Maintaining HIS Performance

Distributed caching provides scalability that can relieve overloaded server farms and re-invigorate listless systems.

By Iqbal Khan

As Web and service-oriented architecture technologies become more popular, hospitals are adopting them rapidly for automating their operations and allowing their administrative staffs, doctors and patients to directly access the hospital information system (HIS). While this has introduced a lot of efficiencies in hospital operations, it has also introduced new headaches for hospital IT management due to an increased load on the HIS.

An HIS consists of an integrated set of Web applications automating hospital operations. This includes applications managing administrative, financial, clinical, radiology and lab operations. There are usually thousands of different people using this system at any given time. However, despite a large number of users, everybody expects the system to perform well even during peak usage hours.

Hospitals try to address scalability by creating Web farms consisting of multiple Web servers tied together through a load balancer to distribute user load. Yet, as the number of users and transactions grows, various data access bottlenecks usually occur thus considerably reducing the performance of those operations.

Typical solutions for resolving this challenge, such as adding more servers, fall short and don't provide the required scalability for sustaining HIS at their normal performance during peak usage. Scalability simply means keeping the same system performance even during peak usage times.

Mission Critical

A hospital must always be operational; therefore, most applications in an HIS are considered mission critical. However, applications with a large number of simultaneous users are more prone to downtime or scalability issues. One such application is a clinical information system (CIS) that deals with electronic medical records. A CIS concentrates on patient-related and clinical-state-related data and is used by hospital staff, patients and even outside partners. This can easily account for thousands if not tens of thousands of people.

Due to its large number of users, a CIS is the most likely candidate for having scalability issues. It can be regarded as a mission critical application because doctors, nurses, patients, business offices and medical records personnel constantly rely on it. Therefore, it must provide 100 percent uptime and with no noticeable drop in performance. When a patient management system slows down due to scalability problems, a domino effect within hospital operations adversely affects business costs.

As thousands of users log in and access the same information performance lags, resulting in sluggish response times for end users. A typical user expecting a two to five second response time experiences 30- to 60-second response times instead, and as the load on the system grows one or more of the Web farm servers may crash.

A Web server can crash if its CPU or network card is constantly using nearly 100 percent of its capacity, or if the CIS application constantly requires more memory than the server has available. All of these conditions can occur if the load on a server goes above a certain threshold and remains there.

Even more serious is the fact that though there may be many Web servers in a Web farm, there is usually only one database server. Although a Web farm can grow, a database server cannot scale in the same fashion. The typical response to this issue is to beef up the database server hardware; however, this only helps scalability slightly and very quickly the bottlenecks re-surface and become a serious problem.

Stumbling Blocks

The fundamental problem of data access bottlenecks has to do with the architecture of ASP.NET and Java Web applications. Increased user requests can be handled by adding more servers to the Web farm, to provide more processing power and more memory. However, data storage and data access always become performance bottleneck in any application; therefore, this bottleneck must be eliminated in order to increase CIS application scalability.

CIS applications use two types of data: general purpose/user specific application data that is stored in the database; and temporary user session data that is kept in the ASP.NET session state and used across multiple Web requests. It is important to note, that an application data storage problem cannot be fixed without an application developer's involvement because fixing the application's internal logic is required. Fixing a user session data bottleneck, however, is easy and requires no development effort. It is even possible to fix bottlenecks in third-party CIS applications that are already running.

How Does Cache Scale?

Distributed cache provides highly effective scalability. Typical applications read data from the database 70 to 90 percent of the time and write data to the database 10 to 30 percent of the time. Interestingly, a CIS application reads the same data over and over again within a brief period of time. When data is cached on the first read operation, all future reads for this data go to the cache instead of the database server. This way, the load on the database server is greatly reduced. Thus, the database server is free to handle additional users, thereby providing a great deal of scalability.

A distributed cache cluster can grow proportionately to a Web farm because it doesn't have the limitations of a transactional database server, which is unable to scale up in the same manner. If there are multiple database servers clustered together then they must be updated simultaneously, which reduces performance. One database server operating within a two to three server Web farm might be able to handle the load. However, as a Web farm grows a single database server can become unreliable and fail to meet a large hospital's IT requirements.

There are various reasons why a distributed cache can scale. One is that most cache operations are single item reads or writes (there are no transactions, query parsing and compilation, or complex updates). Second, is that application data is kept in an object form that an application uses and no effort is required for recreating these objects. Finally, a cache can afford to store data differently than a database due to its nature, and therefore it provides various caching topologies for data storage. All of this combined enables a cache to scale up very nicely compared to a database server.

For a 10-server Web farm, two cache servers in a cluster might be adequate. But, for a 50-server Web farm, it's most likely that 10 cache servers are needed if a 5:1 ratio between Web servers and cache servers is required.

Session State Storage Bottleneck

Microsoft provides three storage options for ASP.NET Session State: InProc, StateServer and SqlServer.

InProc storage is intended for Web applications that are running on a single Web server and puts user session data within an ASP.NET application's worker process. Therefore, a Web garden configuration (a single server multi-processor setup) with multiple worker processes is not supported by InProc. Hence, InProc is ruled out for most real-life environments involving two or more Web servers.

StateServer is the second storage option. It is a server process that keeps the user session data and therefore can handle Web gardens. A StateServer can run on each Web server or all Web servers can access a single common StateServer. However, both situations have serious scalability problems. Keeping a single StateServer for the entire Web farm is a single point of failure and also does not scale up at all as more Web servers are added to the farm. Keeping StateServer on each Web server requires the use of a "sticky session" in the load balancer. Sticky sessions greatly limit application scalability since the load balancer is forced to send all user requests to their original Web server where the ASP.NET Session State was created. This action is taken although that particular server might be heavily loaded and there may be another less-loaded server available to take on this request.

The third storage option is SqlServer where ASP.NET Session State is stored in an SQL Server database. However, this option also has major performance and scalability problems because the SQL Server database cannot scale up in the same manner as the Web farm.

Consequently, all Microsoft ASP.NET Session State storage options are inadequate when it comes to scalability. A hospital IT manager ends up with a situation where adding more servers to the Web farm does not address scalability due to bottlenecks in session state storage.

Database Bottleneck

In addition to ASP.NET Session State storage limitations, a CIS application also faces scalability issues when it comes to accessing application data in the database. Database trips are usually very expensive in terms of their impact on performance and scalability of the application. A CIS application usually makes a lot of database trips for reading and writing patient related data. In fact, for each user request, the application usually makes multiple database trips. Consider how many different types of data an application would fetch from the database just to show information about a single patient. Plus, fetching each type of data requires a separate database trip.

Thus, when an application makes so many database trips for each user request, the database quickly becomes the bottleneck as IT management tries to scale up. The higher number of database trips per user request also means that as the number of user requests increases, the number of database trips will increase at a much faster rate. This will make the database server a scalability bottleneck very quickly and it will grind to a halt.

Buying more powerful hardware will only help slightly in the beginning. Similarly, too many database servers cannot be added in a database cluster, either. This is because the nature of a database makes it very difficult to grow a cluster.

Caching In User Session Data

Distributed caching has emerged as a powerful solution to address this problem and is the ideal storage option for both application data and user session data. A distributed cache is an in-memory storage that can span multiple servers and scale up very nicely. A distributed cache is ideal for storing ASP.NET user session data. It can also be used as a temporary cache for application data to reduce expensive database trips that make a database a scalability bottleneck.

ASP.NET comes with a pluggable user session storage mechanism called Session State Provider (SSP). SSP allows IT managers to replace the built-in session storage with third-party storage solutions without any programming effort; and many commercial grade distributed caching solutions provide SSP plug-ins. This allows IT managers to incorporate one of the commercial grade distributed caching solutions and remove any storage bottleneck for user session data.

With the help of distributed caching for storing user session data, hospital IT management can stop worrying about its CIS applications not being able to scale effectively by simply installing additional Web servers to the farm and adding the required number of cache servers to the cache cluster to scale up to higher levels.

Distributed caching also provides hospital IT management data center disaster replication and thin client capability for grid computing. When a hospital data center goes down, users are immediately switched to the disaster recovery center and their user sessions are instantaneously available. The industry offers several brands of distributed caching software. By implementing distributed caching, scalability is achieved via the clustering of session caching servers.

Conclusion

Hospital CIS applications developed as Web applications either in ASP.NET or Java lack critical scalability to handle peak loads and user increases that are due to inherent data storage and data access bottlenecks, which also inherently limit database servers.

Scalability, however, based on distributed caching provides the vital balancing act for CIS applications. User session data is temporary and needed only while users are logged on and expires afterwards. Therefore, an in-memory distributed cache is an extremely scalable option for storing user session data.

The fewer trips to the database, whether for application or user session data, the faster and more scalable a CIS application becomes. As the load on the database server is reduced, the server is able to handle that many more user requests which helps scale an application. The end result is an environment that has unlimited scalability, allowing hospital IT to be immune from bottlenecks and severe slowdowns. **HMT**

Iqbal Khan is president of Alachisoft. Contact him at iqbal@alachisoft.com.