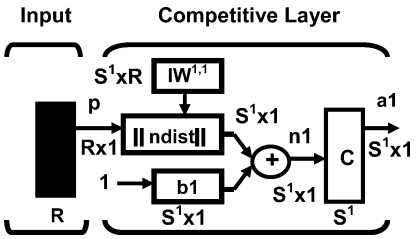

Fig. 18. Competitive predictor.

Hence, from Fig. 16, we observe that the most optimal number of nodes for MLP-based predictor is 5.

Fig. 17 shows the comparison of success rate among MLP with trainable LM backpropagation algorithm, simple perceptron, LS, SS, and best k out of m predictors. The simulation is carried out upon the data sets (file traces) collected from Mozart. From this plot, we conclude that success rate for MLP with trainable LM backpropagation algorithm predictor is 91.2% and is the best among the above predictors.

H. Competitive Predictor

In this model, we put forward a model which uses pattern matching using competitive network. The architecture of the network is shown in Fig. 18. The "*dist*" in the figure accepts the input vector p and the input weight matrix $\mathrm{IW}^{1,1}$, and produces a vector having S^1 elements. The elements are the negative of the distances between the input vector and vectors ${}_i\,\mathrm{IW}^{1,1}$ formed from the rows of the input weight matrix. The net input n^1 of a competitive layer is computed by finding the negative distance between input vector p and the weight vectors and adding the biases b. If all biases are zero, the maximum net input a neuron can have is 0. This occurs when the input vector p equals that neuron's weight vector. The competitive transfer function accepts a net input vector for a layer and returns neuron outputs of 0 for all neurons except for the winner, the neuron associated with the most positive element of net input n^1. The winner's output is 1. If all biases are 0, then the neuron whose weight vector is closest to the input vector has the least negative net input and, therefore, wins the competition to output $a1$.

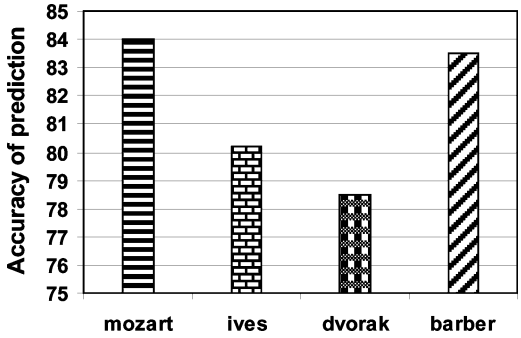

Fig. 19. Accuracy of competitive predictor.

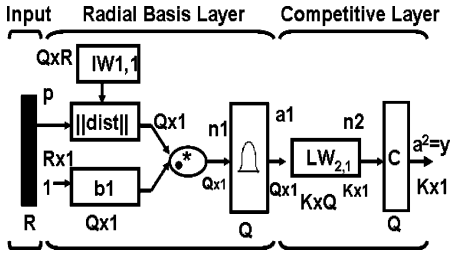

Fig. 20. Architecture of probabilistic predictor. R = number of elements in input vector; a_i1 = radbas ($\|{}_i\mathrm{IW}_{1,1} - p\|\mathrm{bi1}$); $a2$ = compete ($\mathrm{LW}_{2,1}a1$); a_i1 is the ith element of $a1$ where ${}_i\mathrm{IW}_{1,1}$ is a vector made of the ith row of $\mathrm{IW}_{1,1}$; Q = number of input/target pairs; = number of neurons in layer 1; K = number of classes of input data = number of neurons in layer 2.

This model is used to structurally classify the file traces and predicts the target file. The training utilizes competitive learning with Kohonen spread as 1.00.

We obtained the following accuracy values for different file traces as shown in Fig. 19.

I. Probabilistic Predictor

In this model, we used PNN [26] for prediction. The spread constant for this network was taken as 1.0. When an input is presented to this model, the first layer computes distances from the input vector to the training input vectors, and produces a vector whose elements indicate how close the input is to a training input. The second layer sums these contributions for each class of inputs to produce a vector of probabilities as its net output.

A competitive transfer function on the output of the second layer picks the maximum of these probabilities, and produces a 1 for that class and a 0 for the other classes. Fig. 20 represents the architecture for this system.

We obtained the following accuracy values for different file traces as shown in Fig. 21 using generalized Fisher (GF) training algorithm.

V. Summary of Simulations

In this study, we have used NNs for file predictions. In all the above NN predictors, we have simulated for 1000 epochs with 0.3 million DFS file traces (inodes) as training set data and 0.70 million as testing set data, out of the total available 1.0 million DFS file traces (inodes). We found the optimum file prediction accuracy after the complete execution of the fifth file as our assumption. In other words, the worst case prediction time is of the maximum time taken by a system to process completely any

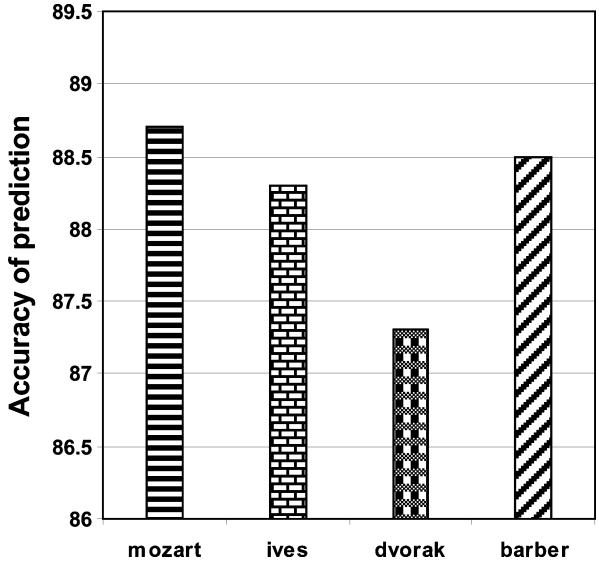

Fig. 21. Accuracy of probabilistic predictor.

five files. This time is elapsed to predict dynamically from sixth file onwards. This is considered as the worst case condition and accordingly the prediction accuracies have been plotted in various figures. We have achieved a good improvement in prediction accuracy over the existing predictors as evident in all the above cases. Hence, whether these algorithms are implemented in a system prediction accuracy can be achieved within few milliseconds i.e., the time taken to process completely any five files.

VI. Conclusion

In this paper, we introduced five different predictors for intelligent and efficient file prediction.

Three simple algorithms, which have proved effective in predicting which file will be accessed next, based on observation of past accesses, are LS, SS, and RP. Although RP is generally considered the optimum of the three, there are times when LS or SS is more accurate. Our NN-based file access predictor is currently the optimum for a given traces of files and selects that predictor to predict the next file access. We were able to significantly improve upon the accuracy, success-per-reference, and effective-success-rate-per-reference of RP when using our NN-based predictor with proper tuning. We reduced the incorrect predictions from 53.11% to 43.6% using a standard configuration for all traces in comparison to RP. With manual tuning with each trace, we were able to improve upon the misprediction rate from 78.43% to 69.02%. Due to our better accuracy, we achieved an ESR 14.71% which is higher than 5.71% of that of RP.

Simulating through various NN, we observe that the exact-fit RBF-based predictor has very high efficiency if we have a high bandwidth between the cache and the file source. With the increase in spatial_spread, the data flow between the cache and the file source increases. This increases the efficiency of the predictor, but it will be bottlenecked if the data transfer rate between cache and the file source is low. Hence, this model can be used in high-end systems where generally a user performs a similar kind of task.

Experimental results for the MLP-based predictor show that it can make approximately 91% correct predictions. We have used the LM backpropagation algorithm for learning the network. Since the BP algorithms are computation intensive, we attempt to predict the files which will be accessed up to five files accessed ahead. This model is better suitable for systems having good computational capability.

We also presented the probabilistic and competitive predictors. Experimental results show that competitive predictor is more efficient for servers where maximum system calls are generated. Overall analysis shows that the probabilistic predictor gives better percentage of correct predictions than competitive learning. These models are most suitable for workstations where we have limited resources to deal with.

VII. Future Work

More work is still needed to reduce the number of incorrect predictions and to evaluate the impact of multilevel caching on the performance of our predictors.

Acknowledgment

The authors would like to thank all the anonymous reviewers for their useful suggestions.

References

[1] A. Amer, "Predictive data grouping using successor prediction," Ph.D. dissertation, Dept. Comput. Sci., Univ. California, Santa Cruz, CA, 2002.

[2] A. Amer and D. D. E. Long, "Dynamic relationships and the persistence of pairings," in *Proc. IEEE Int. Workshop Wireless Netw. Mobile Comput.*, Mesa, AZ, Apr. 2001, pp. 502–507.

[3] A. Amer and D. D. E. Long, "Noah: low-cost file access prediction through pairs," in *Proc. 20th IEEE Int. Performance Comput. Commun. Conf.*, Phoenix, AZ, Apr. 2001, pp. 27–33.

[4] A. Amer, D. D. E. Long, J. -Fran, C. Pâris, and R. C. Burns, "File access prediction with adjustable accuracy," in *Proc. 21st IEEE Int. Performance Comput. Commun. Conf.*, Phoenix, AZ, Apr. 2002, pp. 131–140.

[5] A. Amer, D. E. Long, and R. C. Burns, "Group-based management of distributed file caches," in *Proc. 17th IEEE Int. Conf. Distrib. Comput. Syst.*, Vienna, Austria, Jul. 2002, pp. 525–534.

[6] I. Ari, A. Amer, R. Gramacy, E. L. MillerScott, S. A. Brandt, and D. D. E. Long, "ACME: Adaptive caching using multiple experts," in *Proc. Workshop Distrib. Data Structures 4*, Pâris, France, 2002, vol. 14, pp. 143–158.

[7] K. S. Brandt, "Using multiple experts to perform file prediction," M.S. thesis, Dept. Comput. Sci., Univ. California, Santa Cruz, CA, 2004.

[8] K. S. Brandt, D. D. E. Long, and A. Amer, "Predicting when not to predict," in *Proc. 12th Annu. IEEE/ACM Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Volendam, The Netherlands, Oct. 2004, pp. 419–426.

[9] W. Chen, C. F. Eick, J. -Fran, and C. Pâris, "A two-expert approach to file access prediction," in *Proc. 3rd Int. Inf. Telecommun. Technol. Symp.*, Sao Carlos, SP, Brazil, Dec. 2004, pp. 139–145.

[10] M. Cherniack, E. F. Galvez, M. l. J. Franklin, and S. Zdonik, "Profile-driven cache management," in *Proc. 19th IEEE Int. Conf. Data Eng.*, Bangalore, India, Mar. 1993, pp. 645–656.

[11] C. W. Dawson, R. L. Wilby, C. Harpham, M. R. Brown, E. Cranston, and E. J. Darby, "Modeling Ranunculus presence in the rivers test and itchen using artificial neural networks," in *Proc. 5th Int. Conf. Geo Comput.*, U.K., Aug. 2000.

[12] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[13] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *Proc. Summer USENIX Tech. Conf.*, Boston, MA, Jun. 1994, pp. 197–207.

[14] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. New York: Prentice-Hall, 1999.

[15] T. M. Kroeger and D. D. E. Long, "Design and implementation of a predictive file prefetching algorithm," in *Proc. USENIX Annu. Tech. Conf.*, Boston, MA, Jun. 2001, pp. 105–118.
[16] T. M. Kroeger, "Predictive file system actions from reference patterns," M.S. thesis, Univ. California, Santa Cruz, CA, 1997.
[17] T. M. Kroeger and D. D. E. Long, "Predicting file-system actions from prior events," in *Proc. USENIX Annu. Tech. Conf.*, San Diego, CA, Jan. 1996, pp. 319–328.
[18] T. M. Kroeger and D. D. E. Long, "The case for efficient file access patterns modeling," in *Proc. 7th IEEE Workshop Hot Topics Oper. Syst.*, Rio Rico, AZ, Mar. 1999, pp. 14–19.
[19] H. Lei and D. Duchamp, "An analytical approach to file prefetching," in *Proc. USENIX Annu. Tech. Conf.*, Anaheim, CA, Jan. 1997, pp. 305–318.
[20] L. Mummert and M. Satyanarayan, "Long term distributed file reference tracing: Implementation and experience," Schl. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CS-94-213, Nov. 1994.
[21] M. S. Obaidat and H. Khalid, "Estimating neural networks-based algorithm for adaptive cache replacement," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 28, no. 4, pp. 602–611, Aug. 1998.
[22] M. L. Palmer and S. B. Zdonik, "Fido: A cache that learns to fetch," in *Proc. 17th ACM Int. Conf. Very Large Databases*, Barcelona, Spain, Sep. 1991, pp. 255–264.
[23] J. F. Paris, A. Amer, and D. D. E. Long, "A stochastic approach to file access prediction," in *Proc. Int. Workshop Storage Network Architecture Parallel I/Os*, New Orleans, LA, Sep. 2003, pp. 36–40.
[24] J. H. Park, S. C. Jung, C. Zhang, and K. T. Chong, "Neural network hot spot prediction algorithm for shared web caching system," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2005, vol. 3399, pp. 795–806.
[25] P. K. Patra, M. Nayak, S. K. Nayak, and N. K. Gobbak, "Probabilistic neural network for pattern classification," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Honolulu, HI, May 2002, pp. 1200–1205.
[26] H. H. Patterson, G. A. Gibon, and M. Satyanarayanan, "A status report on research in transparent informed prefetching," *ACM Oper. Syst. Rev.*, vol. 27, no. 2, pp. 21–34, Apr. 1993.
[27] N. Ravichandran and J. F. Paris, "Making early predictions of file accesses," in *Proc. 4th Int. Inf. Telecommun. Technol.*, Dec. 2005, pp. 122–129.
[28] P. Shah, J. F. Pâris, A. Amer, and D. D. E. Long, "Identifying stable file access patterns," in *Proc. 21st IEEE Symp. Mass Storage Syst.*, Apr. 2004, pp. 159–163.
[29] Y. Shirvani, M. Hayati, and R. Moradian, "Multilayer neural network with novel unsupervised training method for numerical solutions of the partial differential equations," in *Proc. Appl. Soft Comput. Sci. Direct*, Feb. 2009, pp. 20–29.
[30] E. Shriver, C. Small, and K. A. Smith, "Why does file system prefetching work?," in *Proc. USENIX Annu. Tech. Conf.*, Monterey, CA, Jun. 1999, pp. 71–83.
[31] W. Sun, J. Shu, and W. Zheng, "Dynamic file allocation in Storage area Networks with Neural network prediction," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2004, vol. 3174, pp. 719–724.
[32] C. Tait and D. Duchamp, "Detection and exploitation of file working sets," in *Proc. 11th Int. Conf. Distrib. Comput. Syst.*, Arlington, TX, May 1991, pp. 2–9.
[33] W. Tian, B. Choi, and V. Phoha, "An adaptive web cache access predictor using neural network," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2002, vol. 2358, pp. 450–459.
[34] G. A. S. Whittle, "A hybrid scheme for file system reference prediction," M.S. thesis, Dept. Comput. Sci., Univ. Houston, Houston, TX, 2002.
[35] G. A. S. Whittle, J. Frann, C. Pâris, A. Amer, D. D. E. Long, and R. Burns, "Using multiple predictors to improve the accuracy of file access predictions," in *Proc. 20th IEEE Symp. Mass Storage Syst.*, San Diego, CA, Apr. 2003, pp. 230–240.
[36] T. Yeh, J. Arul, K.-H. Tien, I.-F. Chen, and J.-S. Wu, "Improving the system performance by a dynamic file prediction model," in *Proc. Int. Conf. Comput. Design/Conf. Comput. Nanotechnol.*, Las Vegas, NV, Jun. 2006, pp. 182–188.

Prashanta Kumar Patra received the B.Tech. degree in electronics engineering from SVRCET (NIT), Surat, India, the M.Tech. degree in computer engineering from the Indian Institute of Technology (IIT), Kharagpur, India, and the Ph.D. degree in computer science from Utkal University, India, in 1986, 1994, and 2003 respectively.

Since March 1988, he has been in teaching profession. Currently, he is a Professor and Head of the Department of Computer Science and Engineering, College of Engineering and Technology, Bhubaneswar, India. He has published many papers in national and international journals and conferences in the areas of neural networks and pattern recognition, which are the subjects of his research interest.

Muktikanta Sahu received the B.Tech. degree in computer science and engineering from Biju Patnaik University of Technology, Orissa, India, in 2003 and the M.Tech. degree in computer science and information technology from College of Engineering and Technology, Bhubaneswar, India, in 2007.

He was a faculty member in Computer Science and Engineering Department, College of Engineering and Technology, Bhubaneswar, India, during 2004–2009. Currently, he is working as a faculty member of the International Institute of Information Technology (IIIT), Bhubaneswar, India. His primary research interests are neural networks and distributed systems.

Subasish Mohapatra received the B.Tech. degree in computer science and engineering from Biju Patnaik University of Technology, Orissa, India, in 2003 and the M.Tech. degree in computer science and information technology from College of Engineering and Technology, Bhubaneswar, India, in 2007.

He was a faculty member at the Computer Science and Engineering Department, College of Engineering and Technology, Bhubaneswar, India, during 2004–2009. Currently, he is an Assistant Professor at the Department of Computer Science and Engineering at the Institute of Technical Education and Research, Bhubaneswar, India. His research interests include image processing and soft computing.

Ronak Kumar Samantray received the B.Tech. degree in computer science and engineering from College of Engineering and Technology, Bhubaneswar, India, in 2008.

Since then he has been working with Microsoft India (R& D) Pvt. Ltd., Hyderabad, India. His primary research interests are centered around information retrieval, artificial intelligence, information security, and theory of computation.