Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

A NEAR OPTIMAL APPROACH TO QUALITY OF SERVICE DATA REPLICATION SCHEDULING

Kevin Adams

Naval Surface Warfare Center Dahlgren Division
17320 Dahlgren Road
Dahlgren, VA 22448, U.S.A.

Denis Gračanin

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061, U.S.A.

Dušan Teodorović

Department of Civil and Environmental Engineering
Virginia Polytechnic Institute and State University
Falls Church, VA 22043, U.S.A.

ABSTRACT

This paper describes an approach to real-time decision-making for quality of service based scheduling of distributed asynchronous data replication. The proposed approach addresses uncertainty and variability in the quantity of data to replicate over low bandwidth fixed communication links. A dynamic stochastic knapsack is used to model the acceptance policy with dynamic programming optimization employed to perform offline optimization. The obtained optimal values of the input variables are used to build and train a multi-layer neural network. The obtained neural network weights and configuration can be used to perform near optimal accept/reject decisions in real-time. Off-line processing is used to establish the initial acceptance policy and to verify that the system continues to perform near-optimally. The proposed approach is implemented via simulation enabling the evaluation of a variety of scenarios and refinement of the scheduling portion of the model. The preliminary results are very promising.

1 INTRODUCTION

Data replication is the act of creating and maintaining multiple copies of data. Data are replicated to enhance the dependability of the system, enhance performance or both. Performance is enhanced via locality. Dependability can be enhanced by creating redundant copies of the data.

Replication can be deployed at the application level, the common services level (e.g. a distributed database) or as a distributed system (a hypermedia system or a file system). Data replication models must determine the replicas placement, updates propagation between replicas and how to keep the replicas consistent. How the model accomplishes these tasks greatly influences the performance and scalability of the solution.

The support for network file systems on slow networks can be provided in several different ways but the general solution is to push the data as close as possible to the site where the data is used and to require only remote communications for updates.

There are two primary update strategies for files in a remote file system: logical and physical. The logical update strategy is based on the file abstraction. The strategy interprets the file system meta-data and writes the file to the remote file system. The current solutions reduce bandwidth usage by relaxing file consistency constraints when used over low bandwidth connections. The physical update strategy duplicates the physical medium on which the files are stored without interpretation. Block level updates are then made when updating a file. Physical update strategies are common on high-end data storage products for server-to-server operations, but have also been implemented for increased disk performance over high-speed links such as fiber channel.

The research described in the paper looks to develop a solution for one-way data replication to a read-only replica when the bandwidth available for replication is low when compared to the bandwidth of the storage systems. The proposed solution looks to maximize bandwidth utilization by exploiting commonality between replicas as in dependent replicas and provide a method where Quality of Service (QoS) guarantees can be defined and enforced. The solution is comprised of three main components: replication request, scheduling, and replication. A simulation enables refinements of the scheduling solution along with the evaluation of a variety of usage scenarios.

The remainder of the paper is organized as follows. Section 2 provides an overview of related work and existing replication algorithms. Section 3 describes the proposed QoS data replication model. Section 4 describes two case studies. Section 5 concludes the paper.

2 REPLICATION ALGORITHMS

A number of file systems have properties that help them tolerate high network latency. The Andrew File System (AFS) (Howard et al. 1988) uses server callbacks to inform clients when other clients have modified cached files. Thus, users can often access cached AFS files without requiring any network traffic. Leases (Gray and Cheriton 1989) are a modification to callbacks in which the server's obligation to inform a client of changes expires after a certain period of time. Leases' relaxing of traditional consistency guarantees reduces the state stored by a server, frees the server from contacting clients who have not touched a file in a while, and avoids problems when a client to which the server has promised a callback has crashed or gone off the network. The NFS4 protocol (Shepler et al. 2000), the most common distributed file system, supports traditional consistency guarantees. It reduces network round trips by batching operations.

The CODA file system (Kistler and Satyanarayanan 1992) evolved from AFS and supports slow networks and even disconnected operation. Changes to the file system are logged on the client and written back to the server in the background when there is network connectivity. CODA requires the client have a copy of the files to be updated before remote or disconnected operations are performed. CODA provides weaker-than-traditional consistency guarantees, allowing update conflicts, which users may need to resolve manually. CODA saves bandwidth because it avoids transferring files to the server when they are deleted or overwritten quickly on the client.

Rsync (Tridgell 2000) is a more efficient replacement for the rcp utility. The rsync utility copies a file or directory tree over the network onto another directory tree containing similar file(s). Rsync has proven useful for updating things like Internet mirror sites. Rsync saves bandwidth by exploiting commonality between files by considering only two files at a time. The recipient breaks its file into non-overlapping, contiguous, fixed-sized blocks and transmits hashes of those blocks to the sender. The sender in turn begins computing the hashes of all (overlapping) blocks. If any of sender's hashes match one of the recipient's hashes, the sender avoids sending the corresponding block, instead the sender tells the recipient were to find the block. An alternative algorithm was proposed in the implementation of a low-bandwidth network file system (LBFS) (Muthitacharoen, Chen, and Mazières 2001). LBFS considers only non-overlapping chunks of files and avoids sensitivity to shifting file offsets by setting chunk boundaries based on file contents, rather than on position within a file. Insertions and deletions therefore only affect the surrounding chunks.

DRDB (Reisner 2001) is an open source device driver, which allows the construction of mirrors over TCP. The connection is dedicated as to provide adequate bandwidth for the mirror synchronization. Mirror synchronization can be 1-safe, 2-safe or asynchronous.

A different approach is the synchronization of mobile devices. SyncML <www.syncml.org> is an industry initiative to develop a single synchronization protocol that works over wireless and wired networks supporting high latency, high cost, limited bandwidth and low reliability. It supports arbitrary networked data and makes use of existing Internet and Web Technologies, i.e., the Extensible Markup Language (XML), and Multimedia Internet Mail Exchange (MIME).

The algorithms for distributed replication require concurrency control protocols to ensure the serialization (Bernstein, Hadzilacos, and Goodman 1987) of updates. One-copy serialization (Bernstein and Goodman 1983) is the correctness criterion for replicated data, a requirement that ensures the performance of logical data operations are reflected on the physical copies of the data even in the event of failures. Distributed data replication algorithms can be generally categorized into two families of protocols, Read One Write All (ROWA) and Quorum Consensus (QC). In ROWA, each replica is updated and the updates occur in the same order as on the primary. Examples of ROWA algorithms are Read One Write All Available (Bernstein and Goodman 1984), Primary Copy Read One Write All (Alsberg and Day 1976) and True Copy Token Read One Write All (Minoura and Wiederhold 1982). The family of ROWA protocols favor read operations by allowing them to proceed with only one copy, while requiring write operations to be out in up to all the replicas.

Voting or QC algorithms allow writes to be recorded only at a subset of the available sites. The subset of sites to be written is known as a write quorum. QC algorithms also require that a read query a subset of the sites, which is guaranteed to overlap the write quorum. This subset is known as the read quorum. Examples of QC algorithms are Uniform Majority QC (Thomas 1979), Weighted Majority QC (Gifford 1979) and Random Weights (Kumar 1991). A more comprehensive coverage of data replication algorithms can be found in Helal, Heddaya, and Bhargava (1996).

The distributed replication algorithms discussed provide a system-wide consistent view of data in the presence of concurrency. If concurrency is not required for a specific application, the consistency constraints can be relaxed. An example of such a special case is where there is a single read-only replica with one-way synchronization. This type of replication is common for Disaster Recovery.

The one-way synchronization is from the primary data source to the replica. Implementations for this case of replication have occurred over data protocols, such as Fiber

Channel or SCSI or over LAN protocols such as TCP/IP and can replicate the entire disk or logical unit number (LUN), without regard to the structure of the data or replicate file or record oriented data. A software or hardware based solution can be used to perform the replication with the basic replication schemes being server or controller based. Replication solutions include server-based, application-level, filesystem, driver-based, and controller-based.

When performing a replication, the replica can be either dependent or independent. An independent replica is stand-alone image or copy of the data at a single point in time. Independent replicas are also known as mirrors. An independent replica will require the identical storage as the primary image. A dependent replica tracks only data that has been over-written and are typically point-in-time images. Dependent replicas require additional storage that is roughly equivalent to the amount of data being written. Hitz, Lau, and Malcolm (1994) provides further discussion of dependent replicas called snapshots in his discussion of Network Appliance's Write Anywhere File Layout (WAFL) design. Figure 1 shows creation and update of a dependant replica.

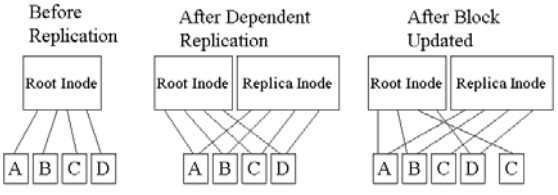

Figure 1: Dependent Replica Creation and Update

Dependent replicas have the advantages of requiring less storage and faster replication but they have two problems. First, if the primary image develops a problem, such as a device going off-line, all of the images become unavailable. Secondly, if the amount of data being written exceeds the capacity of the replica data repository the dependant copy is no longer valid.

Patterson et al. (2002) demonstrated two key concepts for one-way read-only replications. The first one, the reduction in bandwidth, is provided by asynchronous verses synchronous updates. The reduction range of his demonstration was 52% to 98% with an average of 78%. The second, block level updates require substantially less bandwidth than file level updates. His experiments found reductions of 48% and 39% in two different replicas. The time required for the replication was 3.5 and 9 times longer for the file replication. The study used Network Appliance filers with the Snapmirror replication utility.

When bandwidth is restricted, the time delays for synchronous replication quickly become prohibitive. Asynchronous replication allows queuing and scheduling of updates thus defining the recovery point objective (RPO) of the data in the replica. One of the key issues to resolve is the appropriate RPO for each data item as replication requirements will differ throughout an organization and the limited bandwidth makes it infeasible to replicate all data to the shortest RPO. While common, it is undesirable to define the RPO based on the available bandwidth. A better solution is to fine-tune the replication to better reflect and adapt to the current and changing requirements.

3 QOS DATA REPLICATION MODEL

The proposed QoS based data replication model is comprised of three main components: replication request, scheduling of replications and the actual replication (Figure 2). A primary server hosts the data to be replicated. The replication objects on the primary server are identified. Request for replication are made to the replication service. The replication service schedules and controls the replication process. The replication process is controlled via messages between the replication service and the replica server. A replication protocol performs the replication between the primary server and replica server.

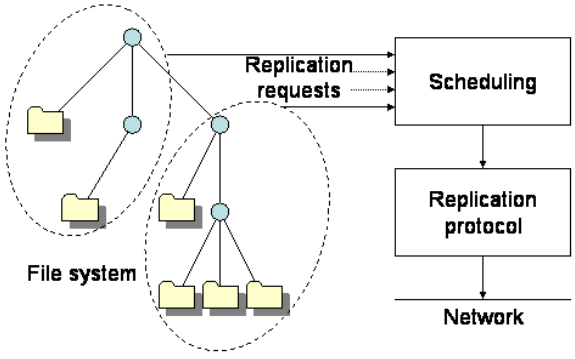

Figure 2: QoS Based Data Replication

The granularity of the replication is at the file level and is defined by the replication object. Replication objects are created based on the services provided within a system and are user defined. Replication objects contain one or more files. Data files associated with the services are what is to be replicated. The attributes associated with a replication object include timing requirements and a priority. The timing requirements define the interval of replication. The priority is a QoS parameter, a relative assessment of how important it is to have the data replicated during a given interval. The scheduler uses the priorities of the replication request as the scheduling criteria, maximizing the priority while fully utilizing the link capacity.

Evaluation of application level replication algorithms (simulation and real-world results) allows general solutions on heterogeneous disk subsystem with a single protocol where hardware dependencies and optimizations do not bias the research. The replication algorithm operates as a peer-to-peer remote copy (PPRC) similar to the remote synchronization algorithm, rsync (Tridgell 2000). The replication algorithm exploits file commonality in an effort to gain the efficiencies of block level updates.

In evaluating scheduling for QoS based data replication three characteristics tend to dominate. The first is the variability and uncertainty of the quantity of data to replicate. Bandwidth reduction techniques, such as exploiting the commonality between replications, are greatly affected by the time intervals between asynchronous replications. While the deltas between a file being modified at different points in time from a single base will tend to grow, this is not a certainty. The remaining two characteristics, the capacity limitations of the communication links and the expense of long haul communication verses the cost of computing are related.

The scheduler is looking to maximize the use of the limiting component, the fixed low bandwidth communications link. The admittance policy of the scheduler can be viewed as an optimization problem with file size and priority as parameters of the replication while the link capacity and the length of time to complete the replication as the limiting factors.

The Knapsack model and its variants provide a rich model for the exploration of alternative scheduling options. An instance of the knapsack problem (KP) can be defined by the capacity c and a set of n items where an item i is described by its profit p_i and weight s_i. A subset of items is selected such that the total profit of the selected items is maximized and the total weight does not exceed c. The KP can be formulated as a solution for the following linear integer program:

$$\begin{aligned} &\text{Maximize O} = \sum_{i=1}^{n} p_i x_i \\ &\text{Subject to} \sum_{i=1}^{n} s_i x_i \le c \qquad (1) \\ &x_i \in \{0,1\}, i = 1,...n \\ &p_i \in \{P_1,...,P_m\}, i = 1,...n . \end{aligned}$$

The profit p_i belongs to the set of priorities $P_1,\ldots,P_m$. Higher priority values represent higher priority items. The decision vector $\mathbf{x}$ identifies which items are to be inserted into the knapsack. A value of one identifies insertion. All of the coefficients are positive integers and O is the objective function. The weight of each item is less than the capacity so that it is possible to be scheduled,

$$s_i \le c, i = 1,...n .$$

Finally, the weight of all items submitted to the scheduler must be greater than the capacity,

$$\sum_{i=1}^{n} s_i > c .$$

In the event that the weight of all items submitted to the scheduler is smaller or equal to the capacity, all items are scheduled. This model is also known as the 0-1 KP. A binary decision is made by the optimization to insert the item into the knapsack are not.

Without loss of generality, the 0-1 KP can be used to model the scheduling of single files or groups of files that must be replicated together. An example would be a database where its indices are stored in one file and the data in a second. In this case, file priorities are the same and the file sizes are treated in the aggregate. The group of files is modeled as a single replication object.

KP is NP-hard and can be solved a number of ways with one such approach being dynamic programming (DP). DP is a common approach as it provides solutions to KP in pseudo-polynomial time. Keller, Pferschy, and Pisinger (2004) provides a further discussion of techniques for solving the KP. The complexities in finding exact solutions to KP problems impact the scale to which the solutions are practical. Furthermore, in order to use the knapsack model in real-time scheduling, knowledge of future scheduling requirements would have to be known.

An alternative approach would be to use the relationship between the expenses of long haul communication verses the cost of computing to our advantage. Gray (2003) equates cost parity between: one database access, ten bytes of network traffic, 100,000 instructions, 10 bytes of disk storage, and one megabyte of disk bandwidth. This parity implies a gigabyte of data transmitted over the Internet would require a CPU day of computation to be in balance. Gray's conclusion is this parity forces the structure of Internet-scale distributed computing to place the data as close as possible to the computation in order to minimize the expensive network traffic. The premise of this work is that the data must be replicated, thus transmitted, but a trade-off of disk bandwidth, disk storage, database access and computation for reduced network traffic is a beneficial trade-off. The benefit is based on the fact, also brought out by Gray, that over the last 40 years telecom prices have fallen much more slowly than any other information technology. The greater is the disparity between price deflations, the stronger is the argument for long-haul bandwidth optimization at the expense of computing resources.

The proposed approach is to divide the scheduler into two parts, an off-line optimization and an on-line implementation of the acceptance policy. The off-line optimization is based on solving the KP as described by the linear integer program (1). The off-line optimization results are used to train a neural network of the appropriate acceptance policy. The neural network makes real-time binary decisions on acceptance of replication requests. The replication requests are also processed by the off-line optimization in order to verify the performance of the network. The offline processing continues to train a new network on the data and when necessary replace the in-line network.

In the next section two case studies are presented which look to simulate this proposed scheduling approach to QoS data replication in the area of disaster recovery.

4 CASE STUDIES

The management of a data replication system includes an assessment of potential risks. This risk assessment includes an evaluation of acceptable loss on a service basis. This evaluation provides a foundation to build business continuity were the cost of recovery can be weighed against the risk of a disaster and the impact on the survival and prosperity of the business. Acceptable losses are identified in terms of lost availability, how old the information supplied to and from the services can be, and the amount of lost data when a service is restored.

The cost of the recovery solution will be directly related to how quickly the business must be restored and how much data needs to be protected. It is a business decision based on cost verses effectiveness and efficiency whether the business should avoid, mitigate or absorb the risk of a disaster condition on each service. The restoration of value to a service is dependent on the restoration of the service's data. The importance of data, the availability requirements and currency requirements of the data for the different services will vary between services and over time.

Disaster recovery (DR) planning requires a separation of resources, assumes data is the single most important component of any DR solution and that the prioritization of critical functions for restoration is a fundamental part of Continuity of Operations Planning. The decision as to what to replicate and how often is dynamic. The observation is made that to support varying levels of data value, availability and currency requirements, a QoS replication mechanism is needed to ensure minimum guarantees of data currency, maximizing the currency and availability of highly valued data and fully use the available network capacity allocated to DR data replication.

4.1 Simple Replication Scheduling Simulation

The case study presented here establishes a simulation to evaluate the approach of a real-time optimization using a Multi-Layer Perceptron Network as a viable scheduler for a QoS based data replication. The purpose of the data replication is to create a remote data vault over a fixed low bandwidth connection. Bandwidth is defined as low when compared to the data bandwidth. The data synchronization between the primary and replica is asynchronous.

4.1.1 Simulation Overview

The simulation consists of a three components: submission of replication request, scheduling the replications, and finally, the replication. This case study is an evaluation of the scheduling portion of the simulation.

When replication request are made they have a timing requirements and a priority. The timing requirement is the periodicity of the replications, which provides the interval for the completion of the replication. The priority is a relative assessment of how important it is to have the data replicated during a given interval. The scheduler uses the priorities of the replication request as the scheduling criteria, maximizing the priority while maximizing the use of the link capacity. For example, a set of files associated with a given service can all be assigned a replication interval of 60 minutes and a priority of 100 (low). Every 60 minutes the files are scheduled for replication. The acceptance policy of the scheduler decides if the replication can complete within the 60-minute interval. If the replication cannot complete within the interval, it is rejected. These replication intervals create a repeating pattern of replication request.

Files submitted for replication consist of a record containing four fields: a system generated primary key, the full pathname for a file, the priority of the file, and the size of the file. The priority of each file is 100, 200, 300 or 400, the higher the number the higher the priority,

$$p_i \in \{100, 200, 300, 400\}, i = 1, ...n \, .$$

A uniform distribution is used to generate the random priorities for the simulation. All of the filenames and sizes were obtained from four file systems on a single workstation. Table 1 is an excerpt from the database used for replication request for the simulation.

Table 1: Replication Request Database Excerpt

PK	PRIORITY	SIZE	FILENAME
0	300	42546201	/…/2.6_Recommended.tar.Z
1	100	351363	/…/gcc-2.95.2/.brik
2	400	345	/…/.cvsignore
3	200	212246	/…/ChangeLog

To take into consideration the effects of file modifications requiring a variable size update over multiple replication intervals, the simulation uses a uniform random percentage of each file to be replicated when it is submitted to the scheduler.

4.1.2 Off-Line Processing

In this simulation, the capacity of the knapsack is defined as the capacity of the fixed bandwidth connection for the replication between the primary data source and replica. The value of the items used for optimization, p, is their assigned priority. The optimization function, O as described by the linear integer program (1), is calculated after a predetermined timeframe or time slice. The resultant decision vector, x, used to calculate O, provides the optimal acceptance policy for a time slice. Based on this policy, two queues are created, one containing the items accepted for

replication and the second for the rejected items rejected. In the next time slice, new replication request are added to the items in the rejected queue and the process repeats. Items that cannot be scheduled during their replication interval specified as an integer number of time slices are rejected. The MinKnap algorithm developed by Pisinger (1997) is used to calculate O.

4.1.3 Real-Time Processing

The neural network works within the pattern in which it is trained. The pattern is a replication interval, which repeats. While the files to replicate repeat over time, whether the file is modified and the size of the modification varies. The neural network used in this study is a Multi-Layer Perceptron (MLP), which is built and trained using the NeuroSolutions software tool. NeuroSolutions uses the back-propagation of errors to train the MLP network (Figure 3). The MLP used has a single hidden layer with three inputs and a single output.

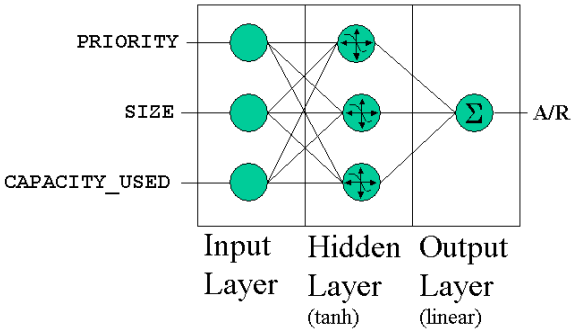

Figure 3: Multi-Layer Perceptron

As items are accepted or rejected for replication from the off-line processing, a comma-separated record is written to an output file. Each record contains seven fields: a primary key (PK), time slice of replication request (TIME), time spent in queue before admittance (TIQ), admittance or rejection (A/R), priority of request (PRIORITY), size of request (SIZE), the percentage of the capacity used when the item is queued for transmission (CAPACITY_USED), and the full file pathname (FILE). The input parameters of the MLP are: PRIORITY, SIZE, and CAPACITY_USED. All of these parameters are numeric.

The desired output is the A/R parameter. The A/R parameter is our decision vector. The parameters PK, TIME, TIQ and FILE are not used by the MLP. The parameters TIME and TIQ are used by the off-line processing to determine when a replication interval has expired. The replication algorithm uses the parameters PK and FILE. This output file is used in training the MLP network. Table 2 is an excerpt from the results of testing a MLP network. As seen in Table 2, the MLP provides real outputs for the accept/reject criteria. To use the MLP results as the acceptance policy for the scheduler, the results are rounded as follows to provide integer results:

$$-0.5 \le x_i < 0.5, x_i = 0 \text{ for i = 0, …, n,}$$

$$0.5 \le x_i < 1.5, x_i = 1 \text{ for i = 0, …, n,}$$

otherwise $x_i = 2$ for i = 0,…, n, (an error condition).

Table 2: MLP Results

Des A/R	Out A/R
0.000000	-0.002276
0.000000	0.004267
0.000000	0.004334
1.000000	1.006103

Once the MLP network is trained and validated, production data is submitted to the network. The results of the MLP network are validated by submitting the same production data to the off-line processing optimization and comparing results. This validation determines a valid MLP network and identifies when the input pattern is changing and the network requires retraining.

4.1.4 Results

The first example presented uses 13,760 events over 10 time slices. The 10 time slices represent one replication interval for each of the replication request. The acceptance policy was biased by varying the capacity of the link plus and minus one order of magnitude from a base value. The off-line processing produced the results shown in Table 3.

Table 3: Off-Line Optimization Results

	Capacity	DP Time	Accept	Reject
Biased reject	18874	22.462s	2,881	10,879
Unbiased	188743	21.879s	6,808	6,952
Biased accept	1887430	22.290s	12,207	1,553

A MLP network was created for each of the cases. Each MLP network used 40% of the data for training (5504 rows), 20% (2752 rows) for cross validation during the training and 40% (5504 rows) for testing previously unseen data. The results of the testing are shown in Table 4.

Table 4: MLP Test Results

	Desired accepted	Desired rejected	Test data correct	Test data wrong
Biased reject	1008	4496	5504	0
Unbiased	2318	3186	5504	0
Biased accept	4768	736	5504	0

A more advanced example is to enhance the simulation to support multiple replication intervals. If a file cannot be replicated during the replication interval, the event is rejected for admission. The training in this example used 30,000 rows of data that represent three replication intervals for 10,000 files. Replication requests were made at a constant rate of 1000 per time unit for ten time units. Three replication intervals were run. Each replication interval consisted of 10,000 request of the same files, same priorities, but with different modification sizes and a resulting different capacity utilization. The format of the test was the same as the previous example, 40% for training, 20% for cross validation and 40% for testing. Table 5 provides an interval breakdown of the results of the off-line programming optimization. The off-line processing took 24.12s. The 30,000 entries were randomized before training, cross validation and testing. The training time was 13:33 with a MSE of 0.000193. The validation test processed 12,000 files; 6125 for acceptance and 5875 were rejected. All files were processed correctly as compared to the optimal solution.

Table 5: Off-Line Processing Optimization Results

Interval	Accepted	Rejected
1	5402	4598
2	5440	4560
3	5448	4552

Once the MLP network was trained and validated, two production runs of six 10,000-file replication intervals were made. The first run consisted of the three intervals used in training, validation and testing plus three additional intervals. To validate the results of the MLP network, the same replication request provided to the MLP network were provide for offline processing optimization. The interval breakdown of the off-line processing is provided in Table 6. The offline processing took 50.138s.

Table 6: Production Run 1 Off-Line Processing Results

Interval	Accept	Rejected
1	5402	4598
2	5440	4560
3	5448	4552
4	5457	4543
5	5390	4610
6	5408	4592

The resultant acceptance policy from the offline processing and the MLP network were compared for the 60,000 files processed. All files were processed correctly based on the optimal solution with 32,545 files accepted and 27,455 files rejected.

The second production run also consisted of 60,000 files, comprised by six 10,000-file replication intervals. All six of the intervals were new data to the MLP. Again the replication requests were provided both to the off-line processor and the MLP network (Table 7). The resultant acceptance policy from the offline processing and the MLP network were compared for the 60,000 files processed. Again all files were processed correctly based on the optimal solution with 32,426 files accepted and 27,574 files rejected.

Table 7: Production Run 2 DP Optimization Results

Interval	Accept	Reject
1	5405	4595
2	5378	4622
3	5396	4604
4	5425	4575
5	5412	4588
6	5410	4590

There were several lessons learned in training the neural network. The most significant was that if the training data was largely biased, not enough training data will be provided for the policy against the bias. In this case, results have shown the acceptance policy for the cases against the bias to, basically, be a guess. For example, one result of the dynamic programming optimization was 1075 accepted and 8 rejected events. The MLP used 40% for training (434 rows), 20% for cross validation (215 rows) and 40% for testing (434 rows). The training time took only 51 seconds and the mean square error was 0.000086. The results of the test showed 433 files processed with 430 accepted and 3 rejected. The MLP network processed 431 of the files correctly with 2 files processed incorrectly. Upon inspection of the rejected events, one was correct and two were incorrect as seen in the output below:

Desired: 0.000000 : Output: 0.117583
Desired: 0.000000 : Output: 0.730678
Desired: 0.000000 : Output: 0.940221.

4.2 Enhanced Scheduler for DR Data Replication

The aforementioned model does not prevent starvation and in some cases promotes it. As each file is submitted for replication, it is placed in a queue. At predetermined times the queue of request is evaluated and the calculation of objective function is made. The files defined by the objective function are replicated; those not identified for replication remain in the queue. If the file is not replicated in the number of time slices that make up the replication interval for the file, the replication request is denied. The file would again be scheduled for replicated for replication, but this replication request would include all former modifications and any modifications made during the previous replication interval. Figure 4 provides an example of replication request for a file, f_0, every four time slices. Unless modifications made during previous replication intervals have been deleted, the file deltas will tend to

increase in size. Since the priority of the file is generally constant, the likelihood of replication tends to diminish slightly with each replication interval, thus increasing the likelihood of starvation from replication for a given file. In order to diminish the impacts of starvation, the model previously presented is enhanced. In the simulation, deltas between replications are allowed to grow or shrink. The delta is determined randomly as a zero to one hundred percentage of file size.

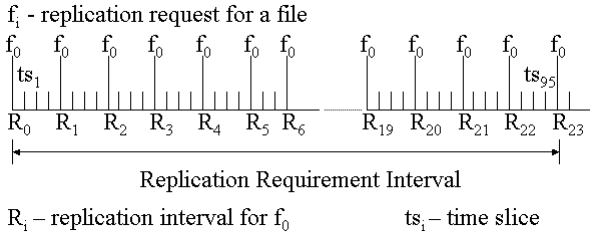

Figure 4: Replication Intervals

The proposed model expands the objective function of the 0-1 Knapsack model to include an exponential time component. An instance of the enhanced knapsack problem (EKP) can be defined as given a set N, consisting of n item i with priority p_i and size s_i, and the capacity value `c`. The priority p_i is a element of the set `{2, 3, 4}`. Select a subset of N such that the total priority of the selected items is maximized, while each item is selected at least once during its replication requirement with high likelihood and the total size does not exceed c.

Formally, an EKP can be formulated as a solution for the following linear integer programming formulation:

$$\text{Maximize O} = \sum_{i=1}^{n} p_i^t x_i$$

$$\text{Subject to} \sum_{i=1}^{n} s_i x_i \le c \qquad (2)$$

$$x_i \in \{0,1\}, i = 1,...n$$

$$p_i \in \{2,3,4\}, i = 1,...n\,.$$

The exponent `t` is defined as the replication interval, `1…m`, until the first replication. After that, `t` is set to 1. The decision vector **`x`** identifies which items are to be inserted into the knapsack. The value m is defined as the ration of `RR/RI` where RR is the required replication and RI is the replication interval. The value `t` is incremented after a replication interval in which the item was not replicated. At the expiration of the RR, the value of `t` is reset to 1 and the filename is logged as not having been replicated. The object function escalates the values faster for the shorter interval replications and slower for the items that have longer replication intervals.

Replication request are made to the scheduler in four flows. Each flow has its own flow rate per replication interval, the items per replication, its own replication interval definition, RI, and replication requirement definition, RR. Table 8 provides simulation parameters. The MLP used in this case study has two hidden layers with three inputs and a single output as before.

Table 8: Simulation Flows

Input	Items / replication	Primary key	RI	RR
Flow1	298	0 – 297	4	96
Flow2	422	300 – 721	16	96
Flow3	200	750 – 949	32	672
Flow4	2038	950 – 2987	96	672

Validation consisted of a simulation run for 1,347 time slices. The training took 4:15:02 and provided a MSE of 0.002338. Testing included 33,546 items, 100,643 items used for training and 33,546 items used for cross validation. The average t value in the simulation is 1.035. The results of comparing the acceptance policy from the offline processing and the MLP network for the 33,546 files produced 33,545 correct decisions, 1 incorrect decision with 27,423 files accepted and 6,123 files rejected. This simulation is 99.997% of optimal. The incorrect decision came from a desired value of 1.000000 with an output value of 1.548140 which was treated as a error condition.

5 CONCLUSIONS

The simulations have shown the approach of a real-time optimization using a Multi-Layer Perceptron Network to work extremely well for determining acceptance given a QoS policy for a replication scheduler. The main issue with this approach is ensuring that the patterns in the data are adequately represented in the training data and recognizing when a pattern is changing or a new pattern is present. The solution presented is to continue the offline processing of the data, comparing scheduler and optimal results with subsequent updating of network weights as required. In situations were patterns are known and repeating, changing to a predetermined MLP works well.

ACKNOWLEDGMENTS

This work has been supported in part by an Academic Fellowship from the Dahlgren Division of the Naval Surface Warfare Center (NSWCDD) and the Submarine Launched Ballistic Missile (SLBM) program.

REFERENCES

Alsberg, P. A. and J. D. Day. 1976. A principle for resilient sharing of distributed resources. In *Proceedings of the*

2nd International Conference on Software Engineering, 627 – 644.

Bernstein, P. A. and N. Goodman. 1983. The failure and recovery problem for replicated databases. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, 114-122.

Bernstein, P. A. and N. Goodman. 1984. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems*, 9(4): 596-615.

Bernstein, P. A., V. Hadzilacos, and N. Goodman. 1987. *Concurrency Control and Recovery in Data base Systems*. Addison-Wesley.

Gifford, D. K. 1979. Weighted voting for replicated data. In *Proceedings of the 7th Symposium on Operating System Principles*, 150-162.

Gray, C. G. and D. R. Cheriton. 1989. Leases: An efficient fault-tolerate mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating System Principles*, 202-264.

Gray, J. 2003. Distributed Computing Economics, IEEE Task Force on Cluster Computing. Available online via <http://www.clustercomputing.org/content/tfcc-5-1-gray.html>. [accessed August 19, 2004].

Helal, A., A. Heddaya, and B. Bhargava. 1996. *Replication Techniques in Distributed Systems*, 13-60, Kluwer Academic Publishers.

Hitz, D., J. Lau, and M.A. Malcolm. 1994. File System Design for an NFS File Server Appliance. In *Proceedings USENIX Winter 1994 Conference*, 235-246.

Howard, J., M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. 1988. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1): 51-81.

Keller, H., U. Pferschy, and D. Pisinger. 2004. *Knapsack Problems*. Springer-Verlag.

Kistler, J. J. and M. Satyanarayanan. 1992. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10 (1): 3-25.

Kumar, A. 1991. A randomized voting algorithm. In *Proceedings of the IEEE 11th International Conference on Distributed Computing Systems*, 412-419.

Minoura, T. and G. Wiederhold. 1982. Resilient extended true-copy token scheme for a distributed database system. *IEEE Transactions on Software Engineering*, 9(5):172-189.

Muthitacharoen, A., B. Chen, and D. Mazières. 2001. A low-bandwidth network file system, In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 174-187.

Patterson, H., S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara. 2002. SnapMirror: File System Based Asynchronous Mirroring for Disaster Recovery. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*,117-129.

Pisinger, D. 1997. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758-767.

Reisner, P. 2001. DRDB, In *Proceedings of UNIX en High Availability*, 93 – 104.

Shepler, S., B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. 2000. NFS version 4 protocol. RFC 3010, Network Working Group.

Thomas, R. H. 1979. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180-209.

Tridgell, A. 2000. Efficient Algorithms for Sorting and Synchronization. PhD thesis, Australian National University.

AUTHOR BIOGRAPHIES

KEVIN ADAMS is a senior Scientist for the SLBM program at NSWCDD and a Ph.D. student in Computer Science at Virginia Tech. He has a B.S in Computer Science from James Madison University in 1986. He has a M.S. in Computer Science and a M.S. in Electrical Engineering from Virginia Tech in 1992 and 1998 respectively. He is a member of the IEEE Computer Society and the ACM. His email address is <keadams2@vt.edu>.

DENIS GRAČANIN is an Assistant Professor in the Department of Computer Science at Virginia Tech. He has a B.S. and M.S. degree in Electrical Engineering from the University of Zagreb, Croatia in 1985 and 1988, respectively. He has a M.S. and Ph.D. degree in Computer Science from the University of Louisiana at Lafayette in 1992 and 1994, respectively, His research interests include virtual reality and distributed simulation. He is a senior member of IEEE and a member of AAAI, ACM, APS, SCS, and SIAM. His email address is <gracanin@vt.edu>.

DUŠAN TEODOROVIĆ is a Professor of Civil and Environmental Engineering at Virginia Polytechnic Institute and State University in Falls Church, VA. He has a B.S., M.S., and Ph.D. degree in Engineering from the University of Belgrade in 1973, 1976, and 1982, respectively. His research interest include Transportation Networks, Air Transportation, Public Transportation, Traffic Engineering, Fuzzy Systems, Neural Networks, Metaheuristics, Swarm Intelligence, Operations Research and Artificial Intelligence Applications in Transportation. His email address is <duteodor@vt.edu>.