On Using Network RAM as a non-volatile Buffer

Dionisios Pnevmatikatos Evangelos P. Markatos
Gregory Maglis Sotiris Ioannidis*
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 227

August 1998

Abstract

File systems and databases usually make several synchronous disk write accesses in order to make sure that the disk always has a consistent view of their data, so that it can be recovered in the case of a system crash. Since synchronous disk operations are slow, some systems choose to employ *asynchronous* disk write operations, at the cost of low reliability: in case of a system crash all data that have not yet been written to disk are lost.

In this paper we describe a software-based Non Volatile RAM system that provides the reliability of synchronous write operations with the performance of asynchronous write operations. Our system takes a set of volatile main memories residing in independent workstations and transforms it into a non-volatile memory buffer - much like RAIDS do with magnetic disks. It then uses this non-volatile buffer as an intermediate storage space in order to acknowledge synchronous write operations before actually writing the data to magnetic disk, but after writing the data to (intermediate) stable storage.

Using simulation and experimental evaluation we demonstrate the performance advantages of our system.

1 Introduction

To speedup up I/O accesses, most file systems use large caches that keep disk data in main memory as much as possible. Published performance results suggest that large caches improve the performance of read access significantly, but do not improve the performance of write accesses by more than a mere 10% [3]. This is not due to insufficient cache size, but mostly due to the fact that data need to be written to disk to protect them from system crashes and power failures. In the Sprite Operating System for example [20], dirty data that reside in the main memory cache are written periodically (every 30 seconds) to the disk(s). Databases require that their data be written to disk even more often: usually at each transaction commit time.

To decouple the application performance from the high disk latency, Fast Non-Volatile RAM (NVRAM), has been proposed as a mean of speeding up synchronous write operations [2, 27, 10].

---

*Sotiris Ioannidis is now with the University of Rochester. Evangelos P. Markatos, Gregory Maglis, and Dionisions Pnevmatikatos are also with the University of Crete.

Applications that want to write their data to disk, first write their data to NVRAM (at main memory speed), and then, they continue with their execution. At the same time, the data are also asynchronously written from the NVRAM to the disk. If the system crashes after the data are written in the NVRAM, but before they are written to the disk, the data are not lost, and can be recovered from the NVRAM when the system comes up again. NVRAM modules can have various forms, but most usually they consist of battery-backed low power SRAM. The main disadvantage of these products is their high price - they cost four to ten times as much as volatile DRAM memory of the same size [2]. To reduce cost and increase performance, researchers have proposed the use of FLASH (EEPROM) memory to construct large non-volatile main memory storage systems [27]. FLASH chips retain their contents even after a power failure, have price comparable to DRAM memory of the same size, but have very high write latency. An NVRAM system consisting of FLASH chips coupled with a small amount of battery-backed SRAM has low read and write latency, but also has a high-cost, making it suitable only for high-end workstations.

In this paper we describe a software-only method to develop an NVRAM system, which does not require specialized hardware. To provide a non-volatile storage space, we use several independent, but *volatile* data repositories. Data that need to be written to non-volatile storage are written to *more than one* volatile storage repository. Data loss would occur, if *all* volatile storage repositories lose their data, an event with very low probability for all practical purposes. Our system uses the main memories of the workstations in a workstation cluster, as the independent storage repositories. Thus, if an application wants to write some data to non-volatile memory, it writes the data in its own memory, and the memory of another workstation in the same cluster. As soon as the data are written in the main memory of another workstation, the application is free to proceed. While the application computes, the data are also asynchronously written to disk. If the workstation on which the application runs crashes before the data are completely transferred to disk, the data are not lost, and can be recovered from the remote memory where they also reside. Both local and remote workstations are connected to independent power supplies, so that a power outage will not result in data loss for *both* workstations.

The notion of redundancy has been extensively used in the past to provide various forms of reliability, ranging from RAIDS [8] to large scale geographically distributed databases. Recent architecture trends in the area of interconnection networks, and Networks of Workstations (NOWs) make the idea of using the main memory of remote workstations (within the same workstation cluster) to develop an NVRAM system more attractive than ever because:

- **Bandwidth in Local Area Networks has increased rapidly**: current computer networks consisting of Fast Ethernet and ATM connections are not uncommon. These connections provide a bandwidth of 100-155 Mbps. Multi-Gigabit-per-second networks have also appeared in the market. Thus, interconnection network bandwidth is now comparable (within the same order of magnitude) to main memory bandwidth, which implies that memory to memory transfers between workstations can happen at a rate similar to memory-to-memory transfers within a workstation, and of course much faster than disk-to-memory transfers.

- **The Latency of Local Area Networks has decreased significantly:** several computer networks provide end-to-end latency as low as a few microseconds [5, 13, 19, 23].

Optimized software implementation on standard ATM networks provide latency as low as a few tens of microseconds [24, 25]. Thus, the transfer of small amounts of information between main memories takes only few tens of microseconds, while disk-to-memory transfers require at least a few milliseconds, even for very small amounts of information.

The above architecture trends suggest that memory to memory transfers between workstations in the same LAN is significantly faster than disk to memory transfers. Both the latency and the bandwidth of remote memory are orders of magnitude better than these of magnetic disks. Thus, an implementation of NVRAM based on memory-to-memory transfers is bound to be significantly faster than an implementation of NVRAM based on memory-to-disk transfers.

In this paper we describe such a software-only implementation of an NVRAM system. Section 2 places our work in the context of the current state of the art. Section 3 describes the design of our system on a network of workstations. Section 4 presents our performance results. Finally, section 5 summarizes our work, and concludes the paper.

2 Related Work

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.

Baker *et al.* have proposed the use of NVRAM to improve file system performance [2]. Through trace-driven simulation they have shown that even a small amount of NVRAM reduces disk accesses between 20% and 90% even for write-optimized file systems, like log-based file systems. Their main concern, however, was that NVRAM may not be cost effective yet. Our work is complementary to that of Baker *et al.*, since we propose an inexpensive software-only method to implement NVRAM, and realize the performance benefits they have predicted.

Recently, research groups have started exploring the issues in using remote memory in a workstation cluster to improve file system performance [1, 9, 11, 20] and paging performance [4, 12, 14, 18]. While all these systems use remote memory as a large file system cache, we propose to use remote memory as a fast non-volatile storage medium. That is, most of the previous work has used remote memory to improve *read* accesses (through caching), while we propose to use remote memory to improve *synchronous write* accesses (through the creation of a non-volatile buffer). We view our work as complimentary to previous work, in that they can easily be extended to incorporate our approach. Most of the mentioned file systems already keep at least two copies of several pages for performance reasons, and thus can be extended to use these copies for reliability reasons as well. In that sense our NVRAM system can improve the performance of Persistent and Recoverable main memory systems [6, 22].

The Harp file system uses replicated file servers to tolerate single server failures [16] and speedups write operations as follows: each file server is equipped with a UPS to tolerate power failures, and disk accesses are removed from the critical path, by being replaced with communication between the primary and backup servers. Although our work and Harp use similar approaches (redundant power supplies and information replication) to survive both hardware and software failures, there are several differences, the most important being that we view network memory as a temporary non-volatile buffer space, while Harp uses full data mirroring to survive server crashes.

The Rio file system avoids 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. 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. In effect, Rio makes a single workstation more reliable, while our approach achieves reliability through redundancy. It is like making a single disk more reliable vs. using a RAID. Vista [17] is a recoverable memory library being implemented on top of Rio.

Ioanidis *et al.* have proposed the use of remote memory to speed up synchronous write operations used in the Write Ahead Log (WAL) protocol [15]. In their approach, they replicate the Log file in two main memories and substitute synchronous disk write operations with synchronous remote memory write operations and *asynchronous* disk write operations. PERSEAS [21] is another user-level transaction library that uses (remote main memory) mirroring to survive crashes. In contrary to [15] and [21], we describe and evaluate a kernel-level system that will benefit all applications running on top of it, not only applications that have been compiled and linked with the special libraries described in [15, 21].

Our work is also related to previous work in RAIDs [8, 26], in that we both use redundancy to improve reliability and recover lost data in case of a crash. However, our work focuses in recovering lost main memory information, while the work in RAIDs focuses on recovering lost disk information.

Compared to previous approaches in developing NVRAM systems, our work has the following advantages:

- We propose an *inexpensive* way to built NVRAM systems. Our system uses the existing and *otherwise idle* main memory in a workstation cluster. Previous approaches to building NVRAM were either too expensive (e.g. battery-backed SRAM systems), or too slow (e.g. magnetic disks), or suitable for systems that require hundreds of MBytes of NVRAM (e.g. FLASH memory).

- Our approach provides a *fast and light-weight recovery* mechanism. In traditional NVRAM systems, if the workstation that has the battery-backed SRAM chips, or the FLASH chips crashes and takes a long time to come up again (e.g. possibly due to irrecoverable hardware problems), the information contained in the NVRAM system is not accessible. The only way to access the information is to manually remove the NVRAM cards from the workstation, plug them in another workstation, and access them from their new place. This procedure may induce several minutes, or even hours of idle time - which is in several cases as frustrating as data loss. In our NVRAM system, if the client workstation crashes, the data are still accessible, since they are in the main memory of another workstation and can be accessed through the network within milliseconds. If, on the other hand, the server crashes, the data are still in the main memory of the client and are being scheduled to be (asynchronously) written on the disk. Data loss would occur only if both client and server do down at the same time. [1]

[1] In this case, however, we could use two or even three servers to make data available even after the crash of two-

3 Remote Memory NVRAM

The computing environment we assume for this paper is a workstation cluster: a set of workstations[2] connected with a high-speed network. Applications that need to use non-volatile RAM are called client applications and the workstations on which they execute on, are called client workstations. Within the same cluster, there exist some server workstations that run NVRAM server processes. Server workstations are either connected to a different power supply from client workstations, or they are connected to a Uninterrupted Power Supply (UPS). In this way, a power failure in a client workstation will not imply a power failure in the server workstation as well, leading to data loss.

The purpose of the NVRAM servers is to accept data from clients so that data reside in at least two workstations: the client and at least one server. Each NVRAM server allocates a main memory segment (hereafter called the NVRAM segment) which it uses to write the data it accepts from the clients. A client that wants to store its data into stable storage, before being able to proceed, sends its data to the NVRAM server and *asynchronously* sends the data to the disk. As soon as the NVRAM server acknowledges the receipt of the data, the client is free to proceed with its own work.

An NVRAM server continually reads data sent by the clients, and stores them in its own memory. If its NVRAM segment fills up, the server sends a message back to the client telling it to synchronously write to the disk all its data that correspond to the NVRAM segment. [3] After the client flushes its data to the disk, it sends a message to the server to recycle the space used by those data. Effectively, the server acts as a fast non-volatile buffer between the client and its disk. This fast non-volatile buffer may improve the performance of the file system significantly as we will see in section 4.

It is possible that a client workstation crashes *after* it sends its data to the NVRAM server, but *before* the data are safely written to the disk. In this case, the data are not lost, and can still be found in the main memory of the NVRAM server. When the crashed workstation reboots, it will read its data back from the NVRAM server.

If the server detects a lost client, it sends all its main memory data to a magnetic disk. If the client detects a lost server, or a disconnected network, it synchronously writes all its data to disk, and continue doing so for all `msync` (synchronous disk write) operations, until the server, or the network is up and running again. We should emphasize that all the above situations do not lead to data loss, but only a graceful degradation of system performance. Since, however, workstation crashes and network disconnections do not last for long, we expect the performance degradation due to them to be unnoticeable. Our system will experience data loss, only when both the client and the server crash within a short time interval, an event with very low probability. If for example, the client and the server crash independent from each other once a month, both of them will crash at the same time interval (of e.g. one minute) once every a thousand years. It is true, however, that several machines may go down within a short time interval, but this is usually a result of a scheduled shutdown, in which case our system can also

three workstations. Although high degrees of data replication seems to add significant overhead our performance results suggest that sending data to remote main memories over a fast network in negligible compared to sending data to a magnetic disk.

[2]In this paper we will use the term workstation to mean either a workstation or a high-end personal computer.

[3]Most probably all these data would already reside in the disk, since they were *asynchronously* written to the disk when they were initially sent to the NVRAM server.

shut down gracefully (i.e. flush dirty data to magnetic disk) as well.

Our NVRAM system instead of using a *synchronous disk write* operation at commit time (to force the data to the disk), it uses a *synchronous network write* operation (to force a copy of the data to the remote server), plus an asynchronous disk write operation. Since both the latency and the bandwidth of modern networks are significantly better than the latency and the bandwidth of modern magnetic disks, we expect that the latency of a synchronous network write operation to be much lower than that of a synchronous disk operation.

4 Performance Evaluation

4.1 Experimental Evaluation

Our experimental environment consists of a network of DEC Alpha 2000 workstations, running at 233 MHz equipped with 128 MBytes of main memory each. The workstations are connected through a 100 Mbps FDDI, and an Ethernet interconnection network. Each workstation is equipped with a 6GB local disk.

In our experiments we will demonstrate how our software-based NVRAM system can improve the performance of software systems that make frequent use of synchronous disk write operations (like file systems and databases). Traditionally, synchronous disk write operations block the system, until the data are safely written to disk. In our NVRAM system instead, a synchronous disk write operation will be implemented as a synchronous network write operation, and an *asynchronous* disk write operation. Thus, instead of waiting for the data to be safely written to the disk, the system will send a copy of the data to the remote memory server, and wait for the data to be written to the remote memory. In this way, our NVRAM system replaces a synchronous disk write operation with a synchronous network write operation, which we expect to result in performance improvement.

We have experimented with three system configurations:

- `DISK`: This is a traditional system unmodified. Synchronous write operations are handled by the operating system without our intervention. The operating system sends the write requests to the disk and waits for them to complete.
- `NVRAM-FDDI`: This is our NVRAM system. Synchronous write operations to the disk are synchronously written to remote memory and asynchronously to disk. Client and server applications communicate via an FDDI interconnection network.
- `NVRAM-ETHERNET`: This is the same system as previously, with the exception that client and server applications communicate via an Ethernet interconnection network.

4.1.1 Sequential Accesses

The first question we set out to answer about our experimental remote memory NVRAM system, is its performance. Specifically, we would like to know how many non-volatile write operations (per second) our system is able to handle, and how it is compared with traditional non-volatile systems (e.g. magnetic disks). Thus, we constructed the following experiment:

> We open a reasonably large file (100 Mbytes long), and write sequentially to it in blocks 8 Kbytes long. After each block is modified, we call the `msync` operation to

	DISK	NVRAM-ETHER	NVRAM-FDDI
Operations per second	22.57	51.2821	86.2069

Table 1: Performance (in operations per second) for various system configurations (NVRAM Segment Size = 1 Mbyte). We see that our NVRAM system achieves 2-4 times higher performance than the traditional magnetic disk system.

make sure that the block is written into stable memory, and continue with writing the next block.

The number of Non-Volatile block write operations per second is shown in Table 1 for the three systems: `DISK`, `NVRAM-FDDI`, `NVRAM-ETHERNET`. We immediately notice that the performance of `NVRAM-FDDI`, and `NVRAM-ETHERNET` is 2-4 times better than the performance of `DISK`.

4.1.2 The influence of NVRAM Segment Size

Next, we set to find out how the size of the NVRAM Segment influences the performance of the system. Our intuition suggests that the larger the size of the remote main memory is used for NVRAM, the better the performance of the system will be. However, we would like to know just how much of remote main memory would be enough to decouple the performance of user applications that need non-volatile storage from the high disk overheads. So, we repeat the previous experiment, but instead of varying the block size, we vary the size of the remote memory used by the NVRAM server. Figure 1 plots the performance of the system as a function of the NVRAM size for the `NVRAM-FDDI`, `NVRAM-ETHERNET`, and `DISK` policies. We immediately see that the performance of the `DISK` is not influenced by the size of the NVRAM memory, which is expected. We also see that the performance of the software-based NVRAM systems improves with the size of the NVRAM memory. Initially, both `NVRAM-FDDI` and `NVRAM-ETHERNET` increase sharply with the size of the NVRAM memory, and then they flattened out. Figure 1 suggests that using 512 Kbytes of remote memory is enough to achieve (almost) the best performance attained. Increasing the size of the NVRAM memory up to 10 Mbytes does not improve performance significantly. These results are encouraging, and suggest that an NVRAM server can support multiple clients with only modest memory requirements.

4.2 Simulation

To further study the performance of the NVRAM approach under more realistic conditions and under varying system parameters, we used trace-driven simulation.

4.2.1 Workload

The workload that drives our simulations consists of traces taken from database processing benchmarks. We use the same benchmarks used by Lowell and Chen [17] to measure the performance of RVM [22], and Vista [17]. The benchmarks used include:

- *debit-credit:* a processes banking transactions very similar to the TPC-B.

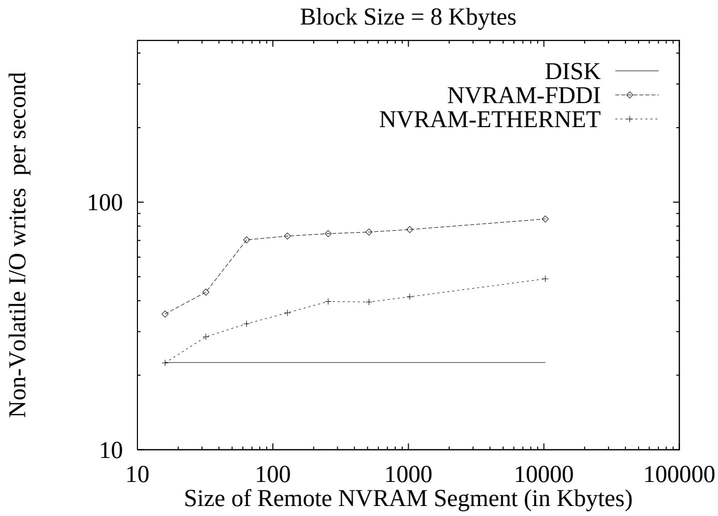

Figure 1: Performance of NVRAM (on top of Ethernet and FDDI) and comparison to synchronous DISK performance, as a function of the size of the available remote RAM. We see that as little as 1 Mbyte of remote RAM is enough to achieve significant performance improvements compared to DISK.

- *order-entry:* a benchmark that follows TPC-C and models the activities of a wholesale supplier.

We run the benchmarks and obtained traces of their `transaction_commit` operations. For each transaction, we recorded -at commit time- a list of all the database regions that are modified and must be sent to stable storage. Our traces record the regions at the locking boundary of the application, and are therefore more detailed than block-based traces. Later, we map each of these regions into (possibly multiple) operating system pages or disk blocks.

4.2.2 The simulator

We simulate three system configurations:

- `DISK`: At each `transaction_commit` time we invoke the (simulated) `msync` system call which synchronously sends to the magnetic disk all operating system pages that have been modified by the transaction. We use the disk simulator provided by Greg Ganger, Bruce Worthington, and Yale Patt from `http://www.ece.cmu.edu/~ganger/disksim/`. The disk simulated is a Seagate ST41601N (modified to include a cache of 320 Kbytes).

- `DISK-ASYNC`: This is the same system as `DISK` with the difference that the write operations to disk proceed *asynchronously*, that is, the `transaction_commit` operation returns as soon as the data have been scheduled to be written (at some later time) on disk. It is obvious that this configuration does not provide the reliability implied by `transaction_commit` or by `msync`. However, we use it as a upper bound on the performance of any system that would use layer of Non-volatile memory between the file system disk and the disk.

- `NVRAM`: This is our proposed system, which when the `msync` system call is invoked, it writes data to remote memory synchronously and to the magnetic disk asynchronously.

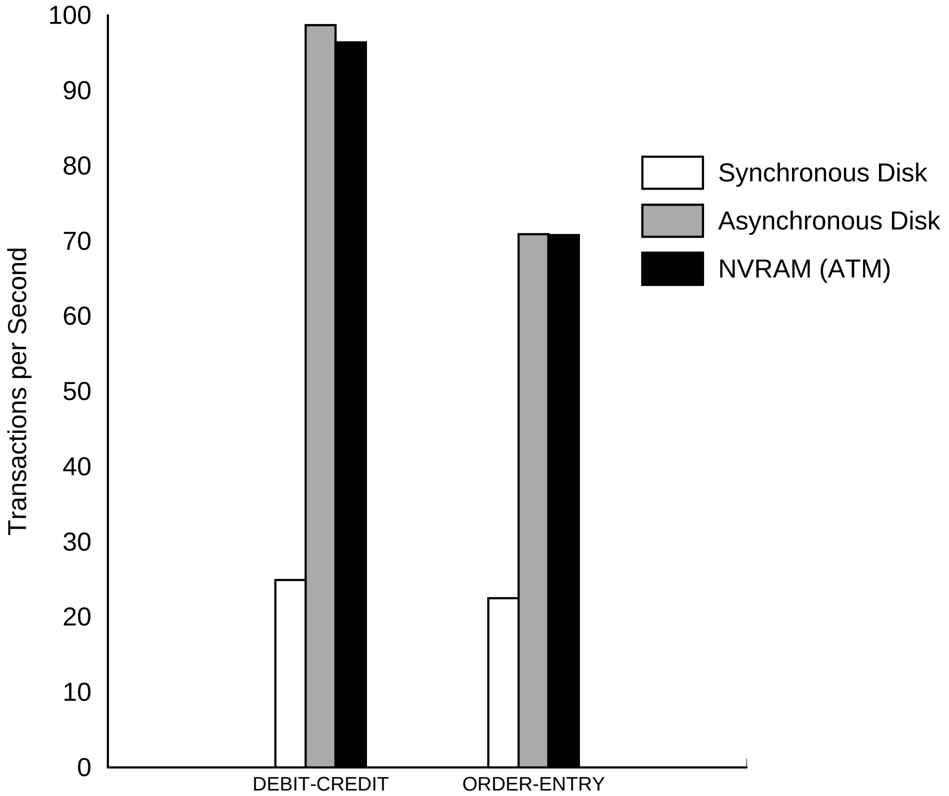

Figure 2: Performance (in transactions per second) of the three system configurations for our two applications.The disk simulated is a Seagate ST41601N.

The `msync` system call returns as soon as the data have been safely written to remote memory.

Our simulations were run on a cluster of SUN ULTRA II workstations connected with an ATM switch. The disk access times are simulated times provided by the disk simulator, while the data that should be sent to remote memory (in the simulated system) are *actually* sent to the remote server's memory using TCP/IP on top of ATM.

Figure 2 shows the (simulated) performance of our system as compared to traditional disk-based systems for a block size of 8Kbytes. We see that a synchronous disk suffers a significant penalty compared to an asynchronous one (roughly 1/3rd to 1/4th of the performance); this result is of course expected and represents the cost of a simple implementation of reliability. Furthermore, we see that our NVRAM system achieves almost the same throughput of the asynchronous disk (within 3%); compared to the synchronous disk, the NVRAM system performance is between 3 to 4 times better. As we describe earlier, the NVRAM system achieves this performance improvement by decoupling all the synchronous transaction operations from the disk. In addition, the asynchronous disk operations may (and actually do) overlap with the synchronous network operations, with the net effect of hiding some part of the synchronous operations. Note however, that even the NVRAM system is still limited by the disk because is is fast enough to saturate the effective bandwidth of the asynchronous disk.

Sophisticated database systems use group-commit to improve the performance of their transactions. In group commit, transactions are commited in groups. All transactions in the group

Benchmark	NVRAM (ATM)	Group-Commit
debit-credit	96.4	92.8
order-entry	70.8	50.04

Table 2: Performance (in transactions per second) of NVRAM and group commit configurations for our two applications.

perform their (synchronous) disk write operations at the group commit time. Thus, the magnetic disk receives several write accesses which can be scheduled and merged effectively, reducing seek and rotational latency.

Table 2 shows the performance of our system vs. the performance of a group commit system that commits 100 transactions at a time (requiring a little more than 3 Mbytes of intermediate buffer space). We see that our approach results in 4%-40% performance improvement, mostly due to the fact that group commit has to make a synchronous disk write operation (every 100 transactions) while our system can proceed fully asynchronously. Even if group commit had the same performance as our system, it would still have two major disadvantages:

- Our approach can speed-up synchronous disk write operations that originate from *any* application running on the system, while group commit applies only to databases, and only to those that it has been explicitly programmed in.

- Group commit results in significant transaction latency, since the first transaction of the group cannot commit until the last transaction of the group starts, executes, and commits. Thus, the latency experienced by each transaction may easily reach up to several seconds. On the contrary, the latency experienced by each transaction in our system is in the order of a few milliseconds.

Figure 3 presents the performance of four system configurations when the simulated disk is a Quantum Viking II 4.55S (4.5 Gbyte disk). As previously, we see that our approach is significantly (3-4 times) faster than the synchronous disk. It achieves 5-40% higher throughput (and two orders of magnitude lower latency) than group-commit. Finally, our approach is within 2-3% of asynchronous disk.

Summarizing, our performance results suggest that our approach combines the performance of asynchronous disk operations (within 3%) with the reliability of synchronous disk accesses.

5 Conclusions

Traditional Non Volatile Memory (NVRAM) systems are constructed either from battery-backed SRAM boards, or from EEPROM (FLASH) memories. Both approaches make the cost of an NVRAM system prohibitively high for low-cost commodity workstations. In this paper we present a software-based NVRAM system, which uses redundancy to improve data reliability, and survive power failures and system crashes. In our approach, instead of writing the data to non-volatile memory, we write the data to (at least) two volatile memories, i.e. to the main memories of two workstations in a workstation cluster, that are connected to different power

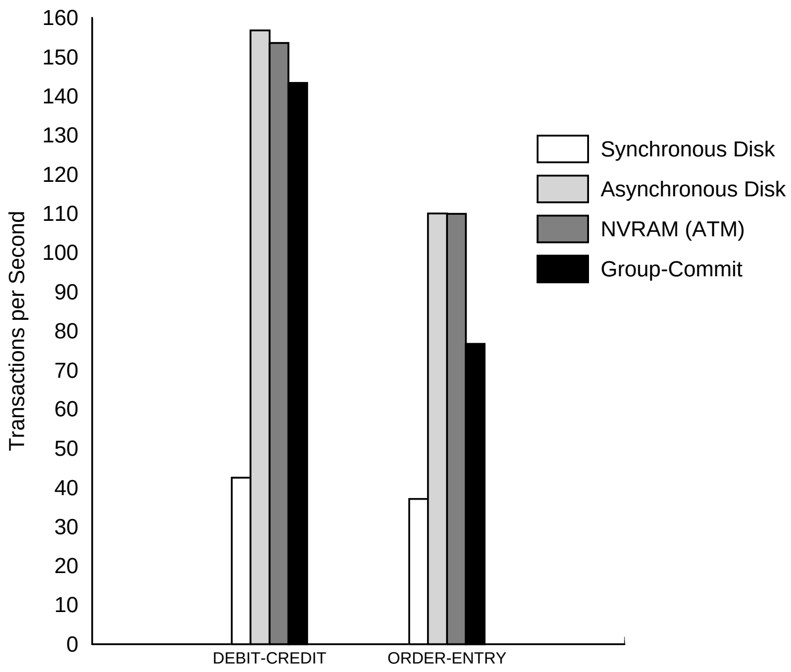

Figure 3: Performance (in transactions per second) of four system configurations. The disk simulated is a Quantum Viking II 4.55S.

supplies, or to a UPS. Our system will experience data loss only if *all* workstations that have a copy of the data fail, an event with extremely low probability.

We have implemented our system on a workstation cluster, evaluated its performance and compared it against the performance of disk-based systems. Based on our experience we conclude:

- *Network Memory provides an attractive alternative to hardware battery-backed NVRAM systems.* The main advantage of our system is its low cost. It uses existing (and otherwise unused) resources to create a non-volatile memory buffer.

- *Network Memory has (practically) the same performance as methods that provide no reliability.* Our results suggest that the performance of our system is the same as the that of a system doing asynchronous disk write operations.

- *The performance benefits of software-based NVRAM will increase with time.* Current architecture trends suggest that the gap between processor and disk speed continues to widen. Disks are not expected to provide the bandwidth and latency needed by synchronous write operations, unless a breakthrough in disk technology occurs. On the other hand, interconnection network bandwidth and latency keeps improving at a much higher rate than (single) disk bandwidth and latency, thereby increasing the performance benefits of using remote memory in software-based NVRAM systems. Meanwhile, the amount of memory in workstation increases, reducing in this way the cost of an NVRAM server.

Based on our experience in building the NVRAM system, and our performance measurements, we believe that software-based NVRAM is an inexpensive alternative to hardware

NVRAM systems, and can be used to speed up several system components, including file systems and databases.

Acknowledgments

This work was supported in part by PENED project "Exploitation of idle memory in a workstation cluster" (2041 2270/1-2-95) funded by the General Secretariat for Research and Technology of the Ministry of Development. We deeply appreciate this financial support. The disk simulator was made available by Greg Ganger, Bruce Worthington, and Yale Patt from `http://www.ece.cmu.edu/~ganger/disksim/`. We thank them all.

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] 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.

[3] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a Distributed File System. In *Proc. 13-th Symposium on Operating Systems Principles*, pages 198–212, October 1991.

[4] M. Bangalore and A. Sivasubramaniam. Remote Subpaging Across a Fast Network. In *Proc. of the Workshop on Communication, Architecture and Applications for Network-based Parallel Computing*, pages 74–87, 1998.

[5] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proc. 21-th International Symposium on Comp. Arch.*, pages 142–153, Chicago, IL, April 1994.

[6] Jeff Chase. A Network Virtual Store, 1995. Duke University. Talk given at the SOSP 95.

[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] P.M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[9] 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 *First Symposium on Operating System Design and Implementation*, pages 267–280, 1994.

[10] Fred Douglis, Ramon Caceres, Frans Kaashoek, Kai Li, Brian Marsh, and Joshua Tauber. Storage Alternatives for Mobile Computers. *First Symposium on Operating System Design and Implementation*, November 1994.

[11] 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.

[12] 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.

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

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

[15] S. Ioanidis, E.P. Markatos, and J. Sevaslidou. On using Network Memory to Improve the Performance of Transaction-Based Systems. In *International Conference on Parallel and Distributed Techniques and Applications*, 1998.

[16] 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.

[17] David E. Lowell and Peter M. Chen. Free Transactions With Rio Vista. In *Proc. 16-th Symposium on Operating Systems Principles*, pages 92–101, October 1997.

[18] 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.

[19] E.P. Markatos and M. Katevenis. Telegraphos: High-Performance Networking for Parallel Processing on Workstation Clusters. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture*, pages 144–153, February 1996.

[20] 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.

[21] Athanasios Papathanasiou and Evangelos P. Markatos. Lightweight Transactions on Networks of Workstations. In *Proc. 18-th Int. Conf. on Distr. Comp. Syst.*, pages 544–553, 1998.

[22] M. Satyanarayanan, 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.

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

[24] C.A. Thekkath, H.M. Levy, and E.D. Lazowska. Efficient Support for Multicomputing on ATM Networks. Technical Report 93-04-03, Department of Computer Science and Engineering, University of Washington, April 12 1992.

[25] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. *Proc. 15-th Symposium on Operating Systems Principles*, December 1995.

[26] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, 1996.

[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.