Lightweight Transactions on Networks of Workstations

Athanasios E. Papathanasiou Evangelos P. Markatos*

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, papathan@ics.forth.gr

http://www.ics.forth.gr/proj/avg/paging.html

Technical Report 209

September 1997

Abstract

Although transactions have been a valuable abstraction of atomicity, persistency, and recoverability, they have not been widely used in programming environments today, mostly because of their high overheads that have been driven by the low performance of magnetic disks. A major challenge in transaction-based systems is to remove the magnetic disk from the critical path of transaction management.

In this paper we present PERSEAS, a transaction library for main memory databases that decouples the performance of transactions from the magnetic disk speed. Our system is based on a layer of reliable main memory that provides fast and recoverable storage of data. We have implemented our system as a user-level library on top of the Windows NT operating system. Our experimental results suggest that PERSEAS achieves performance that is orders of magnitude better than traditional recoverable main memory systems.

1 Introduction

Transactions have been valued for their atomicity, persistency, and recoverability properties, which are useful to several systems, ranging from CAD environments, to file systems and

*The authors are also with the University of Crete.

databases. Unfortunately, adding transaction support to an existing data repository has been traditionally 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. A major challenge in transaction-based systems is to decouple the performance of transaction management from the magnetic disk speed.

In this paper we present PERSEAS,[1] a transaction library for main memory databases that decouples the performance of transactions from the magnetic disk speed. Our system is based on a layer of reliable main memory that provides fast and recoverable storage of data. This reliable memory layer is achieved by mirroring data into more than one main memories of (at least two) different PCs (or workstations), connected to different power supplies. Efficient data mirroring is achieved by copying data from the main memory of one PC to the main memory of another PC over a high-speed interconnection network.

On top of this reliable main memory layer PERSEAS builds an efficient transaction library. The existence of this reliable memory layer allows PERSEAS to implement fast transactions that do not need magnetic disks as a reliable storage medium. If a workstation crashes, all its main memory data can still be recovered, since they have been mirrored in the main memory of another workstation. Data can be completely lost only if all mirror workstations crash (during the same time interval). However, such an event (unless scheduled by the system administrators, in which case the database can gracefully shut down) is unlikely to happen. The most likely reasons that cause a workstation to crash involve (a) power outage, (b) hardware error, and (c) software error. Power outages are unlikely to lead to data loss, since mirror workstations are connected to different power supplies (e.g. UPS's), which are unlikely to malfunction concurrently. Software and hardware errors (in different PCs) usually occur independent from each other, and thus they can not lead to data loss. On the other hand, it is true that different

[1]Perseas was one of the famous heroes of the ancient Greek mythology. Among the several achievements he accomplished the most important one was the elimination of Medusa, a woman-like beast with snakes instead of hair and the ability to turn into stone everyone, who looked at her gaze. Perseas managed to outcome her by using his shield as a mirror. When Medusa tried to petrify him, her gaze fell upon her reflection on Perseas' shield, while Perseas got the chance to approach her and cut her head. In the same way Perseas killed Medusa, the PERSEAS transactional library uses mirroring to support reliable and atomic transactions while at the same time eliminating its own opponent, the overhead imposed to transactions due to synchronous accesses to stable storage (usually magnetic disks).

PCs may block (i.e. hang) together at the same time if they access a common crashed source (e.g. a crashed file server). Although such correlated disruptions in service may happen, they do not lead to workstation crashes and correspondingly to data loss, that is, they may affect the performance, but not the correctness of the mirroring mechanism. Thus, we believe that our approach leads to a level of reliable memory on top of which transactions can be efficiently implemented. The rest of the paper is structured as follows: Section 2 surveys previous work. Sections 3 and 4 present the design and implementation of our system. Section 5 presents our experimental results, and section 6 concludes the paper.

2 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 [2, 6, 16, 22] 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 [12, 18, 21, 24]. Even Distributed Shared Memory systems can exploit the remote main memory in a NOW [11, 7] for increased performance and reliability. For example, Feeley *et. al* describe a *log-based coherent* system that integrates coherency support with recoverability of persistent data [11]. Their objective is to allow several clients share a persistent storage through network accesses. Our approach is significantly simpler than [11], in that we do not provide recoverable support for shared-memory applications, but for traditional sequential applications. The simplicity in our approach leads to significant performance improvements. For example, [11] reports at most a factor of 9 improvement over unmodified traditional recoverable systems (i.e. RVM), while our performance results suggest that PERSEAS results in four orders of magnitude performance improvement compared to unmodified RVM.

Persistent storage systems provide a layer of virtual memory, (navigated though pointers), which may outlive the process that accesses the persistent store [28, 26]. We believe that our approach complements persistent stores in that it provides a high-speed front-end trsansaction library that can be used in conjuction with the persistent store.

The Harp file system uses replicated file servers to tolerate single server failures [22] as follows: each file server is equipped with a UPS to tolerate power failures, and speedup synchronous write operations. Although PERSEAS 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 our work is concerned mostly with user-level transaction-based systems that make lots of *small* read and write operations. In contrast, Harp is a kernel-based file system that sustains hardware and software failure.

The Rio file system changes the operating system to avoid destroying its main memory contents in case of a crash [5]. 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. Vista [23] is a recoverable memory library being implemented on top of Rio. Although Vista achieves impressive performance, it can provide recoverability only if run on top of Rio, which, by being a file system is not available in commercial operating systems. On the contrary, our approach provides performance comparable to Vista, while at the same time, it can be used on top of any operating system. In our current implementation, PERSEAS runs on top of the unmodified Windows NT operating system. In case of long crashes (e.g. due to hardware malfunction) data, although safe in Vista's cache, are not accessible, until the crashed machine is up and running again. In PERSEAS, even during long crashes, data are always available, since data exist in the main memories of (at least) two different workstations: if one of them crashes, the data can still be accessed through the other workstation.

Ioanidis *et al.* have proposed the use of remote memory to speed up synchronous write operations used in the Write Ahead Log (WAL) protocol [19]. 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. Although their approach is related to ours, there still exist significant differences. In case of heavy load, write buffers will become full and the asynchronous write operations of [19] will become synchronous, thereby delaying transaction completion. Moreover, the transaction commit performance of [19] is limited by disk throughput (all transactions write their data to disk even if they do so asynchronously). In PERSEAS, transaction performance is limited only by network performance, and not magnetic disk speed. Current architecture trends suggest that disk latency (throughput) improves 10% (20)% per year, while interconnection network latency (throughput) improves at the much higher rates of 20% (45)% per year [8]. Thus, approaches that get rid of magnetic disk accesses (like PERSEAS) provide increasingly better performance.

Network file systems like Sprite [29] and xfs [2, 9], 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 [13]. They validate their approach using simulation, and report very encouraging results. Griffioen *et. al* proposed the DERBY storage manager, that exploits remote memory and UPSs to reliably store a transaction's data [15]. They simulate the performance of their system and provide encouraging results.

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 [10]. 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 [13] and [10] improve the performance of read accesses (by providing large caches), while our approach improves the performance of write-dominated transaction-based systems.

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 [33], 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-

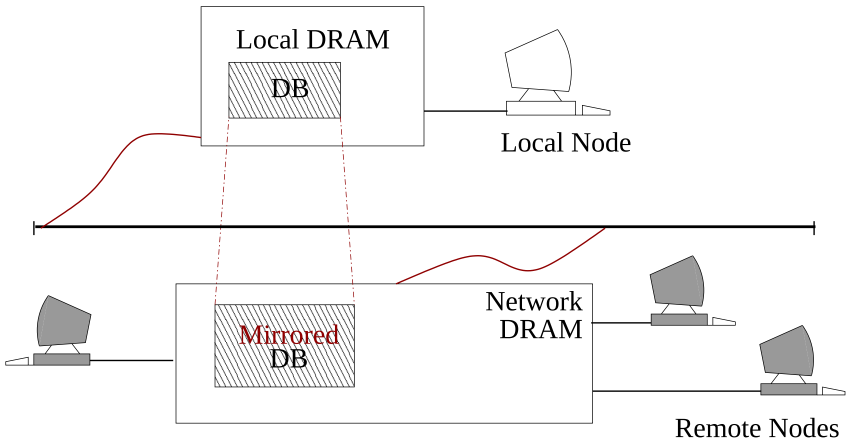

Figure 1: **Reliable Network RAM & Mirroring:** The unexploited memory of idle workstations is used to create a layer of reliable network RAM. Sensitive data, like those of a Main Memory Database System (MMDB), can be mirrored in remote memory to increase their reliability.

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.

Summarizing, PERSEAS is a user-level, easily portable transactional library, implemented on top of a user-level reliable main memory, which can result in good transaction performance. Previous approaches have been mostly based on providing fast recoverability by modifying operating system internals, an approach that significantly limits their widespread use.

3 Design

The work presented in this paper may be separated into two different layers: a level of reliable (or recoverable) network RAM and PERSEAS, a user-level transactional library.

In a network of workstations, significant portions of main memory remain idle for long periods of time. In this project, these segments of "free" physical memory are used as a form of reliable memory. Sensitive data are stored in the main memory of more than one workstations, (mirroring), and may be recovered in case of workstation failures (Figure 1).

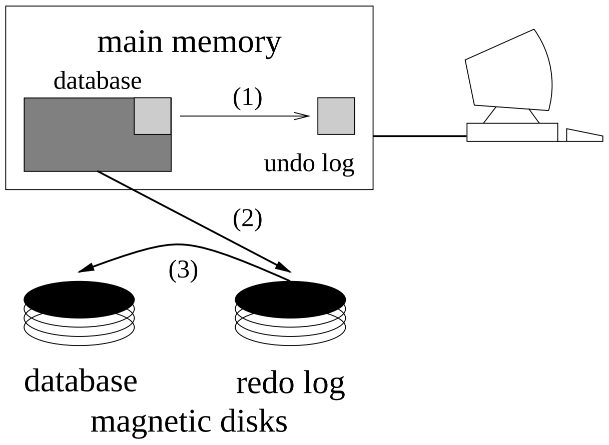

Figure 2: **The Write-ahead logging Protocol:** Three copies are necessary for an update operation. Firstly, the undo log is created with a memory copy operation. Secondly, the modified data propagate to the *redo log*. Finally, when several transactions have commited, data from the *redo log* propagate to the database in stable storage.

Transactional libraries provide the characteristics of atomicity and persistence to transactions, and can be used to support database systems, persistent languages and file systems. To implement reliable transactions, most database systems use a (write ahead) log file, which must reside in stable storage (usually magnetic disks). Accesses to the log file are usually in the critical path of the transaction processing and need synchronous input/output. Some well-known examples of systems that use the Write-Ahead Logging Protocols are RVM [31] and ARIES [27].

As shown in Figure 2, the Write-Ahead Logging Protocol involves three copy operations. When a transaction begins the original data of the portion of the database to be updated are copied temporarily to a memory region called *undo log*. The *undo log* is used to undo quickly any modifications in the database in case the transaction aborts. When the transaction has updated the database, the modifications propagate to a file, which resides in stable storage, the *write-ahead log file* or *redo file*. At this point the transaction commits and the space occupied by the *undo log* is freed. When several transactions have commited, the updates that have been logged in the *redo file* are copied to the original database and space from the redo log is reclaimed.

PERSEAS eliminates the redo log file, used in the Write-Ahead Logging Protocol, as well as synchronous disk accesses by using network memory as reliable memory. A reliable network memory layer may be developped over a high-throughput, low-latency network interface, like Myrinet [4], U-net [32], Memory Channel [14], SCI [17], and ATM. Some Network interfaces have transparent hardware support for mirroring, which makes PERSEAS easier to implement. Such systems include PRAM [30], Telegraphos [25], and SHRIMP [1].

Three major operations are necessary to make this possible:

- **remote malloc:** The remote malloc operation can be used to map physical memory from a remote node to the calling process's virtual address space.
- **remote free:** Remote free is used to free an occupied remote main memory segment.
- **remote memory copy:** The remote memory copy operation is similar to a memcpy or bcopy operation with the exception that it copies data between local and remote main memory segments.

The operations described above create the layer of reliable network RAM, which is used by PERSEAS to support atomic and recoverable transactions without the need for a *redo log file.*

PERSEAS offers a simple interface through which applications can make persistent stores and atomic updates. PERSEAS' interface consists of the following procedures:

- **PERSEAS_init**
- **PERSEAS_malloc**
- **PERSEAS_init_remote_db**
- **PERSEAS_begin_transaction**
- **PERSEAS_set_range**
- **PERSEAS_commit_transaction**
- **PERSEAS_abort_transaction**

After calling *PERSEAS_init*, which initializes the PERSEAS transactional library, the application can call *PERSEAS_malloc* in order to get local memory space for the database records. In addition to the above, *PERSEAS_malloc* prepares the remote memory segments, in which the database records will be mirrored. As soon as the local records have been set to their initial

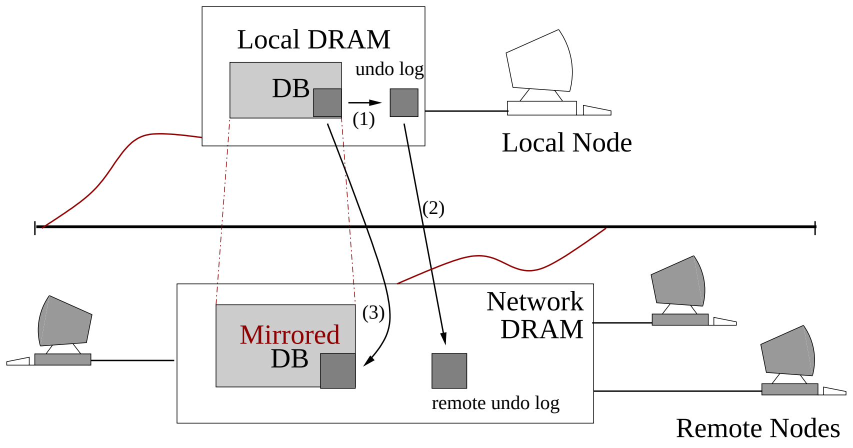

Figure 3: **Atomic & Reliable Transactions with PERSEAS :** Only three **memory copies** are necessary for a transaction. Firstly, the before image of the database is copied in the *undo log* in local memory (Step 1). Data in the *local undo log* propagate to the remote undo log with a remote write operation (Step 2). Finally, the updated portion of the local database is copied to the equivalent portion in the remote database (Step 3). All accesses to stable storage, like magnetic disks, have been eliminated.

values, the application has to call *PERSEAS_init_remote_db* to initialize the remote database segments. At this point the database has been completely mirrored to network DRAM.

Applications start a transaction by calling *PERSEAS_begin_transaction*. Before making any updates, the applicaton should notify the transactional library of the portion (or portions) of the database that is going to be updated. This is done through a call (or more) to *PERSEAS_set_range*. This call has as a result the logging of the segment's original image to an *undo log*. To reassure the correct recovery of the database in case of a system crash the *undo log* is also copied to the remote node's memory. The *local undo log* is used to undo quickly any modifications to the database records in case the transaction aborts. The *remote undo log* might be necessary during recovery, if some modifications of the database propagated to the remote node before the local system's failure. After this step, the application can update any portions of the database, for which *PERSEAS_set_range* has been called (Figure 3).

After the completion of the update operation, the modified portions of the database have to be copied to the equivalent portions in the memory of the remote nodes. This is done through a call to *PERSEAS_commit_transaction*. With this call the *undo logs* are discarded and the transaction commits. In case the transaction aborts, the application may use

PERSEAS_abort_transaction to undo any modifications to the database. This function performs just a local memory copy operation.

In case of failure of the primary (local) node, PERSEAS can use the data found in the network memory to recover the database. If the modified data had started propagating to the remote (secondary) node before the local system's failure, then the original data, which can be found in the remote undo logs are copied back to the remote database, in order to discard any illegal updates. With this memory copy operation the remote database is brought in a legal state and can be used to recover the local database. In any other case, the remote database segments are legal and the local database is recovered with just one (for each database record) remote-to-local memory copy operation.

Another important advantage of PERSEAS is the availability of data. Data in network memory are always available and accessible by every node. In any case of single node failures, the database may be reconstructed quickly in any workstation of the network and normal operation of the database system can be restarted immediately.

4 Implementation

Our current version of PERSEAS is implemented on top of two PCs with 133MHz Pentium processors and 96MB of main memory, running Windows NT 4.0. The network interface used is a PCI-SCI (Scalable Coherent Interface) Cluster Adapter Card manufactured by Dolphin and configured in ring topology. The PCI-SCI card is a high-throughput, low-latency network interface, which can support remote write and remote read operations.

Write operations to contigious remote memory addresses through the PCI-SCI network interface can give throughput similar to the local memory subsystem. The application end-to-end one-way latency for one 4-Byte remote store is 2.6 microseconds. To achieve this kind of performance, the PCI-SCI card contains 16 internal 64-byte buffers. Half of them are used for write operations, while the other half is used by read operations. The PCI-SCI card devides the physical memory of a node into 64-Byte chunks. Every 64-byte memory region is aligned on a 64-byte boundary. Each chunk is mapped to a 64-Byte buffer in the PCI-SCI chip. The six least-significant bits of the address define its offset in the buffer, while bits 6-8 identify the buffer which relates to the specific address (Figure 4). Stores to contigious memory addresses are

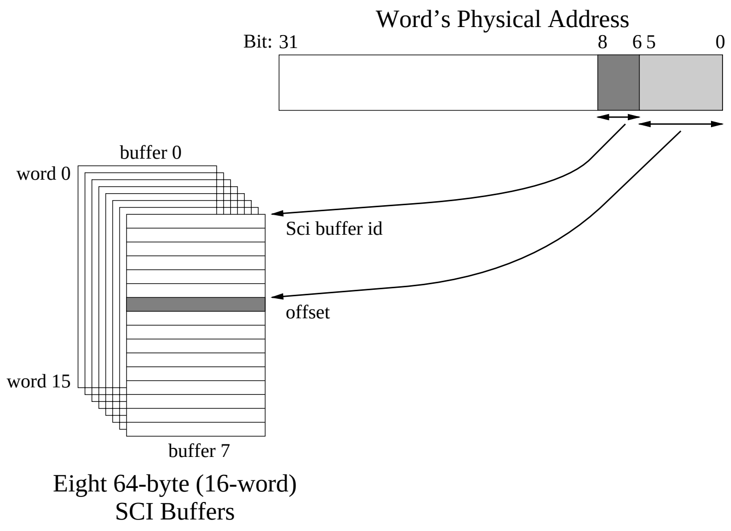

Figure 4: **SCI Internal Buffers & Physical Address Mapping:** The six least significant bits of a word’s physical address define the word’s offset in an SCI buffer, while bits 6 through 8 specify to which of the eight internal buffer the word belongs.

gathered in the buffers (*store gathering*), and each address stream (buffer) can be transmitted or gather data independently of each other (*buffer streaming*). In this way, the overhead of sending an SCI packet through the network is amortized over many store operations. Full SCI buffers are flushed as whole 64-byte SCI packets, while half-filled buffers are transmitted as a set of 16-Byte packets. In addition to the above, store operations which involve the last word of a buffer give better latency results, because of the way the buffers of the PCI-SCI chip are flushed.

Several experiments conducted with the PCI-SCI network interface have shown us that for memory copy operations of 32 bytes or more, it is better to copy 64-byte memory regions aligned on 64-byte boundary (Figure 5). In this way, the PCI-SCI interface transmits whole 64-byte packets, while the *store gathering* and *buffer streaming* techniques work more effectively. For data sizes of 16 byte or less the store operation is performed as is. The PCI-SCI card sends one or two (if the address range crosses the 16-byte alignment boundary) 16-byte packets. This results to an end-to-end one way latency of 2.7-3.0 or 5.3 microseconds respectively. Memory copy operations of 17-32 bytes may be performed as a 64-byte copy operation of a 64-byte aligned memory region, if none of the word addresses maps to the last word of a buffer, or as a 17-32 byte copy operation, if the sixteenth word of a buffer is written. The second case

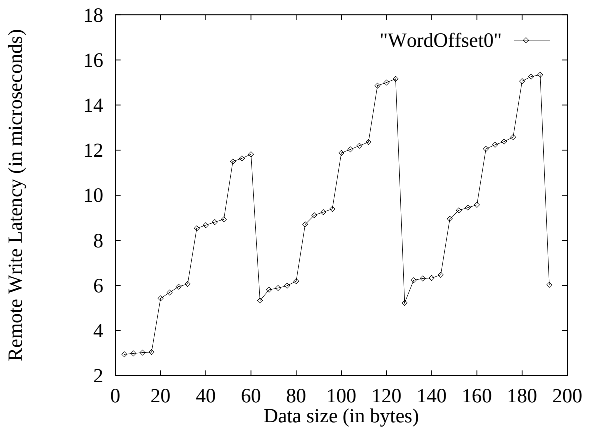

Figure 5: **SCI Remote Write Latency:** This graph shows the application's end-to-end one-way latency for SCI remote write operations for data sizes between 4 and 196 bytes. The first word of every write operation maps to the first word of an SCI buffer. Stores of whole 64-byte regions (aligned on 64-byte boundary) have the lowest latency for all data sizes greater than 32 bytes. Remote stores of 64 and 196 bytes need 5.3 and 6.0 microseconds respectively.

may result in the creation of two 16-byte SCI-packets (if the address range crosses the 64-byte alignment boundary), but the overhead for the creation of the second packet overlaps with that of the first one due to *buffer streaming*.

As mentioned in the previous section, the reliable network memory layer can be used through three major operations: *remote malloc*, *remote free* and *remote memory copy*. To implement these on top of PCI-SCI a client-server model is used. The server process, runs in the remote node and is responsible for accepting requests (remote malloc and free) and manipulating its main memory (exporting physical memory segments and freeing them when necessary). The client process sends requests to the server process and in the case of malloc requests blocks until the request is serviced. As fas as the *remote memory copy* operation is concerned, the memcpy function can be used, since remote memory is mapped to the virtual address space of the client process. However, in our current implementation of PERSEAS, we have used a more complicated *sci_memcpy* function with several optimizations, that take advantage of the PCI-SCI card's behavior, described in the previous paragraph.

The PERSEAS communicates with the reliable network memory layer through the following basic functions:

- **sci_get_new_segment**

- **sci_free_segment**
- **sci_memcpy**
- **sci_connect_segment**

The *sci_get_new_segment*, *sci_free_segment* and *sci_memcpy* functions implement the *remote malloc*, *free* and *memory copy* operations, while *sci_connect_segment* is used by PERSEAS to connect to already created segments. The last function is necessary after a system crash, in which case the remote node has already exported the requested segments, but the local node has lost the pointers, through which it can access them. Specifically, *sci_connect_segment* maps to the calling application's virtual address space the requested remote memory segment.

PERSEAS_malloc calls *sci_malloc* to get memory space from a remote node. In this way, for every database segment created in the local memory an equivalent segment is created in remote memory. When the local database records have been initialized, the application calls *PERSEAS_init_remote_db* to copy them to the remote node. This call results to an *sci_malloc* call. At this point, the whole database has been mirrored to the remote node, and the application can start executing transactions. The other basic PERSEAS functions (*PERSEAS_begin_transaction*, *PERSEAS_commit_transaction*, *PERSEAS_set_range* and *PERSEAS_abort_transaction*) perform only local memory copy operations and remote write operations, using the *sci_memcpy* or simple store commands, according to the occasion.

During recovery the primary node has to reconnect to the portions of memory where PERSEAS metadata are kept, as well as to the remote database segments. Since the remote segments already exist, the *sci_malloc* function cannot be used. Instead, *sci_connect_segment* is called every time a remote segment has to be remapped to the local virtual address space. Firstly, the segments containing the PERSEAS metadata are reconnected. From these, the information, that is necessary to find and reconnect to the remote database records, is retrieved. After this point all the information about the database status becomes available and recovery proceeds as described in the previous section.

5 Experimental Results

To evaluate the performance of our system and compare it to previous systems, we conducted a series of performance measurements using a number of benchmarks, including TPC-B and

TPC-C. We draw on the benchmarks used by Lowell and Chen [23] to measure the performance of RVM [31], and Vista [23]. Actually, we use the exact code distributed from `http://www.eecs.umich.edu/Rio/`. The benchmarks used include:

- *Synthetic*: a benchmark that measures the transaction overhead as a function of the transaction size (i.e. the size of the data that the transaction modifies). Each transaction modifies a random location of the database. We vary the amount of data changed by each transaction from 4 bytes to 1 Mbyte.
- *debit-credit:* a processes banking transactions very similar to the TPC-B.
- *order-entry:* a benchmark that follows TPC-C and models the activities of a wholesale supplier.

All our experiments were run on two PCs connected with an SCI interconnection network [20]. Each PC was equipped with a 133 MHz processor.

5.1 Performance Results

Figure 6 plots the transaction latency as a function of the transaction size. We see that for very small transactions, the latency that PERSEAS imposes is less than 14 μs, which implies that our system is able to complete more than 70,000 (short synthetic) transactions per second. Previously reported results for main memory databases [31], suggest that the original implementation of the RVM main memory database can sustain at most 50 (short) transactions per second - a 3-orders of magnitude performance difference. When RVM is implemented on top of the Rio file cache it can sustain about 1,000 (short) transactions per second [23]. Comparing to Rio-RVM, our implementation achieves two orders of magnitude better performance. The Vista main memory database, which is the fastest main memory database known to us, is able to achieve very low latency for small transactions (in the area of 5 μs) [23].

Table 1 shows the performance of PERSEAS when running the debit-credit and order-entry benchmarks. We have used various-sized databases, and in all cases the performance of PERSEAS was almost constant, as long as the database was smaller than the main memory size.

We see that PERSEAS manages to execute more than 23,000 transactions per second for debit-credit. Published performance results report that RVM barely achieves 100 transac-

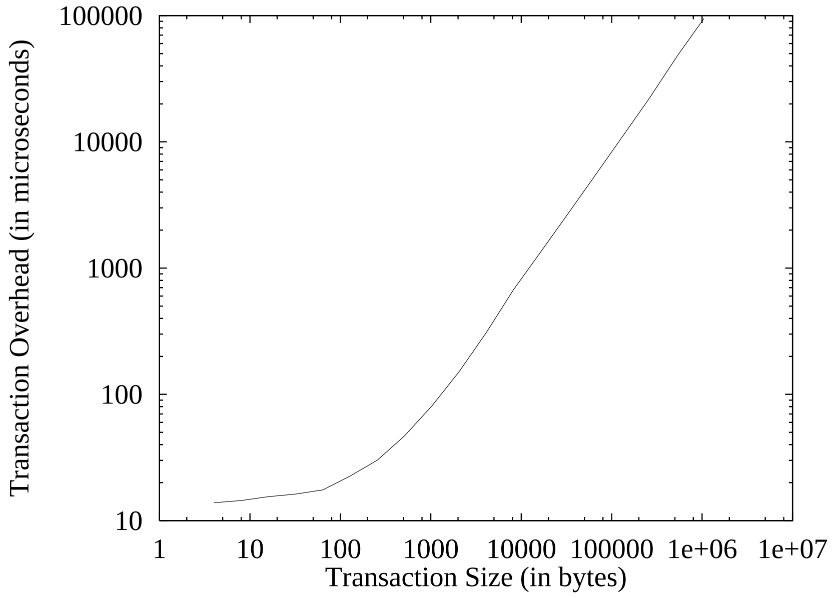

Figure 6: **Transaction Overhead of PERSEAS :** Very small transactions can be completed in as little as 14 microseconds, resulting in a throughput of more than 70,000 transactions per second. Even large transactions (1 MByte) can be completed in less than a tenth of a second.

Benchmark	Transactions per second
debit-credit	23,223
order-entry	8,432

Table 1: Performance of PERSEAS for benchmarks debit-credit and order-entry.

tions per second, RVM-Rio achieves little more than 1,000 transactions per second, and Vista achieves close to 50,000 transactions per second. For order-entry, PERSEAS manages to execute about 8,500 transactions per second. Previously reported performance results suggest that RVM achieves less than 90 transactions per second, and Vista achieves a bit more than 10,000 transactions per second.

Summarizing, we see that PERSEAS clearly outperforms traditional recoverable virtual memory systems by several orders of magnitude. Moreover, PERSEAS performs very close to Vista (which is the most efficient sequential recoverable main memory system today), while it retains its independence from operating system internals, while the performance of Vista depends on extensive operating system kernel modifications, as manifested by the Rio file cache. We believe that PERSEAS composes the best features from RVM and Vista: the great performance of Vista, with the operating system independence of RVM.

6 Conclusions

In this paper we describe how to construct a layer of fast and reliable main memory in a Network of Workstations, and how to build a fast transaction library on top of it. We implemented our approach as a user-level library on top of a cluster of personal computers running Windows NT.

Based on our experiences and performance results we conclude:

- *PERSEAS decouples transaction performance from magnetic disk speed.* The performance of traditional transaction systems is tied to disk latency, which is rather large and improves slowly with time. Sophisticated optimization methods (like group commit, stripping, etc) improve performance by decoupling transaction performance from disk latency and couple it to disk throughput. On the other hand, PERSEAS decouples transaction performance from magnetic disk speed, because it does not use disks as a short-term reliable storage medium. Instead, it uses redundant power supplies, mirroring and fast main memories to provide a layer of recoverable memory, which in turn can be used to implement transaction libraries.

- *PERSEAS results in significant performance improvements.* Compared to traditional RVM, PERSEAS results in 3 orders of magnitude of performance improvement. PERSEAS outperforms even sophisticated optimization methods (like group commit) by an order of magnitude.

- *PERSEAS provides efficient and simple recovery.* Mirrored data are accessible from any node in the network. Thus, in case of any kind of failure in the primary node, the recovery procedure can be started right-away in any available (working) workstation allowing immediate recovery of the database, even if the crashed node remains out-of-order for a long time.

- *The performance benefits of our approach will increase with time.* PERSEAS gains its performance improvements by substituting disk accesses with remote memory accesses (over a fast interconnection network). According to current architecture trends, magnetic disk speed improves 10-20% per year, while interconnection network speed improves much faster, at a rate of 20-45% per year [8]. Thus, the performance gains of our approach will probably increase with time.

Acknowledgments

This work was supported in part by PENED project "Exploitation of idle memory in a workstation cluster" (2041 2270/1-2-95), 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 Dolphin Inc., for giving us access to the PCI-SCI network interfaces, where the described experiments were run.

References

[1] R. D. Alpert, A. Bilas, M. A. Blumrich, D. W. Clark, S. Damianakis, C. Dubnicki, W. Felten E, L. Iftode, and K. Li. Early Experience with Message-Passing on the SHRIMP Multicomputer. In *Proc. 23-th International Symposium on Comp. Arch.*, Philadelphia, PA, May 1996.

[2] 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.

[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] N.J. Boden, D. Cohen, and W.-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29, February 1995.

[5] 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.

[6] T. Cortes, S. Girona, and J. Labarta. PACA: A Cooperative File System Cache for Parallel Machines. In *2nd International Euro-Par Conference (Euro-Par'96)*, pages 477–486, 1996. Lecture Notes in Computer Science 1123.

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

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

[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] 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.

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

[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] 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.

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

[15] 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.

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

[17] Dolphin ICS. PCI-SCI Cluster Adapter Specification. ARCHES project Working Paper No. 12.

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

[19] S. Ioanidis, E.P. Markatos, and J. Sevaslidou. On using Network Memory to Improve the Performance of Transaction-Based Systems. Technical Report 190, Institute of Computer Science - FO.R.T.H., March 1997. Available from ftp://ftp.ics.forth.gr/tech-reports.

[20] D. V. James, A. T. Laundrie, S. Gjessing, and G. S. Sohi. Scalable Coherent Interface. *IEEE Computer*, 23(6):74–77, June 1990.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] M.J.Carey, D.J. Dewitt, and M.J. Franklin. Shoring up persistent applications. In *Proc. of the 1994 ACM SIGMOD Conf. on Management of Data*, 1994.

[27] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. *ACM Transactions on Database Systems*, 17(1):94–162, 1992.

[28] J.E.B. Moss. Design of the Meneme Peristent Storage. *ACM Transactions on Office Information Systems*, 8(2):103–139, 1990.

[29] 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.

[30] Dimitrios N. Serpanos. *Scalable Shared Memory Interconnections (Thesis).* Technical Report CS-TR-227-90, Department of Computer Science, Princeton University, October 1990.

[31] 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.

[32] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proc. 15-th Symposium on Operating Systems Principles*, pages 40–53, December 1995.

[33] 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.