Architecture of the Oasis Mobile Shared Virtual Memory System

William H. Schroeder
University of California
Riverside, CA 92521-0304
Telephone: (909)-790-2760
Fax: (909)-797-3100
Email: whs@cs.ucr.edu

Brett D. Fleisch*
University of California
Riverside, CA 92521-0304
Telephone: (909)-787-7206
Fax: (909)-797-4643
Email: brett@cs.ucr.edu

UCR-CS-96-4
April 25, 1996

Abstract

Oasis is a memory-based storage system designed to support collaborative mobile computing applications. These applications require peer-to-peer and client-server interactions in conditions of less than ideal network connectivity. The architecture is based on a memory model where the address space is logically partitioned into regions used for sharing. Mobile units may disconnect from the backbone, cache portions of the regions, operate, and later reconnect. The backbone is kept strongly consistent, whereas disconnected units operate off local cached copies that have the potential of becoming inconsistent with the backbone. A *time-based coherency approach* is used for voluntary disconnection that limits inconsistency between disconnected units and the backbone. For inconsistencies that do arise, mechanisms are provided to detect and resolve update conflicts. Oasis also addresses issues of autonomous operation, reliability, and locking.

* Supported by a grant from IBM Corporation, the UC Micro program and Computer Marketplace, Inc.

Contents

1 Introduction

The Oasis system supports collaborative applications that operate with mobile laptop computers and personal digital assistants (PDAs). PDAs add a new dimension to previous tightly coupled distributed shared memory (DSM) research, not only because of their ability to move easily, but also because of their limitations as computational and interactive devices[1, 2]. For example, the Apple Newton has a small amount of RAM, a small screen, one PCMCIA slot and no hard disk[3]. Despite these limitations, the value of these devices is their ability to interact with external devices and external services, to perform real-time data collection, their support for wireless communication, and their use in global positioning[4]. A PDA can increase sharing effectiveness and enable collaboration.

Oasis collaborative applications require peer-to-peer and client-server interactions in conditions of less than ideal network connectivity. Oasis has a highly consistent backbone that typically executes either tightly or loosely coupled distributed applications. Mobile units support loosely coupled shared memory applications that may disconnect from the backbone, cache areas of the memory, operate, and later reconnect. The backbone is kept strongly consistent, whereas disconnected units operate off local cached copies that have the potential of becoming inconsistent with the backbone. A *time-based coherency approach* is used which uses a mechanism called a *lease*. During the lease period, no inconsistencies can arise between disconnected units and the backbone.

One possible method for supporting collaboration of small mobile devices is to use DSM algorithms. DSM approaches have been explored for the past several years. These systems have traditionally involved loosely coupled computers that run scientific computations. These applications have a compelling need to share intermediate results that are made during the computation. While scientific computation has been the main application area for DSM, few extensions beyond this realm have been pursued. Nonetheless, there are a great many applications that need to share data from a common database or that can benefit from loosely coupled sharing. We begin by describing these applications to give the reader an understanding of the types of applications such a system would support.

1.1 Oasis Collaborative Computing Applications

We have implemented a representative set of five small demonstration programs using POSIX DSM primitives as examples of collaborative distributed computing tasks. The development of these applications was done for proof-of-concept of the architectural feasibility of Oasis. These applications are best characterized as having loosely coupled sharing patterns. The Oasis Suite consists of: a Hospital Management System, Battlefield Tracking Application, Grocery Shopping Assistant, Distributed Appointment Scheduler, and a Distributed Grading System.

In the Hospital Management system, several health care workers share patient records from a database. These users must have instantaneous access to patient information as they move around in a hospital. The users may update portions of patient records based on their job function and level of authority. Updates are permitted at any time from any location. The program has a mode that allows orderly shutdown and disconnection from the backbone. Disconnected operation is supported by caching patient records when disconnection is requested. Later, when the user is reconnected to the backbone database, missing updates are added and conflicting data is reconciled.

The Battlefield Tracking Application displays positions of moving armies. Peer-to-peer cooperation between mobile units is employed to extend the range of data gathered. For example, one army may be out of range of the backbone, yet it may be able to relay messages to and from the backbone through another nearby army. This form of indirect communication allows a larger number of armies to be monitored. This application could also be used in civilian law enforcement for cooperating police officers that need to track suspects.[1]

The Grocery Shopping Assistant assists shoppers in making shopping, spending, and routing decisions associated with grocery shopping. Specifically, the assistant guides the preparation of a shopping list, the decision of which items to purchase, and the

[1] PCMCIA GPS cards could be used in this application.

design of a travel route through the grocery store that is a shortest path to all of the groceries selected. The program implements a shortest-path-algorithm to guide the shopper to the correct aisle for the groceries requested and implements a bin-packing algorithm to help the user select items within a specified budget. The program uses caching to operate disconnected from the backbone during shopping.

The Distributed Appointment Scheduler allows users to arrange appointments while connected or disconnected from an appointment database on the backbone. A client-server relationship with a central database is employed. It allows each individual to disconnect from the central server, cache the user's recent schedule, and while disconnected create peer-to-peer appointments. Later, when the server and client are reconnected, conflicting appointments are identified and sent back to the user for reconciliation.

The Distributed Grading system provides a user the option to create, delete, and update student records regardless of server connectivity. Disconnected operation is supported by caching student records when disconnection is requested. Later, when the user is reconnected to the backbone database, missing updates are added and conflicting data is reconciled.

2 Background

Distributed systems software remains difficult and expensive to develop[5]. Failures in the environment make correctness and consistency difficult to ensure in distributed programs. Costs are higher because software development is more complex when a richer set of errors arise. Our goal is to increase the effectiveness of software development for *collaborative* mobile applications, which combine distributed computing, mobility, and variable connectivity. These applications involve significant sharing between users or user tasks.

To reduce the costs of developing collaborative mobile applications, we wish to support a programming paradigm that makes writing distributed applications at least as easy as writing shared memory applications for parallel computers[2]. Existing programming paradigms can provide some assistance, such as RPC[6, 7], but they are not entirely satisfactory. RPC is not suitable for sharing large amounts of data between systems, which is becoming increasingly important for collaborative work. In contrast, the message passing approach[8], which has been widely used to communicate data between loosely coupled systems, in addition to RPC, cannot pass data by reference. Furthermore, the message passing approach requires that explicit packetization and communication primitives be included in distributed programs. These additions complicate the programmer's task. Message passing programs lack *referential transparency*. Referential transparency means that an application can be used on a different underlying system, which extends the application to execute in a more general environment or manner, without changing the application code. For example, existing applications that adhere to the POSIX interface could operate unchanged in a distributed system and benefit as a distributed application.

The model of distributed computation that we have been investigating is called *Distributed Shared Memory* (DSM)[9, 10, 11, 12, 13, 14, 15, 16]. In this model, a collection of sites (typically workstations of some kind) in a loosely coupled computer system are connected to a local area network. The sites have access to a paged shared virtual address space from which the sites can share memory; the memory appears as one large, smoothly addressable space that can be accessed from any site transparently. DSM is an attractive paradigm for a network of workstations. The user need not be concerned with physical location when accessing memory, and programs written for tightly coupled shared memory machines need not be rewritten to operate properly for a cluster of workstations. DSM applications are cooperating applications that require synchronization and consistent memory. These applications have a real-time requirement that they share peer-to-peer immediately and consist of programs that use memory as part of an address space that has need for close sharing. Birman calls these applications *tightly coupled distributed applications*.

[2]These programs impose intrinsic difficulties in and of themselves. However, shared memory parallel machines have been around for over a decade and provide a good reference point from which a programmer can begin to program a parallel distributed application in a network of workstations.

We contrast DSM systems with Shared Virtual Memory (SVM) systems. Whereas DSM systems feature a high degree of tightly coupled sharing, SVM systems typically have looser coupling between the programs sharing the memory. These programs use shared memory as a cache for persistent store objects (e.g. files) or contain memory that has very few hot spots. These applications have little write sharing[17, 18] and are best described as *loosely coupled distributed applications*[19]. While Oasis supports a backbone that executes DSM programs, programs that operate on the disconnected PDAs are best characterized as SVM applications. DSM protocols and algorithms are used to support both types of applications in the design.

3 Design

Communication over wireless networks has a fixed service area. Operation around the boundaries of the service area or outside the service area can cause partial, intermittent, or complete communication failures. These failures will cause numerous problems such as the inability of backbone sites to acquire critical resources that are owned at the unreachable mobile sites. Likewise, the unreachable mobile sites will also be unable to acquire resources from the backbone.

Designing and implementing distributed applications can prove to be far more complex than applications designed for a uniprocessor. Aside from these challenges, designing a distributed system to support mobility has its own set of complexities that involve operating partitioned or operating independent of the supporting system. Oasis was designed to meet these challenges.

Oasis has three fundamental design goals. First, Oasis is designed to explore adapting a DSM paradigm for a system of collaborating mobile applications using wireless communication. A number of workstation implementations of DSM were examined as a possible starting point, which include Mirage[14], Treadmarks[20] and Quarks[21]. We rejected the Mirage approach since it requires operating system modifications to support coherency. This limits portability of the enhancements because the operating system may be machine specific. Treadmarks was not selected because it is based on a lazy release consistency, and the coherency policy used for Oasis needs to be sequentially consistent in order to emulate POSIX behavior for the many applications we plan to use[3]. A commercial fee is also required for Treadmarks which limits the redistribution of source code enhancements. Quarks was selected because of its portability to multiple UNIX[4] platforms and because it is a self contained application support library[22]. Quarks also provides flexible support for a wide variety of potentially useful coherency policies. In addition, Quarks is a free public domain DSM system.

The second design goal for Oasis is to support *continual operability* of distributed mobile applications. By continual operability we mean that applications need not be terminated when they move outside the known range of network connectivity. To achieve continual operability, mobile computers must be able to operate detached from the backbone system. Disconnection of mobile computers in a distributed system can occur in three different methods: voluntary, involuntary, and intermittent. Oasis addresses voluntary disconnection by reorganizing shared resources to support extended disconnected system operation. Oasis also handles intermittent disconnected operation by providing communication retries. Retries are timed events that may be easily adjusted to handle variations in the latency of communication. When a specified amount of retries occur and the remote machine fails to respond, Oasis assumes the mobile system has failed. A failed mobile machine is treated as an involutary disconnection and can be handled with recovery protocols[23].

The third design goal of Oasis is to provide referential transparency for programs that use POSIX shared memory and POSIX semaphores. Oasis takes existing applications that adhere to the POSIX interface and execute these programs unchanged in a distributed system, whereas earlier, they may have been only functional on an uniprocessor. With the three design goals of Oasis defined, the remaining sections will discuss the system and the design of mechanisms to support continual operability.

[3] We cannot guarantee applications will be rewritten to use the Treadmarks synchronization primitives.

[4] UNIX was developed an licensed by AT&T. UNIX is a register trademark of AT&T.

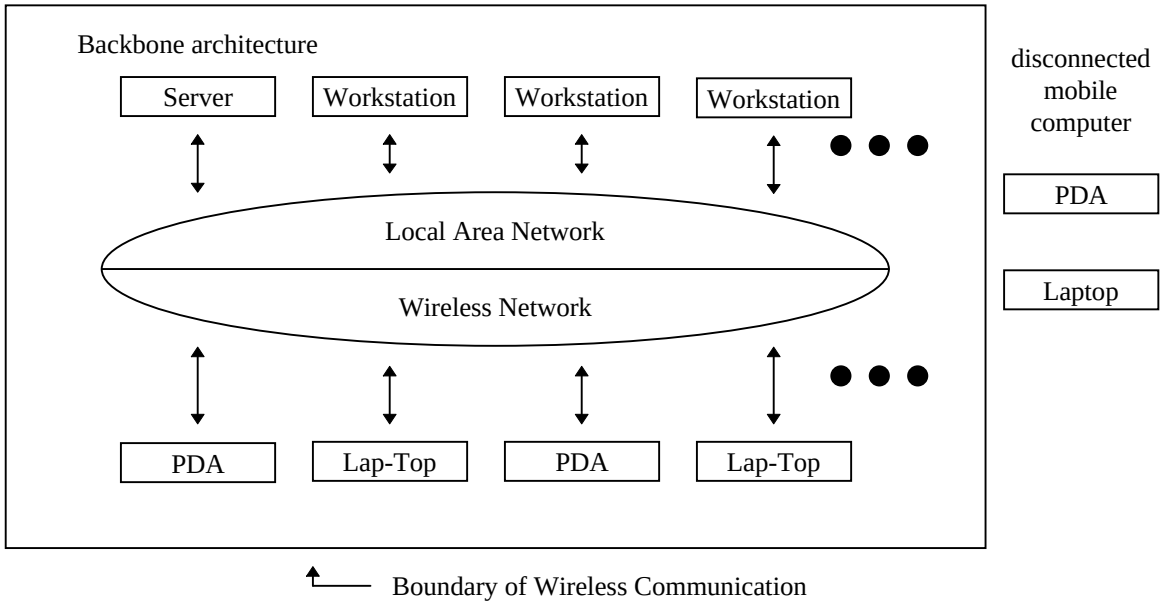

Figure 1) The system configuration of an Oasis distributed system.

3.1 System Architecture

The *backbone* is compromised of a collection of fixed sites and mobile computers connected by using a mixture of fixed site LANs and wireless networking. The backbone system is a network of computers that can range from large mainframe computer systems to smaller systems such as lap-tops and PDAs. The backbone is made up of a *server site* and multiple *client sites*. The server site is a fixed position workstation that maintains the system configuration. The client sites are either fixed position workstations or mobile lap-tops or PDAs. The client sites, when attached to the backbone system, provide protocols to maintain a strongly consistent model for shared memory regions and shared locking mechanisms. The mobile computers (client sites) can disconnect and reconnect at will from the backbone. Figure 1 shows the system configuration of Oasis.

The Oasis server handles the backbone configuration, and manages shared memory regions and locking. Managing the shared resources involves keeping track of which sites are attached to a shared resource, as well as determining when shared resources are no longer referenced and thus can be de-allocated. Oasis uses a client-server relationship for the allocation and de-allocation of shared regions and shared locks, as well as the mobile computers disconnection-reconnection algorithms. These operations occur infrequently and are the only time the server is contacted. In all other cases, a peer-to-peer relationship exists between client sites.

3.2 Site Architecture

Client sites execute application programs linked with an Oasis library. The Oasis library equips an application for mobility. A site is comprised of the components illustrated in figure 2. These components, except for the application component, are provided in the application library. The subsystems that form an Oasis distributed application consist of the following: the client application, POSIX translation subsystem, DSM subsystem, message subsystem, and the reconnection-disconnection subsystem.

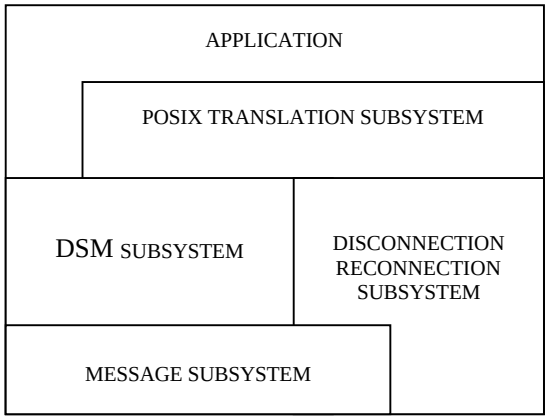

Figure 2) The application library architecture

3.2.1 Posix Translation Subsystem

A goal of Oasis is to provide a system level library that will transform a program that uses POSIX shared memory and semaphores into a distributed application. This transformation is accomplished by adding a header file into the original source code and linking a POSIX-to-Oasis translation library into the executable program to convert uniprocessor shared memory applications into multiple processor DSM applications. This method provides a referentially transparent approach to converting existing programs to Oasis.

3.2.2 DSM Subsystem

The Oasis DSM subsystem manages the shared resources that an application programmer uses. The management of shared memory regions, shared locks, and memory coherency protocol is accomplished by a thread of execution called the *DSM thread*[5] . The DSM thread requests and releases shared resources to and from distributed sites using the messaging subsystem. There are three important components in the DSM subsystem: shared memory regions, memory coherence protocol, and shared locks.

The Oasis DSM system is a shared memory abstraction built on a message passing distributed-memory system. Shared memory regions are allocated and de-allocated by the application programs. The server creates and maintains information about each region which includes the size of the region, and the sites that have attached to it. A shared memory region is composed of a collection of pages, and throughout this paper the term *page* is referenced as the smallest unit of shared memory between client sites. Page information is maintained in a structure called a *page table*. Both pages and page tables are described in more detail in section 5.1.

Memory maintained in the DSM subsystem is kept coherent throughout the backbone. The DSM subsystem implements memory coherence in accordance with the protocol supported. There are several DSM consistency policies that could have been selected including: strict consistency [15], sequential consistency [24], processor consistency [25], weak consistency [26], and release consistency [12]. Oasis uses a sequential consistency model because the consistency behavior for our test suite of POSIX applications should preserve the property that the most recently written value should be returned from a read operation. This policy can be implemented using a number of protocols including write-invalidate or write update.

When multiple sites access information simultaneously, synchronization is required to support mutual exclusion and thus to serialize access to memory. Oasis provides locks for synchronization support. Each site maintains information on the location of the lock through a lock owner field. When a site holds the lock, the site maintains a queue of requests for the lock. When the lock is released, the first site on the queue receives the lock. The waiting queue is also forwarded to the next site to preserve the lock request order. This approach ensures lock acquisition fairness.

3.2.3 Message Subsystem

The message subsystem provides for reliable, in-order delivery of messages between backbone sites. The message subsystem operates as a independent thread of execution that wakes up upon receipt of messages. The messages are placed on messaging queues for threads to use. The messaging subsystem uses both *synchronous* and *asynchronous* send mechanisms to deliver messages. The asynchronous send allows the sending thread to continue processing without waiting for an acknowledgment. Synchronous messages block the sending thread until a response is received. The thread polls for a

[5] The DSM thread is a concept presented in Quarks.

reply message but yields to the processor at every iteration, if the message is not received. Message sequencing is used to ensure the ordering of messages, and enables threads to continue operation without waiting for responses.

In the next section, we examine the final component of our site architecture, the disconnection and reconnection subsystem, in detail.

4 Disconnection/Reconnection Subsystem

Oasis provides mechanisms to application programmers for continual operability during disconnection. The procedures to initiate the disconnection and reconnection process are described in this section. These are independent operations but use mechanisms common to both.

Two protocols were designed to support most operations involved in disconnection and reconnection. High level descriptions of the two protocols are provided below, while the specifics of the operations using these protocols are described in the disconnection and reconnection subsections.

Protocol 1 The protocol is initiated by the mobile machine. The protocol uses the server as the central coordinator of the operations, since the server has control over potential interference from the dynamically changing system participants.

Step 1) An operation-specific message is sent to the server. The server performs an internal operation.

Step 2) The server broadcasts an operation-specific message to backbone client sites.

Step 3) The backbone sites perform an internal operation and respond back to the server.

Step 4) The server gathers the responses from the backbone sites and sends a response to the initiating mobile machine.

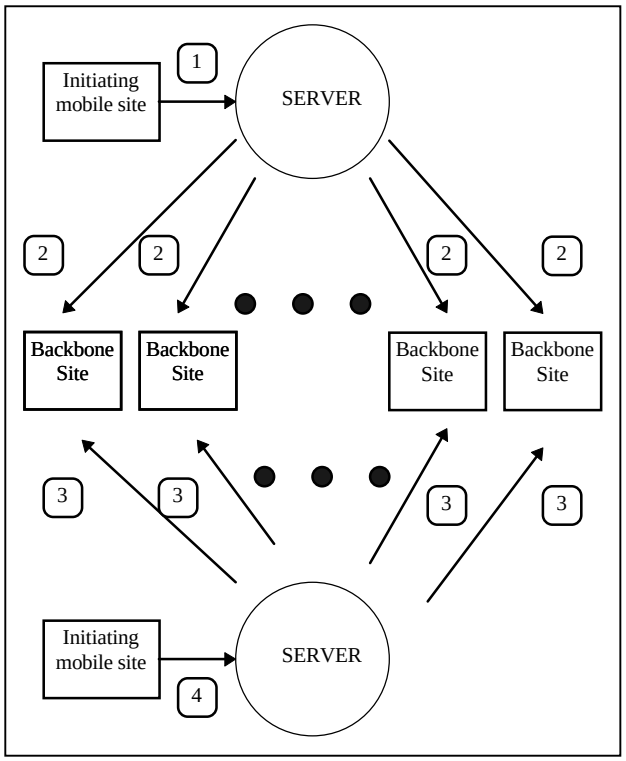

Figure 3) Operational flow of messages for protocol 1

Protocol 2 The protocol is initiated by the mobile machine. The protocol uses the server to get the current configuration of a shared resource. From that point, the initiating mobile machine can contact the backbone sites that have access to the shared resource.

Step 1) The mobile machine queries the server for statistics on a shared resource.

Step 2) The server responds with the statistics on the specified resource.

Step 3) The mobile machine contacts the backbone sites provided in the statistics.

Step 4) The backbone sites perform an operation-specific action and send status messages to the initiating mobile machine which is awaiting responses.

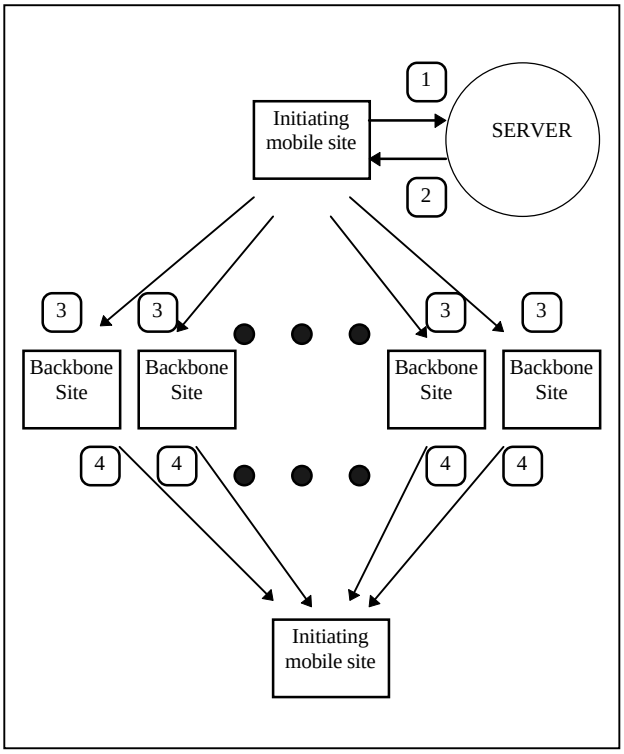

Figure 4) Operational flow of messages for protocol 2.

These two protocols are highly useful in the overall design of voluntary disconnection and reconnection. A significant difference between protocol 1 and protocol 2 is that protocol 2 is coordinated by the mobile site, whereas protocol 1 is coordinated by the server. Protocol 2 contacts the server to find the sites that have access to a shared resource which may be a subset of the backbone sites, while protocol 1 is used by the server between backbone sites.

4.1 Disconnection

Disconnection limits access to shared resources. When an application in a distributed system cannot access a critical resource, the application will either wait or the application will fail. Two Oasis mechanisms were designed to enhance mobility and to ensure that applications do not precipitously fail. The first mechanism is called *hoarding*. Hoarding attempts to cache the shared memory pages required during disconnection. In a similar way, Coda hoards files during disconnection[27,28]. The second mechanism, a *lease*, is used in conjunction with hoarding to enforce the time-based coherency model. The use of a time based protocol addresses an application's inability to access shared resources by returning ownership of resources to the backbone after a fixed time. During the fixed period, the mobile machine may use the resources exclusively. If the mobile machine exceeds its time allocation, two copies will exist of the shared resource. Reconciliation mechanisms are used to merge differences that arise. Section 4.4 discusses the mechanism for time based coherency. A time-based coherence approach is used so the backbone is not indefinitely postponed while waiting to access the leased pages.

Applications follow three steps to disconnect. First, applications query the user for information to be used for disconnection. Second, checks determine if there are any conflicts in the pages that are requested. Lastly, there is a synchronous signal to continue the disconnection process after the previous operations are complete. In the first step, the user is queried for information. The answers provided give the disconnection process necessary parameters for mobility support. Specifically, the parameters include the estimated disconnection time and the hoarding policy for the shared memory pages. Different hoarding policies provide a higher degree of accessibility to backbone sites when leases are used. These hoarding policies range from hoarding all pages, most recently used pages, least recently used pages, or most frequently referenced pages. If the user does not know which hoarding policy to select, a default is provided. The second step checks the requested pages for conflicts. Conflicts arise using leases when the user requests a lease on a page that is already leased or owned by another disconnected machine. In this case, it may not be possible to hoard the entire set of pages required. When this problem occurs, the user will be given the opportunity to terminate the disconnection process or to hoard the desired pages without a lease. Lastly, the disconnection algorithm is inititated by internally or externally generating a signal which will indirectly activate the *disconnection/reconnection thread*. The following operations occur during *resource reorganization*:

- Acquire global disconnection system lock
- Flush stored shared memory pages and locks to the backbone

- Hoard shared memory pages to the disconnecting site and optionally establish a lease for those pages
- Release global disconnection system lock and initiate independent operation

During resource reorganization we must: 1) move shared memory pages and shared locks from a disconnecting machine to the backbone sites and 2) gather needed shared memory from the backbone sites. During reorganization, the system is *globally locked* to prevent ongoing DSM coherence protocol actions from interfering with the resource reorganization actions. In particular, the procedure to globally lock the system uses a centralized server-based locking algorithm. Other distributed mutual exclusion algorithms were investigated including: Lamport's algorithm[24], Ricart-Agrawala algorithm[29], and Maekawa's algorithm[30]. These algorithms were not adopted because Oasis must handle a dynamically changing number of participants, and thus, has chosen a conservative policy that locks all sites to prevent the possibility of errant behavior. For example, new sites entering into the backbone that request shared resources are briefly suspended because they will interfere with the disconnection algorithm.

Locking the system uses protocol 1 by suspending the application thread and the DSM thread. The algorithm begins with the disconnecting mobile machine acquiring the *local system synchronization mutex* which locally prevents resources from migrating. Then, the disconnecting mobile machine contacts the server. The server honors only one disconnecting site by proceeding only if it can lock the *server synchronization mutex* or rejecting the request if the lock is not free. This mutex guarantees serial access of disconnecting and reconnecting mobile machines; it represents a global system lock. Next, the server sends a message to all sites in the backbone, except the disconnection initiator, requesting that their local system synchronization mutexes be locked. The server acknowledges the disconnecting mobile machine when all backbone sites respond. Any subsequent requests are denied, and these requesters must release their local system synchronization mutexes in order to prevent the possibility of deadlock.

Once the global system lock is acquired, shared memory pages on the disconnecting machine are flushed to the backbone so the most recent copies used by the backbone remain continually accessible. The write-back algorithm protocol 2 for all regions and all pages that a disconnecting mobile machine uses. The disconnecting mobile machine uniformly flushes copies of the locally owned shared memory pages to the backbone sites that store the pages. It then receives an acknowledgment from each site.

Similarly, shared locks are also flushed to the backbone in another phase of this algorithm. The algorithm for flushing the shared locks at disconnection is similar to the method used for shared pages. The difference is, instead of transferring the shared memory pages, the lock structure with the queue of waiting requests is sent to the newly elected owner.

When a site is disconnected, locks have a different semantic behavior. The behavior allows backbone sites to access locks normally, and disconnected mobile machines to access locks concurrently with the backbone. Locks are exclusively associated with shared memory and are used to preserve mutual exclusion between backbone sites. Disconnected machines are not concerned with mutual exclusion between disconnected and connected components since no other sites are able to access shared memory except the mobile machine that acquires the memory. Thus, the backbone may be able to acquire locks, but will not be able to proceed unless the resources required can be used. Thus, mutual exclusion is guaranteed. Locks are conceptually unified when the backbone and the disconnected mobile machine are unified.

After flushing the locks, all the locks are returned to the backbone, and all pages owned at disconnected mobile machines have been copied to sites in the backbone. The next step in the disconnection process is to obtain the necessary resources required to operate in a disconnected mode. There are two methods which can be used to hoard shared memory pages: hoarding with a lease and hoarding without a lease. The method is determined by the estimated disconnection time entered by the user. If the disconnection time is non-zero, then the disconnecting machine will register a lease on the backbone and will own the hoarded pages. Otherwise, the disconnecting mobile machine will hoard copies of the shared memory pages from the backbone without a lease. In this latter case, the backbone retains ownership of the pages. Potential inconsistencies can arise when

backbone sites and disconnected machines concurrently update replicas.

The algorithm for hoarding and establishing leases involves creating a shared memory page *hoarding request map* in order to reassign page ownership. The hoarding request map is used to determine the pages to be hoarded, determine the pages to establish leases, and to synchronize the page owner field in the page tables on each backbone site. We begin by assuming all sites have correct up-to-date information stored in their local page tables. The hoarding request map indicates the pages required by the disconnecting mobile machine. First, the hoarding request map is sent by the disconnection initiator to all backbone sites that have the region of memory attached. In response, each backbone site sends the pages that were requested to the mobile machine. In addition, each site determines where the pages will reside after this operation completes and thus, updates its tables accordingly. After the operation is complete and if leases were established, all backbone sites record that the disconnected mobile machine is the primary owner of the pages that were sent to it. A secondary owner field is also maintained and used to revert ownership of leased pages to backbone sites once the lease expires. It is important to note that leased pages cannot be re-leased to other requestors if copies of the pages exist on disconnected machines. Since page ownership is associated with lease reconciliation, discussed in section 4.3.3, re-leasing can not occur with an inaccessible leased page.

Consider the following example with site 6 disconnecting from the backbone with a lease. Assume pages 1, 2, 3, and 5 are requested in the hoarding request map by the disconnecting machine. The four memory pages are stored at sites 1, 2, 3, and 1 respectively on the backbone. When site 1 receives the request map, it will send pages 1 and 5, and adjust the page owner field of the page table entry for pages 1, 2, 3, and 5 to contain a 6. Further, it will store the previous values 1, 2, 3 and 1 into the secondary owner field of the structure. Next, site 2 will send page 2 and take similar action. Then, site 3 will send page 3 and take similar action.

Oasis handles the issue of incomplete hoarding which could potentially cause resources to be inaccessible on demand. In particular, in Oasis, if a client disconnects and is unable to access a shared memory page that is required, the application is suspended while waiting for the resource. In addition, the user is prompted with an information dialog box to reconnect to the backbone. Reconnection may satisfy the request if the resource is available on the backbone and has no associated lease that precludes the page from being hoarded. When there is a lease associated with the resource, Oasis informs the user of the remaining time of the lease period so that the user can retry later.

Lastly, disconnection occurs when the backbone unlocks the system-wide mutual exclusion flag. This algorithm is similar to the one described in section 4.1.1, except that the system mutexes which were previously acquired are released. This release operation allows the backbone sites to continue operation. At this point, the backbone and the disconnected mobile machine are detached, and the disconnecting mobile machine releases its local system synchronization mutex and suspends the disconnection/reconnection thread.

4.2 Disconnected Operation

Mobile machines that disconnect from the backbone must have the ability to reintegrate updates that occurred when disconnected. Our initial approach in the design of Oasis was to create a distributed transaction model to reconcile disconnected shared memory updates into the backbone. Protocols such as the well known *two-phase commit algorithm* were discussed. We analyzed the approach carefully, considering its applicability to our problem. However, ultimately we decided that the difficulty of making the two-phase commit work well in the face of communication failure seemed overly difficult.

At this point, we stepped back and decided to approach the problem differently. In particular, we designed two different mechanisms for reconciliation. The first mechanism supports the consistent memory model of the backbone by using leases, as described earlier. Machines that have a valid lease at reconnection can simply replace pages on the backbone, which are guaranteed to be unchanged from pages updated during disconnection. The second mechanism, *capturing* and *reconciliation*, permits Oasis to integrate changes that occurred on disconnected machines with possible conflicting updates that may have

occurred on the backbone. We describe reconciliation further in the next section.

Capturing is used to record all write to the memory while the client operates disconnected. We have implemented capturing using two approaches: *logging* and *twinning*. With logging, an update log is maintained at the client site, and begins at the time of disconnection and continues until reconnection. In particular, Oasis mechanisms trap all writes made to shared memory and record these writes to an in-memory log. In order to gather the writes to shared virtual memory, the pages are placed in read only mode when hoarded at the disconnecting machine. When a write occurs to the shared memory page, the privilege level for the memory page is upgraded to read-write. After the completion of the write instruction, the page protection is lowered to read-only. The execution of the instruction is interpreted, otherwise the processor would be allowed to continue operation and perform additional writes while the privilege level is set to read-write. Thus, each write is individually trapped during disconnection and logged.

Oasis also addresses the issue of the log becoming too large. In particular, an in-memory copy of the log is maintained as long as the log stays within a prescribed boundary. Once the in-memory log becomes full, the log is written to persistent storage[7] and the in-memory log is emptied. While the buffering of writes is a common technique, in addition, in order to reduce the number of I/O operations and reduce the amount of disk space utilized, Oasis compacts the writes to memory. Since programs often reference memory in patterns that exhibit a high degree of locality, existing entries are simply replaced. This process seems like it would corrupt the total ordering of the memory updates, but the log is not used as a transaction playback log. Instead, only the most up-to-date entries need to be retained. A fixed size is allocated to the persistent storage log. When the size reaches the maximum allocated the application must reconnect.

Figure 6 below shows the structure of the update log. The log consists of an array of records that include the size of the write to memory, the virtual address of the write, the value of the memory location before the write occurred, the value of the memory location after the write was performed, a region identifier, and the page number.

The second approach we use is called twinning. Using a twin copy of a memory page preserves the original state of memory from the point of disconnection. The most up-to-date memory values are kept in a replica of the original memory, while new values are kept in the twin. An advantage of using twins is that shared memory accesses during disconnected operation need not trap write accesses nor is persistent storage accessed.

Char	unsigned long *	long	long	long	int
1) size	virtual_address	before_value	after_value	regionid	pagenum
2) size	virtual_address	before_value	after_value	regionid	pagenum
3) size	virtual_address	before_value	after_value	regionid	pagenum
4) size	virtual_address	before_value	after_value	regionid	pagenum
5) size	virtual_address	before_value	after_value	regionid	pagenum
6) size	virtual_address	before_value	after_value	regionid	pagenum
7) size	virtual_address	before_value	after_value	regionid	pagenum
8) size	virtual_address	before_value	after_value	regionid	pagenum

Figure 6) Structure of the update log.

4.3 Reconnection Algorithm

Reconnection is initiated by either an internally or externally generated interrupt which will indirectly activate the disconnection/reconnection thread. The reconnection algorithm must perform the following operations to maintain system coherency. First, the backbone global system mutual exclusion lock is raised. The procedure for locking and unlocking the system is exactly the same as the ones described in section 4.1.1 and 4.1.5. Second, the reconnecting machine must validate the lease, if one was used. Thus, when a disconnected machine reconnects to the backbone, outstanding leases are examined to remove any invalid leases. Third, after determining if the lease is valid, two different situations arise: reconnection with a valid lease, or reconnection using reconciliation. The next sections discuss these issues.

4.3.1 Reconnection With A Valid Lease

The procedure to reconnect involves releasing all outstanding leases stored on the backbone that pertain to the reconnecting machine. The server holds the lease timer, and this timer must be cancelled during reconnection. Specifically, using

[7] Persistent storage may either be a hard disk for laptops or Flash memory for PDAs

protocol 1, the reconnecting machine requests the server to instruct the backbone sites to drop leases associated with it. If the server has already dropped locks due to timer expiration, the server responds with a negative acknowledgment that indicates that the reconnecting machine must use reconciliation methods described in the next section. If the lease timer is still valid, the server will instruct all backbone sites to drop leases that are held. Each backbone site purges leases held on pages which are owned at the reconnecting machine. The server will also terminate the timer task since the mobile machine connected before the timer elapsed. Lastly, the reconciliation log or all original replicas can be discarded and all pages updated during disconnection can simply replace pages stored on the backbone.

4.3.2 Reconnection Using Reconciliation Methods

Reconciliation is a process that applies the shared memory modifications made during disconnection to the backbone. Two methods for reconciling disconnected mobile machines with the backbone can be selected. The first method involves ignoring changes that occurred when disconnected. This method would, in effect, have given the reconnecting machine read-only access while disconnected. This method could be useful when disconnection is anticipated and numerous resources were hoarded so that they could be examined during the disconnection period. This method is used as the default for applications that have not supplied a *merge procedure* which is an application-specific rule-based approach to integrate disconnected memory changes.

The second method updates the shared memory pages on the backbone using the reconciliation log stored from the point of disconnection. This policy is called a *merge procedure*. An Oasis merge procedure is similar to the ones used in Bayou[31], except Oasis operates on memory based modifications where Bayou operates with database transactions. Using a rule based application-specific integration module, updates from the log can be reintegrated into the pages. Using rules to describe the actions to merge updates, structure is added to the memory regions.

The Shopping Assistant application, for example, has a merge procedure to reconcile item counts for each product. The item count is a positive number and represents a product unit count that exists on a store shelf. There are two operations on an item count. The first operation is that the shopper can puchase. This request will decrement the item count. A request can only be satisfied if the item count is nonzero. The second operation occurs when the restocker adds items to the shelf. This will increase the item count by the restocked amount. When shoppers are attached to the backbone, the item count will remain consistent and the item count will always reflect the accessible amount of an item. However, disconnected operation requires reintegration with a merge procedure because of potential inconsistencies. For instance, if an item count has 6 units when a mobile machine disconnects, the mobile machine will operate using 6 as the item count. If the disconnected machine requests 4 units of the 6 items, the item count would reflect 2 items. Concurrently, another shopper connected to the backbone acquires 2 units of the 6 items leaving the item count at 4. When the disconnected unit reconciles, the merge procedure determines the number requested, while disconnected, was 4. The backbone item count which is 4 will be compared to verify whether the disconnected item request can be satisfied. In this case the request of 4 items can be satisfied which will leaves the backbone item count at zero. However, if the disconnected machine acquired 5 units, the merge procedure will determine that the disconnected request cannot be satisfied and will only give the reconnecting customer 4 units of the item.

4.4 Backbone Expiration Of Time-Based Coherency

Leases are used to maintain consistency. Once the leases expires, the backbone must resume processing shared memory requests or requests for locks. When leases are removed from the server, a potential race condition exists between the server and a reconnecting machine. Both lease expiration and reconnection of a machine may be competing to remove the remnants of the lease state. In addition, the server is in control of releasing leased locks thus no mobile machine can perform a voluntary disconnect and reconnection because of potential lease interference. Therefore, before the server can proceed, the system must be globally locked.

The algorithm for expiration of leases is similar to protocol 1 except for steps 1 and step 4. These steps are not needed since this algorithm is not initiated by a mobile machine. Instead, this algorithm is started at the server in response to the timer notification. The server first acquires the global system lock before the lease is invalidated. The backbone sites scan the page tables to find any pages that are leased by the identified disconnected mobile machine. Before the lease is dropped, the page owner fields in the page table are replaced with the secondary owner field. Each backbone site acknowledges the server, which then releases the global system lock when all responses are received.

5 Implementation

Oasis uses eight Digital Equipment Corporation (DEC) Alpha AXP/150 computers. The eight machines were networked together using 10Mbit Ethernet. The Alpha computers are the Jensen model which operate at 150 MHz and are configured with 32 Megabytes of RAM. The operating system is OSF1 UNIX version 2. The application programs described in section 1.1 were implemented as a proof-of-concept to verify the system design. The applications were initially developed using POSIX shared memory and semaphores on a uniprocessor. They are event driven using a graphical user interface created with TCL and TK[32,33]. The Oasis POSIX translation library, written in C, allowed these applications to support mobility with only minor changes. An Oasis client application is composed of three threads: the communication thread, the DSM thread, and the disconnection-reconnection thread. Each thread is implemented as a POSIX thread.

6 Evaluation

6.1 System Performance and Basic Costs

The basic costs in Oasis are the time required for a page fault, network messages, and memory copies. A page fault takes 17.35 milliseconds to complete and is comprised of a short network message of 32 bytes from the requester and long message of eight 1K messages from the responder, and two memory to memory copy operations (`bcopy`) of the 8K page. Message transmission time was measured for two types of messages, a short-short message which takes 1.95 milliseconds round-trip and a short-long message which takes 11.71 milliseconds round-trip. The time required for a memory-to-memory copy was .98 milliseconds which is performed at the requesting and responding sites. Table 1 compares three DSM systems whose page fault times have been accurately measured in the literature[37]. Oasis compares favorably with these implementations given that Oasis faults 8K pages and is a non-kernel invasive implementation.

DSM Systems	Technology	Operating System	Page Size	Page fault Time
Mirage	VAX 11/70	Locus	512 bytes	27.5ms
Mirage +	PS/2 486/25	AIX/TCF	4K	19.9ms
Oasis	DEC Alpha AXP/150	OSF1 UNIX version 2	8K	17.3ms

Table 1) Comparison of Page Faults

6.2 Component Cost of Disconnection Algorithm

The evaluation of disconnection was performed by creating a test program with a configurable amount of shared memory pages and a single shared lock. The system configuration consisted of the Oasis server, one disconnecting site, and one to six backbone sites. Each site started a test program which allocated six shared memory pages and one shared lock. The test was run in two different configurations. The first configuration, *case 1* has all the shared resources stored locally at the time of disconnection. The second case, *case 2* has all shared resources located on backbone sites. These two configurations represent the worst and best case, respectively. When shared resources are distributed between the disconnecting computer and the backbone, the case is bounded from below by case 2 and from above by case 1. Figure 7 provides comparative data summarizing the component cost of each part of disconnection. Disconnection consists of five components:

- Acquire Global System Lock
- Flush Shared memory Pages
- Acquire and Sync Pages
- Flush Shared locks
- Release Global System Lock

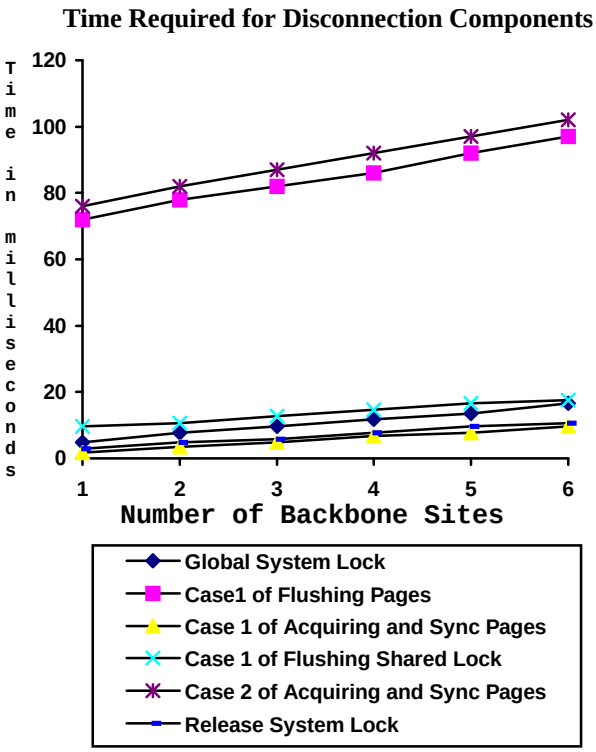

Figure 7) Component cost of disconnection of case 1 vs. case 2

The results indicates that disconnection component costs in Oasis increase linearly when more sites are added into the backbone. The extra cost occurs because each component requires an extra request message be sent. Flushing pages and acquiring pages are the most time consuming part of the disconnection since they both require short-long messages. All other components are much faster because they require short-short messages. The flushing of shared pages and flushing of shared locks in case 2 takes no time since there are no pages or locks are stored locally on the disconnecting machine. Figure 8 below represents the cumulative component costs of figure 7 for case 1 and case 2.

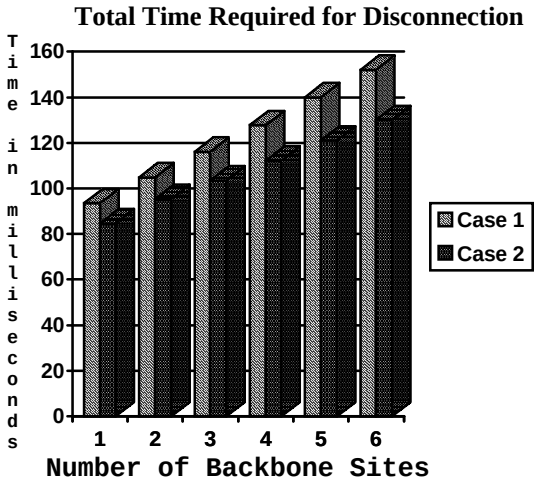

Figure 8) Total time required for disconnection

6.3 Cost of Disconnection Operation

Disconnected operation used two methods of recording updates: logging and shadowing. These two methods were compared by creating a simple test program that allowed a mobile computer to disconnect from the backbone. Once disconnected the mobile computer could perform N writes to a shared memory page. When using the logging method each write costs 546 microseconds which includes the cost of local interrupt handling. The shadowing method generates a write fault initially and allocates a twin memory page. This requires an 8K memory `bcopy` of .98 milliseconds to copy the twin page. After the creation of the twin page, the memory is set writeable and no other faults occur. The initial creation of the twin along with interrupt overhead costs 3.25 milliseconds. Figure 9 compares the repeated hardware interrupt cost at the user level for logging with the one-time memory allocation and memory-to-memory transfer cost of shadowing.

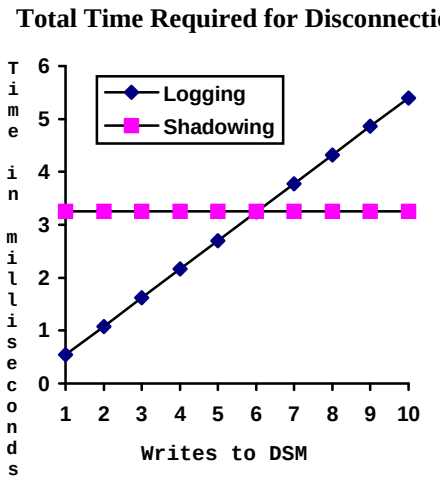

Figure 9) Time comparison writes to DSM for logging vs. shadowing

6.4 Cost of Reconciliation Algorithm

A reconciliation handler is an application-specific routine that depends on application rules. The rules will have different costs depending on whether logging or shadowing is used. The logging method has advantages over the shadowing method because the log completely describes the address of the modification that occurred while disconnected, while the shadowing method records page level changes. If the amount of disconnected shared memory updates is small, logging saves space because the whole twin page is not needed. Shadowing has an advantage over logging because logging may require a search algorithm when log compaction is used to compress the size of the log. Shadowing allows direct access of memory values. A space-time trade off between logging and shadowing exist and depending on the behavior of the application, one will be more suitable than the other. This tradeoff is heavily influenced by the machine architecture page fault handling cost versus memory allocation and a memory-to-memory copy. Figure 9 shows that if more than six writes occur, shadowing is less expensive. However, this tradeoff must be balanced against the compactness that logging provides in the face of many short accesses to many different pages.

6.5 Further thoughts and experiences

The Oasis application suite was developed to test proof-of-concept for Oasis . The applications were initially created for uniprocessor POSIX shared memory. The POSIX-to-Oasis translation was used to convert these applications so they could operate in a distributed manner. Thus, with minimal application changes, the Oasis suite became functional under an integrated DSM/SVM support model. Oasis disconnection and reconnection methods provide continued operability when disconnected.

Two protocols were developed under Oasis that were used to perform all actions that pertain to disconnection and reconnection. We were quite surprised that such limited recoding of an existing DSM system could provide such a powerful and easy-to-understand set of mechanisms for users. In addition, the use of time based coherency provides a flexible method to maintain system integrity during disconnection. Indeed, only one simple issue arose during actual use of the system. In particular, when the backbone was unable to continue operation due to inability to access shared memory pages held under a lease, the backbone was frozen for too long. To solve this problem, a maximum lease time was imposed so the backbone was not suspended indefinitely.

We plan to continue to use Oasis and the Oasis suite to gain experience with the flexibility and practicality of this paradigm. Our current assessment is quite positive based on the status of our implementation.

7 Related Work

Oasis is the first DSM system designed to support mobility. It is unique in that it integrates DSM applications and SVM applications under one common platform for distributed computing. Other distributed systems handle the concept of disconnection and reconnection but none use a memory-based approach..

- The terminology “hoarding” was adopted from Coda. Hoarding indicates that resources are being cached locally during disconnection.

- The use of time based coherency was adopted from concepts in Mirage[14] and Leases[34]. The time interval used in Mirage was designed to improve system performance by preventing page thrashing in DSM systems.

- Reconciliation merge procedures are comparable to the approach taken in Locus[35], Ficus[36] and Bayou[31].

8 Conclusions

In this paper, we have described the architecture and design of Oasis. This design is a proof-of-concept and strongly indicates that loosely coupled sharing applications remain functional, not only in distributed systems, but in mobile distributed systems. Application conversion to Oasis was done with minimal intrusiveness to the application source code. Oasis further provides a convenient programming methodology in which programmers need not worry about the intrinsic mechanisms required for mobility. We believe that mobile DSM is a promising paradigm to support collaborative applications that exhibit a loosely coupled degree of sharing.

9 Future Work

A focus for the next phase of this research includes purchase of laptops or PDAs communicating through wireless networking. Several areas of continuing research include incorporating multiple DSM consistency protocols, as well as experiments with different hoarding policies to obtain pages during disconnection. Future mobile distributed architectures are being evaluated which may be designed around a shared object model instead of a paged region model.

References

[1] Marvin Theimer, Alan Demers, Brent Welch. Operating System Issues for PDAs. *Fourth Workshop on Workstation Operating Systems*, 2-8, October 1993.

[2] Ramon Caceres, Fred Douglis, Kai Li, and Brain Marsh. Operating System Implications of Solid-State Mobile Computers. *Fourth Workshop on Workstation Operating Systems*, 21-27, October 1993.

[3] Michael Culbert. Low power hardware for high performance pda. In *Proceedings of the 1994 IEEE Computer Conference*, San Francisco, CA, USA,1994.

[4] Ivan A. Getting. The global positioning system, *IEEE Spectrum*, 30(12), December 1993.

[5] S.Mullender, *Distributed Systems*, Addison-Wesley Publishing, New York, NY, USA, 1993

[6] Andy Birrel and Brian Nelson, Implementing remote procedure calls, *ACM Transactions on Computer Systems*, 2(1):39-59, February 1984

[7] Michael D. Schroeder and Michael Burrows. Performance of Firefly RPC. *ACM Transactions on Computer Systems*, 8(1):1-17, February 1990

[8] P Brinch Hansen. The nucleus of a multiprogramming system. *Communications of the ACM*, 13(4):238-241, April 1970

[9] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. In *Proceedings 5th ACM SIGACT-SIGOPS Symposium of Principles of Distributed Computing*, pages 229-239, Canada, August 1986. ACM Press.

[10] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321-359, November 1989.

[11] Roberto Bisiani and Mosur Ravishankar. PLUS: A distributed shared-memory system. Technical Report, School of Computer Science, Carnegie Mellon University, 1990.

[12] John K. Bennett, John Carter, and Willy Zwaenepeol. Munin: Distributed shared memory based on type specific memory coherence. In *Proceedings of 1990 Conf. Principles and Practice of Parallel Programming*, pages 168-176, New York, NY, USA, 1990 ACM Press.

[13] Umakishore Ramachandran, Mustaque Ahamad, and M Yousef A. Khalidi. Unifying synchronization and data transfer in maintaining coherence of distributed shared memory. Technical Report GIT-ICS-88/23, Georgia Institute of Technology, Atlanta, GA, USA, June 1988.

[14] Brett D. Fleisch and Gerald J Popek. Mirage: A coherent distributed shared memory design. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, published in Operating Systems Review 23(5) Special Issue, pages 211-223, The Wigwam, Litchfield Park, Arizona, USA, December 1989. ACM SIGOPS, ACM Press.

[15] B. Nitzberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *IEEE Computer*, 24(8):52-60, August 1991.

[16] Virginia Lo. Operating systems implementations of distributed shared memory. *Advances in Computers*, 39,1994.

[17] John K. Ousterhout, Andrew R. Cherenson, Fredrick Douglis, Michael N. Nelson, and Brent K. Welsch. The sprite network operating system. *IEEE Computer*, pages 23-36, February 1988.

[18] M.N. Nelson, B.B. Welsch, and J.K. Ousterhout. Caching in the sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134-154, February 1988.

[19] K.P. Birman and Robbert Van Renesse, Reliable Distributed Computing with Isis Toolkit. *IEEE Computer Press*, Los Alamitos, CA, USA, 1994

[20] P. Keleher, A.L. Cox, S. Dwarkadas and W. Zwaenepeol. TreadMarks: distributed shared memory on standard workstations and operating systems. *Proceeding of the 1994 Winter USENIX conference*, pp115-131, January 1994.

[21] Dilip Khandekar, Quarks: Portable DSM on UNIX, Tech Report Computer Systems Laboratory Department of Computer Science University of Utah Salt Lake City, May 1995.

[22] J.. Carter, D. Khandekar, and L. Kamb, "Distributed Shared Memory: Where We Are and Where We Should Be Headed," In the *Proceedings of the 5th Workshop on Hot Topics in Operating Systems*, pp.119-122, May 1995.

[23] Brett D. Fleisch and Niels C. Juul. A Memory Approach to Consistent, Reliable Distributed Shared Memory. In the *Proceedings of the 5th Workshop on Hot Topics in Operating Systems*, pp.108-112, May 1995.

[24] L. Lamport, How to Make a Multiprocessor computer that Correctly Executes Multiprocessor Programs. *IEEE Transactions on Computers*, 29, 9 1979, 690-691

[25] J. Goodman, Cache Consistency and Sequential Consistency. Technical Report. 61, *IEEE Computer Society*, SCI Committee, 1989

[26] M. Dubois, C. Scheurich, and F. Briggs, Synchronization, Coherence and Event Ordering in Multiprocessors. *IEEE Computer* 21, 2 1988, 9-21.

[27] James Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File *System. Proceedings of the Thirteenth ACM Symposium on Operating System Principles*, 213-215 Oct. 1991.

[28] M Satyanarayanan, James Kistler, Puneet Kumar, Maria Okasaki, Ellen Siegel, David Steere. Coda: A Highly available System for a Distributed Workstation Environment. *IEEE Transactions on Computers* , 447-458, Apr 1990.

[29] G. Ricart, and A. Agrawala, An optimal algorithm for mutual exclusion in computer networks. *Comms ACM*, vol. 24 no. 1, pages 9-17, 1981

[30] Maekawa, Oldehoeft and Oldehoeft, Operating Systems: Advanced Concepts, The Banjamin/Cummings Publishing Company, Inc., 1987 Edition

[31] D. Terry, M Theimer, K Peterson, A. Demers, M. Spreitzer, and C Hauser, Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System, In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, December 3-6, 1995

[32] J.K. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley Publishing Company 1994.

[33] K. Petersen, Tcl/Tk for a Personal Digital Assistant, *Proceeding of the Usenix Symposium on Very High Level Languages*, 1994.

[34] C. Gray, D. Cheriton, Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency, In *Proceeding of the 12th ACM Symposium on Operating System Principles*. December, 1989

[35] B. Walker, G. Popek, R. English, C Kline, G. Thiel, The LOCUS Distributed Operating System. *In Proceedings of the 9th ACM Symposium on Operating System Principles*. October, 1983.

[36] R. Guy, J. Heidemann, W. Mak, T. Page, Jr., G. Popek, and D. Rothmeier. Implementation of the Ficus replicated file system. *In USENIX Conference Proceedings*, pages 63-71. USENIX, June 1990.

[37] B. D. Fleisch and R. L. Hyde, and N. C. Juul. Mirage +: A kernel implementation of distributed shared memory on a network of personal computers. *Software- Practice & Experience*, 24(10):887-909, Oct. 1994