# IMPLEMENTING A VERY LARGE SINGLE ADDRESS SPACE ACROSS MULTIPLE NODES: MEMORY PARTITIONING, PROTECTION DOMAIN MIGRATION, KERNEL REPLICATION, CONSISTENCY AND FAULT TOLERANCE

## ASU 64-bit OS Group White Paper 10

**Alan C. Skousen and Donald S. Miller**[1]

**Computer Science and Engineering Department**
**Arizona State University**
**Tempe, AZ 85287-5406**

**Technical Report TR-95-021**[2]
**January 1996**
**(Revised May 10, 1999)**

## ABSTRACT

Sombrero's protection model and design that makes use of the properties of a very large single address space on a single node were intended from the outset to be extensible to a multiple node system such as a set of workstations, servers and LANs. This White Paper describes how Sombrero makes use of the naming, persistence and protection features of a single address space in a network setting. The principle issues here are replication of the kernel structures and associated consistency issues, partitioning of the virtual address space, fault tolerance and domain migration. In the end, all of these topics must deal with the basic additional feature of a LAN: latency and its consequent, inconsistent local views of the virtual address space, which ultimately stem from the impossibility of atomic access to corresponding memory locations on distinct machines. The principle mechanism that Sombrero uses to deal with this problem is called token tracking. In this White Paper we look at the properties of a single address space that make possible network-wide optimizations beyond those found in conventional process-oriented operating systems and then describe the token tracking mechanism that makes use of these properties.

We have two objectives in writing these White Papers. First, we want to get our ideas out on the net and hear what you think of them. Second, they serve as an information basis for proposals we are making to industry to obtain hardware and software platforms on which to build a prototype and for proposals to government and industry for funding for the prototype effort.

[1] The authors can be reached as follows:
e-mail: alan.skousen@asu.edu and donald.miller@asu.edu
phone: (602)-965-5935
fax: (602)-965-2751

[2] Web page for technical reports: HTTP://www.eas.asu.edu/~sasos

## 1. INTRODUCTION

The assets of a Single Address Space Operating System design include: a flat name space which obviates the need for intermediary naming systems such as a file system; a unique name for every storage location thus eliminating conflicts in assigning uniqueness to an object; and a natural visibility between every object which makes resource sharing a simple and inexpensive matter. As for protection, processes in a process-oriented or Many Address Space Operating System (MASOS) are protected from one another because they cannot address data in another process's virtual address space, keeping one from corrupting the other. The Single Address Space strategy does not provide this natural barrier, rather all of the threads of execution are visible to each other as under a single multithreaded process. Protection in a SASOS is done with protection domains laid atop a single virtual space forming hard walls between principal activities as effective as the separate or private address spaces in a process oriented system.

We think the issues of protection, naming and sharing noted above work well to improve the performance in a SASOS over a conventional OS, and what's more these same issues that work so well on a single node can be made extensible to an entire network. *This extended SASOS, which Sombrero is designed to implement, has the appearance of a single large multithreaded process distributed over many nodes.* The extensibility is achieved largely because the kernel, which contains all of the management structures and code, is distributed, making the assets created on one node transparently available to all the others via kernel services. Sombrero is properly named a Very Large Single Address Space *Distributed* Operating System. The network features of Sombrero are not an add-on.

This paper builds on the concepts introduced in White Paper 2 which describe the OS operations, structures and hardware on a single node.

## 2. SOMBRERO ARCHITECTURE CONCEPTS THAT SUPPORT A NETWORK

### *2.1 Overview*

Sombrero takes advantage of the Single Address Space to implement architectural abstractions which could not otherwise exist. The consolidation of namespaces and the extension of the cache hierarchy beyond main memory to include mass storage and the network are two of these. A third, which we refer to as *VM Magic*, takes advantage of the ability to permanently bind data and code to virtual addresses, integrate persistent storage into the VA space, and additional single address space properties that derive from removing the encapsulation of protection, execution and principals by process boundaries. These single address space properties allow a permanent network wide binding of pages to resources, i.e., to objects. They also permit a permanent binding of access rights to objects. This latter is independent of principals and their associated protection domains. These two bindings imply that demand paged virtual memory management can transparently support object management. Finally, objects (with their associated VAs and access rights) can be associated in whatever way is desired with protection domains, or alternatively, protection domains can be modified to include/exclude whatever objects we want. So VM Magic is just the transparent network wide support for sharing and protection supplied automatically on every node by the already present demand paged virtual memory management of the 64-bit VA space[3].

[3] There is a price paid for all of this - additional protection hardware is needed within the CPU - see WP 2.

### 2.1.1 Network Name Space

Modern operating systems use several name spaces to provide network wide services. These name spaces include the many separate VM spaces on each network node, the file system on each node, a page management system such as a DSM, and the name space supporting IP addresses for IP networks. For processes to interact a considerable cost is incurred to translate between all of these name spaces. Sombrero offers an opportunity to integrate all of these name spaces into one allowing the advantage of global persistent pointers and uniform resource management. The only remaining translations will be the physical to virtual translations supporting the cache hierarchy discussed in the next section.

Sombrero can provide this integration since every storage location has a network wide unique name in the VM namespace and every one of the name space structures mentioned above require storage allocation to support the resource. In other words whenever storage is allocated to support a resource a unique name is allocated as well thus all of the above mentioned services can be supported in the same name space without conflict.

### 2.1.2 Network VM Access and Storage

To design an operating system which operates on a single node the designer can always count on the atomicity of a bus transaction. Even in a multiprocessor environment there is only one bus master at a time. Simple reads and writes to main memory RAM are atomic. Some primitives perform an atomic read/modify/write which provides atomicity for multiprocessors. Snoopy caches or other strategies enforce the atomic view of these accesses by updating or invalidating cached data to force compliance with the atomic bus cycle view so that all threads will always see the latest version of data. Even so a latency exists because most modern CPUs run much faster than the bus will support. This latency is overcome by providing a faster interface in which the CPU and a first level cache communicate at nearly the speed of the CPU and consistency is propagated down the cache hierarchy as needed. Data consistency is maintained from the operational level of the CPU at its first level cache up the hierarchy to the RAM or main memory of the computer. For modern processors there may be three caches in the cache hierarchy between the CPU registers and the main RAM. Hardware enforcement provides the data consistency at this level. The operating system counts on this level of consistency on which to build a new level of services including multiple threads, access to slower storage such as disk or tape or CD ROM and a user space in which applications can run.

Operating systems such as UNIX provide some services to enforce the consistency of data at levels above that which the hardware supports. There are also add-on network services such as Distributed Shared Memory (DSM) which perform complicated consistency management on pages of data across the network. The user level programs count on these OS services to provide a reliable environment on which to build, just as the OS counts on the hardware for a starting point.

#### *2.1.2.1 RAM-Centricity*

Most OS's build their services around the main RAM. File buffers are read into RAM where programs act on them. Until virtual memory was invented programs in execution and data buffers were limited by the size of available RAM. Network interfaces map data across the network into the local RAM as well. RAM or Main Memory is the central focus of the OS

architecture. Of late, files are being memory mapped and the logical distinction between RAM and slower disk storage is beginning to disappear.

Sombrero takes advantage of the Single Address Space to simplify the view of storage by skipping the RAM-centric view. Unlike the Many Address Space OS's which create a virtual memory space for every new task, translate the code and data into the VM space from the file system and then destroy the VM space when the task is over, Sombrero has only one persistent VM space and no file system. The Sombrero design is intended to provide a seamless continuation of the cache hierarchy rooted at the CPU. Sombrero does not just extend the cached view from the CPU to the main RAM, where the hardware consistency support ends, but to the disk storage and the entire network. RAM and disk are in fact considered as additional levels of cache to the network-wide Virtual Memory, the 64-bit virtual address space. Each node treats the network connections as a slower bus to a lower level cache in a manner similar to the way that separate CPUs connect to main RAM via snoopy caching.

#### *2.1.2.2 Object View*

As a consequence of the non RAM-centric view and with the support of the Range Protection Loookaside Buffer (RPLB - see White Paper 2) it makes sense for Sombrero to manage data as memory objects at the VM level instead of as pages of data in main RAM. VM pages provide the block transfer unit between main RAM and disk cache level storage or the network. For Sombrero the VM paging subsystem acts transparently much like any other cache. Since a VM page is the same page for all nodes, has a unique name and each node has only one memory map to deal with, it is not difficult to locate a page on a node or to call for it from the next higher level in the cache hierarchy when needed. The page handling mechanism is independent of object management.

Just as the CPU counts on atomic transactions to access main RAM, Sombrero's users can count on a view which provides atomic access to user Memory Objects when needed. Instead of shuttling and tracking pages which belong to a Memory Object, Sombrero shuttles Memory Object Control Blocks (MOCBs) and the pages are located through the cache tracking process when needed. The method by which Sombrero makes this process fairly painless is called Token Tracking.

Token Tracking, Sombrero's network distribution mechanism for kernel resources, achieves kernel extensibility for all kernel resources not just memory objects. The basic strategy is to take advantage of the uniqueness of every object by creating local structures to handle the management on the local node and make all of these structures visible to every other node. Taken together these structures provide a distributed control structure for the entire network. The control data itself is distributed as needed and kept up to date via a graph called the Copy Set graph (CS graph) which is used to connect interested parties via a distributed set of control blocks. The access to a memory object, depending on the per-object implemented R/W semantics, is controlled by a token which is used to manage the consistency of an object.

## 3. VERY LARGE MEMORY PARTITIONING

Before we discuss token tracking it is necessary to describe the manner in which memory is partitioned or allocated in the Sombrero VM space.

Each node has a unique segment of the address space from which objects that originate at that node are allocated. MONADS and Mungi two other SASOS's, preallocate a set of fixed sized partitions which they assign to nodes [Rosenberg, Heiser et al.]. This fixes the size of the regions and the number of nodes which can be attached to the network under the same virtual space. These VM region bindings may often be unnecessary and overly restrictive depending on the application.

Sombrero uses a different strategy to partition the memory but without the fixed regions or node count and without maintaining an allocation table. These features fall out from the use of Memory Object Control Blocks (MOCBs) to manage memory. White Paper 2 describes the MOCB [2].

When the first node in a new network is brought on-line the entire 64 bit address region is assigned to that node under a single root MOCB. As new memory objects are created they are assigned via sub-MOCBs out of the lowest aligned address with a sufficient range to cover the requested block of memory leaving the upper ranges of memory unpartitioned.

When additional nodes are brought on-line they initially operate in their own virtual space and send out a broadcast requesting a parent from which they can obtain a root MOCB for themselves. One of the parents offering the largest split is accepted. The parent fissions its root MOCB into two equal aligned parts, the lower half containing all of its local memory allocations and the upper unpartitioned half is sent to the new node. The new node joins the virtual space and may later fission its root MOCB as well. When a node is removed from the network it may send its root MOCB back to be fused with its parent or another node may pick it up as a second disjoint root MOCB. This second root MOCB may later be given to a new node complete with its set of existing sub-MOCB’s, etc.

In the network the base virtual address of a node’s root MOCB also becomes the name of the node for routing purposes. Nodes owning multiple root MOCBs are treated as multiple logical nodes. This allows a request for access to an address to always be directed to the inception node for that object. This is done by having the network router route to the location of the root MOCB with the highest address in its routing table which is less than or equal to the address in the request.

Memory allocations recognized network wide are as simple as allocating on the local node since each node manages its own unique piece of the virtual space.

A final issue is memory fragmentation. In such a large address space memory fragmentation is not a major issue except that the fissioning process may be affected by widely distributed memory allocations. Sombrero’s allocation algorithm and garbage collection schemes reduce memory fragmentation by allocating from the open regions closest to the root MOCB. A hashing table can be used to grade the fragments by size for faster allocation. Also in Sombrero, adjoining free memory objects are automatically fused.

## 4. TOKEN TRACKING

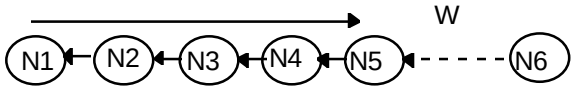
The Protection and Resource Access List (PRAL) described in White Paper 2 contains the control blocks which Sombrero uses to manage its resources. These control blocks include the

MOCB or Memory Object Control Block, the RCB or Region Control Block which is subordinate to the MOCB, the GCB or General Protection Domain Control Block which is part of the Ubiquitous Protection Domain (UPD), the TCB or Thread Control Block which also contains the Carrier Protection Domain part of the UPD, and the PCB or Principle Control Block which governs resource ownership.

In general, tokens are used to enable a Control Block (CB). When a CB is created it comes with a token (just a flag bit) which enables the resource or other entity which the CB represents to be used on the node. When a CB is copied to another node the setting of the token flag is dependent on the semantics which govern the memory object. RCBs carry the tokens for their MOCBs. Consider the RCB of an MOCB which represents exclusive Write access to an object's memory. When the RCB is created its token is initially enabled. If it is copied to another node the token flag is cleared on the original node and enabled on the new node. By this mechanism, the original node will not be able to gain write access to its memory object until the token flag is returned (disabled on the other node and reenabled on the original node). The original node knows where to find the token because its local MOCB stores a pointer to the node it last knew held the token, i.e., the node it passed it to. As each node in turn gains access to the token the sending node remembers to what node it sent it. These senders can be viewed as the nodes of a directed graph. We call this graph the *Write Last Graph*. It forms one of the four basic graphs that support token tracking. It is further discussed in section 4.2 below.

For the MOCB, a CS_Parent pointer (CS is Copy Set), CS_Graph pointer, CS_Arity count (arity is the maximum number of children a node can have), Write_Last pointer, Write_cache pointer and Inception_node pointer are data which aid in determining the location and state of every distributed token and its virtual pages. These data about token distribution provide an automatic means of enforcing data consistency and Memory Object coherence without depending on a broadcast. Four of the pointers are the basis for the four graphs.

### *4.1 Write Cache Graph*




The Write Cache graph: N5 currently holds the write token W which originated at N1. N6 is attempting to read the Memory object.

Figure 1 Write Cache graph

The *Write Cache Graph* is a directed graph with computer nodes as vertices that indicates the path along which the latest version of each of the pages in an MOCB is. The basic strategy for the Write Cache Graph is to maintain a directed graph with edges pointing back to the nodes which have previously owned the write token. Its direction is generally opposite to that of the Write Last Graph introduced in the previous section. This graph terminates at the *owner node* (the owner node is where the CS graph is rooted and is usually where the MOCB was originally created) and the nodes in the graph collectively provide a cache of the complete set of up-to-date virtual pages belonging to a Memory Object. The node pointers are stored in the Write_cache pointer in the MOCBs.

Figure 1 shows a write cache graph in which N5, the graph 'starting' node, possesses the most recently changed virtual pages of a Memory Object controlled by an MOCB holding it's write token W. Successive nodes in the Write Cache graph contain pages of the Memory Object modified at increasingly earlier times. When N6 seeks to load a page of W it finds the most current copy of the page by reading from the starting node of the graph, N5. If the page is not there, N4 is searched, etc. In this way the most recent version of the page is accessed first, guaranteeing consistency. Recently altered pages will be in a graph node close to N5 while page fault-based lazy updating leaves less often modified pages nearer N1 at the 'end' of the graph. The benefit of this strategy is that pages in the memory object are moved only if they are needed, and when a page is read, the reader will access the most recent version first, guaranteeing consistency. This is done without using version numbers or logical page pointers and accommodates any number of pages belonging to the Memory Object with no change in complexity to the control structures. This simple mechanism works for two reasons. First, in a single global namespace there is a one-to-one relationship between every data unit and the virtual address that names it, and the demand-paged virtual memory management already present on every node automatically generates the addressing and protection page faults needed. This is an example of the "VM-Magic" we referred to earlier. The second reason is the per-object organization of Sombrero's PRAL protection data structures described in White Paper 2. In Sombrero protection is associated with resources, i.e., memory objects. This grouping of pages by object and objects with protection coupled with the VM support for paging enables the simple graph structures described in this section.

The write cache graph is pruned automatically by Sombrero when it intersects itself. In Figure 2, starting with the Write Cache graph shown in (a) with the write token in the MOCB on Node 5, and a request to write a page by Node 3, the pruning is accomplished by gathering all of the pages updated in N4 and N5 and copying them to N3 with N5 pages having higher priority than N4 pages. The resulting Write Cache graph is shown in Figure 2(b). To close the graph and return all pages to the owner node the write token need only be transferred back to N1 which causes the graph to prune itself to a single node and copy back all of the updated pages. The nodes which are pruned out of the write cache graph remain in the Copy Set graph, described in section 4.3, unless they remove themselves.




Figure 2. Write Cache graph pruning. The Write Cache graph intersecting itself in A prunes itself in B when token W is transferred from N5 to N3.

### *4.2 Write Last Graph*

Whenever the write token is moved, the Write_Last pointer is also updated with the new node name to preserve a path to the current write token holder. This path, called the Write Last graph, is used primarily by readers who want to locate the start of the Write Cache graph in order to obtain the most recent version of the data. Pruning for the Write Last graph is shown in Figure 3.

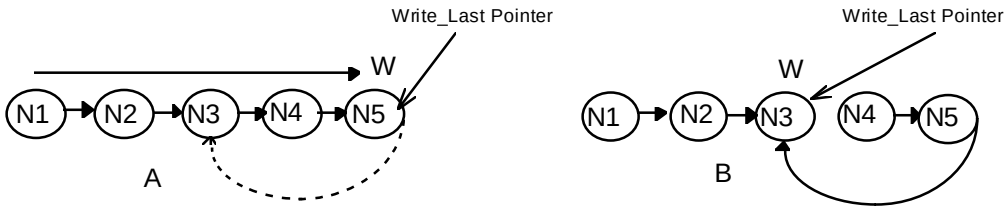
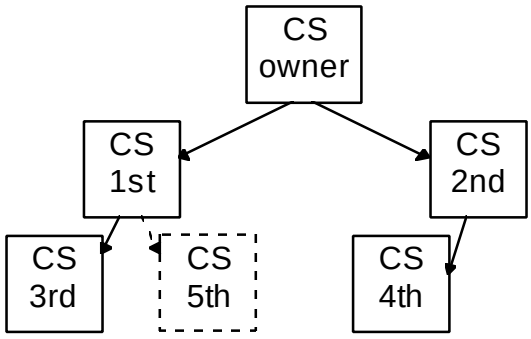



Figure 3. Write Last graph pruning. The Write Last graph before (A) and after (B) pruning.

### *4.3 Copy Set Graph and Parent Graph*

The *Copy Set (CS) Graph* (see Figure 4) contains all the information necessary to define,




A Copy Set graph for a Control Block with an arity of two with the owner node as its root.

Figure 4. A CS graph with arity two and four nodes currently in the copy set. A fifth node, when added,will continue to fill out the balanced graph. The first node was the first to make a request, and so on. There is no attempt at any other ordering.

control and provide some fault tolerance for the copy set of a distributed memory object. For a distributed MOCB the CS graph is a tree rooted at the owner node. Each additional vertex corresponds to a node at which there is a copy of the MOCB. Additions to the tree occur when any node first requests access to an MOCB. The CS_Graph pointer of an MOCB on a given node is a vector with arity elements that point to the descendants of that node in the CS graph tree. The CS graph is used to notify all parties holding a given Control Block (CB) of important changes pertaining to it such as:

- **Obsolete Pages**

  One or more of the pages of an MOCB have become obsolete by a write. A reader on the given node who needs a current copy will have to obtain it by reading from the top of the Write Cache graph. Page invalidates are initiated when a dirty bit is set on a page and this information is sent to the nodes on the CS graph. The effect is to leave only one primary

copy of the page and, depending on the semantics, force the nodes on the CS graph to invalidate the page copies and reload them from the write cache graph as needed.

- **Node Requests for Exclusive Access**

  An RCB token is to be surrendered by a node to a request elsewhere for exclusive access. All nodes will retain a tokenless copy of the RCB. Page invalidates are still sent to these nodes since they remain on the CS Graph.

Messages sent to the CS Graph are confirmed when the leaf nodes acknowledge back to parent nodes recursively back to the owner node via the CS_Parent Graph (see Figure 5). This removes any doubt of the successful reception of a message and informs the owner node so that it may broadcast a *recovery message* to all nodes if it appears that any children are logically severed from the CS tree. Recoverable children are nodes which see the broadcast message without first having seen it through their parent node. They are placed elsewhere in the tree under new parents. When the old parent node, whose failure caused the initial sever, recovers it is informed of the reconfiguration upon checking with its parents and is also recovered. Special handling is needed in the important case of owner node failure (see Appendix 1).

The CS graph has an arity greater than one in order to shorten the path to any given node holding a CB. If many nodes have a CB, the path to the furthest CB can be reduced to $\log_{arity} n$. By distributing the CB requests evenly to each branch of the tree a balanced CS graph can be achieved and a very large graph can be supported to provide scaleability. This graph is balanced and directed but unordered and is a network multicast in function.

Memory object dirty bits are stored in the in-core inverted page table with the first and last bits redundantly stored in the MOCB. The redundancy enables Memory Objects not aligned with a page boundary or smaller than a page to maintain consistency for ranges that share a page. The CPU sets dirty bits on the first attempted write of a clean page and also initiates page invalidate notification[4]. The dirty bit is reset and a writeback performed if necessary, when the page is copied out to another node. Sombrero provides hooks for higher levels of software to modify the default multiple reader/single writer sequential consistency semantics.

Tracking of the general and thread control blocksGCBs and TCBs uses simpler graphs depending on the assigned semantics. GCBs and TCBs have two basic semantics: replication or unique. Unique semantics mean that only one active copy of a token for a CB can exist, and hence the CB can be active on only one node in the network. Replication means any number of token copies, one per node called *clones*, can operate with the access rights for the GCBs or TCBs as a whole being the sum of the rights of the GCBs or TCBs on the separate nodes. MOCB RCB semantics will still prevail for memory access but multiple GCBs or TCBs operating on different nodes act in parallel and exchange data when useful to do so. TCBs and GCBs use the CS graph to perform actions on all of the clones for some operations such as determining if an access permission exists to some MOCB.

[4] The details of this are hardware dependent.

### 4.4 Distributed Shared Memory vs. Token Tracking

Until the Sombrero prototype can be developed it is not clear how well token tracking will work but it appears to have some advantages over strategies like Distributed Shared Memory (DSM). DSM is an effort to provide a global name space to a network of process-oriented nodes. While it succeeds it also creates another occasion for the need to translate from one name space to another. For example, a block in the DSM name space needs to be translated to a local VM name space and a translation table maintained for every process. The properties of a single address space used in the Sombrero design naturally provide what the DSM concept is attempting to patch into the MASOS type operating systems with no translation requirements. In a single address space a virtual address names the same data independent of what node it is located on. No additional mapping is necessary. Under DSM messages between processes must be informed of and carry the differing VA mappings. In Sombrero the VA naming mechanism is at the bottom and all other mechanisms are built on top of it. The kernel is distributed and VM support information travels between nodes as low level intra-kernel message communication. The rationale for shared memory as the basic communication mechanism has been presented elsewhere, albeit with page rather than object mapping and protection domains defined by page maps [Angel, Arius].

DSM can also be viewed as a distributed process forced to have different threads on each node. Sombrero overcomes this limitation as well, since not only is memory shared, but all non-hardware resources are distributed. Protection domains are shared from node to node, principals can be represented simultaneously on multiple nodes and threads can move between nodes. Sombrero even supports having the *same* thread on multiple nodes.[5]

DSM suffers at least the complexity that Token Tracking does but the complexity of DSM is compounded by the need to manage every page. Token Tracking manages objects and one object may contain many pages without additional cost. Sombrero VM is managed by one hardware assisted inverted page table for main RAM and one indexed inverted page table per disk on each node. A single per-node inverted page table is not practical under DSM in a process-oriented system because of the need to support many different name space translations.

## 5. MIGRATION, REPLICATION AND CONSISTENCY

Data Distribution conveys the notion that data is made accessible across a network. Migration is a more extended form of data distribution in which control structures, services and data are conveyed to the point of physical access providing better local performance. Replication is a mechanism by which kernel control structures can migrate and consistency is the matter of enforcing the selected distribution semantics. In Sombrero Token Tracking is the general mechanism upon which migration, replication and consistency are built.

### 5.1 Data and Control Migration

Conventional computer networks are usually managed by a server which provides distributed data to requesting nodes. One purpose of a server is to provide a control between its data set and the network, i.e., it manages the coherency and consistency of its data set to some degree. The servers in a MASOS network are not themselves typically distributed but provide a centralized point of access of limited capacity. Major factors which promote this centralization are the difficulty of reconciling multiple name spaces and maintaining data consistency between the

[5] This can only happen, of course, under a somewhat less restrictive definition of thread that that provided in White Paper 2, i.e., the definition of thread cannot include current CPU state.

name spaces on physically separated nodes. Doing this and having a reasonable response time are in opposition. The need for response time usually wins out. If the non RAM-centric view proposed in Sombrero is adopted, the need for translation between name spaces is eliminated. If consistency is relegated to being a transparent service of the operating system some services can be designed to exist in the VM space without a knowledge of the underlying connecting network hardware. That is, identical copies of kernel and user level server code as well as replicated or copied kernel and server data structures can be distributed. The service thus becomes automatically distributed and migrates to the point(s) of greatest demand which, in general, will provide greatest performance. Client activities migrate in the same manner.

### 5.1.1 Migration over the VM cache

Migration comes into play because the kernel will tend to move resources and user services to the points of physical activity (nodes) to decrease the response time. This tendency is inherent in the VM cache mechanism. This tendency will also have a beneficial effect on the sharing of data between nodes.

### 5.1.2 Dynamic Concurrency

Sombrero will offer to higher layers a service that dynamically and transparently alters the granularity of the MOCB protection mechanism based on current concurrency to reduce the effects of false sharing. Object level memory management can elevate the false sharing problem by requiring frequent token transfers. For those relatively few instances where false sharing consumes excessive communication bandwidth or CPU time this mechanism alleviates the problem. The cost is increased complexity of the kernel control structures for the affected memory objects.

### 5.1.3 Control and Execution Migration

The VM caching mechanism is built upon Token Tracking. Token Tracking manages objects by regulating when and where the VM page manager can send its pages. To do this the MOCB structures must also be mobile. The Token Tracking graphs of the proceeding section provide this mobility as well as mobility for GCBs, TCBs and PCBs.

Sombrero has the following general rules governing how and when MOCBs, GCBs, TCBs and PCBs migrate:

- Whenever a thread attempts to access a memory object accessible to the current UPD but the MOCB is not present the MOCB is located and replicated to the local node. This makes the VM pages in the range of the MOCB available to the node. This is object migration and can include data or code.

- Whenever a thread attempts to access a memory object on a node where the GCB for the current UPD is not present the thread will travel to the node where the GCB is located to continue execution. This is execution migration.

- If the thread from the last step locates a GCB which can migrate, indicated by a *migratable* bit, the thread is returned unexecuted with the replicated GCB to its node of origin and executed there. This is domain migration.

- Since PCBs bind physical resources such as input and output devices to a user the PCB will generally migrate to the point of user connection, i.e., in general to the node on which a user is working.

The above rules provide a general decision making mechanism for managing the migration of control structures on Sombrero. Sombrero is designed to permit refinements and specific semantic features that can override how migration occurs. The token semantics for GCBs and TCBs also determine whether execution is concurrent on more than one node or not.[6]

### *5.2 Kernel Replication via Surrogates*

Kernel Replication describes the distribution of the kernel control structures across the network. Unlike page distribution which simply copies pages to the copy set, kernel replication builds local representations of permissions found globally in the copy set as needed. Local representations of control structures which originated elsewhere are called *surrogates* and have the same *name* field entry as the original control structure (the VA on the originating node) but have a different VA for the local CB (see WP 2 for more on CB data structures). This has several advantages in the Sombrero environment. The edges of Token Tracking graphs point to nodes as vertices and the graphs are sets of connected nodes. The pointers are actually the replicated surrogates' VAs in the nodes. This allows network routing to forward graph messages not only to the proper node based on the local surrogate VA, but to the exact control structure. The local hardware also uses the local surrogate VA in the RPLB to perform object loads. These two optimizations reduce the need to search the kernel structures (their VAs are always the local structure name) and allow a compact representation for the various control blocks by avoiding fragmentation of locally allocated physical space[7]. The drawback is that the relatively few instances of resource access by the global namefield name on a foreign node will have to search the protection access lists there to find a match rather than using the VA name as a pointer. Since in many cases, i.e., on the originating node, the local and global VA names will be the same this tradeoff seems worthwhile.

### *5.3 Data Consistency*

Data Consistency is expected to be managed on a per object basis. Each MOCB has a method which defines and controls the semantics for that object. Consistency semantics can range from no consistency requirement to as strict as needed. Since consistency is implemented as a method, an override method to that for the base class can be provided on a per object basis as well. There are parallels here to the SPIN operating system kernel extension structures []. The Token Tracking graphs provide a structure on which a variety of semantics can be implemented.

## 6. OTHER ISSUES

### *6.1 Fault Tolerance*

In a conventional MASOS great difficulty occurs if a key storage device becomes unavailable to a network. Various techniques have been established to improve the availability of key devices but nothing provides 100% availability. One of the strategies is called RAID or

[6] As noted earlier, it is not possible for two threads to share processor state. However, Sombrero's design permits common Thread ID and common software protection domain and memory object data structures that originate from some initial shared state.

[7] Although control block coherency could be set to allow multiple writer semantics (as, for example, memory mapped I/O), the physical memory fragmentation engendered by locally allocating VAs for control blocks originating on other nodes would cause greater problems than the use of suoorgates.

Redundant Array of Inexpensive Disks which distributes data in such a way so that if one of the drives fail the others can regenerate the missing data [5].

In Sombrero, which depends on the availability of the Virtual Memory, the failure of a node can leave a hole in the VM. This hole may cause anything from no difficulty to failure of the network depending on the importance of the data and how it has been distributed. In what follows we present some of our thoughts on ways to provide acceptable tolerance to VM faults in Sombrero.

Any fault tolerant solution that works well will have to be simple. Token tracking provides only marginal fault tolerance. The key is to view Sombrero as a distributed multithreaded process. Most conventional solutions are RAM-centric. The normal mindset is to regard the separate RAM spaces as important rather than the space as a whole. If we continue to think of each host as an autonomous computer our solutions continue to give priority to the picture of single entities trying to be consistent in a heavily interdependent environment. There is no good solution for that. There is actually only one problem to be solved. The virtual memory space has to be able to look the same to any set of CPUs working on the same data set in order to have object consistency. The corollary is that CPUs not working on the same data set do not need to have the same view of virtual memory.

The network, mass storage and dynamic ram and bus are all just storage manipulators or cache (storage) and carry (communications) devices for the virtual space. The network is the same as the bus just slower and the disk is the same as the RAM only slower but all of these devices are just parts of a cache for the virtual space. In addition, networks are becoming faster. The increased bandwidth of contemporary network hardware enables latencies less than disk latencies [Feeley's global memory work].

The latter observation leads to a basic approach. Ideally, every dirty page which needs to be flushed from RAM should be broadcast to all nodes at once. That way if any node should fail all of the memory would still be accessible. This scheme requires too much storage redundancy. We can accomplish the same thing to a lesser degree of fault tolerance by using a RAID like concept called RAIN (Redundant Array of Independent Nodes) spread across a LAN instead of across a dedicated bank of disk drives. As long as the various consistency models are supported the shared memory should be well protected. As soon as a node gives up its write token the dirty pages are written across the LAN. These pages can then be reconstructed if one of the nodes fails. This strategy affects normal reading, sharing implementations, scalability and available bandwidth.

Another more scaleable approach is to take advantage of the cache hierarchy to place a fault tolerant layer at the top of the hierarchy. If memory objects which comprise an entity such as a database, control their write tokens in such a way as to release them only when the database is in a consistent state then an outside agency, such as the fault tolerance manager, can only obtain the entire set of write tokens for the database if the database is consistent. By periodically taking control of a group of write tokens a fault tolerant layer can take a snapshot of the memory objects which represent some consistent state. If a failure occurs all pages can be invalidated for the write token group allowing the last consistent version to then be recovered from the fault tolerant layer by normal caching operations.

We note that an important part of the Sombrero research is to investigate additional mechanisms to support fault tolerance. Especially promising is the lazy evaluation of distributed persistent checkpoints described for the Arius SASOS [6].

As a final note we point out that Sombrero's distributed kernel is not less fault tolerant than other distributed OS kernel designs. The reason is that the PRAL pointers are all local, leading to autonomous essential object access and protection functions. The loss of essential global system data will affect any system.

### *6.2 Kernel Extensibility*

Kernel objects in Sombrero are readily extensible to the entire distributed system. This is accomplished mainly via the Control Blocks and tokens. Tokens enable a CB, and combined with the tracking features extend the ability of any node holding a token or a CB to determine its distributed state. For example, suppose an attempt is made to access memory object O in distributed GPD G on node N. If node N has no access rights for O in G the CS graph of G is accessed to check other participating nodes. Following the CS graph will either provide a copy of an access right to O in G from another node or report no access which results in a no-access fault on N.

Another powerful feature is that individual nodes need only concern themselves with their local address translation since memory distribution is defined by the set of MOCBs not address maps. This means that pages which are not actively stored on a node need have no allocation or representation. As node pages are created or copied from other nodes, entries are made in the node's inverted page table, and as pages are removed its page table entries are invalidated. Pages which are not primary copies (known by a flag in the page table entry) can be discarded when room is needed on node backing store.

### *6.3 Relationship of Sombrero's features to modern LAN hardware*

The ubiquitous nature of Sombrero’s protection mechanism is designed to present a view to a user program that it is running in a single large (networkwide) unfettered memory. As long as no attempt is made by a thread to access resources not allowed to it this abstraction should succeed transparently. Since Sombrero has a distributed kernel the abstraction will hold over a few or over many nodes. The practical constraint will be the non-uniform access times to different pages of the memory. Where this matters, the VM space involved can be confined or wired to a particular set of hardware.

The Token Tracking scheme was designed to provide service in a point to point environment where every Copy Set acts in a multicast setting. This will be especially appealing to ATM networks. Internetworked Ethernet LANs can optimize the local portion of multicasts to broadcasts instead of waiting to traverse the CS graph. And, as detailed in White Papers 2 and 16, in a single node setting Sombrero should outperform current process oriented strategies with their many name space translations.

## 7. SUMMARY

Sombrero is an ambitious proposal that synergistically combines a number of strategies. Token Tracking, the UPD including its protection matrix model, PRALs and RPLB, Migration, Kernel Replication, MOCB resource sharing, and others are concepts which if investigated as isolated

modifications to a conventional process oriented system would lose much of their significance. It is the combination of all of these features that produces a networkwide new design.

We believe the proposed Sombrero operating system has great potential over existing popular designs in both multinode and single node environments. Use of the single ubiquitous very large memory feature by itself provides a considerable opportunity for application modules to work together far more efficiently and responsively. The distribution mechanism is valuable to groups which must work together using common databases for design, transaction processing and other applications which are currently impeded by the many namespace translations which must occur. In addition we believe that Sombrero is superior to other SASOS designs which propose the use of capabilities and other non ubiquitous mechanisms which act as obstructions to the use of the single large memory abstraction because of their program serial instructions for crossing protection domains which in addition to being very inefficient, imply a user level requirement to manage the protection domain boundaries. Sombrero is also far more scaleable in its object oriented design because it establishes a networkwide common set of general structures and places them in the kernel which controls their distribution. By then distributing the kernel, the user level applications are automatically distributable, hiding the allocation of CPUs etc. from the users, while at the same time storing only the locally required data in each node.

Other White Papers describe our implementation ideas which would build Sombrero over a conventional operating system to give it backward compatibility and a stable development environment. A complete listing of all of the White Papers can be found in the bibliography at:

http://www.eas.asu.edu/~sasos.

## 8. POSTSCRIPT

We hope that distribution of these White Papers will lead to a constructive dialog on the design of single address space operating systems, and in conjunction with our external proposals, to funding for Sombrero prototype development and acquisition of a platform to develop it on.

## 9. REFERENCES[8]

[1] "The Sombrero Single Address Space Operating System", *ASU 64-bit OS Group WP 1*, A. C. Skousen and D. S. Miller, ASU Computer Science and Engineering Department Technical Report TR-95-012, September 1995.

[2] "Range Protection Lookaside Buffer and Sombrero Software Protection Data Structures", *ASU 64-bit OS Group WP 2*, A. C. Skousen and D. S. Miller, ASU Computer Science and Engineering Department Technical Report TR-95-013, September 1995.

[3] "Protection, Persistence and Naming in a Single Very Large Address Space", *ASU 64-bit OS Group WP 16*, D. S. Miller and A. C. Skousen, ASU Computer Science and Engineering Department Technical Report TR-95-020, September 1995.

[4] "The Mungi Manifesto", G. Heiser, 1994. Available by anonymous ftp @ vast.unsw.edu.au:pub/Mungi/Mungi-Manifesto.

[8] See the ASU 64-bit OS Group White Paper Bibliography in for latest revisions and status on all ASU 64-bit OS Group White Papers. References in this White Paper may be out of date or the referenced White Paper may not yet be available.

[5] "16 Fast Reliable RAID Subsystems", M. Guy, Byte, September 1995, pp. 248-261.

[6] "Implementing Fault Tolerance on a 64-bit Distributed Operating System", Ph.D. Dissertation, Department of Computer Science, City University of London, England, July 1993.

## 10. APPENDIX 1 - ADDITIONAL DETAILS

### *10.1 Failure of Owner Node*

If the owner node fails the earliest complainer is appointed the new owner node by the name service and a recovery message is sent out. Any blocks needed from the owner node and not found in the cache hierarchy are unrecoverable and can be handled by reallocating the page or failing. The owner node is rebuilt when the owner node fails. Rebuilding the CS graph on the new owner node is done by sending a recovery message without first sending a message by the CS graph. When the original owner node recovers it does not automatically become the owner node again.

### *10.2 Extensions to a WAN*

The migration effect of the token tracking and hierarchical cache will be most pronounced over larger networks. Other matters enter in as well as the network expands from a LAN to a WAN. Since the kernel is distributed the security of the kernel may become an issue in a setting where access to the extended hardware is not closely managed. The use of authentication codes or encryption may be warranted.

There is no inherent or logical restriction which prevents a Sombrero kernel from spanning $2^{32}$ nodes each having $2^{32}$ bytes of local memory. Separate implementations of a Sombrero kernel can communicate with other Sombrero kernels via traditional Inter Process Communications and Remote Procedure Call methods now in use in conventional process oriented settings.

In overview, the tradeoffs for Sombrero on a WAN are the necessity of increasingly complex mechanisms to supply fault tolerance and security versus the advantages of a single namespace. The latter becomes increasingly desirable for 128 bit addresses where, for example, unique worldwide IP and node socket addresses could be easily managed and every byte of data in every database on earth could have its own VA.