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, GREECE
si@cs.rochester.edu, {markatos|sevas}@ics.forth.gr

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, the performance of transaction-based systems is usually limited by the magnetic disks that are used to hold the data. 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

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) [1].

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 from a (local) magnetic disk [2]. Architecture trends suggest that this disparity between magnetic disks and interconnection networks will continue to increase with time [2].

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. 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.

The first of the above issues has been somewhat explored in the areas of file systems [3, 4, 5], paging [6] and global memory databases for workstation clusters [7, 8]. The thrust of this paper is on exploring the second issue. Transaction-based systems make many small synchronous write operations to stable storage, and thus they are going to benefit significantly from any improvements to synchronous disk write operations.

*This work was supported in part by PENED project "Exploitation of idle memory in a workstation cluster" (2041 2270/1-2-95). Several of the described experiments were performed at the Parallab High Performance Computing Centre and the University of Rochester. We thank them all for their support.

†Current affiliation: Computer Science Department, University of Rochester, Rochester, NY 14627–0226

‡Evangelos P. Markatos is also with the University of Crete, Department of Computer Science

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 it in two existing transaction-based systems: the EXODUS storage manager [9], and the RVM (Recoverable Virtual Memory) System [10]. Section 2 describes the design and the implementation of our systems. 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 memory), makes a few calculations on the data, writes its results back (via a Log file), and then commits. Disk read operations can be sped up with the help of large main memory caches. Disk write operations at transaction commit time, however, are difficult to avoid since the data and meta-data have to reach stable storage in order to protect the application from crashes.

We believe, that in NOW 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. The main sources of system crashes that may lead to data loss are:

Software failures are the result of software malfunctions, operating system crashes, etc. Data that must survive software failures are replicated to the main memories of (at least) two workstations so they are able to survive software failures with high probability. [1]

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).

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.

2.2 EXODUS and RVM

To illustrate our approach we have modified a lightweight transaction-based system called RVM [10] and the EXODUS storage manager [9] 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 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 transaction commit time using synchronous disk write operations. In our modified systems, we substitute each synchronous transaction commit operation with the following operations:

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.

[1] 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.

3. Eventually, the data reach the magnetic disk asynchronously.

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). [2]

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).

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

3.1.1 Experimental Environment

Our experimental environment consists of a network of eight DEC Alpha 2000 workstations with Digital UNIX running at 233 MHz, equipped with 128 MBytes of main memory each and a 6GB local disk. The workstations are connected through an Ethernet, an FDDI, and a Memory Channel Interconnection Network [11].

In our experiments we demonstrated the performance of RRVM and compared it with its predecessor RVM [10]. We have experimented with four system configurations:

- `RVM`: Is the unmodified RVM system.
- `RRVM-ETHERNET`: Is RRVM running on top of Ethernet.

[2] 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).

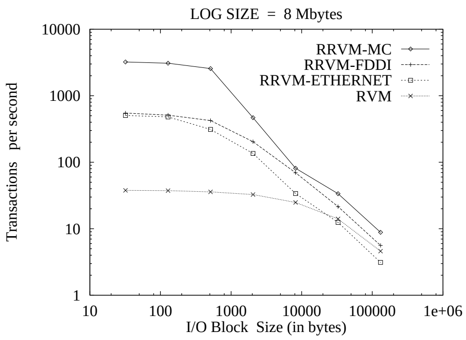

Figure 1: Performance of RVM as a function of the I/O block size. Data file = 100 Mbytes, Log File = 8 Mbytes.

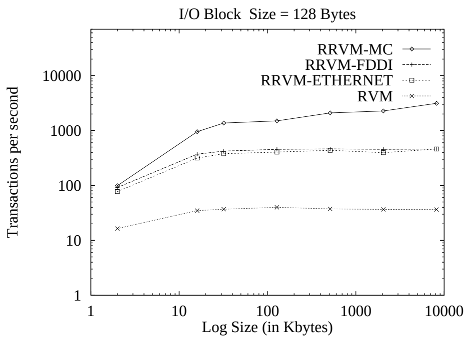

Figure 2: Performance of RVM as a function of the size of the log. Data File = 100 Mbytes.

- `RRVM-FDDI`: Is RRVM running on top of FDDI.
- `RRVM-MC`: This is RRVM running on top of the Memory Channel 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, which agrees with previously reported results [10]. However, the performance of `RRVM-ETHERNET`, `RRVM-FDDI` and `RRVM-MC` manages to sustain close to 3,000 transactions per second: almost two orders of magnitude improvement over unmodified `RVM`. As the I/O block size increases, badwidth and not latency is the dominating factor, so the performance of all systems converges.

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 beyond a threshold, `RVM` reads it, truncates it and updates the data file.

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 size of 128 bytes are plotted in Figure 2. In all cases the performance of both RRVM systems is significantly better than the performance of the unmodified `RVM` system.

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 3.

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 perfor-

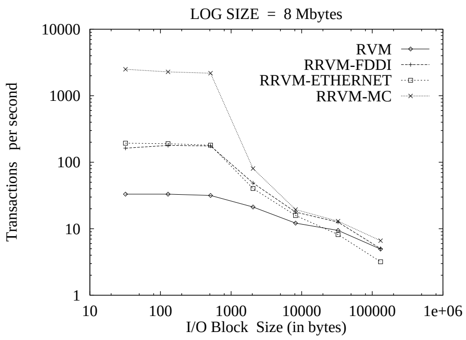

Figure 3: Performance of RVM as a function of the I/O block size - random accesses. Data File = 100 Mbytes, Log File = 8 Mbytes.

mance 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 accesses.

3.5 Network and Server Load

Our next set of experiments explore how well our system performs under a loaded network and under a loaded server.

To answer the first 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. 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 4 presents the number of transactions per second for *each* RRVM system (client-

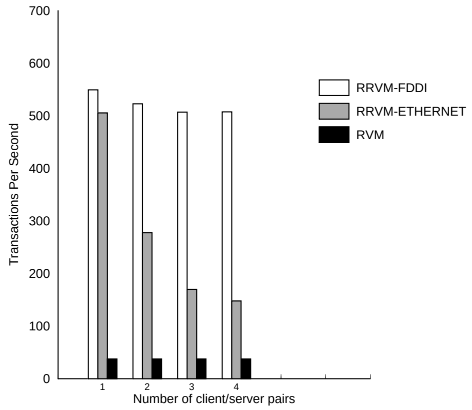

Figure 4: Network Load: Performance of RVM as a function of the network load - sequential accesses - all servers and all clients run on different workstations - I/O block size = 32 bytes.

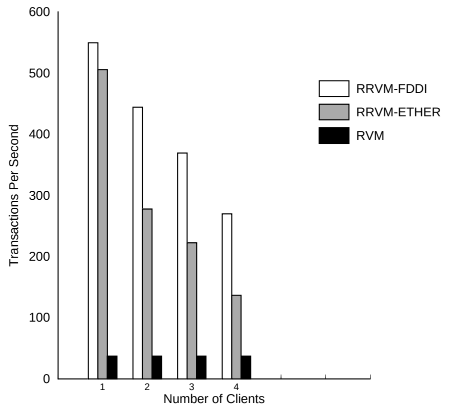

Figure 5: Server Load: Performance of RVM as a function of the number of the participating clients - sequential accesses - all servers run on a single workstation - I/O block size = 32 bytes.

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` only accesses 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 whereas the performance of `RRVM-FDDI` is practically constant. This is expected since FDDI has ten times the throughput of Ethernet.

The next experiment puts pressure not only on the network, but on the single server workstation as well. It is the same as the previous

Accounts (×1024)	RVM (Unmodified)	RRVM ETHER	RRVM FDDI
TPCA-A: Sequential Accesses			
32	44.24	203	262
128	44.41	201	261
512	44.35	199	260
1024	31.08	193	250
TPCA-A: Random Accesses			
32	43.2	182	230
128	40.4	144	171
512	41.6	89	96
1024	21.6	53	67
TPCA-A: Localized Accesses			
32	43.4	186	239
128	41.8	159	197
512	41.9	127	154
1024	28.7	84	93

Figure 6: Transactions per second sustained by the TPCA-A Benchmark.

one with the difference that all RRVM servers are running on a single workstation. The performance of RRVM as a function of the number of participating clients is shown in Figure 5, for I/O block size of 32 bytes. As expected the performance of RRVM decreases as the number of participating workstations increases, but it is still significantly better than that of `RVM`.

3.6 TPC-A

To place our RRVM system in the right perspective with previously published performance results, we ran the widely used TPC-A database benchmark (which was also used to evaluate the original system [10]) , on top of it.

Our results, summarized in Figure 6, show the number of transactions per second that each system can achieve. The experiments present sequential, random and localized accesses. `RRVM-ETHERNET` and `RRVM-FDDI` outperform the original RVM system by an order of magnitude in almost every case.

3.7 EXODUS

3.7.1 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.

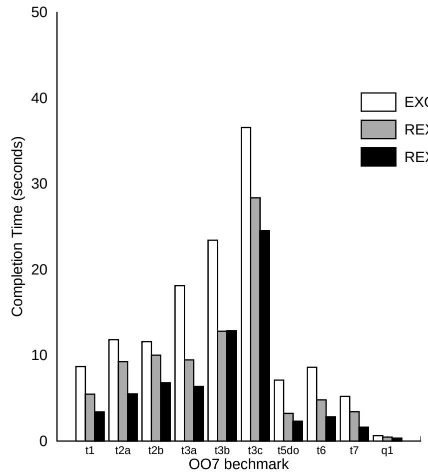

Figure 7: Performance of OO7 running on top of EXODUS.

3.7.2 OO7

On top of EXODUS we ran a common database benchmark called OO7 [12]. We used three versions of EXODUS:

- `EXODUS:` Is the unmodified EXODUS system [9].
- `REX-FDDI:` Is our modified EXODUS (REX) system running on top of FDDI.
- `REX-ETHERNET:` Is REX running on top of Ethernet.

Figure 7 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 NOW has been previously explored in the literature in file systems [3, 4, 5], pagers [13, 14, 6], even Distributed Shared Memory systems [15].

The closest to our research is the Harp file system [5]. 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 we use similar techniques, there are several differences between our work and Harp.

Data Granularity: Our work is focused on transaction-based systems that make a lot of *small* read and write operations. Harp operates on file blocks and is able to sustain several tens of such operations per second, according to the published results. This is an order of magnitude less than RRVM for small transactions.

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.

The Rio file cache has been designed to survive operating system crashes by not destroying its main memory contents in case of a crash [16]. Systems like Rio may simplify the implementation of our approach significantly. However it requires operating system kernel changes not necessary in our approach.

Network file systems like Sprite [17] and xfs [3, 18], 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.

Franklin *et al.* have proposed the use of remote main memory in a NOW as a large database cache [7]. 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 [19]. We believe that our approach complements this work in the sense that both [7] and [19] improve the performance of read accesses (by providing large caches), while our approach improves the performance of synchronous write accesses.

Papathanasiou and Markatos [20] describe a system to improve performance of main memory databases on top of Networks of Workstations. The approach described in this paper is applicable to all databases, whether they fit in main memory or not.

Griffioen *et al.* proposed the DERBY storage manager, that exploits remote memory and UPSs to reliably store a transaction's data [8]. 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 on the transaction managers who have better knowledge of how to manage the various resources (memory, disks) in the system.

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 [21], and Baker *et al.* have proposed the use of battery-backed SRAM [22]. Our approach has the advantage that is does not need any specialized hardware

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. Based on our implementation experience and performance results we conclude:

Our approach can be easily incorporated in existing database systems. It only took a few weeks of programming to implemented our approach on top of two different transaction based systems.

RRVM provides significant performance improvements over RVM, even on top of Ethernet interconnection networks. RRVM on top of Ethernet is able to sustain 70–500 transactions per second compared to 30– 50 of the unmodified RVM.

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`.

The performance benefits of our approach will increase with time. 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.

Based on our experiments, we believe that remote memory 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

We like to thank Manolis Katevenis and Catherine Chronaki who provided useful feedback in earlier versions of this document, as well as the anonymous referees.

References

[1] T.E. Anderson, D.E. Culler, and D.A. Patterson. A case for NOW. *IEEE Micro*, February 1995.

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

[3] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. *TOCS*, February 1996.

[4] J. Hartman and J. Ousterhout. The zebra striped network file system. *14th SOSP*, December 1993.

[5] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams. Replication in the Harp file system. *13th SOSP*, 1991.

[6] E.P. Markatos and G. Dramitinos. Implementation of a reliable remote memory pager. In *Usenix Technical Conference*, 1996.

[7] M. Franklin, M. Carey, and M. Livny. Global memory management in client-server dbms architectures. In *18th VLDB Conference*, August 1992.

[8] J. Griffioen, R. Vingralek, T. Anderson, and Y. Breitbart. Derby: A Memory Management System for Distributed Main Memory Databases. In *RIDE '96*, February 1996.

[9] M. Carey et al. The EXODUS extensible DBMS project: An overview. In *Readings in Object-Oriented Database Systems*. 1990.

[10] M. Satyanarayanan, Henry H Mashburn, Puneet Kumar, David C. Steere, and James J. Kistler. Lightweight recoverable virtual memory. *TOCS*, 1994.

[11] R. Gillett. Memory channel network for pci. *IEEE Micro*, 16(1):12–18, February 1996.

[12] M. Carey, D. DeWitt, and J. Naughton. The OO7 bechmark. In *ACM SIGMOD*, 1993.

[13] L. Iftode, K. Li, and K. Petersen. Memory servers for multicomputers. In *COMPCON 93*, 1993.

[14] K. Li and K. Petersen. Evaluation of memory system extensions. In *ISCA*, 1991.

[15] M. Costa, P. Guedes, M. Sequeira, N. Neves, and M. Castro. Lightweight logging for lazy release consistent distributed shared memory. In *OSDI*, 1996.

[16] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The Rio file cache: Surviving operating system crashes. In *ASPLOS*, 1996.

[17] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite network file system. *TOCS*, February 1988.

[18] M.D. Dahlin, R.Y. Wang, T.E. Anderson, and D.A. Patterson. Cooperative cahing: Using remote client memory to improve file system performance. In *OSDI*, 1994.

[19] 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 *15th SOSP*, 1995.

[20] A. E. Papathanassiou and E. P. Markatos. Lightweight transactions on networks of workstations. In *ICDCS*, 1998.

[21] Michael Wu and Willy Zwaenepoel. eNVy: a non-volatile main memory storage system. In *ASPLOS*, 1994.

[22] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *ASPLOS*, 1992.