On using Network Memory to Improve the Performance of Transaction-Based Systems

Sotiris Ioannidis[*] Evangelos P. Markatos[†] Julia Sevaslidou[†]
Computer Architecture and VLSI Systems Group
Institute of Computer Science (ICS)
Foundation for Research & Technology – Hellas (FORTH), Crete
P.O.Box 1385
Heraklio, Crete, GR-711-10 GREECE
markatos@ics.forth.gr
tel: +(30) 81 391655 fax: +(30) 81 391601

Technical Report 190, ICS-FORTH
htpp://www.ics.forth.gr/proj/avg/paging.html

Abstract

Transactions have been valued for their atomicity and recoverability properties that are useful to several systems, ranging from CAD environment to large-scale databases. Unfortunately, adding transaction support to an existing data repository was traditionally thought to be expensive, mostly due to the fact that the performance of transaction-based systems is usually limited by the performance of the magnetic disks that are used to hold the data repository. In this paper we describe how to use the collective main memory in a Network of Workstations (NOW) to improve the performance of transaction-based systems. We describe the design of our system and its implementation in two independent transaction-based systems, namely EXODUS, and RVM. We evaluate the performance of our prototype using several database benchmarks (like OO7 and TPC-A). Our experimental results indicate that our system delivers up to two orders of magnitude performance improvement compared to its predecessors.

1 Introduction

Transactions have been valued for their atomicity and recoverability properties that are useful to several systems, ranging from CAD environment to large-scale databases. Unfortunately, adding transaction support to an existing data repository was traditionally thought to be expensive, mostly due to the fast that the performance of transaction-based systems is usually limited by the performance of the magnetic disks that are used to hold the data repository. A major challenge in transaction-based systems is to decouple the performance of transaction management from the performance of the disks.

In this paper we describe a novel way to improve the performance of transaction management by using the collective main memory (hereafter called remote memory) in a Network of Workstations (NOW) [2, 4, 16]. The main idea behind our approach is to reduce the number of disk accesses by substituting them with (remote) main memory accesses. There are two main areas where remote memory can be used to improve performance of a transaction-based system:

- *Speeding up Read Accesses:* The collective main memory in a NOW can be used as a large cache of the transaction-based system. This cache is larger than any *single* workstation can provide, and thus can be used to hold large amounts of data. Reading data from remote main memory (over a high speed interconnection network), was shown to be significantly faster than reading data

[*]Current affiliation: Computer Science Department, University of Rochester, Rochester, NY14627
[†]Evangelos Markatos and Julia Sevaslidou are also with the University of Crete, Department of Computer Science

from a (local) magnetic disk [10]. This is due to the fact that current high-speed networks provide higher throughput and lower latency than current high-speed disks. Architecture trends suggest that this disparity between magnetic disks and interconnection networks will continue to increase with time [10]. Thereby, the use of remote memory instead of magnetic disks (whenever possible) will continue to result in everincreasing performance improvements.

- *Speeding up Synchronous Write Operations to Reliable Storage:* Transaction-based systems frequently use synchronous write operations to force all modified data to disk at transaction commit time. This is done so that the system will be able to recover in a consistent state after a hardware or a software failure. We advocate (and show in this paper) that the set of remote main memories in a NOW has comparable reliability to a magnetic disk, and thus can be used to hold sensitive data that must survive a system crash. [1] Synchronous write operations are dominated by the latency of the medium where the write is performed. Remote memory latency is dictated by network latency and is usually significantly lower than disk latency. Moreover, remote memory latency is mostly induced by communication protocol processing (e.g. TCP/IP), which improves with processor speed. On the other hand, disk latency is mostly induced by mechanical arm movements, which does not improve at as fast. Thus, data can be synchronously written to remote memories much faster than they can be written to a magnetic disk. This performance disparity will continue to increase, according to current architecture trends.

The first of the above issues (reading from remote memory) has been somewhat explored in the areas of file systems [1, 18, 21], paging [22] and global memory databases for workstation clusters [15, 17]. All previous work suggests that the use of remote main memory as a large file (database) cache results in significant performance improvements. The thrust of this paper is on exploring the second issue: using remote memory to speed up synchronous disk write operations. We believe that transaction-based systems make lots of small synchronous write operations to stable storage, and thus they are going to benefit significantly from any improvements to synchronous disk write operations.

Recent architecture trends in the area of interconnection networks, and Networks of Workstations (NOWs) make it more attractive than ever to use the main memory of remote workstations (within the same workstation cluster) to speedup synchronous disk I/O operations, because the latency of Local Area Networks has significantly decreased over the last few years. Traditional interconnection networks (like Ethernet and FDDI) have latency in the range of several hundred microseconds. ATM networks have latency in the range of a few hundred microseconds [26], while more recent networks (like SCI [24] and Memory Channel [16]) have latency in the range of few microseconds. At the same time, disk latency has remained in the order of a few milliseconds for several years now, and is not expected to improve at a significant rate. Thus, operations dominated by disk latency (i.e. synchronous disk write operations) will remain in the millisecond range. On the contrary, operations dominated by network latency (e.g. synchronous remote memory write operations) may complete within microseconds.

Based on the current architecture trends, we believe that transaction-based systems should make use of the remote main memory of a NOW, in order to avoid (synchronous) disk data transfers and substitute them with (synchronous) network data transfers. To demonstrate our approach, we implemented our approach within two existing transaction-based systems: The EXODUS storage manager [6], and the RVM (Recoverable Virtual Memory) System [25]. Section 2 describes the design and the implementation of our systems. We have run several benchmarks on top of the modified transaction systems and have observed performance improvements up to two orders of magnitude. We report our performance results in section 3. Section 4 places our work in context by surveying previous work and comparing it with our approach. Finally section 5 concludes the paper.

2 Remote-Memory-based Transaction Systems

2.1 Reliable Main Memory

The performance of transaction-based systems is usually limited by slow disk accesses. During its lifetime, a transaction makes a number of disk accesses to read its data (if the data have not been cached in main

[1] We increase the reliability of remote main memory by using redundant power supplies (UPS) to survive power failures, and mirroring of data to survive software failures.

memory), makes a few calculations on the data, writes its results back (via a Log file), and then, if all goes well, it commits. Although disk read operations may be reduced with the help of large main memory caches (or even network main memory caches [1, 15]), disk write operations at transaction commit time are difficult to avoid, since the transaction's modified data and meta-data have to reach stable storage before the transaction is able to commit, otherwise a system crash would leave the data repository in a non-consistent state. Several current transaction based systems use a magnetic disk as the stable storage, and force all dirty data to it using the `fsync(2)` system call [6, 25]. [2] Magnetic disks can usually survive power and software failures, thereby providing a stable medium to store data that must survive crashes.

We believe however, that in Networks of Workstations the collective main memory of all workstations in the system can be made reliable in such a way as to survive power outages and software failures, and thus become a viable alternative to disk storage for sensitive transaction data. We believe two are the main sources of system crashes that may lead to data loss: (i) software failures, and (ii) power loss. We deal with each of them in turn:

- *Software failures* are the result of software malfunctions, operating system crashes, etc. When the operating system crashes and reboots, it may destroy the contents of its main memory, thereby eliminating all data that have not been written to stable storage. Data that must survive software failures are replicated to the main memories of (at least) two workstations that are connected in two different power supplies (e.g. one is on a UPS, and the other is on the main power supply). Since different workstations run different copies of the operating system, they will probably crash (due to software errors) independent of each other. Thus, if the data that must survive software crashes are replicated to two workstations, they will survive software failures with high probability. [3]

- *Power losses* are the result of malfunctions in the power supply system. To cope with power losses we assume the existence of *two* power supplies: one could be the main power supply, and the second could be provided by an uninterrupted power supply (UPS). [4] A UPS for a workstation can cost less than \$100.00, making it a small percentage ($< 5\%$) of the cost of a workstation. If workstations are connected to UPSs, they will retain their main memory contents even after a power loss. However, if a power loss is detected, the UPS gives plenty of time to workstations to save their sensitive data to magnetic disks. [5]

Based on our description we advocate that using mirroring and UPSs, we can make the (remote) main memory, a storage medium as reliable as the magnetic disk. Thus, sensitive data that need to be *synchronously* written to disk, can be (synchronously) written to remote main memory with the same level of reliability. Our described main memory system suffers from data loss once every several years, which is the same level of reliability current magnetic disks provide.

2.2 EXODUS and RVM

To illustrate our approach we have modified a lightweight transaction-based system called RVM [25] and the EXODUS storage manager [6] to use remote memory (instead of disks) for synchronous write operations. After studying the performance of the systems, we concluded that they spend a significant amount of their time, synchronously writing transaction data to their log file, which is used to implement a two-phase commit protocol. When a transaction commits, all the data the transaction modified are synchronously written to the log (stored as a UNIX file on a magnetic disk). After the mentioned data are successfully written to the log, the system is allowed to proceed.

We have modified both EXODUS and RVM so as to to keep a copy of their log file in remote main memory (as well as the disk). The unmodified systems force all their sensitive data to the disk at

[2] Although optimizations like group transaction commit have been proposed, these may increase the complexity of the transaction-based system, and thus have not been incorporated in the EXODUS and RVM systems.

[3] Simple math calculations suggest that if each workstation crashes once every few months, and stays crashed for several minutes, two workstations will crash within the same time interval once every several years, which leads to higher reliability than current disks provide. If, however, this level of reliability is not enough, data can be replicated to three main memories, etc.

[4] Alternatively, both power supplies may be provided by two different UPSs.

[5] Note, however, that the UPS *itself* may malfunction, leading to power loss. Such malfunctions however, happen once every several years, making the UPS more reliable machine than the magnetic disk. If, however, a UPS malfunctions more frequently, sensitive data can be replicated to different workstations connected to *different* UPSs that malfunction independent of each other.

transaction commit time using synchronous disk write operations. In our modified systems, we substitute each synchronous write operation with the following two operations:

1. *A synchronous write to the log "file" in the main memory of one remote workstation.*
2. *An asynchronous write to the log file on the magnetic disk.* This operation is being carried in the background and is used to preserve a local copy of the data in case the remote main memory crashes.

Essentially, we substitute a *synchronous disk write* operation with a *synchronous network write* operation plus an *asynchronous disk write* operation (which has no effect on completion time since it proceeds in the background, as long as adequate data buffering is provided). At the same time, our systems do not compromise data reliability. Let's examine what are the steps in writing data in our systems:

1. At transaction commit time, the transaction's sensitive data are synchronously written to the log in remote main memory
2. At the same time, these data are asynchronously written to the local magnetic disk
3. Eventually, the data reach the magnetic disk. [6]

The transaction is committed *after* step 2 completes. It seems that there is a "window of vulnerability" between steps 2. and 3., that is *after* the data have been safely written to remote memory (and scheduled to be written on the disk), but *before* the data have been safely written to magnetic disk. If the local system crashes during this interval, then the data that are still in the local main memory buffer cache will be lost during the crash. Fortunately, our system can still recover the seemingly lost data, since the same data reside in the remote memory as a result of step 1. Data loss may happen only if both local and remote systems crash during this interval. However, we have argued that the probability of both systems (which are equipped with UPSs) crashing during the interval of few minutes is comparable (or even lower) than the probability of a magnetic disk malfunction. Thereby our system provides levels of reliability comparable to a magnetic disk.

2.3 Recovery

In the event of a workstation/network crash, our system needs to recover data and continue its operation. If the *local* workstation crashes, and reboots, it will read all its "seemingly lost" data from the remote memory, store them safely on the disk, and continue its operation normally. If the *remote* workstation crashes, the local transaction manager will realize it after a timeout period. After the timeout, the local manager may either search for another remote memory server, or just stop using remote memory, and commit transactions to disk as usual. If the *network* crashes, the local workstation will stop using remote memory and will commit all transactions to disk. In all circumstances, the system can recover within a few seconds, in the worst case. The reason is that at all times there exist two copies of the log data: if one copy is lost due to a crash, the system can easily switch to the other copy quickly.

2.4 Implementation

We have made the described changes to RVM and EXODUS. We call the resulting systems RRVM (Remote RVM) and REX (Remote EXODUS). [7]

Our systems have been completely implemented in user space, without any operating system modifications. For each transaction manager, we start a user-level remote memory server on a remote workstation. The purpose of this server is to accept synchronous write requests from the transaction manager and acknowledge them. In the case of a transaction manager crash, the remote memory server is responsible for providing the contents of the Log file it keeps in its main memory. At all times, data written by *committed* transactions either reside safely on the disk, or are stored in the main memory of at least two workstations (the local transaction manager, and the remote memory server).

[6] UNIX-derivative systems force all their modified data to the disk every 30-60 seconds.

[7] Our modification were rather small. Out of about 30,000 lines of RVM code, we modified (or added) less than 600 lines (2%). Out of 200,000 lines of EXODUS code, we modified (or added) less than 700 lines (a 0.3% change).

3 Experimental Evaluation

In this section we report the performance advantages of our systems RRVM and REX compared to the original RVM and EXODUS systems.

3.1 RVM performance

We have implemented our RRVM system on top of Digital UNIX on a network of workstations. In this section we will describe some of our experiments to measure its usability and performance.

3.1.1 Experimental Environment

Our experimental environment consists of a network of eight DEC Alpha 2000 workstations running at 233 MHz, equipped with 128 MBytes of main memory each. The workstations are connected through an Ethernet interconnection network, a 100 Mbps FDDI, and a Memory Channel Interconnection Network [16]. These three interconnection networks represent three different generations of LANs: Ethernet is a traditional low-bandwidth network that provides low-cost connectivity without ambitious performance goals. FDDI is a high-bandwidth network that provides higher speed at a slightly increased cost. Memory Channel is a very high-bandwidth network that was designed to make possible a wide range of high performance applications in Networks of Workstations. Each workstation is equipped with a 6GB local disk.

In our experiments we demonstrated the performance of RRVM and compared it with its predecessor RVM [25].

We have experimented with four system configurations:

- `RVM`: This is the unmodified RVM system [25]. It makes no use of the interconnection network, since all its data are stored in a local disk.
- `RRVM-ETHERNET`: This is RRVM running on top of an Ethernet interconnection network.
- `RRVM-FDDI`: This is our RRVM system running on top of an FDDI interconnection network.
- `RRVM-MC`: This is RRVM running on top of the Memory Channel Interconnection Network.

3.2 I/O Block Size

In our first set of experiments we would like to find out how many transactions per second our RRVM system is able to sustain, compared to the number of transactions per second the unmodified RVM system is able to sustain. For this reason we constructed the following experiment:

> We create a file 100 Mbytes long. Then, we start a sequence of 10000 transactions. Each transaction writes a segment of the file and commits. Transactions modify the file in sequential manner. The size of the file segment modified by each transaction (also called I/O block size) is the parameter of our experiments.

Figure 1 plots the number of transactions per second, for the original version of `RVM`, and our `RRVM-MC`, `RRVM-FDDI`, and `RRVM-ETHERNET`. We see that the unmodified `RVM` system is able to sustain up to at most 40 transactions per second for small transactions. As the size of the I/O block gets larger, the number of transactions per second successfully executed gets even lower. Our results agree with previously reported results, in which `RVM` is able to sustain at most 50 transactions per second ([25], figure 8(b)). However, the performance of `RRVM-ETHERNET` and `RRVM-FDDI` is significantly better than that of `RVM`. For small transactions both `RRVM` systems are able to sustain close to 500 transactions per second: an order of magnitude improvement over unmodified `RVM`. Even better, `RRVM-MC` manages to sustain close to 3,000 transactions per second: almost two orders of magnitude improvement over unmodified `RVM`. By looking closely at figure 1 we see that the performance of `RVM` is practically the same for I/O block sizes between 0 and 10 Kbytes, which implies that the transactions overhead (for the above I/O block sizes) is dominated by disk seek and rotational latency and not by the data transfer itself.

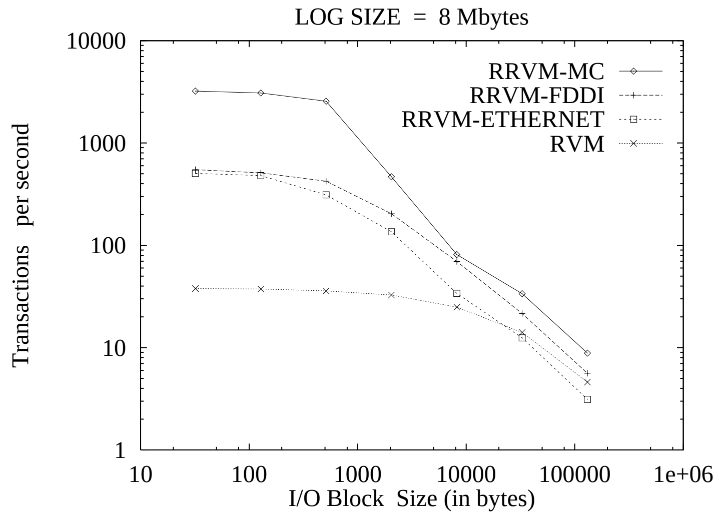

Figure 1: Performance of RVM as a function of the I/O block size. Data file = 100 Mbytes, Log File = 8 Mbytes.

3.3 The Size of the Log

In this section we set out to answer how the size of the log file kept by RRVM influences the performance of the system. The log file is synchronously written by transactions during their commit phase. When the log file fills above a threshold, `RVM` reads it, truncates it and updates the data file. Thus, the log file is used as a buffer between synchronous transaction writes and asynchronous data file updates. The larger the log file, the better the performance of the system is expected to be.

To measure the performance effect of the log file size, we constructed the same experiment as previously, but instead of varying the I/O block size, we vary the log size and we keep the I/O block size constant.

The results of our experiment (number of transactions per second) as a function the log size for I/O block sizes of 128 and 512 bytes are plotted in figures 2 and 3 respectively.

We see that all systems have poor performance for small log sizes (a few Kbytes long), especially for the larger I/O block size. The reason is that very small logs force applications to suffer almost two synchronous write operations per transaction: one to write the dirty data to the log (at transaction commit time), and one to empty the log to the data file. Fortunately, the performance in all systems gets better and almost constant for logs larger than 32 Kbytes. In all cases the performance of both RRVM systems is significantly better than the performance of the unmodified `RVM` system: between one and two orders of magnitude for small transactions.

3.4 Random Accesses

Next, we set out to explore the performance of our transaction-based system in a random accessed environment. Thus, we repeated the previous experiment, but instead of accessing the data file sequentially, the transactions access the data file completely randomly. The performance of our systems for log size 8 Mbytes is shown in figure 4.

We see that all `RRVM-MC`, `RRVM-FDDI`, and `RRVM-ETHERNET` perform much better than `RVM`, as expected. However, the number of transactions per second sustained by `RRVM-FDDI` and `RRVM-ETHERNET` is a little less than 200 (for small transactions), and around 2,500 for `RRVM-MC`: a reduction in the performance observed so far. There are two reasons for this performance reduction: increased number of page faults, and disk I/O operations. When a file is accessed by 10000 transactions, and each transaction accesses 32 bytes of data, a total of 320 Kbytes of data are accessed. If these transactions access sequential data, they will access in total 320/8 = 40 pages. If, however, the transactions access data randomly, they will access many more pages. Thus, both the number of page faults, and the number of disk I/O operations are significantly lower in the case of sequential transaction accesses, as compared to random transaction

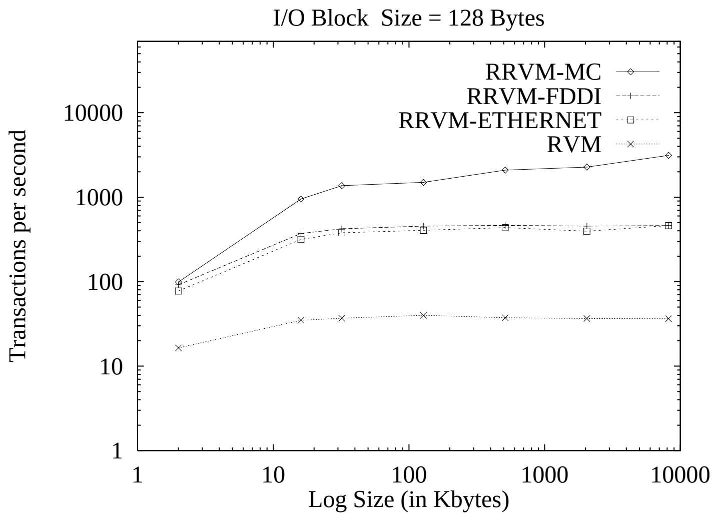

Figure 2: Performance of RVM as a function of the size of the log. Data File = 100 Mbytes.

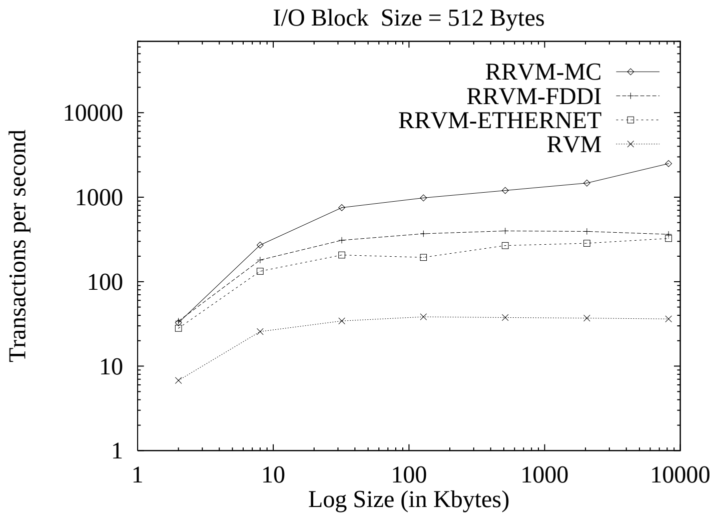

Figure 3: Performance of RVM as a function of the size of the log. Data File = 100 Mbytes.

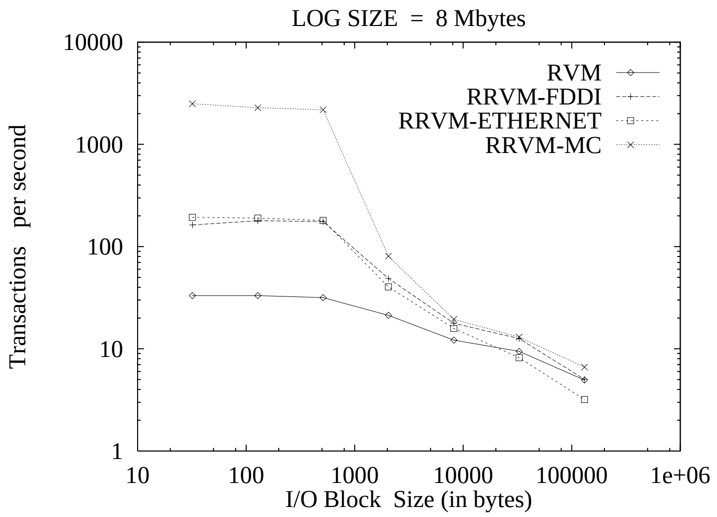

Figure 4: Performance of RVM as a function of the I/O block size - random accesses. Data File = 100 Mbytes, Log File = 8 Mbytes.

accesses. Even though, the transactions per second sustained by RRVM is very good (around 2,500 transactions per second for small transactions, on top of Memory Channel).

3.5 Network Load

Our next set of experiments explores the network load that our RRVM system imposes. [8] Our previous experiments suggest that on top of a lightly-loaded network, RRVM performs much better than traditional `RVM`. What is interesting to explore now, is what will be the performance of RRVM if several instances of it run concurrently and put significant pressure on the interconnection network. Will the network be able to handle the increased communication demands, or will it collapse? To answer this question we constructed the following experiment:

> We create several instances of RRVM clients. For each client, there is also an RRVM server. All clients and all servers run on different workstations. All workstations are connected to the same interconnection network (Ethernet or FDDI). We progressively increase the number of client/server pairs participating in the experiment and measure the transactions per second each RRVM system (client/server pair) is able to sustain. Each RRVM system executes the experiment described in section 3.2.

In our experiments the log size was set to 8 Mbytes and the I/O block size to 32 Bytes. Figure 5 plots the number of transactions per second for *each* RRVM system (client-server pair) as a function of the number of workstations participating in the experiment. First of all, we see that the performance of the unmodified `RVM` system stays the same independent of how many workstations participate in the experiment. This is as expected, since `RVM` makes all traffic to its local disk, and does not put any network load. We also notice that the performance of `RRVM-ETHERNET` decreases with the number of workstations. If there is only one pair of workstations in the network, `RRVM-ETHERNET` sustains close to 500 transactions per second, while when four pairs of workstations participate in the experiment, it sustains close to 150 transactions per second. [9] Even so, its performance is three times better than the performance of unmodified `RVM` which sustains less than 50 transactions per second. Finally, we see that the performance of `RRVM-FDDI` is practically constant even when eight workstations participate in the experiment. The reason is simple: FDDI has ten time more throughput than ETHERNET, and can sustain several heavily-communicating workstations. Furthermore, under heavy load, Ethernet may suffer from increased number of collisions that may lead to throughput collapse.

Figure 6 plots the results of the same experiment for transaction size equal to 2 Kbytes. Again, we observe that the performance of `RRVM-FDDI` stays practically the same, independent of the communicating workstations. Although the performance of `RRVM-ETHERNET` decreases with the number of participating workstations, it is much better than the performance of unmodified `RVM` system.

3.6 Server Load

Although RRVM does not impose any significant network load on top of modern interconnection networks (e.g. FDDI), we would like to investigate whether RRVM imposes any significant computation load on the server workstation. For this reason, we repeated the previous experiments, with the difference that all RRVM servers were running on a single workstation. This experiment puts pressure not only on the network, but on the single server workstation as well. The performance of RRVM as a function of the number of participating clients is shown in figures 7 and 8 for I/O block sizes 32 bytes and 2 Kbytes respectively. Both figures suggest that there is some performance decrease as the number of participating workstations increases, esp. for small block sizes. On the other hand, the *total* number of transactions per second sustained stays particularly high in all cases ranging from 712 to 1076 transactions per second for FDDI-based systems, and from 352 to 544 transactions per second or Ethernet-based systems.

The performance of `RVM` is much lower, reaching at most 128 *total* transactions per second for the four RVM systems executing.

[8] Due to technical difficulties we were not able to run these experiments on top of Memory Channel.

[9] Since *each* client/server pair sustains 150 transactions per second, the *total* number of transactions per second sustained in the Ethernet-based system is $4 \times 150 = 600$.

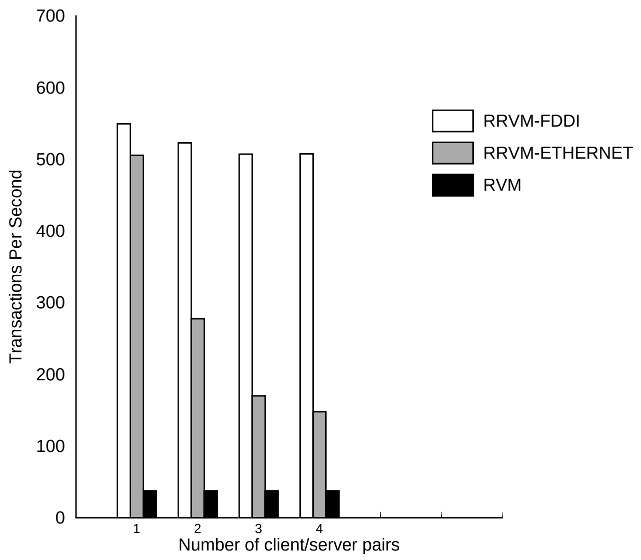

Figure 5: Network Load: Performance of RVM as a function of the network load - sequential accesses - all servers and all clients run on different workstations - I/O block size = 32 bytes.

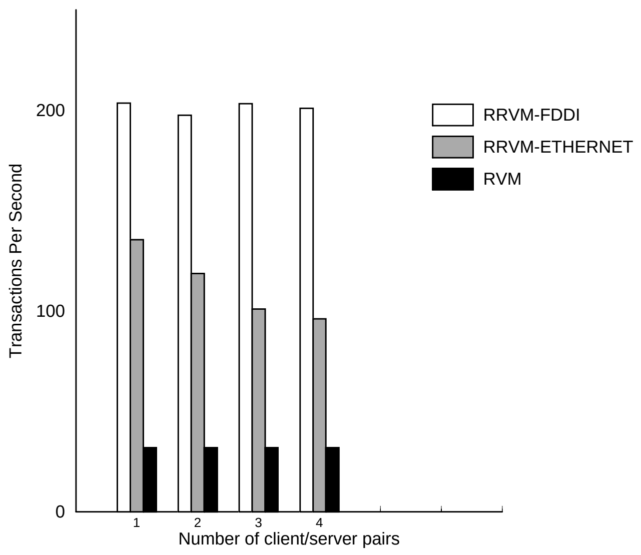

Figure 6: Network Load: Performance of RVM as a function of the network load - sequential accesses - all servers and all clients run on different workstations - I/O block size = 2 Kbytes.

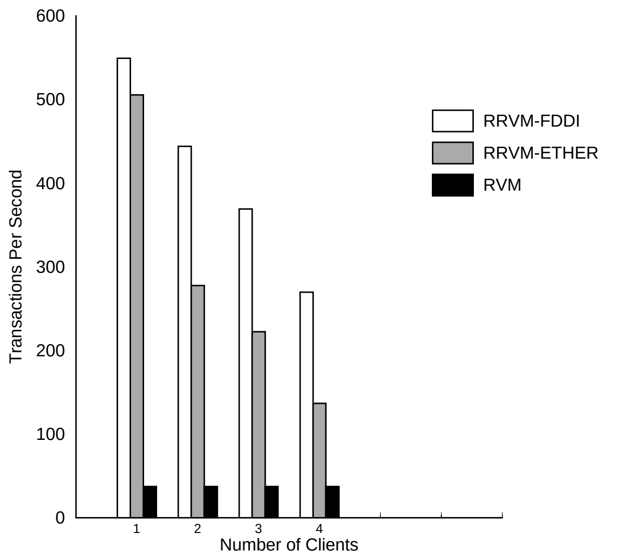

Figure 7: Server Load: Performance of RVM as a function of the number of the participating clients - sequential accesses - all servers run on a single workstation - I/O block size = 32 bytes.

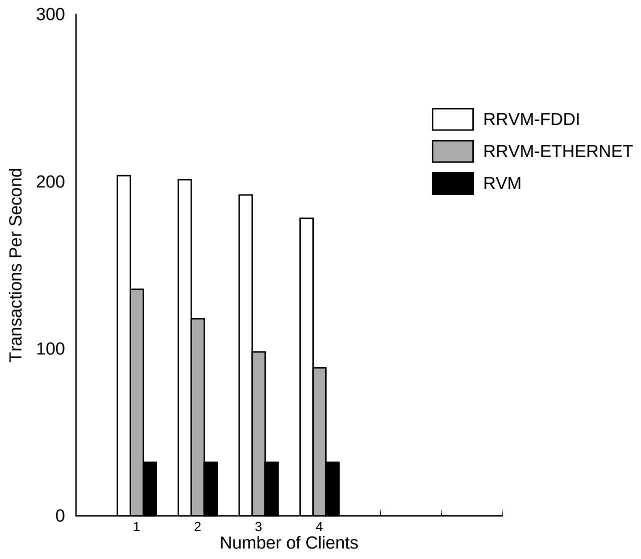

Figure 8: Server Load: performance of RVM as a function of the number of the participating clients - sequential accesses - all servers run on a single workstation - I/O block size = 2 Kbytes.

Accounts (×1024)	Unmodified RVM	RRVM-ETHER	RRVM-FDDI
32	44.24	203	262
128	44.41	201	261
512	44.35	199	260
1024	31.08	193	250

Figure 9: TPCA-A: Sequential Accesses

Accounts (×1024)	Unmodified RVM	RRVM-ETHER	RRVM-FDDI
32	43.2	182	230
128	40.4	144	171
512	41.6	89	96
1024	21.6	53	67

Figure 10: TPCA-A: Random Accesses

3.7 TPC-A

To place our RRVM system in the right perspective with previously published performance results, we run the widely used TPC-A database benchmark on top of it. The same benchmark was run on the original RVM system and its performance was reported in [25]. In the original RVM system, the log file and the data file were stored on *different* local disks, so as to eliminate any interference between the accesses to the different files.

Figure 9 shows the performance of our RRVM systems, and the original (unmodified) RVM system as a function of the number of accounts. In this experiment accesses to the database are sequential. We see that although the original RVM system barely achieves more than 40 transactions per second, `RRVM-ETHERNET` achieves 200 transactions per second, while `RRVM-FDDI` achieves more than 260 transactions per second - an order-of-magnitude improvement over the original RVM.

We make similar observations by looking at the experiments for random accesses and localized accesses in figures 10 and 11 respectively. Both RRVM systems are significantly better than the unmodified RVM. Their performance improvements range from a factor of 2.6 to 6.

3.8 EXODUS

3.8.1 Experimental Environment

Our experimental environment for EXODUS consists of a network of Supersparc-20 workstations. The workstations are connected through a 100 Mbps FDDI, and a 10Mbps Ethernet interconnection network.

3.8.2 OO7

On top of EXODUS we run a common database benchmark called OO7 [5]. We used three versions of EXODUS:

Accounts (×1024)	Unmodified RVM	RRVM-ETHER	RRVM-FDDI
32	43.4	186	239
128	41.8	159	197
512	41.9	127	154
1024	28.7	84	93

Figure 11: TPCA-A: Localized Accesses

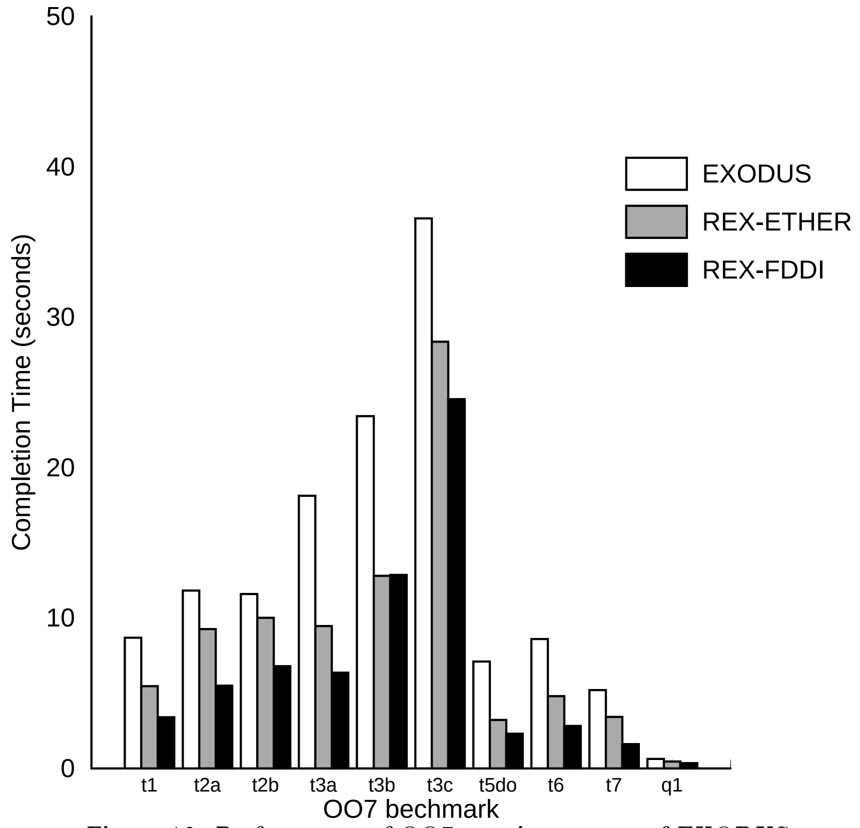

Figure 12: Performance of OO7 running on top of EXODUS.

- `EXODUS`: This is the unmodified EXODUS system [6].
- `REX-FDDI`: This is our modified EXODUS (REX) system running on top of an FDDI interconnection network.
- `REX-ETHERNET`: This is REX running on top of an Ethernet interconnection network.

Figure 12 plots the completion time of various parts of the OO7 benchmark on top of EXODUS. We see that in all cases REX has superior performance compared to the unmodified EXODUS systems. Actually, REX-FDDI is sometimes more than 3 times faster than EXODUS (see for example t5do, and t6). Although we do not see the impressive performance difference we demonstrated in the previous section (since OO7 stresses all aspects of the system, not just transaction commit), our measurements suggest that REX results in noticeable performance improvement over the unmodified EXODUS storage manager.

4 Related Work

Using Remote Main Memory to improve the performance and reliability of I/O in a Network of Workstations (NOW) has been previously explored in the literature. For example, several file systems [1, 8, 18, 21] use the collective main memory of several clients and servers as a large file system cache. Paging systems may also use remote main memory in a workstation cluster to improve application performance [14, 19, 20, 22]. Even Distributed Shared Memory systems, can exploit the remote main memory in a NOW [13, 9] for increased performance and reliability.

The closest of these systems to our research is the Harp file system [21]. Harp uses replicated file servers to tolerate single server failure. Each file server is equipped with a UPS to tolerate power failures, and speedup synchronous write operations. Although RRVM and REX use similar approaches (redundant power supplies and information replication) to survive both hardware and software failures, there are several differences between our work and Harp:

- *Data Granularity:* Our work is concerned mostly with transaction-based systems that make lots of *small* read and write operations. Being able to efficiently read and write a small amount of data is particularly important for the performance of these systems. On the contrary, Harp (by being a file system) can not address data at a granularity finer than a file block. Thus, to read/write even a single byte of data, Harp will have to read/write an entire block of data, which leads to significant performance degradation. Our performance results suggest that on top of a workstation cluster connected via Ethernet, RRVM is able to sustain several hundred (short) transaction operations per second (see figure 1). Published results for Harp [21], suggest that it is able to sustain several tens of NFS operations per second. This implies, that if each transaction needs at least one NFS operation, then the number of transactions per second that Harp will be able to sustain is an order of magnitude lower than our RRVM system.

- *Open User Level Implementation:* RRVM is linked with user applications as a library, outside the operating system kernel. Thus, it is portable and easily modifiable; any ordinary computer user can link RRVM to their program and run it. In contrast, Harp runs inside the operating system kernel, which makes it difficult to port and install. Users will be able to benefit from Harp, only if the file system installed by the system administrators is Harp, or a Harp derivative. Currently there are very few file systems (if at all) that provide functionality similar to Harp. Moreover, Harp adds significant overhead to applications that do not want/need the reliability Harp offers, but are forced to use it and pay for it. On the contrary, in our systems, data replication (for reliability) is only done for database applications, that need it and are willing to suffer its overhead. All other applications run without any intervention from our systems.

Summarizing, Harp is a kernel-level file system that sustains hardware and software failures, while our approach leads to open, portable, flexible, and lightweight user-level transaction-based systems.

The Rio file system changes the operating system to avoid destroying its main memory contents in case of a crash [7]. Thus, if a workstation is equipped with a UPS and the Rio file system, it can survive all failures: power failures do not happen (due to the UPS), and software failures do not destroy the contents of the main memory. Systems like Rio may simplify the implementation of our approach significantly. Unfortunately, few file systems (if any at all) follow Rio's approach (although they should). However, even Rio may lead to data loss in case of UPS malfunction. In these cases, our approach that keeps two copies of sensitive data in two workstations connected to two different power supplies, will be able to avoid data loss.

Network file systems like Sprite [23] and xfs [1, 11], can also be used to store replicated data and build a reliable network main memory. However, our approach, would still result in better performance due to the minimum (block) size transfers that all file systems are forced to have. Moreover, our approach would result in wider portability since, being user-level, it can run on top of any operating system, while several file systems, are implemented inside the operating system kernel.

Franklin, Carey and Livny have proposed the use of remote main memory in a NOW as a large database cache [15]. They validate their approach using simulation, and report very encouraging results. Griffioen *et. al* proposed that DERBY storage manager, that exploits remote memory and UPSs to reliably store a transaction's data [17]. They simulate the performance of their system and provide encouraging results. Although our approach is related to the DERBY system, there are significant differences: (i) we provide a full-fledged implementation of our approach on two independent transaction-based systems, (ii) we demonstrate the performance improvements of our system using the same benchmarks that demonstrated the performance of the original RVM and EXODUS systems, (iii) DERBY places the burden of data reliability to the clients of the database, while we place it to the transaction managers who have better knowledge of how to manage the various resources (memory, disks) in the system.

Feeley *et. al.* proposed a generalized memory management system, where the collective main memory of all workstations in a cluster is handled by the operating system [12]. Their experiments suggest that generalized memory management results in performance improvements. For example, OO7 on top of their system runs up to 2.5 times faster, than it used to run on top of a standard UNIX system. We believe that our approach complements this work in the sense that both [15] and [12] improve the performance of read accesses (by providing large caches), while our approach improves the performance of synchronous write accesses. Thus, if used both, they improve the performance of database applications even further.

To speed up database and file system write performance, several researchers have proposed to use special hardware. For example, Wu and Zwaenepoel have designed and simulated eNVy [27], a large non-volatile main memory storage system built primarily with FLASH memory. Their simulation results suggest that a 2 Gbyte eNVy system can support I/O rates corresponding to 30,000 transactions per second. To avoid frequent writes to FLASH memory, eNVy uses about 24 Mbytes of battery-backed SRAM per Gbyte of FLASH memory. Although the cost of eNVy is comparable to the cost of a DRAM system of the same size, eNVy realizes its cost effectiveness only for very large configurations: for hundreds of Mbytes. Furthermore, although the chip cost of eNVy may be low, its market price will probably be much higher, unless it is massively produced and sold. Thus, eNVy would be used only for expensive and high-performance database servers, and not for ordinary workstations. As another example, Baker *et al.* have proposed the use of battery-backed SRAM to improve file system performance [3]. Through trace-driven simulation they have shown that even a small amount of SRAM reduces disk accesses between 20% and 90% even for write-optimized file systems, like log-based file systems.

5 Conclusions

In this paper we described how to use several workstations in a NOW to provide fast and reliable access to stable storage. Our approach consists of using network main memory to avoid synchronous disk I/O as much as possible. By using data replication and redundant power supplies we increase the reliability of remote main memory, and use it as a short-term non-volatile storage medium.

We have implemented our approach within the EXODUS storage manager and the RVM recoverable virtual memory system. We have experimented with both systems running on top of a Network of Workstations connected with a variety of interconnection networks ranging from the traditional Ethernet, to the high-speed Memory Channel. Based on our implementation experience and performance results we conclude:

- *Our approach can be easily incorporated in existing database systems.* We have implemented our approach on top of two different transaction based systems. Each implementation took only a few weeks of programming, and consists completely of user-level software.
- *RRVM provides significant performance improvements over RVM, even on top of Ethernet interconnection networks.* Our results suggest that for small transactions, RRVM on top of Ethernet is able to sustain 70–500 transactions per second, depending on the transaction access patterns. At the same time, unmodified RVM sustains 30–50 transactions per second - an order of magnitude less. Most of the benefits of our approach are realized on top of the high-speed Memory Channel interconnection network, where RRVM manages to sustain a little more than 2,500 (short) transactions per second.
- *RRVM is able to sustain several clients on top of the same interconnection network.* Our results suggest that even when four RRVM systems operate on top of the same Ethernet network, their performance is 3-4 times better than the performance of `RVM` (for small transaction size). The performance of the same clients on top of FDDI is 5-7 times better than the performance of `RVM`.
- *The performance benefits of our approach will increase with time.* The performance of our approach is highly dependent on the latency and the bandwidth of the interconnection network on top of which it runs. Since the latency and the bandwidth of interconnection networks quickly improve with time, we expect the performance benefits of RRVM to improve at similar rates, at least in the foreseeable future.

Based on our experiments, we believe that remote memory (as demonstrated through RRVM and REX) is a viable alternative to synchronous disk I/O and should be considered seriously for implementation in databases and transaction-based systems in general.

Acknowledgments

This work was supported in part by PENED project "Exploitation of idle memory in a workstation cluster" (2041 2270/1-2-95), in part by the USENIX Association through project "Network RAMDISK", and in part by the ESPRIT/OMI project "ARCHES" (ESPIRIT 20693), funded by the European Union. We deeply appreciate this financial support.

We also thank the Computer Science Department of the University of Rochester, and the Parallab Supercomputing Center, for giving us access to their equipment where several of the described experiments were run. Finally, we thank Manolis Katevenis and Catherine Chronaki who provided useful feedback in earlier versions of this document.

References

[1] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless Network File Systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.

[2] T.E. Anderson, D.E. Culler, and D.A. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[3] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile Memory for Fast, Reliable File Systems. In *Proc. of the 5-th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 10–22, Boston, MA, October 1992.

[4] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. An Implementation of the Hamlyn Sender-Managed Interface Architecture. In *Second USENIX Symposium on Operating System Design and Implementation*, October 1996.

[5] M. Carey, D. DeWitt, and J. Naughton. Ther OO7 Bechmark. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 12–21, 1993.

[6] M. Carey and D. DeWitt et. al. The EXODUS Extensible DBMS Project: An Overview. In S.Zdonik and D.Maie, editors, *Readings in Object-Oriented Database Systems*. Morgan Kaufman, 1990.

[7] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The Rio File Cache: Surviving Operating System Crashes. In *Proc. of the 7-th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 74–83, 1996.

[8] T. Cortes, S. Girona, and J. Labarta. PACA: A Distributed File System Cache for Parallel Machines. Performance under Unix-like workload. Technical Report UPC-DAC-1995-20, Departament d'Arquitectura de computadors, Universitat Politecnica de Catalunya (UPC), June 15 1995.

[9] M. Costa, P. Guedes, M. Sequeira, N. Neves, and M. Castro. Lightweight Logging for Lazy Release Consistent Distributed Shared Memory. In *Second USENIX Symposium on Operating System Design and Implementation*, pages 59–74, October 1996.

[10] M. Dahlin. *Serverless Network File Systems*. PhD thesis, UC Berkeley, December 1995.

[11] M.D. Dahlin, R.Y. Wang, T.E. Anderson, and D.A. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *First USENIX Symposium on Operating System Design and Implementation*, pages 267–280, 1994.

[12] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *Proc. 15-th Symposium on Operating Systems Principles*, pages 201–212, December 1995.

[13] Michael J. Feeley, Jeffrey S. Chase, Vivek R. Narasayya, and Henry M. Levy. Integrating Coherency and Recovery in Distributed Systems. *First USENIX Symposium on Operating System Design and Implementation*, pages 215–227, November 1994.

[14] E. W. Felten and J. Zahorjan. Issues in the Implementation of a Remote Memory Paging System. Technical Report 91-03-09, Computer Science Department, University of Washington, November 1991.

[15] M. Franklin, M. Carey, and M. Livny. Global Memory Management in Client-Server DBMS Architectures. In *Proceedings of the 18th VLDB Conference*, pages 596–609, August 1992.

[16] R. Gillett. Memory Channel Network for PCI. *IEEE Micro*, 16(1):12–18, February 1996.

[17] J. Griffioen, R. Vingralek, T. Anderson, and Y. Breitbart. Derby: A Memory Management System for Distributed Main Memory Databases. In *Proceedings of the 6th Internations Workshop on Research Issues in Data Engineering (RIDE '96)*, pages 150–159, February 1996.

[18] J. Hartman and J. Ousterhout. The Zebra Striped Network File System. *Proc. 14-th Symposium on Operating Systems Principles*, pages 29–43, December 1993.

[19] L. Iftode, K. Li, and K. Petersen. Memory Servers for Multicomputers. In *Proceedings of COMPCON 93*, pages 538–547, 1993.

[20] K. Li and K. Petersen. Evaluation of Memory System Extensions. In *Proc. 18-th International Symposium on Comp. Arch.*, pages 84–93, 1991.

[21] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams. Replication in the Harp File System. *Proc. 13-th Symposium on Operating Systems Principles*, pages 226–238, October 1991.

[22] E.P. Markatos and G. Dramitinos. Implementation of a Reliable Remote Memory Pager. In *Proceedings of the 1996 Usenix Technical Conference*, pages 177–190, January 1996.

[23] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1):134–154, February 1988.

[24] Dolphin Interconnect Solutions. DIS301 SBus-to-SCI Adapter User's Guide.

[25] M. Stayanarayanan, Henry H Mashburn, Puneet Kumar, David C. Steere, and James J. Kistler. Lightweight Recoverable Virtual Memory. *ACM Transactions on Computer Systems*, 12(1):33–57, 1994.

[26] Alec Wolman, Geoff Voelker, and Chandramohan A. Thekkath. Latency Analysis of TCP on an ATM Network. In *Proceedings of the USENIX Winter '94 Technical Conference*, pages 167–179, San Francisco, CA, January 1994.

[27] Michael Wu and Willy Zwaenepoel. eNVy: a Non-Volatile Main Memory Storage System. In *Proc. of the 6-th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 86–97, 1994.