Using DSVM to Implement Transparent Distributed File Access

Ramon Lawrence
Department of Computer Science
University of Manitoba

May 29, 1998

Abstract

This paper attempts to demonstrate how the use of distributed shared virtual memory (DSVM) can make the implementation of a distributed file system easier. Distributed file systems based on DSVM and client-server architectures are proposed and compared. Prototypes of both architectures are implemented, with the DSVM version constructed using Treadmarks.

1 Introduction

Distributed Shared Virtual Memory (DSVM) is a global shared address space accessible by processes on different machines. Instead of the usual interprocess communication (IPC) methods needed for distributed processes to communicate, DSVM provides this communication transparently to the processes. Thus, process cooperation is more easily achieved in a distributed environment.

This project will focus on showing how DSVM can be used to simplify the implementation of a global application directory for a set of applications distributed over an Ethernet. Two implementations, one DSVM-based, will be presented which attempt to provide a flat name space for applications across a limited network domain. The advantages/disadvantages of both approaches will be discussed and explanation on why a flat name space for applications may be beneficial in such an environment.

2 Related Work

Distributed file systems were examined in great detail in the late eighties. Extensive research was carried out at Cornell University where a prototype distributed file system called Deceit [5] was developed. It contained many of the features desirable for efficient file sharing and transparent file access. The work also produced a set of distributed management tools [2] which simplified the task of creating distributed software. The system was built on top of Sun NFS [4].

The Deceit system was a client/server architecture which allowed multiple servers. To improve scalability, clients were divided into clusters called cells each served by a server. The system implemented a superset of NFS functionality and behaved like NFS without client modification. In NFS, the global directory hierarchy is divided amoung servers in a static fashion. Each server contains a portion of the hierarchy and the complete hierarchy can only be formed by combining their portions. Although clients may communicate with many servers, servers don't communicate amoung themselves. The main difference between Deceit and NFS is that in Deceit files are not statically bound to any server. Clients communicate only with one server, and if the files requested are non-local to the server then the server contacts other servers to fill the request.

Using this architecture, Deceit was able to provide user-configurable replication of files and a file version control mechanism. Replicas couldn't cross cell boundaries, as cells from a logical unit implementing replication and security access mechanisms. Thus, full transparency wasn't achieved as servers at different cells had different functionality and stored different files. The system provided global one-copy serializability which is that all clients see the whole file system as if only one copy existed of all files. There are many other such systems demonstrating similiar functionality [7, 6]. The goal of this project is not to duplicate the past accomplishments, but to show that by using DSVM, the implementation of the distributed system is simpler.

3 Environment Specification

The networking domain is a series of interconnected PCs on an Ethernet. Each PC is considered a site where applications may be stored. An application is assumed to have a unique name across the entire network. A global name table (GNT) is defined to manage the applications on the network. The global name table permits the operations: add application, remove application, move application, and find application. In future work, it would be beneficial to have the table support more advanced operations including sharing and replication of applications. The GNT is managed by the operating system (OS). Operations may come from the applications itself during an install, the user wishing to relocate the application, and the OS itself. Reasons why the OS may want to manipulate the GNT are discussed later.

The entries of the GNT consist of the application name and the network location of the application. Currently, the table does not support replication, so an application can only exist at one site on the network. Although a name and a location are sufficient to find and retrieve the application, another structure is defined to capture application information. This structure is an application descriptor record (ADR) which contains information about an applications resource usage (files,hardware,etc.), run-time statistics, and sharing options. Although an exact definition of the ADR is not necessary here, a structure storing all the ADRs of applications on the network could increase the utility of the GNT.

Currently, the GNT is simply a table-lookup mechanism to find applications on the network. It provides the user with a flat application name space, effectively hiding the network from them. By defining application requirements formally in an ADR, it may be possible to hide the network from the application. This permits dynamic load balancing which is the balanced distribution of application processes to take advantage of changing network access statistics. The GNT provides a mechanism to make application access to the user transparent which allows the OS to freely move application files. The ADR provides a mechanism for hiding the network from a running application thereby allowing the OS to dynamically choose sites for executing applications.

One assumption the GNT makes is that all file access by an application is done relative to its home directory. Although this is not strictly necessary for the GNT itself, it is important for later definition of the ADR. The GNT stores an absolute location path along with a machine name to uniquely find an application. Clearly, if we allow applications to be executed at different sites, absolute paths to files would pose a problem. It is feasible for all files that belong to an application to be accessed relative to its home directory, but user, OS, or other application files may pose a problem.

To achieve this relative path constraint, a few restrictions must be placed on file naming and locating conventions. For files belonging to an application, file access must be specified relative to their home directory. This restriction is not to severe as most PC applications already create and work in their own directory. OS files used by applications can be assumed to exist in a fixed directory at all sites because the OS must exist at all sites. If an application references files of another application, it can do so by giving the unique application name and then specifying the file relative to the other application's home directory. Enforcing that an application name be unique across all networks is a harder problem which can only be suitably solved by having the software vendor register the name with the OS provider. This name may be different than the local network name which can always be tailored to a specific network. In order for applications to reference each other meaningfully, some sort of global naming must be done.

The most complicated problem is user files because users have grown accustomed to creating complicated directory hierarchies and have had the luxury of scattering their files freely. Being designed for a network, it is not unreasonable to assume that each user has a home directory. This home directory can function like an application's home directory, so most of the user file access should be specified relative to their home directory. There is one nasty exception in that some files must be at specified sites at fixed location. This can be overcome by concatenating the machine name, path, and filename to give unique names. These files do not pose much of a concern because there is no reason for the user or OS to move them frequently.

Since applications are invoked from all sites on the network, the GNT must be in some way distributed across it. The ease with which this distribution can be achieved illustrates the power of DSVM.

4 Architectural Differences

4.1 Non-DSVM Architecture

The non-DSVM architecture is shown in Figure 1. Its distinguishable features are distinct client and server processes, and the explicit communication between the two. When there are separate client and server processes, the problem of dividing the tasks between the two becomes a major issue. To enhance efficiency, many tasks are propagated to the client side. This often results in more complicated logic for both the client and server processes. For example, client buffering can be used to avoid network access to the server, but then processing updates becomes more complicated as copies stored in client buffers must be invalidated. This causes both increasing overhead and complexity.

Further, a dedicated server introduces a point of failure as the data is not distributed across all sites. Multiple servers compensate for this problem, but then the scalability of the system suffers as the servers must maintain consistency amoung themselves. The synchronization and communication is explicitly handled by each process, so IPC code can easily obscure control logic in a large implementation.

The client/server architecture is based on the idea of machines in isolation. The distribution is achieved by pessimistic sharing as sharing is only accomplished by explicit requests and responses. This explicit communication minimizes sharing. Although this may be desirable for implementing security protocols, with today's trend for internetworking and communication, it may be desirable to have a sharing protocol which is optimistic, in that it allows data sharing unless explicitly prevented.

Finally, the reason why this architecture is so popular is its implementation simplicity. With a single server, there is no replication and the message passing is easy to handle. The communication overhead is also minimized as only the minimum amount of information need be passed between the client and server. This allows for great flexibility in communication form. The only drawback of using a client/server architecture with the GNT is that the majority of GNT accesses will be location searches in the global directory. This would result in a network access to the server everytime an application is started.

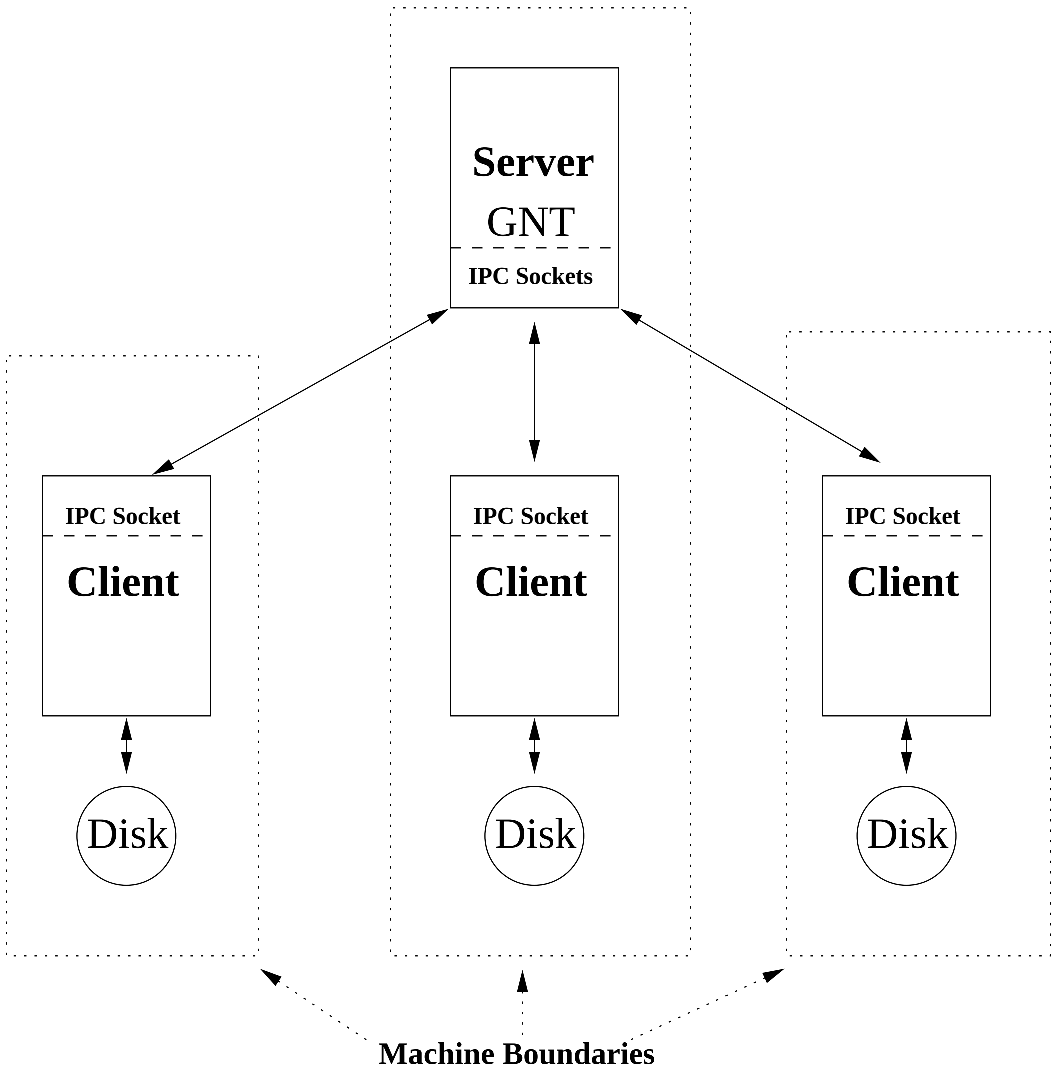

Figure 1: Non-DSVM architecture

4.2 DSVM Architecture

The DSVM architecture is illustrated in Figure 2. In this architecture, there are no distinguishable client and server processes. Each process performs the functions of both a client and a server as would be expected in a truly distributed system. The DSVM provides the clients at all machines with a single machine view. The DSVM stores the GNT, so the GNT appears to be local to all processes. DSVM provides optimistic sharing in that all data in the DSVM region is visible to all processes unless explicit security protocols are implemented on top of it. All communication between the processes is handled by the DSVM manager transparently. The processes would not even know that they are sharing the GNT data structure if it were not for the locks that maintain consistency across all sites.

The DSVM provides maximum replication because all sites have access to the GNT. A failure at one site would allow the other sites to continue as long as process failure occuring while holding the table lock can be detected. This optimistic sharing may not be desirable if there are priviledged processes or security constraints, but in this situation each process is assumed to be a well-behaved OS process.

There are hidden costs not explicitly shown in the diagram. The memory at each processor can only be made consistent using message passing mechanisms. To implement DSVM efficiently, the size of the DSVM blocks are normally equal to the page size. This may cause high network traffic in the presence of frequent updates and false sharing if the granularity of the data on the pages is small. In this case, the majority of GNT accesses are reads with updates only occurring when an application is added, moved, or removed; all of which are rare events. Hence, the DSVM implementation of the GNT should scale well due to the infrequency of updates.

4.3 Architectural Differences

The main difference between the two architectures is that the non-DSVM version does explicit communication while the DSVM version does implicit communication. Explicit communication is almost always more efficient as the programmer can decide on the minimum amount of data to be passed and design specific programming tricks to improve efficiency. Unfortunately, it is harder to code, maintain and standardize across implementations. The implicit communication done by the DSVM manager closely parallels object-

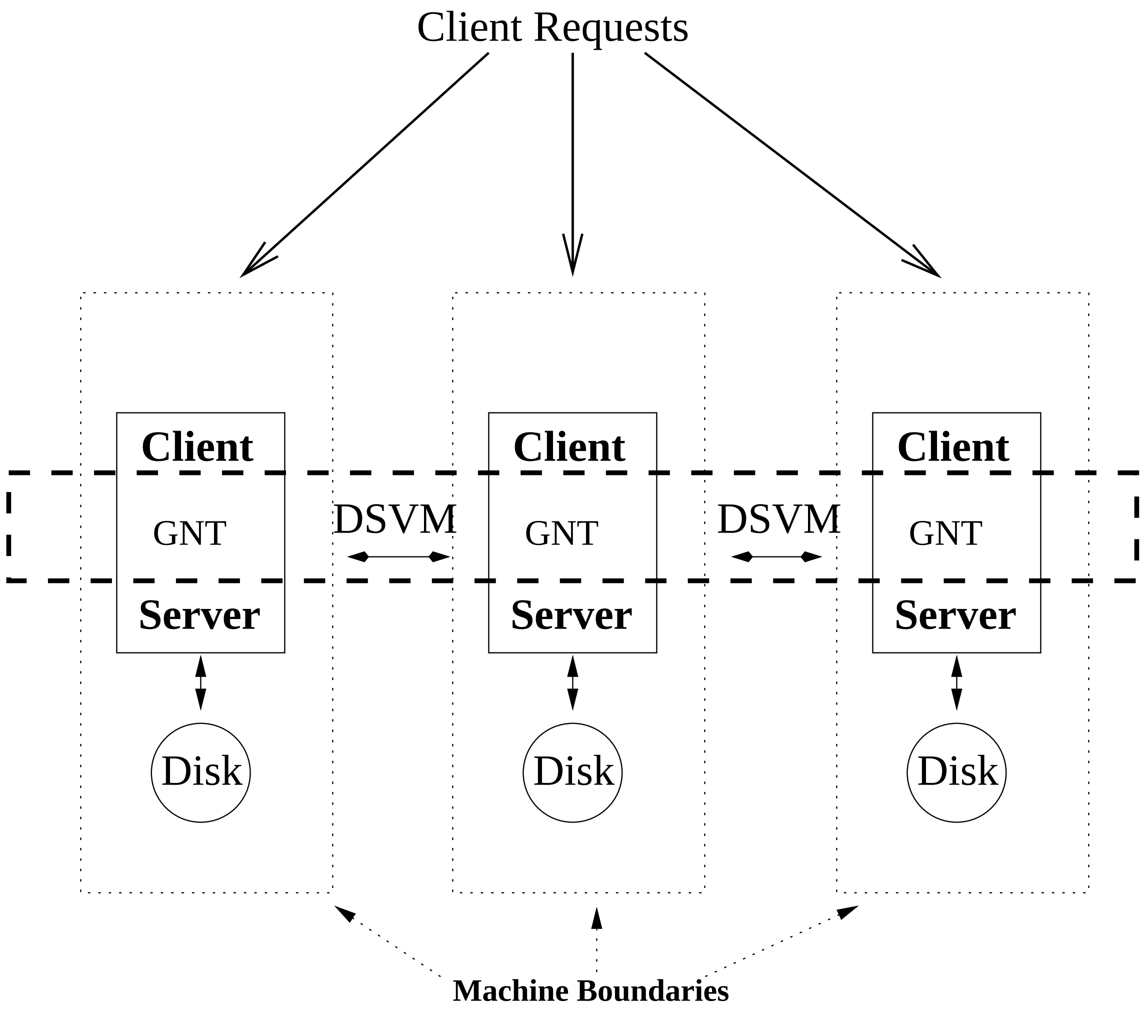

Figure 2: DSVM architecture

oriented principles. The code is easier to maintain and debug as the communication is isolated from the rest of the program. The cost is increased communication and overhead. The increased overhead arises from monitoring memory accesses to determine if they access the shared meomry region.

The second major difference is in the sharing methodology. By its nature, message passing is a pessimistic sharing method as you only share the information that is requested. The non-DSVM version has machines cooperating by passing commands in the form of messages. The processes are not isolated because they must know about the others to communicate (at least the server), but are not in a default sharing situation as each controls its own resources which are only shared by requests through the server.

The DSVM presents a single machine view. All processes appear to be sharing a global memory resource, but in addition each process is given the illusion of isolation as the sharing with others is done implicitly. This parallels closely the idea of a transaction in a database system. DSVM access to the GNT is much like transactions accessing a DB. Individual transactions do not have to worry about other transactions, locking data items, and sharing updates with other transactions. These features are all handled by the DBMS and provide the database system with its increased functionality. Each process accessing DSVM does have to worry about locking, but it doesn't need to know about other processes. Thus, it is under the illusion that it is the only process operating on the GNT much like a transaction is under the illusion that it is the only one operating on the DB.

The final difference is in the amount of replication. The client/server architecture only has one copy of the data while the DSVM version distributes it to all processes. Thus, access to the GNT are more efficient as they can be done locally. Since the number of updates is limited, this distribution scales well in the DSVM case even though all processes much be informed of any change.

5 Benefits of a Global Name Table

One of the main benefits is that all users are presented with a consistent and identical view of the network regardless of their site location. Actually, the network would not be in their "view" as all applications would appear to be local. This substantially enhances user familiarity with the system. The network no longer appears as separate computers who happen to share an application. Instead, it appears as a large computer storing all the applications with several input terminals. Obviously, machines on the network differ in power and storage capabilities, but presenting a consistent view of the network reduces the cognitive load on the user. Currently, icons in windowing environments provide this location transparency. Once an icon is configured, the application's location is unimportant. The GNT provides a way to simultaneous configure all icons or network links during an application install.

The GNT makes applications more movable. Instead of having to reconfigure all icons or links at all stations on the network when an application is moved, only one entry in the GNT needs to be updated and the effect is propagated to all sites. This allows a user to move an application without it effecting the views of other users. It also allows the OS to move an application without effecting the users' view. If used in combination with an ADR table, application execution also becomes site independent. Enforcing the relative path constraint, allows the OS to find the files referenced by an application or user at any site. Other system resources or configurations are site dependent such as display characteristics, sound, and graphics. Hence, to provide application execution transparency, standardized access to these resources must also be provided. This will be considered in future work.

Several beneficial OS functions arise if the application and its execution are site independent. Benefits include load balancing, application shipping and replication, and increased/easier sharing capabilities. The only function directly achievable with a GNT alone is transparent application shipping which allows the OS to reconfigure the application allocation across the network.

6 Implementing the GNT without DSVM

The GNT must be visible at all sites, so it could be implemented by replicating the GNT across all sites and using IPC to propagate updates to all sites. Each site could have a GNT server process which manages the copy of the GNT for its site. This has the advantage of distribution in case of system failure, but also results in a lot of message passing between all the sites whenever an update occurs. Furthermore, the tables are not consistent in the time it takes an update to propagate to all sites.

Instead, a client-server architecture has been implemented with one site being chosen as the server and the others designated as clients. This introduces a single point of failure at the server site, but also greatly reduces the message passing as only one message has to be passed per update. The biggest disadvantage is that every time an application is invoked a network access to the server is required whereas in the distributed case the query would be local. Whenever an application is referenced, its found by name in the GNT. If it is stored at a non-local site, requested files are shipped to the client site. Currently, the GNT is only stored in memory, but any full implementation would store the GNT persistently in case of system failure.

Clearly, the hardest part of implementing the GNT is achieving the required distribution. Each process must do explicit IPC to get the required information. This increases the programming complexity and possibility of errors or inconsistencies in the way the structure is maintained.

6.1 Non-DSVM code explanation

The server process manages the GNT in memory. For simplicity, the GNT is an unsorted array indexed by application name. The server registers a UNIX socket and waits for client requests. A client request can be either add application, remove application, move application, or find application. The GNT does not check for duplicate application names nor proper application locations.

The client connects to the server by a UNIX socket. If it does an add application, it prompts for an application name and uses the machine name that it is running on as a network location. Remove application just removes the application given its name, while move applications moves it to another site. The site name is not checked for validity. The find application request returns the site of the current application.

The queries and responses are formulated as simple records. The server process is capable of handling up to 20 clients at the same time. The server does busy waiting while waiting for queries because all sockets were made non-blocking. This allows each client socket connection to be polled in round-robin fashion by the server process.

The client basically acts like a dumb terminal for requests as every request is sent to the server for processing. More complicated implementations may allow client buffering, but this adds unneeded complexity for this demonstration. The server is very flexible in its form of communication. New clients can connect to it at any time and may connect/disconnect several times. This is a desirable feature in this situation as not all network machines may be running at the same time. Everytime a machine reboots, it could reconnect to the server and issue requests. Thus, only the server need be active at all times.

7 DSVM

In theory, DSVM is a global address space accessible by any number of processes which may reside at different sites across a network. Essentially, it is a form of IPC which implements message passing between processes in a way that is transparent to the application programmer. Instead of passing explicit message across network channels, processes read and write to the global address space, and it is the responsibility of the DSVM manager to insure the information they see is consistent.

Clearly, the main advantage is the distribution transparency. A process implementor does not have to worry about communicating with other processes explicitly. Communication is achieved by shared memory accesses of the same form as regular local memory accesses. DSVM, like any other high-level protocol, does not come without a price. Message passing still must be done by the DSVM manager to maintain the consistency of the global memory, but this is handled separately from the process requiring the DSVM. The advantages of DSVM are similiar to the advantages of object-oriented programming. The implementation is hidden from the user, in this case the intercommunicating processes, allowing for more modifiable code and easier task decomposition.

8 Treadmarks implementation of DSVM

Treadmarks [3, 1] is a commercial implementation of DSVM intended for running parallel algorithms over a network. It was designed as an alternative to message passing systems. It is implemented at the user level, as a C++ library which is linked into the parallel applications requiring DSVM, and requires no modification to the OS kernel.

Treadmarks does not implement the full-functionality of DSVM. Its main limitation is that all processes accessing DSVM must be of the same type because it does a fork() system call to distribute the parent process to the different sites and uses properties of the similiarity of their address spaces' to make the implementation of DSVM more efficient. A DSVM region can be allocated by any process dynamically with a call similiar to malloc() in the C environment. A separate call then distributes a pointer to this memory region to all running processes. This pointer must be declared as a global variable because Treadmarks copies it directly into the same location into each of the other processes address spaces. By unmapping the shared memory region from a processes address space, the DSVM manager of Treadmarks can capture accesses to the region and propagate changes to the other sites. This unmapping causes the process to generate SEGV signals when accessing DSVM, limiting its usefulness in system-programming where system calls are interrupted by signals. Treadmarks provides exclusive locks and barriers as synchronization primitives so access to the DSVM region can be controlled explicitly by the programmer.

The DSVM management protocol, lazy release consistency, is defined in terms of these synchronization primitives. A barrier stops all processes at the barrier point until all processes reach that point. Exclusive locks are not associated with any data item in DSVM in particular but are used to guarantee that data access in the DSVM region is consistent. An acquire primitive is either when a process acquires an exclusive lock or it leaves a barrier. The local memory at the process must reflect global updates at that point. That is, before a processor can continue after an acquire, all accesses to the DSVM region that preceeded the acquire must be reflected at this processor.

There are several mechanisms for maintaining the consistency of DSVM. Sequential consistency is the method used by most parallel machines where updates are reflected at all sites immediately after they are performed. This type of consistency is undesirable in the network environment due to the high amount of unnecessary communication. The alternative is release consistency which guarantees that shared memory is consistent only after synchronization primitives occur. That is, broadcasting of the updates is delayed until a release or an acquire. In eager release consistency, the updates are sent to all sites when the lock is released at one site. This involves too much overhead as the other sites may never need to know about the update. In lazy release consistency, the updates are propagated only to the site doing the acquire as that is the only site that needs to know the changes at this point. Release consistency is equivalent to sequential consistency if all synchronization operations are done using system-supplied primitives, and there is a release-acquire pair between conflicting accesses to the shared memory at different locations. Treadmarks enforces this constraint by requiring the programmer use its synchronization primitives to guarantee that the shared memory is consistent.

The second major optimization Treadmarks does is allowing multiple-writers. This is to reduce the amount of false sharing as the DSVM region is broken (and guaranteed consistent) by pages which is often a much larger granularity than objects in the DSVM region. Basically, when a processor first updates a page, the page is duplicated at that processor and all following updates to the page are performed on the copy. When a synchronization primitive occurs, the page is compared to the original to form a "diff" and the "diffs" are sent to other sites. The other sites then merge the "diffs" to form the new updated pages. Besides allowing increased concurrency the size of the "diffs" is often smaller than a page which reduces the amount of transmitted data. If two "diffs" access the same memory region on the same page, then a data race occurs, and the result is timing dependent. If this is undesirable, use the synchronization primitives (ie. locks) to insure that this does not occur.

Although Treadmarks was designed to be used in coding parallel algorithms, it is still useful for prototype demonstrations like this one. Its usefulness for system programming is limited by its use of signals and the requirement that all processes be homogeneous. A more general DSVM implementation would be needed for any practical project.

9 Implementing the GNT using DSVM

The GNT structure is still an unordered array of application names and their locations, but in this case it is dynamically allocated in DSVM. This guarantees that the structure is seen by all processes at all sites. Since the GNT is now distributed, concurrency control mechanisms must be applied to insure that the data structure is manipulated correctly. Thus, the table is locked whenever an access is done. Treadmarks does not provide shared locks, so only one process is allowed to access the table at a time.

The main limitation of the implementation is that all clients must be forked at startup, but this is a limitation of the Treadmarks package itself. Also, process failures after lock acquire are not handled which would be vital in a production system. The simplicity in the code is evident, as except for the locks, each process thinks that it is operating on its own copy of the GNT and does not have to worry about what the processes at other sites are doing.

9.1 DSVM code

Treadmarks requires a file specifying machine names on the network where processes will be forked. When the parent process is started, Treadmarks forks processes to the designated sites. All processes stop at the barrier until the parent process can create and initialize the GNT structure and "distribute it" to the other sites. Once the barrier is passed all processes are identical. There are no distinguishable client or server processes because each process functions as both a server and a client. On termination, the parent process is responsible for deallocating the GNT.

10 DSVM vs. Non-DSVM Implementation

The obvious advantage of the DSVM implementation is its simplicity. DSVM access is virtually identical to local memory access making it easier to code and debug than explicit IPC mechanisms like sockets, message queues, or UNIX shared memory. The migration to DSVM parallels the trend in object-oriented programming. DSVM is an abstraction of IPC principles into a simplified interface (object methods) provided transparently (object encapsulation) to the application implementor.

Object encapsulation is not free. At its lowest level, DSVM still must be implemented as a form of message passing, but abstracting the IPC into a DSVM interface allows for the implementation of DSVM to be improved without effecting applications in which it is used. This is similiar to the implementation hiding which object-orientation provides. It is easier and more practical to improve a single DSVM algorithm than to optimize code based on lower-level IPC.

A non-DSVM implementation can be more efficient than a DSVM implementation if coded intelligently, but much like the debate over using C instead of C++ for efficiency, devising such an implementation may prove more work than it is worth. It is time to stop reinventing the wheel everytime a new application or protocol is designed and start accepting higher-level protocols which EFFICIENTLY implement lower-level constructs. Future work on the GNT and the ADR are a push in this direction. More efficient and powerful systems can result by optimizing a layered-protocol system and abandoning the idea that an application must do everything in the system for sake of efficiency.

In this case, the non-DSVM implementation is less efficient than the DSVM implementation in terms of server performance. In the non-DSVM case, the server monitors numerous sockets at a time forcing it to continually poll existing sockets for queries and check for any new communication from new clients. This results in a lot of busy waiting. The clients are efficient because they can block on their one communication socket. On the positive side, communication is minimal, but this is mostly because of the limited distribution of data. If the GNT was distributed across all sites like the DSVM case, then each process would need at least 2 sockets connecting to the others and practically would have to be connected to all sites. This would result in message passing between all processes for every update, and all processes would have to perform busy waiting as blocking cannot be done on more than 1 socket.

11 Conclusions

DSVM is a higher level IPC protocol in the same vein that an object is a higher level data structure. It provides an easier programming interface and a mechanism for IPC standardization that essentially provides network distribution transparently to the application designer. As shown by a simple implementation of the GNT, the benefits to the programmer are very similiar to the benefits of object-oriented design: resource encapsulation, standardized interface, and implementation hiding.

The GNT in its own right is essentially a higher-level OS function providing transparent network file access to the user. Just as DSVM does not provide anything that existing IPC mechanisms cannot, the GNT does not provide the user with any more functionality than icons or aliases in windowing systems, but it is a simpler method to accomplish the same task. The GNT is a mechanism which abstracts the file naming problem into a simple interface which could be readily standardized to provide a file access mechanism independent of OS implementation and optimized as a separate entity to make file allocation more transparent to all aspects of the system.

12 Future Work

The GNT, as described, provides just the shell of functionality required for any real system. The operations, access functions, and data stored in the GNT must be refined to make it usable. By simultaneously defining the ADR as a method for abstracting application execution from network location, a truly dynamic network may result which allows application files and executing processes to be freely migrated throughout the network by the OS to improve system efficiency and utilization.

References

[1] Cristiana Amza, Alan L. Cox, Sandhya Dwarkada, , Peter Keleher, Honghui Lu, Ramakrishnan Raja-

mony, Weimin Yu, and Willy Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. Technical report, Rice University, 1994.

[2] Kenneth P. Birman, Robert Cooper, Keith Marzullo, and Mark D. Wood. Tools for distributed application management. Technical Report TR90-1136, Cornell University, Computer Science Department, July 1990.

[3] Pete Keleher, Alan L. Cox, Sandhya Dwarkada, and Willy Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. Technical report, Rice University, 1994.

[4] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network file system. Technical report, Sun Microsystems, Inc., 1985.

[5] Alexander Siegel, Kenneth P. Birman, and Keith Marzullo. Deceit: A flexible distributed file system. Technical Report TR89-1042, Cornell University, Computer Science Department, November 1989.

[6] Andrew S. Tanenbaum and Sape Mullender. An overview of the Amoeba distributed operating system. *ACM*, 15(3):51–64, July 1981.

[7] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS distributed operating system. In *Proceedings of the Ninth ACM Symposium on Operating System Principles*, pages 49–70. ACM, October 1983.