

Distributed Scheduling for the Sombrero Single Address Space Distributed Operating System

Donald S. Miller
Department of Computer Science
and Engineering
Arizona State University
Tempe, AZ, USA

Alan C. Skousen
Department of Computer Science
and Engineering
Arizona State University
Tempe, AZ, USA

Milind Patil
Department of Computer Science
and Engineering
Arizona State University
Tempe, AZ, USA

Keywords: distributed operating systems, scheduling, single address space operating systems, operating systems, load balancing

Abstract - *This paper presents a distributed scheduling algorithm for the Sombrero single address space operating system. This algorithm uses the properties of a single address space and the Sombrero support for thread migration to modify and extend scheduling algorithms developed for multiple address space operating systems to the single address space environment. Threads in the distributed system are scheduled among the different nodes so that CPU usage is balanced. A dynamic local-information-based distributed scheduling algorithm is designed, implemented on a simulation and evaluated. The simulation includes nodes, routers, load tables, and use of them to demonstrate that the distributed scheduling algorithm is correct, works in a single address space distributed shared memory environment and scales in a constant manner with respect to the number of messages and tables required for load balancing. A hierarchical clustering mechanism is used to make the algorithm scale well. The paper also presents, and the simulation includes, the Sombrero fission/merge mechanism for distributing the virtual address space across multiple nodes and the Sombrero use of virtual addresses as an inter-node addressing mechanism.*

1. Introduction and Motivation

Sombrero [1] [2] [3] [4] is a very large single address space operating system that is intended to be extensible to a multiple node system consisting of a set of homogeneous workstations, servers and LANs. In a process-oriented operating system, the process boundary inhibits processes from addressing data or executing code in another processes' virtual address (VA) space. However, in a single address space

operating system, neither threads nor programs nor the access rights associated with these programs are bound by a reused VA space. Only a single address space exists and the VAs have the same meaning anywhere on a local area network. This leads to important reductions in the complexity of application and system software and consequent reductions in the costs of software development [5]. Sombrero is designed to have the appearance of a single large multithreaded process distributed over many nodes. This paper addresses the issue of thread scheduling over multiple nodes in the system.

Distributed scheduling in Sombrero takes advantage of the properties of a single address space and Sombrero support for thread and data migration in a single address space. This can provide considerable performance gains as opposed to process-oriented systems. In process-oriented systems threads are encapsulated within a process and true protection domain crossing and machine migration without an address space and thread context switch is essentially impossible. Threads are bound in a single environment including VA space, memory objects, protection domain and resources reachable by the process such as files. In a SASOS, objects are bound permanently to VAs that provide a system-wide available namespace. The thread control block, stack, other private static data, thread ID, and other thread state, e.g., run status and priority, retain their VAs irrespective of their physical locations. This means a thread can be suspended on a particular computer node and can be restarted with the same context on some other node. General register contents can be loaded into the CPU general registers on the destination node. Through the caching effect, any missing data can migrate to the thread as well. In summary, a SASOS cannot only cache data across the network, but it can also migrate execution across the system in a straightforward manner. The ability of threads to migrate in a simple manner across machines has a potentially far-reaching

affect on the design and performance of a distributed scheduling mechanism.

2. Distributed Scheduling Overview

One of the techniques for scheduling processes of a distributed system is the transparent load-distribution approach. The aim of this technique is to transparently equalize workload amongst the nodes to maximize total system throughput. Transparency can only be achieved by using dynamic load distribution techniques, which operate with absolutely no a priori knowledge of the characteristics and resource requirements of the executing threads. This user transparent activity provides a greater amount of processing power to all users of the system. Any good scheduling algorithm should be dynamic in nature, have a quick decision-making capability, strike a good balance between system performance and scheduling overhead and also be stable, scale well and be fault-tolerant. [6].

Distributed scheduling techniques for load-distribution are either *global-data-based* or *local-data-based* based on the information they gather to make the scheduling decisions. In global schemes, the state information is collected from all nodes in the system. The state information could be the load for each node or the performance profile of each node. In local schemes, the nodes are organized to form groups of particular sizes. In this case the state information exchange and the load distribution is limited within each group.

Alternatively, distributed scheduling techniques for load-distribution is either *centralized* or *distributed* depending on whether the scheduler is located at one master node or if the scheduler is distributed among the nodes, respectively [7]. So there are four possible strategies: global centralized scheduling, global distributed scheduling, local centralized scheduling and local distributed scheduling

The centralized approach to scheduling relies heavily on the performance of the centralized scheduler. If the node that hosts the scheduler fails then all scheduling in the system will come to a halt. In addition, this centralized server can cause a bottleneck for collecting state information and distributing scheduling decisions in the system. In contrast to the centralized scheme, a distributed scheme involves all nodes in the system in making scheduling decisions. Since a centralized server is not solely responsible for making all scheduling decisions, the scheme becomes fault-tolerant. However, the global scheduling algorithms are inherently more complex. The state information now has to be made available to all nodes in the system.

Depending on the type of node that initiates a global search for a suitable node for the migration of a process, the distributed scheduling schemes can be *sender initiated*, *receiver initiated* or *dynamic* [6]. In a sender-initiated scheme the sender node of the process decides where to send the process. In this scheme, heavily loaded nodes search for lightly loaded nodes to which work may be transferred (Figure 1a). This scheme works well when the system load is low, but as the number of heavily loaded nodes increases its performance decreases due to the large number of messages being passed around. In a receiver initiated scheme the receiver node of the process decides where to get a process from. In a receiver-initiated scheme, lightly loaded nodes search for heavily loaded nodes from which work may be transferred (Figure 1b). This scheme performs well when the system load is high, but its performance decreases if the number of lightly loaded nodes increases due to the large number of messages being passed around. For dynamic schemes at low system loads the sender-initiated scheme is employed while at high system loads the receiver-initiated scheme is used.

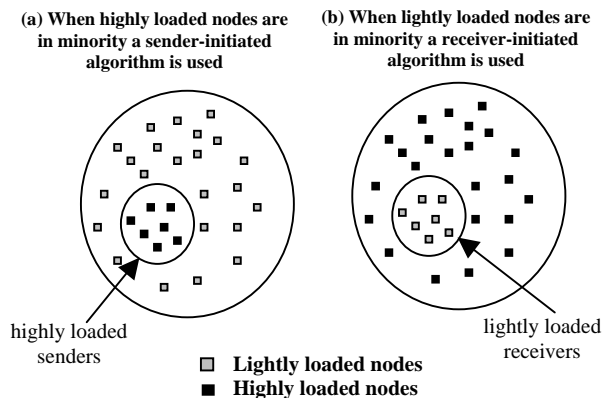


Figure 1. Load Balancing Algorithms

3. Sombrero Distributed Scheduling Design

Sombrero is an operating system designed to be extensible to large LANs. On a large LAN, the overhead of distributing state information to all nodes in the system can be very great. Hence, it seems intuitive that a local-data-based scheduling scheme should be adopted to perform thread scheduling for Sombrero so that it scales well. For concurrency and fault tolerance a distributed scheduling technique for load balancing should be used. Finally, for good performance at both low and high system loads the scheduling algorithm should be able to switch between sender and receiver initiated algorithms. For these reasons a dynamic local-data-based distributed scheduling scheme has been selected for Sombrero.

In process-oriented systems, processes being in different virtual address spaces are, by definition, in different protection domains. Hence inter-process data exchange and synchronization involves cross-domain communication, which leads to considerable overhead as, for example, in a remote procedure call. This is because addresses in a process do not have the same meaning across the process boundaries. Inter-boundary data access and procedure calls require stubs to perform a software communication protocol that checks access rights to entry points, performs marshalling/unmarshalling of parameters/results, etc. In contrast, in a hardware supported SASOS an ordinary procedure call can be used for inter-domain operations [4]. After the first call, in which authorization is obtained, a hardware-based protection cache holds the <VA, access rights> pair representing access privileges to the protection domain. The allowed domain crossings can now happen transparently (hardware controlled rather than under software control) and the run-time overhead is the same as that for any intra-program procedure call. These unique features of Sombrero provide the functionality needed by a scheduler with components distributed over the entire system. The scheduler components communicate transparently through shared memory for data distribution and entirely avoid using cumbersome IPC and file access protocols required by the middleware runtimes needed by process-oriented operating systems.

The entire system is divided into scheduling clusters to limit the distribution of the state information. Clusters help to achieve a balance between the desired system performance and the overhead required to gather the state information.

It is expected that LANs will be used to organize the Sombrero nodes into clusters. A large sized LAN is usually made up of a number of smaller LANs depending upon the division of an organization into different subgroups. These LANs can be viewed as the logical clusters required for efficient distributed scheduling. The clusters have a hierarchical organization in which a cluster can be made up of several other clusters. Scheduling of threads takes place amongst nodes within a particular cluster or between clusters that form a bigger cluster.

A mechanism is needed to distribute load information of the nodes throughout the system. The shared memory inherent to a distributed SASOS provides an excellent mechanism for this purpose. Shared memory is used to set up a data structure per cluster for storing load information of the cluster members. The data communicated amongst the members of a cluster consists of load information and thread related data (e.g., thread control blocks). The Sombrero nodes are designed to communicate with

each other through router like services. These are described in section 4.1.

In Sombrero a fission/merge address space allocation mechanism is used to distribute the 2^{64} byte address space amongst the nodes in the system [8]. As part of its startup initialization, a Sombrero node contacts its router with a request for an address space region. The router keeps track of a parent node that can grant such a region upon request. When the size of the address space of the parent node falls below the lower threshold, the router looks for an alternative parent node in its cluster. Deallocated regions for nodes removed from the system are merged with adjacent free regions. Also in Sombrero, there is a token tracking mechanism used for distributed consistency management [2] [9] that enables Sombrero programs to be unaware of the physical location of the VAs. Token tracking uses a communication mechanism based on VAs instead of IP addresses.

4. Sombrero Distributed Scheduling Implementation

The Microsoft Development Studio and the Microsoft Visual C++ development system were used to implement the distributed scheduling mechanism. The code is written using the Win32 API. Important implementation parts of this research are described next.

4.1 Inter-node Communication

The Sombrero system is organized into hierarchies of clusters for managing distributed scheduling. The lowest level cluster consists of nodes and a router is assigned to each cluster. A group of clusters forms a higher level cluster. Each router uses a table to store VA to node mappings for the nodes in its cluster. The routers also manage consolidated router tables for VA to node mappings of other routers and the associated VA ranges of the nodes in their clusters. All the messages exchanged between any of the nodes are redirected through the routers. It's the router's responsibility to find a node depending upon the VA inserted as data in the message package. If the target VA is not listed in the router table of VA to node mappings for its own cluster, a router forwards the message to an appropriate router.

Figure 2 shows two first level clusters. A message from the node containing VA 0x1000 directed to a node containing 0x2000 (message A) goes through the router 0x1 as shown. The router 0x1 has the VA to node mapping of both these nodes. A message from the node containing VA 0x1000 directed to a node containing VA 0x3000 (message B) goes through the router 0x1 and the router 0x11. The router 0x11 stores

the VA to node mapping of the node 0x3000. In addition, both of the routers have tables indicating the range of VAs managed by other routers.

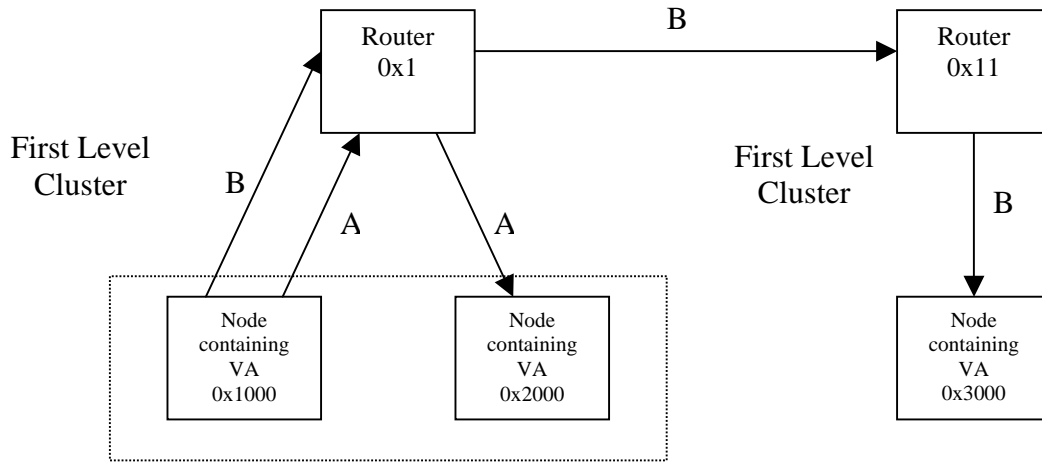


Figure 2. Sombrero nodes communicate with each other through the routers

4.1.1 Sombrero Node

In the simulation, the only function of the Sombrero nodes (Figure 3) is to demonstrate load balancing. They use Windows message-based notification of network events to a socket.

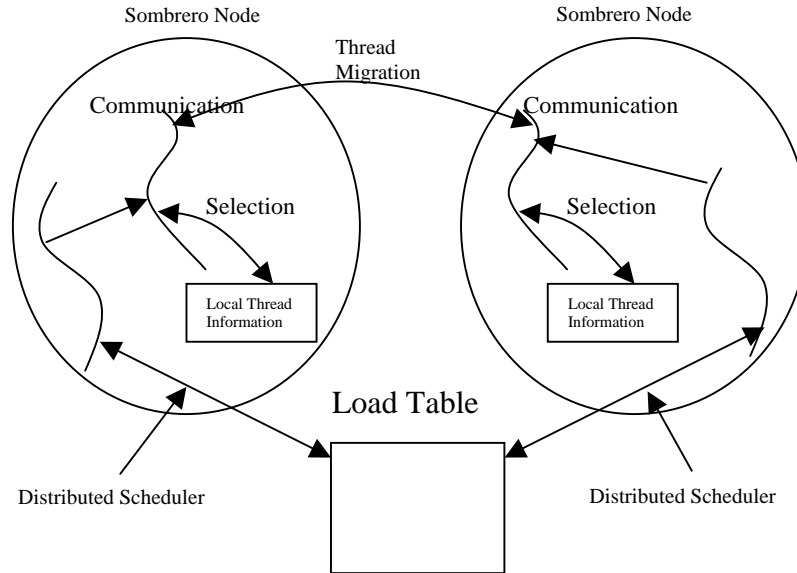


Figure 3. Architecture of Sombrero Nodes

4.1.2 Sombrero Router

A Sombrero router's main job is to direct messages sent from a node in the system to the destination node using VA to node mapping tables. Each router has a well-known port on which it listens for new node and new router messages. As part of its initialization, a node sends a new node message to its router. On receiving this message, the router enters the VA to node mapping for that node into its tables. Similar initializations take place as part of a new router's setup. A new router message is broadcast to all existing routers.

I/O completion ports have been used for the communication involving sockets (Figure 4). All the sockets created by the router are associated with an I/O completion port. Whenever a message is sent to any of these sockets, an I/O completion request is queued to the port. A pool of threads is created to service the packets queued at an I/O completion port. This architecture prevents the server from having to create one thread per client. At the same time having multiple threads for servicing requests allows the router to handle several requests at a time increasing the parallelism of the router.

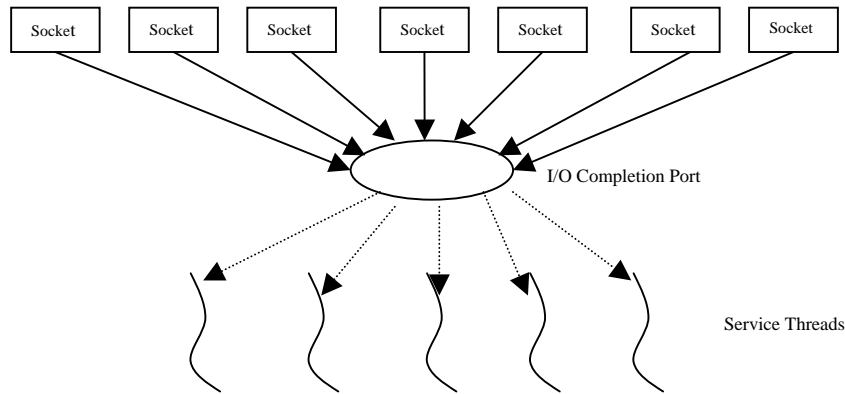


Figure 4. Architecture of a Sombrero Router

4.2 Distributed Scheduler

A simple round-robin algorithm has been implemented in the distributed scheduler simulation to take care of the local scheduling needs of a Sombrero node. The algorithm uses multilevel priority queues for scheduling the Sombrero threads. The system has been organized into a hierarchical structure of clusters. The nodes in the system are at the lowest level of the organization. This makes the scheduling algorithm scalable. Load information exchange is thus limited between members of a cluster, which themselves may be clusters or nodes. Load tables are allocated to maintain load information about members of the cluster. Distributed scheduling primarily occurs at the node level. As the load on clusters at lower levels increases, the distributed scheduling starts occurring at higher levels. The mechanism for scheduling at levels one and above (between clusters) is identical. However, at the lowest level (between nodes) slightly different rules are applied. The two different mechanisms are explained below.

4.2.1 Level Zero (Node Level)

The measure of load at the node level is the number of Sombrero threads. There are three thresholds of load at this level, viz., high, medium and low. For the simulation, these values are chosen as some arbitrary constants. When the Sombrero prototype is fully operational, these values will be determined empirically using profiling of the actual prototype.

In dynamic distributed load balancing only the smaller group, i.e., the minority group will be active. This means that at lower loads the algorithm is sender initiated (the few heavily loaded nodes try to send their threads to other nodes) and at higher loads, the algorithm is receiver initiated (the few lightly nodes try to get threads from other nodes). As far as possible the distributed scheduler operates at this level to achieve load balancing. The distributed scheduler gets involved in thread migration at a higher level only if it is not possible at a lower level. The key steps followed by the algorithm are:

1. If this node is loaded in the medium range no initiation of scheduling is done for the node
2. If node belongs to the majority group then distributed scheduling is not to be done.
3. If node belongs to the minority group then distributed scheduling is to be done. This condition is true in the following two cases:

- The node is heavily loaded and the algorithm is sender initiated. In this case, a lightly loaded node is chosen at random and a message is sent to it to initiate thread migration to it.
- The node is lightly loaded and the algorithm is receiver initiated. In this case, a highly loaded node is chosen at random and a message is sent to it to initiate thread migration from it.

- low--number of Sombrero threads in the ready queue < 30
- medium -- $50 >$ number of Sombrero threads in the ready queue ≥ 30

For all cases of testing, the load values were chosen so that the lightly loaded nodes can accept the entire extra load from the heavily loaded nodes. For example, consider the case that there is only one lightly loaded node present in a cluster and all the other nodes are heavily loaded. In this case the loads were chosen so that all the heavily loaded nodes could be brought down to medium loads through load balancing with the single lightly loaded node.

4.2.2 Levels One and Above

We illustrate this for the case when there are no lightly loaded nodes in a cluster. (A similar protocol is followed for the case where there are no heavily loaded nodes in the cluster.) No load balancing can now occur at the node level. The heavily loaded nodes in the cluster cannot share their load with any other node in that cluster. In this case, the nodes participate in thread exchanges with nodes of other clusters. At any level above the node level the three thresholds are defined as given:

- high: - no cluster members are lightly loaded and at least one member is highly loaded
- low: - no cluster members are highly loaded and at least one member is lightly loaded
- medium: - all other cases of loads where load balancing can occur within the cluster members or when all members of the cluster are medium loaded

Load balancing is intermittently attempted at higher levels of clusters. A suitable n^{th} level target cluster is found through the corresponding load table and a message for inter-cluster thread migration is sent to the target. The target node will check whether the first level cluster it belongs to is a suitable one. If the cluster is suitable it is locked and an acknowledgement is sent to the initiating node. The initiating cluster now sends threads to the target cluster.

The Sombrero routers then count the number of messages required to achieve load balancing within the Sombrero nodes. This procedure is repeated for all the important cases of load values at each of the three cluster levels tested. Note: it is expected that distributed consistency management will necessitate some additional messages to support thread migration.

Tests were carried out at the node level for single clusters consisting of 8 nodes, first level clusters in which three clusters each had eight Sombrero nodes and second level clusters in which two second level clusters each contained three first level clusters each having eight Sombrero nodes. Table 1 gives a high level slightly simplified overview of the results. The details can be found in [10].

Table 1. Load balancing messages required for distributed scheduling algorithm.

Cluster Organization Examined	Load Balancing Messages
One zero level cluster with eight nodes	3-14
Three first level clusters each cluster containing eight nodes	5-52
Two second level clusters of three first level clusters each with eight nodes	5-81

5. Testing

The distributed scheduling algorithm has been tested to verify its correctness, stability and scalability. The testing has been conducted at the node level and Sombrero cluster levels one and two. Script files were used to initialize the Sombrero nodes and routers. The script files start the Sombrero routers and nodes with an appropriate setup of load tables and initial load values (number of Sombrero threads). Load thresholds were defined as follows:

- high--number of Sombrero threads in the ready queue ≥ 50

Within single clusters three messages were required to transfer the load from a single highly loaded node to a lightly loaded node. The number increased to 14 for the case of 7 highly loaded nodes and a single lightly loaded node. For first and second level clusters, 5 messages were required to transfer a thread from a highly loaded node to a lightly loaded node in another cluster. The number of messages increased proportional to the number of heavily loaded nodes from 5 to 81. The number of messages required for load balancing in the first and second level cluster tests is the same if the ratio of total heavily loaded to total lightly loaded nodes is kept constant in both tests.

Only one additional load table is required per additional level of clustering. Hence, the required number of messages is expected to increase by a small constant factor as the number of levels of clustering increases. It can therefore be safely concluded that the algorithm is scalable and will be stable as the number of nodes in the system increases.

6. Summary

The testing of the dynamic local-data-based distributed scheduling algorithm using the simulator verified that the algorithm functions correctly. The algorithm was tested for three levels of Sombbrero clusters. The level of clustering was increased as the number of nodes in the system increased. The algorithm scaled well as the required number of messages increased by a small constant factor as the number of levels of clustering increased. It should be noted that the simulation has certain limitations. One is that the thread migration algorithm will produce an additional message-passing overhead for consistency management of the shared virtual address space. In addition, after any thread control block has been transferred to some other processor, the number of page faults is expected to rise for access to the thread data and instructions. These overhead messages were not simulated.

The principal objective of this work was to design, implement and test a distributed scheduling mechanism for a distributed single address space operating system. Specifically, the following issues were addressed:

1. The design of a dynamic local-data-based distributed scheduling algorithm to achieve load balancing for a distributed single address space operating system.
2. The design of a router facility for communications in a single address space operating system that uses a VA to node-address translation service.
3. Verification by simulation of the correctness, stability and scalability of the distributed scheduling algorithm for single node clusters and first and second level hierarchical clusters of nodes.
4. Implementation of an address space fission algorithm for distributing the large address space among the nodes.

More information on the Sombbrero Project can be found at [11].

References

- [1] Alan Skousen and Donald Miller, "Operating System Structure and Processor Architecture for a Large Distributed Single Address Space", *IASTED 10th International Parallel and Distributed Computing and Systems Conference Proceedings*, October 1998, pages 631-634.
- [2] Alan Skousen and Donald Miller, "Using a Single Address Space Operating System for Distributed Computing and High Performance", *18th IEEE International Performance, Computing and Communications Conference*, February 1999.
- [3] Alan C. Skousen, "SOMBRRERO: Implementation of a Single Address Space Paradigm for Distributed Computing Exhibiting Reduced Complexity", Ph.D. Dissertation, Computer Science and Engineering Department, Arizona State University, August 2002.
- [4] Donald B. White, "Implementation of a Lower Level Architecture for the Sombbrero Single Address Space Operating System", MS Thesis, Computer Science and Engineering Department, Arizona State University, December 2005.
- [5] Ron Feigen, Alan Skousen and Donald Miller, "Reduction of Software Development Costs under the Sombbrero Distributed Single Address Space Operating System", *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2002)*, June 2002.
- [6] P. K. Sinha, *Distributed operating systems: concepts and design*, Chapter 7. IEEE Press, 1997.
- [7] M. J. Zaki, W. Li, and S. Parthasarthy, "Customized Dynamic Load Balancing for a Network of Workstations. Computer Science Department", The University of Rochester, Technical Report TR-602, December 1995.
- [8] Alan C. Skousen, "Sombbrero: A Very Large Single Address Space Distributed Operating System", MS Thesis, Computer Science and Engineering Department, Arizona State University, December 1994.
- [9] Alan Skousen and Donald Miller, "The Sombbrero Single Address Space Operating System Prototype A Testbed for Evaluating Distributed Persistent System Concepts and Implementation", *The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, June 2000, pages 557-563.
- [10] Milind H. Patil, "Distributed Scheduling in Sombbrero, A Single Address Space Distributed Operating System", MS Thesis, Computer Science and Engineering Department, Arizona State University, December 1999.
- [11] Sombbrero Web Site, <http://www.eas.asu.edu/~sasos/>