Reduction of Software Development Costs under the Sombrero Distributed Single Address Space Operating System

Ron Feigen Alan Skousen Donald Miller
Computer Science and Engineering Department
Arizona State University, Tempe, Arizona 85287, USA
E-mail: r.feigen@motorola.com, {alan.skousen,donald.miller}@asu.edu
Telephone: (480)-965-5935 Fax: (480)-965-2751
Presenter: Ron Feigen

Keywords: single-address-space operating systems, software complexity, distributed operating systems, distributed shared memory, persistent systems

ABSTRACT

Sombrero is a distributed single address space operating system (SASOS). DSM and persistence are part of the fundamental nature of a distributed SASOS because the common virtual address space is distributed, shared and persistent. We believe that the major reason for using a SASOS is that many useful applications would be less complex if written to run on a SASOS rather than under a conventional process-oriented multiple address space operating system (MASOS). This paper examines the potential for reduction in software complexity of applications developed under the Single Address Space Operating System Sombrero versus those developed under a conventional process-oriented operating system, Windows 2000. To test Sombrero's impact on development costs, a set of sample database applications with varying functionality was developed under Windows 2000 and Sombrero. Using accepted software engineering metrics, the software complexity of these systems was compared. The results obtained indicate a substantial reduction of software complexity and software development costs in a single address space environment such as that provided by Sombrero. Reasons for this based on the code required to perform the same tasks in the Sombrero and Windows 2000 environments are presented.

Introduction

Sombrero [1-5] is a Distributed Single Address Space Operating System (SASOS) that, among other things, provides distributed persistent shared memory. The premise of the work presented in this paper is that the hardware/software architecture of Sombrero reduces the complexity of application software and hence the cost of software development. We believe that the powerful system abstractions it provides developers make many system implementation details transparent to them and allow programmers to concentrate on their applications. For example, in [5] we compared a simple client/server file copy program written for a process-oriented Multiple Address Space Operating System (MASOS) with the same program written under Sombrero. The SASOS file copy was three lines of codes plus a few definitions while the MASOS version contained dozens of lines of code and in addition referenced many more lines of file access and manipulation code via function calls. These also would not be in the SASOS version. Because of this and numerous other examples we decided to see if we could quantify the reduction in software complexity in a SASOS environment and parse its causes. To do this, we coded what we felt was a representative application, a simple client/server database system under Windows 2000 and Sombrero. We coded four increasingly functional versions of the application, starting with a single client and server, then adding multiple clients, next adding multithreading to the server and finally, adding multiple servers with replication and load balancing. We ran the first version on a single PC system and the rest on a 2 PC system to check for correct operation. We collected software complexity data using industry standard metrics: Lines of Code, Halstead (vocabulary) and McCabe (complexity) for each version under each operating system. The factors of 2:1 and 3:1 reduction in complexity (see Table 1) – depending on the metric and application version - were much larger than we anticipated and very large by software engineering standards. We examine these results and give some reasons for them. The bulk of the paper goes through these steps. At the end we describe the current state of the Sombrero prototype and pull together our observations. More information on the Sombrero project can be found at our web site [8].

Database Systems used as Sample Applications

Application used to Evaluate Software Complexity

A simple Database System (DBS) was selected to evaluate software complexity because it embodies significant and complex system services and process interactions. These include: management of large amounts of data, data persistence, message passing between processes, distributed processing and load balancing. In addition it can be identified as an application that parallels what is considered to be commercially viable. Database management systems are good examples because, typically, they are distributed applications that rely on many system services including: memory mapping of large files, message passing, data persistence and multithreading. A commercial database also requires process distribution, replication and load balancing.

The DBS developed for this project is not a commercial grade application. It consists of several thousand lines of C++ (as contrasted with the tens or hundreds of thousands of lines of code in a commercial product). However our evaluation DBS can manage very large datasets (limited only by disk size, not available virtual memory) and provides the following primitive functionality:

Create new database; open; close; search (key, OID, Iterate); insert; delete; and replace. Details on design, implementation and usage of the database class are provided in [6].

The sample database was developed in four phases, with each stage of development adding additional features, such as client-server architecture, or multi-threading, to the base set of database primitives outlined above. For Windows 2000, a fifth version of the database was developed that explores how adding distributed shared memory would effect development in a MASOS environment. The distributed shared memory version of the code was simulated, but meaningful metrics were captured.

Description of the Four Phases of Functionality

The four levels of functionality describe architectural characteristics of the database (i.e. data distribution or load balancing) that are implemented in addition to the basic database primitives described in above. We describe them from the more familiar Windows 2000 architecture point of view.

Phase 1, the base application, is a database class, whose methods act on a local dataset, and are called directly by a driver application, as depicted in Figure 1. The database class and the driver application run in the same thread of execution. The class does not support multiple instantiations of the database class accessing the same dataset.

Phase 2 of the application requirements adds a multiple client-server paradigm to the Phase 1 development as shown in Figure 2.

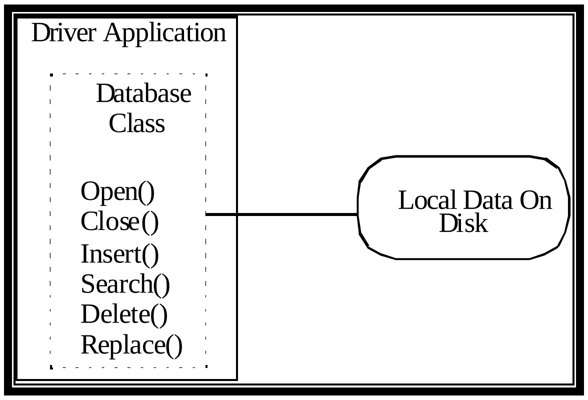

Figure 1. Phase 1 Functionality – Simple DBS

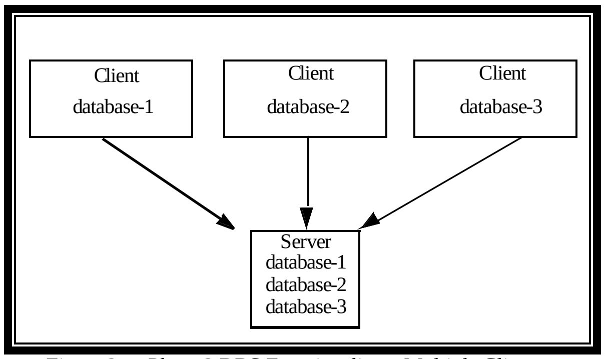

Figure 2. Phase 2 DBS Functionality – Multiple Clients

The Phase 2 database server class has the ability to handle requests from multiple clients directed to the same database or different databases. This implies that multiple clients can access the same database, requiring that the server must maintain state information per client and that open and close may be affected if multiple clients can open and close the same dataset.

The Phase 3 iteration of development, as shown in Figure 3, adds a multithreading capability to the server class, requiring that the server manage critical sections.

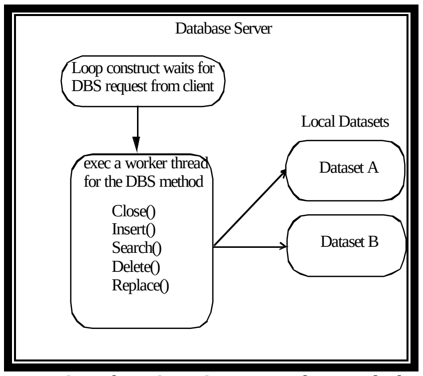
Database Server
Loop construct waits for DBS request from client
Local Datasets
exec a worker thread for the DBS method
Close()
Insert()
Search()
Delete()
Replace()
Dataset A
Dataset B

Figure 3. Phase 3 DBS Functionality -Multithreading

The algorithm selected to manage critical sections permits either simultaneous readers or a single writer. A thread wishing to update (write) the database must wait for all the readers to exit before proceeding and no readers or writers can enter a critical section if a writer is writing.

Phase 4, seen in Figures 4 and 5 adds the following functionality: a load-balancing algorithm for read operations; data distribution and operations that must ensure data consistency when the dataset is modified.

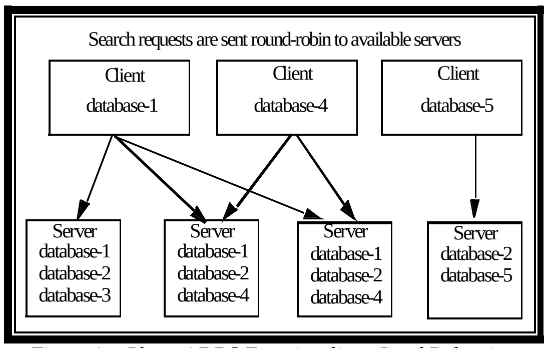
Search requests are sent round-robin to available servers
Client database-1
Client database-4
Client database-5
Server database-1 database-2 database-3
Server database-1 database-2 database-4
Server database-1 database-2 database-4
Server database-2 database-5

Figure 4. Phase 4 DBS Functionality – Load Balancing

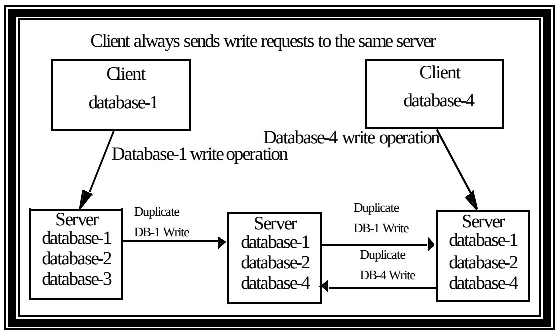
Client always sends write requests to the same server
Client database-1
Client database-4
Database-4 write operation
Database-1 write operation
Duplicate DB-1 Write
Duplicate DB-1 Write
Duplicate DB-4 Write
Server database-1 database-2 database-3
Server database-1 database-2 database-4
Server database-1 database-2 database-4

Figure 5 Phase 4 DBS Functionality - Replication

Software Complexity Metrics

In order to improve software methods that reduce costs incurred in the software lifecycle, software engineers have developed metrics to quantify the complexity of a software design and to predict development costs and project quality (in terms of defects). We used 3 such metrics: Lines of Code (size); Halstead (vocabulary) and McCabe (complexity). Lines of Code (LOC) is a sheer volume metric that can include comments. In addition we used the Basic Constructive Cost Model (Basic COCOMO). It uses LOC plus two factors related to the nature of the design team and the tightness of their constraints. Halstead's Software Science metric measures a program's difficulty in terms of its vocabulary based on the number of distinct operators and operands and the total number of these. Based on this raw data, Halstead's metrics estimate 'program effort' and 'predict bugs'. McCabe's Cyclomatic Complexity metric measures complexity in terms of the decision structure of a program. Essentially it uses a flow graph to determine the number of decision statements in a program.

Native Sombrero Services used to Implement the Simple DBS Application

A Sombrero application does not typically need to deal with issues such as message passing, data migration, replication, and distribution. In many cases multithreading is abstracted to the point where the application must protect critical sections but not deal with the semantics of creating and managing worker threads. The native Sombrero services are currently being developed. The basic set of system services are described in [7] but the syntax and semantics for each command is not finalized. In this project we approximated the interface and simulated its implementation. We do not believe that this will significantly affect the metrics. We describe the three Sombrero system service directives that accomplish most of the work.

Memory allocation primitives

Creating persistent storage in Sombrero is similar to allocating virtual memory from the heap in a MASOS. Because the memory allocated is persistent, the allocation operation must allow for additional semantics that parallel *fopen()* in traditional systems. Memory is allocated with the allocate primitive:

char *allocate(name, size, sharing, mode, replicate, migration)

Name: A text-based alias for identifying the base address of the persistent virtual memory, analogous to a filename in a process-oriented system.

Size:	Size in bytes
Sharing:	Protection and sharing attributes: rwx, share/noshare/share with group
Mode:	Attach to previously allocated virtual memory or Allocate new virtual memory
Replicate:	None, default, on all nodes, on *n* nodes, on a specific set of nodes
Migration:	None, all nodes, *n* nodes, on a specific set of nodes

Object Creation

Objects are created in Sombrero using the new() directive.

new (class, object)

creates an instantiation of an object of the type class. Note: this object is not animated, as it has no thread of execution. This is how Sombrero creates passive servers.

Resource Control –Root Memory Object Control Block

The root memory object control block provides an application with a directory of all resources available on the Sombrero network. In addition, the application can access root memory control methods to direct the Sombrero scheduler to run a thread of execution on a specified subset of the available resources. This is similar to modifying the scheduling policies of a symmetric multiprocessing system. As an example, a calling application can modify the scheduler to force a thread to execute on the local node by passing the constant THIS for the list of servers:

err = rmcb(schedule, <thread id>, THIS);

The list of servers specifies to the scheduler to which server it can migrate the thread. The default is that the scheduler can migrate the thread to any server in the Sombrero network.

Test Results

Table 1 provides the metrics captured for each phase of Windows and Sombrero Development permitting comparisons across and within an environment. The results show that applications developed under Sombrero compared to applications developed under Windows 2000 are: less complex (McCabe); require less effort to develop (Halstead, LOC, COCOMO) and less likely to have defects (Halstead).

Table 1. Metrics Captured From Sample Database Application

	Halstead Program Effort	Halstead Bug Prediction	McCabe	LOC	Basic COCOMO
Windows					
Phase 1	2,637,658	6.36	85	852	2.03
Phase 2	4,287,457	8.80	121	1348	3.28
Phase 3	4,420,421	8.98	127	1420	3.46
Phase 4	6,204,183	11.26	145	1612	3.96
DSM[1]	4,987,458	9.73	132	1481	3.62
Sombrero					
Phase 1	1,025,712	3.39	67	639	1.50
Phase 2	1,084,757	3.52	65	647	1.52
Phase 3	1,135,836	3.63	68	719	1.70
Phase 4	1,135,836	3.63	68	719	1.70

1.This is the sample application under Windows 2000 with distributed shared memory.

This is true for all phases of development and increases as the application feature set requirements increase. By Software Engineering standards, these advantages, (2:1 to 3:1 depending on the metric) – these improvements are very large. In the following section we present the reasons for the sharply lower software complexity in Sombrero. Appendix I briefly describes the test environment and mechanism to gather metrics.

Comparison of Sombrero and Windows 2000 Programming Required for Sample DBS

In overview, the reduction in programming complexity in Sombrero was derived from the way it provides services in four areas: mapped memory management; messaging and IPC; thread management and control transfer; data distribution and load balancing. To a rough approximation the relative magnitude of each factor can be estimated by observing at which phase of development the needed functionality was introduced. Mapped memory management is present at the start; messaging begins in phase 2; multithreading is introduced in phase 3 and data distribution and load balancing are introduced in phase 4.

Mapped Memory Management

Efficient manipulation of datasets like an array of objects (database) or sorted arrays of key/indices pairs requires direct, random access to the data. Thus in a process-oriented operating system, the sample database application maps the user database and associated indices into the process' virtual address (VA) space. For large data sets, the data being manipulated can easily exceed the available VA space. To resolve this under Windows 2000 requires developing a memory mapping page handler to ensure the proper pages from disk are mapped into virtual memory. It provides a windowing mechanism to map subsets of the object's namespace into a process's address space and to return an offset the application can use for swizzling pointers into that mapping of memory. Moreover, this page handler is not transparent to the programmer like a memory management unit. The application must call the routine prior to using any pointer into mapped memory. In Sombrero the VA space is persistent and includes the backing store of the local node and all peer nodes. Additional mapping mechanisms are unnecessary. Pointers remain constant and swizzling is unnecessary.

Messaging and IPC

To support communication between processes, as needed from Phase 2 on in a process-oriented operating system, the sample DBS must pass messages in order to move data from one namespace to another. In Windows 2000 this required the creation of a communication class to handle creating TCP/IP client and server ports, establishing IPC, message read, message write and close connection. In addition the reading and writing operations require the sending and receiving applications to agree on a message format. Mechanisms to pack and unpack messages and marshal and unmarshall arguments must be written. As TCP/IP is a streams protocol, the read method must make sure it gets all the bytes before it returns to the calling routine. This requires reading incoming data in a looping construct and building a message buffer. Furthermore, as is typical in client/server systems a connection and binding mechanism is needed. In Sombrero all processes share the same VA space and there is no need to pass messages. Either one process signals the other that data is ready, or as discussed in the next section, the client accesses the server's methods, in a manner not unlike a local procedure call.

Thread Management, Control Transfer and Passive Servers

The underlying Sombrero single address space, hardware protection mechanism and non-hierarchical software architecture facilitate the implementation of passive server objects. These are class instantiations that persist but do not have their own threads of execution. The client supplies a thread to a passive server when it accesses a server method. The implementation enables the client to access the server via an ordinary procedure call. In terms of control transfer, the client thread migrates to the passive server and 'animates' it. The client may or may not reside on the same node as the server and any required migration of service or thread across nodes is transparent to the developer and the application. Passive servers, as mentioned above are one mechanism used in Sombrero to eliminate message passing. Windows 2000 from Phase 3 (multithreading) on requires a thread dispatcher in the server to create worker threads. Sombrero does not need this.

Data Distribution and Load Balancing

Load balancing in the Windows 2000 DBS is implemented in the client. It distributes its read requests, round robin, to a predefined pool of available servers. Data consistency under replication is implemented in the DBS server. Each server forwards open, close and write requests received from a client to a single neighbor server, which, in turn forwards the request to its neighbor, etc., resulting in all operations that modify the state of the database being executed on all copies of the database. By default, all applications are distributed in Sombrero. The Sombrero scheduler will, unless instructed to do otherwise, consider all resources in the network when determining where a thread of execution should run. It will transparently migrate data (i.e., replicate) the data to the node running the thread or migrate the thread to the node with the data, based on its knowledge of current system state. Scheduling, data consistency and load balancing are transparent to the application and developer.

Examining the LOC metric gives some insight into how the above factors affect software complexity. In Windows 2000 each additional feature has associated lines of code. For example, phase 2 requires a client/server paradigm and a Windows 2000 communication class is created to provide IPC methods for: creating a client, creating a server, reading and writing messages. Under Sombrero, there is an increase in LOC for phases 1 through 3, but the change is only minor. There is no change in LOC between phases 3 and 4. The reason for this is because in most cases, the feature requirements that mapped to additional code required by a Windows 2000 programmer are, under Sombrero, a natural extension of the operating system. Thus, many of the feature requirements trace to Sombrero not the application, so no sample application code is required. For example the only difference between Sombrero Phases 1, 2, and 3 are how locks are managed. In Phase 1 a single lock is used to grant exclusive access to the passive server. In Phase 2 locks are used to serialize access to passive server methods. In Phase 3 locks are used to protect critical sections. The Sombrero operating system satisfies the other requirements. For example, in essence, Phase 3 and 4 have the exact same code base. The only difference is how the database virtual memory is allocated. The differing feature requirements between phases 3 and 4 are implemented without adding additional code, but rather by passing different parameters to the Sombrero memory allocation primitive. Basic COCOMO, being a tuned LOC metric, tracks LOC. Halstead’s metrics track the LOC measurements because, as additional features are added, the total number of operators, operands and unique operators and operands increases. McCabe’s complexity matrix indicates that in Sombrero, restricting the application to run on a single node, distributing the data to all or a set of nodes required few if any additional logical paths through the application. This was not true for the Windows 2000 DBS.

Test Environment and Configurations

All phases of the sample database, for both Windows 2000 and Sombrero, were developed using Microsoft Visual C++ version 6.0. The development platform was a dual 550MZ Pentium III-based CPU, running Windows 2000 in 2-Way SMP mode. The sample application was tested on a network consisting of the development platform and another Intel based Pentium III platform running Windows 2000. The processors were connected with an Ethernet link.

Generating the metric data was a two-step process repeated for each phase of development. For each phase of the project, a single file for all of the code in that phase was created. Then the metrics tool was applied to the single project file to generate metrics for the given phase. The metrics tool we used was Krakatau Professional (C++) Metrics for which Power Software Limited was kind enough to provide us an evaluation license.

The software and hardware configuration for Phase 1 of the application runs on a single PC. The Database class is linked directly into the single threaded driver application. There is a one-to-one relationship between the client/driver application and the dataset. Phase 2 of the application provides a single threaded database server that can accommodate multiple database clients running on a separate PC. Each Database client class is linked to a single threaded driver application. The client class provides access to the database server via its public methods. Database server is a stand-alone single threaded application that listens for requests from multiple clients. This allows multiple clients to access the same data. Physically the clients can be on either of the 2 test PCs and the server is on one of them. The configuration for testing phase 3 was identical to that of phase 2. Phase 3 adds multithreading to the database server. There are no changes to the database client. Phase 4 adds data distribution to the phase 3 database server and load balancing to the phase 3 database client class. The test configuration for phase 4 is similar to that of Phases 2 and 3 except that there are two servers, one on each PC. Correct execution was obtained in all cases for each of the development phases described. Details can be found in [6].

Sombrero Prototype

The obvious disadvantage of the Sombrero approach is that it premises changes to protection hardware, a different arrangement of memory management hardware and an additional small amount of hardware features not present in contemporary CPUs. CPU manufacturers are resistant to this. Roughly a 50% performance improvement or equivalent in cost reduction needs to be shown before they will introduce changes of this magnitude in the core of a processor. Sombrero's needed hardware can be simulated, as we are doing in Alpha PALCode and low-level kernel code. The Sombrero project is currently being prototyped on four sets of two or three networked Alpha 21164 boxes with sufficient resources to compile and run Windows 2000 source code. The development environments are a combination of Red Hat Linux 7.1 and Windows 2000. One box in each set can be booted under Windows 2000, Linux or Sombrero as needed. Three of these research systems are located in the homes of the developers and one in the OS laboratory at ASU. We are currently coding the low-level protection, memory management and thread scheduling mechanisms and finalizing service class instantiations and their interfaces. In addition persistence via a SCSI disk controller and disk drive and associated mapping/paging are being written. The prototype is expected to be operational by June 2002. In our view, only proof of concept, via an operational prototype, will be sufficient to convince chip manufacturers to invest in a chip with its base hardware targeted at single address space operation.

Conclusions

In overview, Sombrero is able to provide simplified access to complex services. Its persistent single VA space and thread and data migration allow additional powerful abstractions that simplify the development environment. Because Sombrero itself manages thread scheduling and data consistency and can migrate threads and data transparently, the application is freed of unwanted load balancing, distribution/replication and fault tolerance issues. Message passing is unnecessary – a passive server can supply services via a local subroutine call. The single persistent name space means no translations between namespaces are necessary – no pointer swizzling or memory mapped data operations are needed, as data is always available at the same address. Sombrero's object-grained protection provides native support for OOD and OOP. These features combine to reduce the complexity of software needed to perform many tasks, which in turn reduces the costs of their software development.

References

[1] Skousen, A. and Miller, D., "Operating System Structure and Processor Architecture for a Large Distributed Single Address Space", *Proceedings of PDCS'98: 10th International Conference on Parallel and Distributed Computing Systems*, October 1998.

[2] Skousen, A., and Miller, D., "Using a Distributed Single Address Space Operating System to Support Modern Cluster Computing", Proceedings of *32nd Hawaii International Conference on System Sciences, HICSS-32*, January 1999.

[3] Skousen, A., and Miller, D., "Using a Single Address Space Operating System for Distributed Computing and High Performance", *Proceedings of 18th IEEE International Performance, Computing, and Communications Conference*, February 1999.

[4] Skousen, A. and Miller, D., "The Sombrero Distributed Single Address Space Operating System Project", *2nd USENIX Windows NT Symposium*, August 1998, p. 168.

[5] Skousen, A and Miller, D., "The Sombrero Single Address Space Operating System Prototype, a Testbed for Evaluating Distributed Persistent System Concepts and Implementation", *Proceedings of PDPTA'2000: The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2000.

[6] Feigen, Ron, "Reduction of Software Development Costs under Sombrero, A Single Address Space Distributed Operating System", Computer Science and Engineering Department, Arizona State University, *MS Thesis*, December 2001.

[7] Carnes, Mark, "Sombrero System Interface", Computer Science and Engineering Department, Arizona State University, *MCS Project Report*, December 1996.

[8] Sombrero Web Site http://www.eas.asu.edu/~sasos