Scaling Web Applications on Server-Farms Requires Distributed Caching

A White Paper from ScaleOut Software

Dr. William L. Bain
Founder & CEO

Spurred by the growth of Web-based applications running on server-farms, a new category of data is beginning to play a key role in driving high performance. Called *workload data*, this information claims the unique properties of being simultaneously mission critical, frequently accessed, and often short lived. Workload data can either be generated by the application or it can represent data cached from recent access to a database server. Examples include intermediate business logic state, Web session-state, and cached data sets.

This unusual combination of characteristics has created the fast-growing need for applications to store workload data in a distributed cache that is shared across the server-farm. A distributed cache holds key workload data in memory for high performance. It also allows simultaneous access from any server in the farm to simplify application design. Distributed caching enables server-farms to meet the new performance, scalability and availability challenges created by the management of workload data.

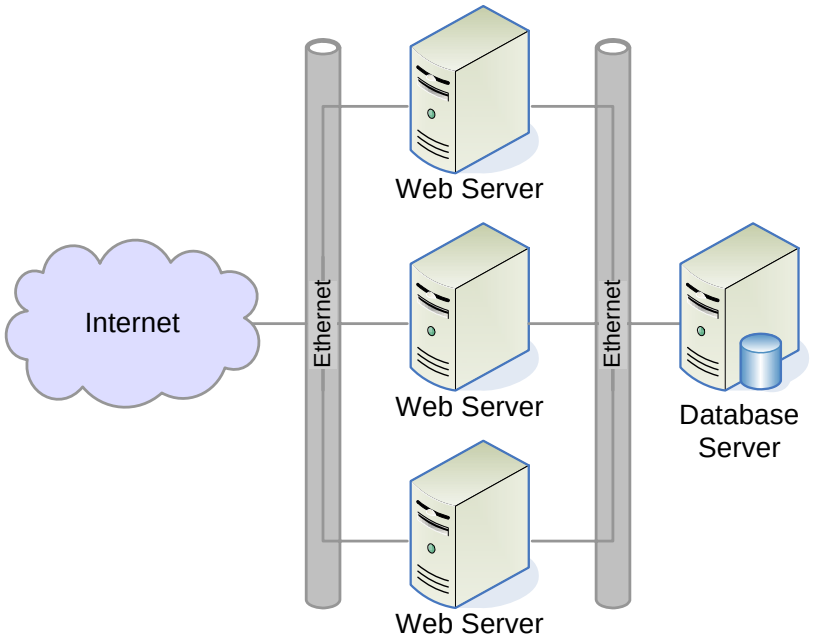
This white paper examines the need for distributed caching and its importance in building server-farm applications. It describes ScaleOut StateServer, a distributed cache for the .NET platform, and shows how it can be used to easily integrate distributed caching into the design of new applications.

The traditional method of storing fast-changing application state in database servers no longer meets the needs of today's server-farms. The growing use of server-farms in mission-critical applications has made distributed caching a key requirement in delivering high performance and scalability.

Growth of Server-Farms

Over the past several years, server-farms have rocketed in popularity because they remove a crippling barrier to the growth of Web-based applications. By enabling applications to both scale in capacity and provide non-stop availability, server-farms have played a major role in the rapid expansion of Web-based computing. The emergence of hardware and software IP load-balancers in the late 1990s catalyzed the development of server-farms by transparently distributing incoming client requests among the servers of a farm. However, Web-based applications and services were not designed to run on server-farms. As a result, all application data had to be stored in a database server so that the application could retrieve it from any server in the farm.

This constraint limits the application's performance and scalability, and it imposes an additional load on the database server. In essence, server-farms move the performance bottleneck from the Web server to the database server.



Typical Web Farm Architecture

Distributed caching solves these problems by enabling Web-based applications to maintain memory-based state and avoid repeated round-trips to the database server. For distributed caching to be effective, application state must be uniformly accessible from all servers. This enables the IP load-balancer to distribute client requests to any server in the farm. Distributed caching thereby creates a breakthrough in application performance, and it eliminates a bottleneck at the database server. It also can help maintain the simplicity of application design.

Workload Data

For Web-based applications, workload data represent the application's state information that must be maintained between client requests. For example, e-commerce applications use shopping carts to record each customer's progress in a shopping session. Online banking applications might maintain pro forma account balances and login state to back-end mainframe servers during customer sessions. In addition, all Web-based applications, typically maintain cached database information, such as category lists, for rapid access in producing output pages.

Other types of server-farm and grid applications also make use of workload data that represent the state of a distributed application. For example, scientific modeling and weather forecasting applications keep application state in the form of a distributed "grid" that is partitioned among the servers. Likewise, multi-player games often distribute the various regions of a game and its players among the servers of a farm.

All of these examples of workload data share a common set of unique characteristics that are important in their usage on server-farms:

- *mission-critical*: Loss of data can have a significant impact on the overall application; for example, loss of a shopping cart can result in the loss of a sale in an e-commerce transaction.
- *short lived*: The data are only used for the duration of a session or of the overall application, and once a transaction is committed, they may no longer be needed; for example, after an e-commerce purchase is made, the shopping cart is removed. Note that some workload data cached from a database server may persist beyond the duration of an individual session.
- *frequently accessed*: Workload data are frequently accessed and updated during their relatively short lifespan; for example, the state of an online banking transaction may change rapidly as the user progresses through a session. Unlike cached database data, fast-changing workload data have a much higher ratio of writes to reads.

The characteristics of workload data present a striking contrast to long-lived, line-of-business (LOB) data, such as accounting information, customer profiles, scientific data, census information, and other similar data, which are typically held in database servers. The following table shows how these two types of data tend to differ:

	LOB Data	Workload Data
Volume	High	Low
Lifetime/turnover	Long/slow	Short/fast
Access patterns	Complex	Simple
Data preservation	Critical	Less critical
Fast access/update	Less important	More important

Comparison of Workload Data to LOB Data

These differences create new challenges for effectively storing workload data on server-farms.

Traditional Approaches for Storing Workload Data

As the need to efficiently manage workload data on server-farms has become increasingly important, Web architects have explored various approaches to this problem, each of which has important shortcomings:

- *client-side cookies*: This approach, which is often used in client/server systems, transparently stores session data on the client system. Since the information must travel back and forth between client and server on each page access, the data size must be kept very small, and security can be a challenge. This approach is only suitable for client-server applications.

- *in-memory storage on the server*: This technique stores workload data in memory on the server. For client-server applications, it requires that session affinity to a particular server be maintained by the load-balancer, and this limits scalability. Many ISPs routinely change IP addresses mid-session, which can cause sessions to be lost. Also, a server cannot be taken down for maintenance until all sessions are drained from it. Two additional limitations of this approach are that in-memory storage is lost if a server fails, and this storage is not accessible to other servers in the farm.
- *memory-based storage server*: This solution uses a dedicated server to hold workload data for a server-farm and solves the issues associated with session affinity by making workload data accessible to all servers. However, it also introduces a single point of failure that can make all session data unavailable farm-wide in the case the storage server fails. Also, this method of storage does not scale with growth in the server-farm and can easily become a performance bottleneck for larger server-farms.
- *database server*: Because database servers have been optimized over many years for reliably storing and searching large volumes of long lived, line-of-business data, they typically are not well suited for holding workload data. The repeated reads and writes of workload data from a database server tends to impede application performance and slow down the database server. Also, unless the database server is running on a server cluster, it represents a single point of failure for the farm. Because clustering tends to be expensive and complicates management of the farm, it usually is only implemented in very high-end operations.

The limitations of these traditional storage techniques can be summarized in the following table. Note that none of these techniques simultaneously solves the challenges of providing performance, scalability, and high availability. The table also shows that distributed caching meets all these challenges; this will be explored in detail in the next section.

	Required for Server-farms		
Storage method	**Performance**	**Scalability**	**Hi-av**
Client-side cookies		✓	
In-process	✓		
Memory-based server	✓		
Database (not clustered)	✓		
Database (clustered)	✓		✓
Distributed caching	✓	✓	✓

Comparison of Storage Methods on Server-Farms

Because there was no choice in the past, Web architects were forced to choose among the traditional alternatives for storing workload data. However, these methods of

storing workload data are no longer sufficient for serious applications running on server-farms. The growing use of server-farms in mission-critical applications has made the effective handling of workload data increasingly important in maintaining high performance and availability.

Why are each of these techniques, insufficient for today's server-farms? The potentially voluminous and sensitive nature of workload data rules out the use of client-side cookies as a general solution; cookies are primarily used only to hold login information. In-process storage is too volatile for mission-critical workload data, and its use impedes the IP load-balancer's ability to scale performance. A stand-alone memory-based server or database server introduces a new bottleneck to performance scaling, and it also creates a single point of failure. Clustered database servers avoid the single point of failure, but they do not remove the performance bottleneck. The key limiting factor for scalability on server-farms is the performance bottleneck for workload data storage.

The Next Generation: Distributed Caching

The recent recognition of workload data as being uniquely demanding has required the development of new technology. To maximize performance and scalability, workload data should be kept as close as possible to the application that uses it and ideally in memory on the server-farm. This can be accomplished using a distributed, in-memory cache readily accessible to applications on all servers. Distributed caching boosts performance and eliminates the bottleneck of repeatedly saving and retrieving workload data in a database server. It also minimizes overall cost by avoiding the use of expensive clustered database resources. Interestingly, this approach has been successfully used for many years to maximize performance in high-performance scientific computing.

However, keeping workload data in a distributed, in-memory cache creates three important new challenges that the cache designer must solve for this approach to be viable:

- *scalable access*: To avoid any bottlenecks to scaling performance as the server-farm grows, workload data must be automatically distributed, or *partitioned*, across the server-farm for parallel access. This allows applications running on all servers to simultaneously access different stored data. Partitioned data must be dynamically load-balanced among the servers to avoid creating hot spots in the farm over time.
- *uniform accessibility*: Workload data must be uniformly accessible to all servers in the farm. This allows an application to access workload data that may be stored on a different server, and it enables the load-balancer to direct client requests to any server. By doing this, all workload data can be readily accessed independent of where it is stored on the server-farm.
- *high availability*: Storing workload data in a distributed, in-memory cache also requires that it be able to survive the failure of a server. The cache must copy, or *replicate*, workload data to additional servers in order to avoid data loss if a server becomes unavailable. However, to maintain scalability, workload data

must not be replicated to all servers; otherwise, storage requirements would increase too quickly as the server-farm and its workload grows.

To be effective in managing workload data on a server-farm, the distributed, in-memory cache must incorporate and integrate these key mechanisms, i.e., load-balanced data partitioning, uniform accessibility, and replicated storage. Together, these mechanisms, along with numerous capabilities such as optimized internal buffering and storage strategies, enable applications to quickly store and access workload data, and they help maintain uninterrupted service after failures occur or as servers are added to the farm.

Implementing these mechanisms in turn requires that the distributed cache be able to effectively coordinate its actions on all servers and to detect and recover from server failures. This in turn requires that the distributed cache implement an efficient error detection and recovery mechanism that allows it to detect the health of servers within the farm and to heal the distributed cache when a failure occurs. By doing so, the storage architecture can present a very simple and compelling view to applications that hides the complexity involved in handling these distributed computing issues. In fact, applications need not be aware that they are running on a server-farm and that their workload data has been intelligently distributed across the farm to scale performance.

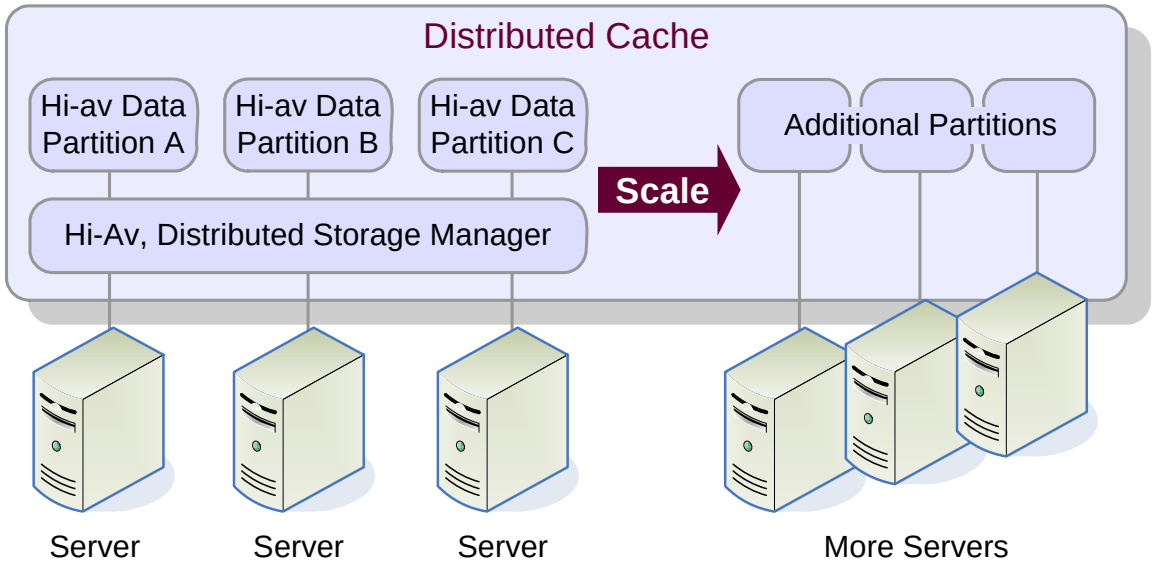
Another advantage of a distributed, in-memory cache is that applications can keep workload data in its natural, object-oriented form and avoid converting it to a relational form for storage in a database server. A distributed, in-memory cache typically stores workload data as an opaque, binary stream of data that has been automatically serialized from the objects that the application manages. No additional application code is necessary to convert an application's objects into the form needed for out-of-process, cache storage.

ScaleOut StateServer

ScaleOut StateServer™ from ScaleOut Software provides distributed caching for server-farms. This product runs as a software service on a Web or application server-farm and implements the cache within the farm's random access memory (RAM). It can be used to create, read, update, and remove opaque workload data objects based on an identifying key. Stored objects either have been "serialized" from datasets or record sets that were previously accessed from the LOB database, or are generated as business logic objects.

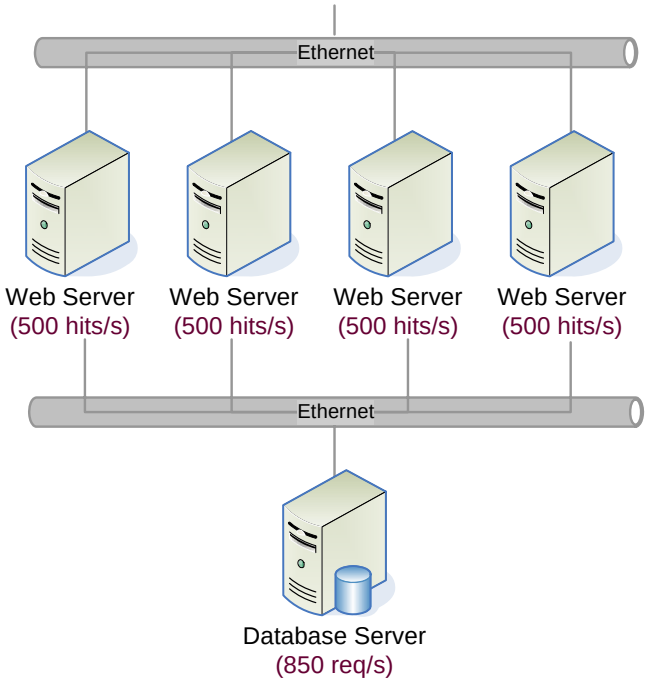
To ensure scalable performance and high availability, ScaleOut StateServer incorporates the three key mechanisms of data partitioning, uniform accessibility, and replicated data storage. It delivers scalable throughput by partitioning and dynamically load-balancing workload data across the servers within a farm, as illustrated in the following diagram. It works seamlessly with an IP load-balancer and enables every server to access any workload data object stored in the cache. Using patent-pending technology, it efficiently replicates all data across selected additional servers to ensure high availability and requires less than ten seconds to failover after a server failure (versus one or more minutes for a clustered DBMS).

When a new server is added to the farm, ScaleOut StateServer automatically integrates the server into the distributed cache, and a portion of the load is migrated to it. The cache automatically self-heals after a server failure to restore full redundancy. The product employs a peer-to-peer architecture that eliminates the use of a master server and enables the distributed cache to recover from multiple server failures.



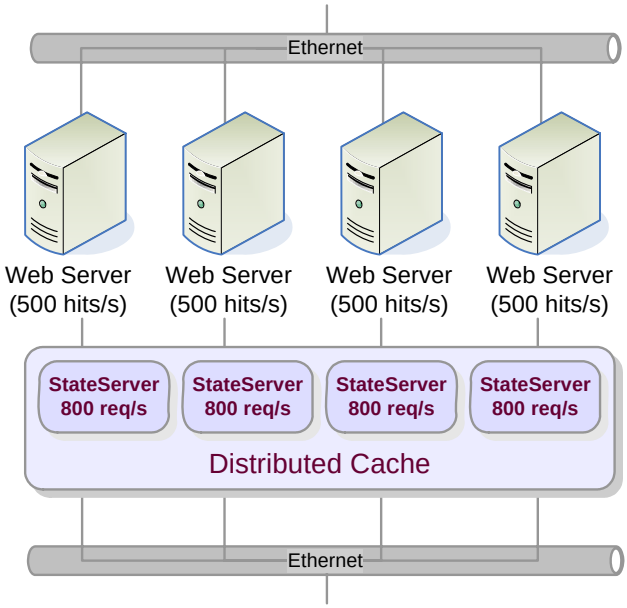
Data Partitioning in ScaleOut StateServer

The distributed, in-memory cache provides much faster response time than a database server by eliminating the need for all servers to repeatedly access a shared database. For example, consider the scenario of using a distributed cache for an e-commerce Web farm's session-state objects. Although the aggregate Web server load grows with the size of the farm (for example, 500 hits per second per server), the backend database server handles a fixed maximum request rate (850 requests per second in this example). If the farm doubles in size, the database server can only deliver half the original throughput to each Web server. In addition, each database update request may require a millisecond or more to write to disk and then confirm completion, and the database server's throughput decreases as the database table grows to accommodate many user sessions. As the Web server-farm grows in capacity to meet demand, the database server must be migrated to faster and more expensive multiprocessor hardware to increase its throughput rate.



Web Farm Scenario with a Conventional DBMS Server

Compare this scenario to the use of ScaleOut StateServer hosted directly on the Web server-farm. The distributed cache's throughput grows as the sum of the throughput for each storage partition (800 requests per second in this example) so that overall throughput and application response time per Web server remains constant as the farm grows in capacity. Unlike a database server, the distributed cache's throughput increases to match the growing workload on the server-farm. Further, each update request completes more quickly since no disk access is required to commit the update.

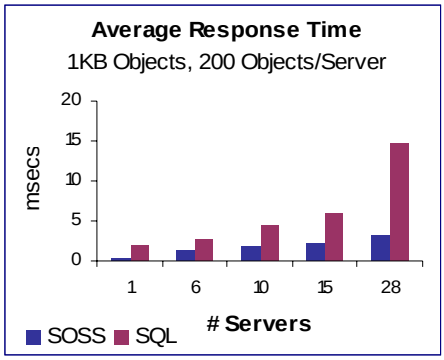


Web Farm Scenario with ScaleOut StateServer

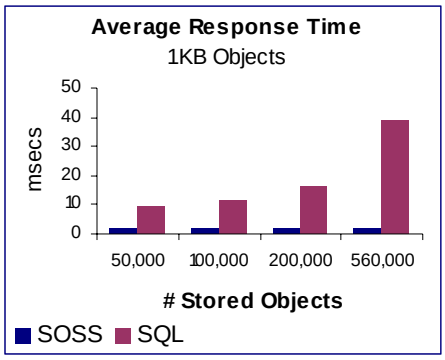
Performance Tests

To measure the performance advantage of a distributed, in-memory cache, such as ScaleOut StateServer (SOSS), over a database server (DBMS) for storing workload data, a .NET test application was used to compare response times during repeated accesses to these two storage mechanisms. This test measured the average response time for a repeated sequence of reads and updates to a pool of stored objects. The tests were run while varying the following parameters: object size (10B to 1MB), number for stored objects (2K to 5.6M), and number of servers.(1-28). (Note that for a Web site storing session-state, the number of stored objects corresponds to the number of concurrent users on the site.) The server-farms consisted of IBM BladeCenter® servers with 3-3.5 GHz CPUs and Gigabit Ethernet. DBMS accesses were made to an IBM xSeries 366 server with 2-4 3.6 GHz CPUs,

The test results clearly demonstrate the performance advantages of using a distributed cache for storing fast changing workload data. The first chart shows the average response time for accessing 200 objects per host on a farm with from one to 28 servers. SOSS's response time (blue bars) remains low as the farm and its storage load grows, while DBMS response time (red bars) grows with additional load. SOSS's scalable throughput delivers performance gains ranging from 2.3 for 6 hosts to 4.5 times for 28 hosts.



The performance advantages of using a distributed cache increase as more objects are stored. The second chart shows the average response time for stores with 50K, 100K, 200K, on a 10 server-farm and for 560K objects on a 28 server-farm. SOSS's response time (the blue bars) remains low as more objects are added, while the DBMS's response time grows significantly. SOSS's performance gain, which ranges from 4.6 to 16.7 times, results from the distributed cache's ability to distribute the workload for accessing and updating objects across many servers. In contrast, for each access request, the DBMS must search a single database table that grows as more objects are stored. In tests with 10KB objects, SOSS lowered average response time by up to 25.3 times in comparison to the DBMS. This enables online applications such as very large Web sites to maintain fast response times while handling workload data for very large numbers of concurrent users.



Summary

As companies move more and more key business applications to the Web, server-farms have become increasingly critical in maintaining the scalability and high availability of these applications. Also, as applications manage increasing volumes of workload data, effective storage techniques must be employed to avoid bottlenecks to scaling or loss of

mission-critical data. Traditional approaches to storing workload data, in particular, the use of a database server, do not meet today's requirements for delivering high performance on server-farms.

Recent breakthroughs in workload data storage have eliminated the need to make a trade-off between performance, scalability, and high availability. Distributed, in-memory caching hosted directly on the server-farm, offers the opportunity to simultaneously meet all of these requirements and also to reduce the load on expensive database resources. However, this exciting new storage architecture presents challenges of its own. By integrating the techniques of data partitioning, uniform accessibility, and data replication, distributed caching can be used to dramatically scale the performance of server-farm applications while maintaining or even simplifying the application's view of the storage architecture.

ScaleOut StateServer, a software product for distributed caching from ScaleOut Software designed for .NET server-farms, incorporates patent-pending technology for scaling performance and replicating data. These features enable it to deliver highly scalable performance while maintaining high availability for today's server-farms. Measurements have shown that ScaleOut StateServer's performance quickly outpaces the performance of a database server as the load on a server-farm grows. By using a distributed, in-memory cache such as ScaleOut StateServer to provide the workload data storage, application architects can be assured of a server-farm that will meet all the needs of their latest Web-based applications.

About ScaleOut Software: ScaleOut Software was founded in 2003 by William L. Bain. Bill has a Ph.D. (1978) from Rice University and has worked at Bell Labs research, Intel, and Microsoft. Bill founded and ran three start-up companies prior to joining Microsoft. In the most recent company (Valence Research, Inc), he developed a distributed Web load-balancing software solution that was acquired by Microsoft and is now called Network Load Balancing within the Windows Server operating system.

SCALEOUT SOFTWARE, INC.

Headquarters
10900 NE 8th Street
Suite 900
Bellevue, WA 98004
425-450-3216

Sales and Support
15075 SW Koll Parkway
Suite J
Beaverton, OR 97006
503-643-3422

Email:
sales@scaleoutsoftware.com
support@scaleoutsoftware.com
Web:
www.scaleoutsoftware.com