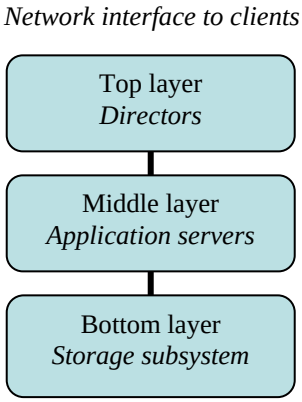
Using Sharing to Simplify System Management

Michael D. Schroeder
Microsoft Research Silicon Valley
9 April 2003

The cost of ownership for many computer systems in non-home environments is dominated by ongoing system management. This paper addresses the management issues around storage-intensive systems that serve many network-attached clients, particularly file servers, mail and calendaring servers, and database servers. The paper begins by describing a three-layer structure for large server systems that is often employed where availability and scale considerations require the use of multiple computers to implement a single service. It then contrasts a system organization called the uniserver model, in which the permanent state is partitioned among the application servers, with an organization called the multiserver model, in which the permanent state is shared among all the application servers. Reviewing the relative advantages and disadvantages of the two models suggests the option of combining them by using a multiserver as a uniserver. The sharing from the multiserver model makes a system easier to manage than a uniserver. But if the sharing is avoided in normal operation, as in a uniserver, then the combined system avoids many of the drawbacks of both models.

Three-layer systems

A useful structure for a multi-computer system that maintains significant permanent state and has network-attached clients is to organize the hardware components into three layers by function.



At the bottom is the storage subsystem, consisting of large numbers of disks and their controllers. These days the storage subsystem is usually interconnected with a storage area network, such as Fibre Channel, to which all the computers are also attached. In the middle layer are the computer systems that implement the service: file servers, mail servers, or database servers. At the top layer are the computer systems that front the system to the network. They collect client requests from the network and distribute them

to the middle-layer computers. These top-layer computers can be simple directors that pass requests directly to the middle layer, or they can be web servers that implement the visible interface and formulate the needed middle-layer requests.

The bottom-layer storage subsystem is usually responsible for reliably storing the permanent state, although in some systems the middle layer participates, too. Reliability is achieved by using data redundancy techniques such as RAID, as well as by replicating the controllers and network components. Data is usually backed up to offline media. In addition, if the bottom layer provides storage virtualization then it can do things like load balancing to improve performance. This possibility is discussed later.

The three-layer system organization can be used in two different ways: uniserver or multiserver. With the uniserver model, each middle-layer server acts on a unique partition of the permanent state of the system stored by the bottom layer. With the multiserver model, each middle-layer server can act on all of that permanent state. These two models have different strengths and weaknesses.

The uniserver model

Today, the most common organization for dividing the work among the middle-layer servers is to partition the permanent state of the system among them. I call this organization with partitioned state the uniserver model.

For example, in a file system, different sub-trees of the naming hierarchy will be implemented by different file servers in the middle layer; in an email system, different sets of user accounts will be implemented by different mail servers; in a database system, different tables of the database will be implemented by different database servers. The top-layer directors understand the partitioning scheme and direct each request to the middle-layer server that “owns” the permanent storage needed to answer it. Sometimes a request needs to be divided into several pieces, each of which is directed to a different server, and the results combined in order to respond to the client, although atomicity is usually not provided for requests spanning multiple servers

As the patterns of client requests evolve and as the system state grows, it is sometimes necessary to redistribute the state among the middle-layer servers and add new ones to maintain good performance. Available tools can detect load pattern changes and overloaded servers, suggest optimal partitioning of the state, and reorganize the storage layers to achieve the optimum.

Another reason for changing the distribution pattern is the failure of a middle-layer server. If high availability is a goal, the system will be provisioned with extra standby servers to take over from failed servers. Failover requires detecting the failure, detaching the associated permanent state from the failed server, attaching it to a standby server, starting the standby server with the transferred state, and cleaning up any unfinished business found in that state. The top-level directors are then told to direct requests to the new owner of that partition of the permanent state.

There is some global shared state in a uniserver system: the list of member servers and the characterization of the partition of the data they each serve. This global state needs to change when the partitioning is changed and when failover occurs, but it changes infrequently and is small.

The uniserver organization is sometimes called the shared-nothing approach because middle-level servers share no permanent state. The partitions of the permanent state are attached to one middle-level server at a time. The shared-nothing approach was once mandatory, since there were no storage interconnects that enabled disks to be accessed by more than one computer at a time. But as this constraint has been removed by the march of technology, people continue to argue the enduring value of the shared-nothing approach. The shared-nothing model is used widely in commercial products. For example, Microsoft's SQL, Exchange, or NTFS servers deployed on Microsoft Cluster Server [1] are examples of this organization. I use the term "uniserver" instead of "shared-nothing" because uniserver contrasts better with its alternative, the multiserver model, discussed below.

The multiserver model

An alternative to uniserver model for a three-layer systems is the multiserver model. In this approach all middle-layer servers in a system can operate on all the permanent state contained in the bottom-layer storage subsystem. For example, with a multiserver file system a single (large) hierarchical name space is served by all servers in the middle layer. Any of the servers can operate on any folder or file. Over the last ten years or so, progress in storage area networks, systems area networks, and local area networks has made shared access to storage affordable and scalable with good performance. With the multiserver model the top-layer director function is still required in order to do load balancing and avoid failed servers. In the case of a multiserver, however, the directors can make dynamic decisions that are not completely dictated by data location.[1] The multiserver model makes extra demands on the implementation of the middle-layer servers. In particular they need to coordinate their access to the permanent state. Coordination is usually done using a global locking service that allows middle-layer servers to set locks on portions of the permanent state. A lock prevents conflicting access from other servers. Choosing the best granularity for the locking, *e.g.*, per folder, per file, or per byte range in the case of a file service, depends on the pattern of expected client requests. The need to coordinate also complicates the management of data caches in the servers.

---

1 For both uniserver and multiserver systems it is possible to put the director function in a clerk module in the clients. The clerk module retrieves configuration information directly from the middle-level servers and uses it to send each client request directly to the appropriate server. With this structure the top-layer directors are bypassed. Clerk modules work best when clients are modest in number and well-connected to the server system. For large-scale systems with many distant clients, it is best to have the director run on top-layer servers of the system, as described here.

Expanding a multiserver system is done by attaching a new middle-layer server to the storage subsystem, updating the membership list so that the top-layer directors know about the new server, and letting the new server initialize itself by reading from the permanent state.

Failure of a middle-layer server can be covered by directing requests to another server, because all servers can operate on all parts of the permanent state. When servers encounter locks still held by the failed server, they must take a special action to recover the lock and complete or abort the operations it protected. This is similar to cleaning up the unfinished business of a failed server when doing failover for a uniserver system. In both cases the new server reads and acts on the operation log written by the failed server.

Multiserver systems have been around for some time. An early successful example was the DEC VAX/VMS cluster [2] that provides a multiserver file system. More recently the Frangipani global cluster file system prototype [3] has demonstrated good performance and automatic operation using these techniques.

Arguments in favor of uniservers

Uniserver systems realize several benefits directly from their organization. By having each partition of the permanent data under control of a single server, undesirable interactions among the servers are minimized. Each server has a free hand in managing and caching that data and in accessing the permanent state without the interference of other servers. Lack of interference can lead to good performance. It also allows the server to be “near” its data, in the sense that the connection from the data storage subsystem to the server for the associated partition doesn’t need to go anywhere else until a failover or repartitioning occurs. When a server crashes it cannot affect the operation of other servers or other partitions. This lack of unwanted interaction contributes to system stability. Finally, failover is an activity confined to the chosen standby server, without system-wide repercussions other than temporary unavailability of data from the affected permanent state.

Arguments against uniservers

The partitioning of the permanent state that characterizes the uniserver organization also generates some problems. Perhaps most important is that partitioning is a major management burden in operating such a system. Growth in the load, changes in the access patterns, and growth of the permanent state require repartitioning the system. Such repartitioning can involve copying the data. Repartitioning can be time intensive and can take the system entirely or partially offline. In typical implementations failover is slow: getting the standby server up to speed from scratch can take minutes. Addressing this problem by having a hot standby mitigates some of the simplicity and non-interference advantages mentioned earlier.

Arguments in favor of multiservers

Multiserver systems also have their benefits. Requests can be dynamically distributed according to load. Requests to read-only hot spots in the data, for example, can be satisfied from multiple middle-level servers in parallel without any pre-positioning of the data. The needed data would find its way from the shared storage subsystem into the caches of all the servers where it could be accessed rapidly at each, increasing throughput of the overall system. Recovery from the failure of a middle-level server can be fast because all other servers are automatically "hot." Repartitioning the permanent state is never necessary since all servers can access all permanent state. The result is that management overhead for such a system is low and there are no lengthy outages for reconfiguration.

Arguments against multiservers

Problems with the multiserver organization include interference between servers needing temporary exclusive access to the same data. Such lock conflicts can result in unpredictable performance. Also, implementing the global lock service as a high-performance, scalable, distributed program is complex. With a multiserver system, failures can impact the operation of all other servers as they recover the locks held by the failed server and take over the load. Another negative for the multiserver organization is that many existing commercial file servers, mail servers, and database servers are not designed to share access to their permanent data. While there is general ignorance about how hard fixing this would be, it clearly would be a major development task. Finally, the storage subsystem has to be able to deliver all permanent state to all servers with good performance, a requirement that has been difficult to achieve.

Using a multiserver as a uniserver

It seems possible to combine the advantages of the uniserver and multiserver models and lose most of drawbacks. The idea is to use the top-layer directors and distribution tables from a uniserver system on a multiserver system with the same permanent state. The uniserver directors will route requests in a pattern that prevents the multiservers from sharing items from the permanent data, even though they could share. Under this scheme, at system start-up or reconfiguration there would be an initial flurry of activity at the global locking service while each application server collects the locks it needs as requests come in. There would never be contention for these locks, since the directors are implementing the same routing decisions that they would for the uniserver system having a partitioned permanent state. Eventually lock requests would largely stop occurring as each server obtained all the locks it needed. The steady state would be characterized by a background level of lock renewals without contention. The performance concerns surrounding contention in a multiserver system would not surface with this scheme.

But have we gained any of the advantages of multiservers? I think we have. Repartitioning, scaling, and failover can happen faster and with less management intervention or service disruption in a multiserver system. Consider each in turn.

Repartitioning — As with a uniserver system, monitoring tools watch for signs that "repartitioning" is needed. In addition to server load, lock contention is a good tell tale. For the multiserver, however, repartitioning is accomplished by changing only the routing pattern implemented by the top-layer director computers. No changes in the organization of the storage subsystem are required. As the middle-layer servers start seeing requests that require access to new parts of the permanent state they obtain the corresponding locks and fulfil the requests. The previous lock holders release their claim because of these requests for contending locks from other servers. After some interval, locking service activity would drop to a background level again. The system continues offering service while the reconfiguration is stabilizing, perhaps with some small loss of performance due to increased locking traffic.

Scaling — An added middle-layer server attaches itself to the permanent storage of the system and internalizes the meta-data it needs to commence operation. All state needed is available to the new server either in the shared storage subsystem or in the membership list and locking service. The membership list for the system is updated to record the new server and the distributors adjust the routing algorithm to allow the new server to operate on a virtual partition of the permanent state. Again, no management intervention is required other than policy direction as appropriate. The system continues to provide service during scaling.

Failover —A failure of a middle-layer server is detected by monitoring mechanisms that are largely similar in the uniserver and multiserver cases. Once detected, the multiserver system adjusts the routing decisions made by the top-layer distributor computers to effectively assign the failed partition of the permanent data to one or more other servers. In the multiserver case, as with the uniserver case, there can be standby servers waiting to receive the load. Lock redistribution follows until the locking service activity quiesces in the new state. When acquiring broken locks abandoned by a failed server, a new server inspects the operations log of the failed server, available from the storage subsystem, to determine the cleanup actions required.

In summary, use of the multiserver organization, but with directors that minimize or eliminate actual sharing among active middle-layer servers, can substantially reduce the cost of management for such systems without much impact on system performance, reliability, or cost.

More on the storage subsystem

As described so far, the bottom-layer storage subsystem is a collection of disks, controllers, and network components with the property that all middle-layer servers can access all disks. Storage reliability is achieved by the use of redundancy within the storage subsystem. This black-box model of the storage subsystem is appropriate for discussing the distinction between uniserver and multiserver systems. Achieving minimum intervention management and good performance for the overall system, however, may demand additional functionality from the storage subsystem. The extra

features are equally useful in uniserver and multiserver systems. In particular, it may be useful for the storage subsystem to implement load balancing, incremental growth, and failover on its own. The key technique for adding these features is storage virtualization, in which the storage subsystem implements one or more virtual storage volumes that are addressed like very large disks. A mapping from the blocks of the virtual volume to the physical storage hides the redundancy scheme and the distribution of the data among controllers and disks.

Using a volume virtualization scheme, failed disks and controllers can be replaced and new disks and controllers added on demand. The only change the middle-layer servers see is that the virtual volumes get larger. Automatic algorithms operating in the background copy data among the attached disks to achieve capacity and load balance and to restore the desired level of data redundancy. No management intervention is required. Operator intervention is required to replace or add hardware, but not to configure it. The Petal storage management system [4] is one example of this kind of storage virtualization.

A shared storage subsystem with volume virtualization clearly would be an asset to a uniserver system as well as a multiserver system and would mitigate some of the management burden associated with uniserver systems.

Discussion

The multiserver organization requires distributed systems software in addition to shared physical access to the storage subsystem. Over the last ten years considerable progress has been made on this software technology. There now are good algorithms for the global state management needed to maintain the system membership list. Perhaps the best algorithms are those in the Paxos family.[5] A global locking service built using leases and depending on server operation logs for lock recovery, as in the Frangipani example, can have good performance and scaling characteristics. This design is a simplification of the traditional distributed lock manager [6]. Because a partitioned multiserver system operates in a way that minimizes or eliminates actual sharing, the locking service and the coordination mechanisms for the server data caches are not stressed.

The distinction between uniserver and multiserver systems focuses on two ends of a spectrum of implementations. Many of the ideas I have associated with multiservers can be applied in some form to uniservers. For example, in a uniserver using standby servers for failover, the idea of hot standbys can be pushed to the point where the standby server is tracking the active server, operation by operation, so that its internal state is almost complete and up-to-date when the failover occurs. This can make failover faster. In this case the experienced system designer will be wary, however, since we would be adding a special purpose mechanism used only to support the unusual case of failover, whereas the similar machinery in a multiserver would be part of the base functionality of the system and thus more likely to be correct.

Summary

In this paper I have argued that sharing is a good organizational technique for a multi-computer server system, especially if the system is configured so that sharing is not on the critical path of high-volume operations. Instead, the sharing mechanisms can make the inevitable system transitions caused by reconfiguration, failure, and growth fit more seamlessly into system operation, minimizing the management attention required to perform them. A system organization that combines the good features of the uniserver and multiserver models has the potential to realize this goal.

Acknowledgements

These ideas have benefited from discussions with my colleagues Kurt Friedrich, Jim Gray, Chandu Thekkath, and Chad Verbowski. In addition Ulfar Erlingsson, Andrew Herbert, Michael Isard, Bill Laing, Butler Lampson, Roy Levin, Fred Schneider, Leslie Schroeder, Chuck Thacker, and Lidong Zhou made useful suggestions.

References

1. MICROSOFT, Windows server 2003: server cluster architecture, available as http://www.microsoft.com/windowsserver2003/docs/ServerClustersArchitecture.doc
2. KRONENBERG, N., LEVY, H. AND STECKER, W., 'VAXClusters: a closely-coupled distributed system,' *ACM Transactions on Computer Systems*, vol. 4 no. 2 ,May 1986, 130-146.
3. THEKKATH, C., MANN, T. AND LEE, E., 'Frangipani: a scalable distributed file system,' *Proc. 16th ACM Symposium on Operating Systems Principles*, ACM ,October 1997, 224-237.
4. LEE E. AND THEKKATH, C., 'Petal: distributed virtual disks,' *Proc. 7th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS-VII*, ACM, October 1996, 84-92.
5. LAMPORT, L., 'The part-time parliament,' *ACM Trans. on Computer Systems*, vol. 16 no. 2,May 1998, 133-169.
6. SNAMAN, W. JR., AND THIEL, D., 'The VAX/VMS distributed lock manager,' *Digital Technical Journal*, vol. 1 no. 5, Sept 1987, 29-44.