Introduction

What is IBM WebSphere eXtreme Scale and how does it work? Let’s take a two-pass approach to answer this question. First, an explanation as you might find it in the Information Center:

WebSphere eXtreme Scale operates as an in-memory grid that dynamically processes, partitions, replicates, and manages application data and business logic across hundreds of servers. It provides transactional integrity and transparent fail-over to ensure high availability, high reliability, and consistent response times. WebSphere eXtreme Scale is an essential distributed caching platform from IBM for elastic scalability and the next-generation cloud environments.

Elastic means the grid monitors and manages itself, allows scale-out and scale-in, and is self-healing by automatically recovering from failures. Scale-out allows memory capacity to be added while the grid is running, without requiring a restart. Conversely, scale-in allows on-the-fly removal of memory capacity.

Got that? If not, let’s look at an example and try again to explain what this groundbreaking product is all about.

Quite simply, the goal of WebSphere eXtreme Scale is to dramatically improve application performance. As the name suggests, one of its primary objectives is to dramatically scale-up the number of users that an application can support. This scale-up could be to serve more users in less time, or to serve many more users with reasonable, predictable response times.
For example, one organization using WebSphere eXtreme Scale improved access time for key data from 60 ms (milliseconds) to 6 ms -- with 450,000 concurrent users and 80,000 requests per second. The implementation and savings took six weeks from concept to production and helped ease prior limitations that were brought on because the database had reached maximum performance. WebSphere eXtreme Scale technology puts them well on their way to accommodating requirements for handling a load 10 times larger, scaling to one million requests per second, and enabling over five million concurrent users.

But how does WebSphere eXtreme Scale do this? The remainder of this article looks under the covers to answer that question, beginning with a few of WebSphere eXtreme Scale’s underlying principles.

**Principles of extreme scalability**

Integral to the idea of WebSphere eXtreme Scale are these fundamental principles:

- Put data in memory
- Partition the data to enable linear horizontal scale-out
- Caching

The next sections describe these principles in detail.

**Put data in memory**

Data is typically stored on a disk (probably in a database on a disk). To process this data, it must be brought into the computer’s memory so the program can process it (Figure 2). This input/output (I/O) operation is time-consuming, compared with the amount of time required to access data that's already in computer memory.
For example, it can take 20 ms, (or .020 seconds, or 20 x 10^{-3} seconds) to read in a chunk of data from a disk. Data already in memory can be accessed in tens of nanoseconds (or 10^{-9} seconds). Accessing data in memory is thus literally a million (or 10^{6}) times faster.

**Figure 2. Memory hierarchy**

![Memory hierarchy diagram](image)

### 32-bit vs. 64-bit JVMs

32 and 64 refer to the number of bits in the address space of the underlying hardware:

- Most machines have 32-bit hardware, which means an address space can be 2^{32} bytes (or 2 GB) max.
- Most machines have 32-bit hardware, which means an address space can be 2^{32} bytes (or 2 GB) max.

If 2 GB in a 32-bit JVM is not enough, can you use a 64-bit JVM which would provide 4 Exa Bytes (or 4x10^{18} bytes)? JVM memory is virtual. A 64-bit JVM is limited by the physical memory of the server running it. When the machine’s physical memory is full, you must get the rest of the data from disk at the slow disk speed. It is the size of the machine’s physical memory that determines your performance, and so putting all your data into one 64-bit JVM creates a single server bottleneck and defeats the horizontal scale-out principle (next section) you are trying to achieve.

There are also practical limits involving Java™ garbage collection (GC) times; a general rule of thumb is to stay below 5-6 GB in a single 64-bit JVM or else GC could eat up large chunks of your processing power. It might be a better option for you to use smaller 32-bit JVMs to scale out and spread the load than to use fewer larger 64-bit JVMs. In testing, this option shows that 64-bit JVMs use 50% more real memory than 32-bit JVMs to store the same number of objects.

WebSphere eXtreme Scale does support 64-bit JVMs and you should use them when they help, but be careful not to use them so they hurt you. 64-bit JVMs come with modest additional overhead, even if you avoid getting eaten up by the GC. (Explore [WebSphere Real Time](#) if you have GC concerns.)

Given that most data lives on disk, how do you scale applications so they are not ultimately waiting on getting data to and from slow disks? The answer is to put as much of your data as is possible and practical into memory to eliminate these costly I/O operations. Faster data access makes your applications run faster, so you have better user response time, better application throughput, and a better ability to support more users.
Sounds good and seems obvious. So, why isn't everybody doing this?

Two reasons:

- Normally, this important data must be persistent, like bank account records. A problem with computer memory is that when the power is turned off, or the machine is rebooted, data in memory is lost. Disk storage is persistent because the data is not lost if and when the power is terminated.
- All the data won’t fit into computer memory! Computer memory and address spaces are getting larger; a 32-bit JVM means about 2 GB of usable memory. That is a fair amount of data, but your middleware and application logic must also share that space.

Another place to consider putting data in memory is on another machine. While it can take 20-40 ms to get data from a local disk, it takes only 2 ms to get data over a fast network from a nearby machine. Using memory on other machines can be great for scaling. A machine’s memory is used for the operating system, and middleware and application logic, among other things, in addition to the data you put in it. Using remote machines, possibly dedicated to storing data, gives you much more memory for data that you can get more quickly than from disk.

Using other machines is also great for availability. If you store a replica of the data on another machine, you can recover from the failure of the first machine.

**Partition the data to enable linear horizontal scale-out**

The fundamental approach to performance analysis and tuning is to find and eliminate bottlenecks. Assume a processor gives you X throughput. As the green linear scaling line in Figure 3 shows, each time you add a processor to your solution, you get X more throughput. The blue line shows non-linear scaling, were you get less than X additional throughput each time you add a processor. The red bottleneck line shows what happens when you have a resource that is at full capacity (in this example, the database) and cannot be increased in size. As the load increases, your throughput remains constant, at the highest allowed rate of the constrained resource.

**Figure 3. Performance, scaling, and bottlenecks**
The first approach to alleviating this bottleneck is to get a bigger, faster machine to run the database. This is a **vertical scaling** approach; that is, make the one bottleneck machine bigger and taller, if you will. Vertical scaling will get you only so far (namely to a somewhat higher saturation point), plus it is an expensive solution to get leading-edge hardware.

More on this topic

Section 1.1 of the *User's Guide to WebSphere eXtreme Scale* contains an excellent discussion on linear horizontal scale-out. Much of this section is summarized from this Redbook.

So, what if you want or need to scale beyond that?

The obvious next step is to throw more machines at the problem. This is **horizontal scale-out**. But can that help the database? The answer for the most part is no, not that much. Figure 4 shows the situation in a realistic setting:

- The application servers scale out nicely horizontally, thus eliminating any bottleneck in that tier.
- Vertical scaling is applied to the database, but is not enough to remove the database as the bottleneck for the type of scaling that you want.

![Figure 4. Scaling options in a traditional three-tier application](image)

The second principle is to **partition** the data: Use two machines, put half the data on each, and route data requests to the machine containing the data. If you can partition the data in this fashion across two machines, then each takes half the overall load and you have doubled your performance and throughput over that of one machine.

Not all data and applications are partition-able. Those that are can be greatly scaled. Those that are not won’t benefit.
The great part about partitioning is that it scales really well. If splitting the load across two machines is good, how about using 10, 100, or even 1,000 machines? That's exactly what WebSphere eXtreme Scale enables you to do, and exactly the scalability it offers.

Partitioning enables linear scaling (the green line in Figure 3). Requests are efficiently routed to the machine hosting the required data. Without partitioning, much overhead is incurred from getting the current data to the right place for processing, and this overhead grows exponentially -- not linearly -- as more machines are added. With partitioning, each machine you add works happily and efficiently on its own, with no overhead or tax incurred by any number of other peers. As machines are added, all machines share the overall processing load equally. You can add as many machines as you need to process your workload.

In the initial example above, the key data was user profiles, which could be distributed (partitioned) across a set of machines for parallel processing. To achieve the ten-fold improvement in response time, ten servers were used. WebSphere eXtreme Scale can scale up to thousands of machines, which means that this solution can scale 100- to 1,000-fold, to support 50 to 500 million users, all with 6 ms response time. Data in memory and partitioning enabled the drop in response time from 60 ms to 6 ms, but it is the partitioning alone that will enable the 100- to 1,000-fold growth in the number of users supported, with the same response 6 ms time. This is what is meant by linear scaling.

**Caching**

A key feature of WebSphere eXtreme Scale is that it provides a large, scalable, elastic cache.

A **cache** is a chunk of memory used to hold data. The first place to look for an item is the cache. If the data is there, then you have a **cache hit** and you get the item you want at memory speed (tens of a nanosecond). If the data is not in the cache, then you have a **cache miss** and you have to get the item from disk, at disk speed (tens of milliseconds). Once you get the item from disk, you put it in the cache and then return it. This basic cache usage is illustrated by the **side cache** shown in Figure 5. As will be discussed later, the application talks to both the side cache and the disk.

**Figure 5. Side cache**

![Side cache diagram]

The effectiveness of the cache is directly proportional to the **hit ratio**, which is the percentage of requests satisfied by having the item in the cache. If all the data fits in the cache, then the data is read from disk only once; after that, all requests are satisfied from the cache at memory speed.
There are two major benefits to using a cache:

- The first benefit is to the process requesting the data in the cache. This process is accelerated because the data is delivered at memory speed. Any cache hit means that the read operation to get the data from the disk has been eliminated. Thus, the load on the disk has been reduced.
- This reduced load on the disk is the other benefit of the cache. The disk can now support more requests for other data. In the initial example above, the reduction of key data access time from 60 ms to 6 ms was a direct result of this caching benefit and the load reduction on the back-end database.

The hit ratio is a function of the cache size, the amount of underlying data, and the data access pattern. If an application's access pattern is to continuously and sequentially read all the data you are trying to cache, then caching will not help you unless all of the data is in the cache. When the cache filled up, data would be evicted before the application got around to accessing it again. Caching in this case is worse than useless; because it costs more to put data in the cache to begin with, this sort of application would perform poorer overall than with no cache at all.

In many cases, the cache is in the same address space of the process that wants to use the data. This placement provides the ultimate in speed but limits the size of the cache. WebSphere eXtreme Scale can use other JVMs for caching. These JVMs can be dedicated cache JVMs so that most of the memory of the JVM is used and available for application data -- no sharing with middleware, infrastructure, or application code or memory.

These JVMs can be on the same machine. Local JVMs provide quick access to memory, but ultimately, the physical memory and CPU cycles must be shared with the operating system and other applications and JVMs on the machine. Therefore, to get large caches, and balanced and parallelized access loads, you might want to use JVMs on other remote machines to get access to more physical memory. It is physical memory that is the key to fast access. Remote JVMs provide for a much larger cache, which is more scalable.

What happens when the data in a cache is changed?

Writes can complicate things with caches. An important factor in determining what data to cache is the write-to-read ratio. Caching works best when the data does not change (that is, there are no writes and so the write-to-read ratio is zero) or does not change often (for example, if user profiles were cached, they change infrequently and so the write-to-read ratio is small.) When discussing writes, it's beneficial to discuss an in-line cache (Figure 6). As will be discussed later, the application talks only to the cache, and the cache talks to the disk.
There are two significant write use cases:

- The process using the cache changes the data.
- A process not using the cache changes the data.

In the first case, there are two options:

- In a write-through cache, the data is written to the cache and to the disk before returning to the writing process to continue. In this case, writes occur at disk speed, which is bad. The good news is that the data on the disk is consistent with the data in the cache, and this is very good.
- In a write-behind cache, once the data is in the cache, the request is returned, and processing continues. In this case, your writing process continues at memory speed, not disk speed, which is good. But the data on the disk is stale for a time, which is bad. If the new cache copy is not written to disk before shutdown or failure of the cache, the update is lost, and this is very bad. However, WebSphere eXtreme Scale can keep additional copies of the cache on other machines to provide a reliable write-behind cache. Thus, a write-behind cache gives you memory speed for accessing data on all writes, and for all cache-hit reads, and has other benefits that you will see later.

If another process changes the underlying data without going through the cache, that is a problem. If you can’t make all processes use the cache, you will have to live with stale data (for a time). Many users employ a shadow copy of their database (Figure 4) to support large and ad-hoc queries against the operational data. These shadow copies are out of date by a few minutes, yet this is understood and acceptable.

There are two approaches to addressing this cache staleness:

- Have all cache entries time out. For example, every entry is automatically evicted from the cache after some user-defined interval, say 10 minutes. This forces the cache to be reloaded from the disk, and thus the cache will be no more than 10 minutes stale.
- Detect changes in the underlying data, and then either evict the changed data from the cache (which will cause it to be reloaded on demand), or reload the updated data into the cache.

WebSphere eXtreme Scale supports both approaches.

Figure 7 shows where caching fits into the overall picture described earlier in Figure 4.
Implementation details

WebSphere eXtreme Scale is non-invasive middleware. It is packaged as a single 15 MB JAR file, has no dependency on WebSphere Application Server Network Deployment, and works with current and older versions of WebSphere Application Server Network Deployment, WebSphere Application Server Community Edition, and some non-IBM application servers, like JBoss and WebLogic. WebSphere eXtreme Scale also works with J2SE™ Sun™ and IBM JVMs, and Spring. Because of this independence, a single cache or grid of caches can span multiple WebSphere Application Server cells, if necessary.

While WebSphere eXtreme Scale is self-contained, it requires an external framework for installing applications and to start/stop the JVMs hosting those applications -- unless you want to do it all through commands -- like WebSphere Application Server (WebLogic and JBoss frameworks are also supported). WebSphere eXtreme Scale can be managed and monitored by IBM Tivoli® Monitoring, as well as with Hyperic HQ and Wily Introscope.

WebSphere eXtreme Scale uses just TCP for communication, no multicast or UDP. No special network topologies (shared subnet) or similar prerequisites are required. Geographically remote nodes can be used for cache replicas.

Partitions

WebSphere eXtreme Scale uses the concept of a **partition** to scale out a cache. Data (Java objects) is stored in a cache. Each datum has a unique key. Data is put into and retrieved from the cache based on its key.

Consider this simple, if quaint, example of storing mail in a filing cabinet: The key is name, last name first. You can imagine a physical filing cabinet with two drawers, one labeled A-L, the other M-Z. Each drawer is a partition. You can scale your mail cache by adding another cabinet. You re-
label the drawers A-F, G-L, M-R, S-Z, and then you have to re-file (yuck!) the mail into the proper drawer. You have doubled the size of your mail cache, and you can continue expanding.

One problem with your mail cache is that people's names are not evenly distributed. Your M, R, and S drawers might fill more quickly than your Q, X, and Y drawers. This will cause uneven load on your cache in terms of memory and processing.

The way partitions really work is that a hash code is calculated on the key. The hash code is an integer that should return a unique number of each unique key. (For example, if the key is an integer, this value might serve as the hash code. In Java, the String hash code method calculates the hash code of a string by treating the string, in a way, as a base-31 number.) In WebSphere eXtreme Scale, you decide on the number of partitions for your cache up front. To place an item in the cache, WebSphere eXtreme Scale calculates the key's hash code, divides by the number of partitions, and the remainder identifies the partition to store the item. For example, if you have three partitions, and your key's hash code is 7, then 7 divided by 3 is 2, remainder 1, so the item goes in and comes from partition 1. The partitions are numbered beginning with 0. The hash code should provide a uniform distribution of values over the key space so that the partitions are evenly balanced. (WebSphere eXtreme Scale calculates hash codes on keys for you automatically. You can provide your own, if desired.)

A key to linear scaling out is the uniform distribution of data amongst the partitions. This enables you to choose a large enough number of partitions such that each server node can readily handle the load on each partition.

How does WebSphere eXtreme Scale expand its cache?

In the mail cache example above, a drawer is a partition, and you used file cabinets with two drawers each. Both drawers and file cabinets are of fixed size. If you extend this analogy to WebSphere eXtreme Scale, you can compare a file cabinet to a container server, which runs in a JVM. This means that the file cabinet can hold, at most, 2 GB of items (or much larger, if 64-bit JVMs are used). In WebSphere eXtreme Scale, a file cabinet (container server) can hold a variable number of drawers (partitions). Thus, a partition can hold at most 2 GB of items (assuming 32-bit vs. 64-bit JVMs). (If your objects consume $n$ bytes of storage each, then a partition can contain, at most, $2\text{ GB}/n$ items each.)

In WebSphere eXtreme Scale, the number of partitions is set up front to contain a projected number of items. Suppose you use a large number of partitions; for example, 101. This means your mail cache can store up to almost 202 GB (101 partitions * 2 GB per partition) of items. Container servers can be spread over multiple remote machines, and several can share the same machine. In the initial simple case with one container server (which corresponds to the one filing cabinet case), you will have all 101 partitions in the one container server, and so you can hold only 2 GB of items. To expand, you just start another container server, preferably on another machine to spread the load.

Having another available container server, WebSphere eXtreme Scale would automatically balance your mail cache load between the two container servers, so that 50 of the 101 partitions...
would be moved to the second container server. If you added a third container server on a third machine, the load would be rebalanced by moving 17 partitions from the first container server and 16 partitions from the second container server to the new container server. The new container server would have 33 partitions, and the first two 34 each.

In this way, WebSphere eXtreme Scale provides linear scale-out. With 101 partitions and one container server, you can have up to 2 GB of items. With 101 partitions, you can scale up to 101 container servers for a total of 202 GB of items. These container servers should be spread over multiple machines. The idea is that the load is spread evenly over the partitions, and WebSphere eXtreme Scale spreads the partitions evenly over the available machines. Thus, you get a dynamic, elastic cache with linear scaling.

**Replication**

Scale-out is good, but what if a machine fails and you lose servers?

WebSphere eXtreme Scale provides availability in the face of failure by enabling data replication. A partition of the cache is stored in one or more shards. The first shard is called the primary shard. When you define a WebSphere eXtreme Scale cache, you specify the level of replication by specifying the number of replica shards. Replica shards are backup copies of all the data in the primary shard, and their main purpose is to recover from a primary shard failure to provide high availability. In some cases, they can also be used to satisfy read requests to offload the primary shard.

The difference between a partition and a shard might be confusing at first:

- A partition is a logical concept. It represents $\frac{1}{n}$th of the data in the cache, where the cache is partitioned into $n$ parts or partitions.
- A shard is a real, physical chunk of memory that stores the contents of a partition.

To ensure fault tolerance and high availability, WebSphere eXtreme Scale shard distribution algorithms ensure that the primary and replica shards are never in the same container. When the primary fails, a surviving replica is promoted to become the primary, and a new replica is created in another container.

Replicas can be synchronous or asynchronous; the distinction is important when data is written to a WebSphere eXtreme Scale cache. Transactions are used for all WebSphere eXtreme Scale cache reads and writes. A write transaction is not completed until all synchronous replica shards have confirmed receipt of the new data. Asynchronous replica shards are updated after the transaction is complete, thus they provide faster transaction performance (typically at least six times faster), but increase the risk of losing data in the face of failures.

Choose the number of synchronous and asynchronous replicas based on the performance and availability requirements of your application. For example, you might have a synchronous replica on another machine in the same data center, and an asynchronous replica on a machine in another data center. In WebSphere eXtreme Scale, zones are used to define the notion of data centers, and WebSphere eXtreme Scale handles this automatically. By selecting the number
of synchronous and asynchronous replicas, and their zone placement, you can balance your performance versus your availability and resilience in the face of failures.

Let’s revisit the filing cabinet example above with A-L and M-Z mail drawers. This time, suppose you define one replica shard. Figure 8 shows an example, although with only 3 partitions. Initially, with one container, there are 101 primary shards and no replica shards in it. (Replica shards are pointless with only one container.) When the second container is added, 50 primary shards are moved to the second container. The other 51 replica shards are also moved to the second container so that you have balance, and, for each shard, its primary and replica are in separate containers. The same thing happens when you add a third container. Each machine thus has 33 or 34 primary shards, and 33 or 34 different replica shards.

**Figure 8. Partition example showing shard placement**

Now, if one of the containers fails or is shut down, WebSphere eXtreme Scale reconfigures your cache so that each of the two servers has 50 or 51 primary and replica shards, and, for each shard, its primary and replica are in different containers. In this server failure/shutdown scenario, the failing server had 33 or 34 primary shards, and 33 or 34 replica shards. For the primary shards, the surviving replica shard is promoted to a primary shard, and new replica shards are created on the other server by copying data from the newly promoted primary. For the failing replica shards, new replica shards must be created by copying them from the primary, and placing them on a server other than the one hosting the primary shard. WebSphere eXtreme Scale does all this automatically, while keeping the data and load on the servers balanced. Naturally, this takes a little time, but the original shards are in use during this process, so there is no downtime.

WebSphere eXtreme Scale makes a best effort to evenly balance shards across all available containers, but there can be cases when the shards are not exactly evenly balanced. One notable case in point is the example of scaling out from one container to two: since replica shards don’t make sense if there is only one container, there are none. When the second container is started, it is populated with all the replica shards. The third container would then get a third of the shards from each of the first two containers, so that it does not have a primary and replica shard of the same partition.
APIs

What APIs does your application code use to access the data?

There are many choices. For example, data is commonly stored in a database. JDBC (Java Database Connectivity) is an established choice here, and JPA (Java Persistence API) is the new standard. JPA is a specification for the persistence of Java objects to any relational data store, typically a database, like IBM DB2®. Many users have their own data access layer, with their own APIs, to which the business logic is coded. The access layer can then use JDBC or JPA to get the data. These underlying data access calls are thus consolidated in the data access layer and hidden from the business logic.

In a side cache, there are two APIs in use: interfaces to get data from the disk, and different interfaces to get data from the cache. Therefore, you typically have to change application code to use a side cache, as described earlier. On a read request, first check the cache. On a miss, get the data from disk, then put it in the cache. The application must code this logic, using the disk and cache interfaces. If a data access layer is used, the code might be consolidated there and hidden from the using application.

The in-line cache looks better in that you only have one interface: to the cache. Typically, however, applications start out with an interface to the disk, and must be converted to an in-line cache interface. Again, if a data access layer is used, the code can be consolidated there and hidden from the using application.

WebSphere eXtreme Scale offers three APIs to access the data it holds:

- The ObjectMap API is the lower-level, higher performance API. It is much like a regular Java (Hash) Map, with simple create, read, update, and delete operations.
- The EntityManager API is the higher-level, easier to use interface. It is very similar to the JPA interface. If your code uses the JPA interface, using the EntityManager API makes the conversion easier.
- The REST data service enables HTTP and Microsoft .NET clients using the ADO.NET Data Services Protocol to access WebSphere eXtreme Scale data grids

From an API perspective, there are three general ways to use WebSphere eXtreme Scale:

- **No coding change required, just configuration changes.** In these cases, WebSphere eXtreme Scale provides plug-ins to the existing infrastructure that applications are already using to extend the infrastructure’s caching capabilities with those of WebSphere eXtreme Scale. WebSphere eXtreme Scale is used as a side cache. Use cases A, B, and C below are examples.
- **Use WebSphere eXtreme Scale as a side cache for relief.** In these caches, you might insert a small amount of code in a few key areas and use ObjectMap APIs to cache data. Use case D below is an example.
- **Dramatic scaling improvements come from changing the application’s data interface to WebSphere eXtreme Scale, such that it becomes the system of record.** In this case, WebSphere eXtreme Scale is an in-line cache. If you currently use JPA, you might use the
WebSphere eXtreme Scale EntityManager API to make the conversion easier. Use case E describes this approach.

The side cache leverages the data-in-memory and caching principles, while the in-line cache also enables the leveraging of the partitioning/horizontal-scale-out principle.

WebSphere eXtreme Scale offers the **Data Grid API** as an additional means to advantageously leverage partitioning. The idea is to move the processing to the data, not move the data to the processor. Business logic might be run in parallel over all or just a subset of the partitions of the data. The results of the parallel processing can then be consolidated in one place. Beyond distributing the storage and retrieval load of data over the grid, processing of this data can also be distributed over the grid partition servers -- and thus itself be partitioned -- to provide more linear scaling.

### Other implementation elements

This article has focused so far on a cache, a container that stores Java objects based on keys. In WebSphere eXtreme Scale terminology, a cache is considered a **map**. Here are some other items and terms that are important in reference to WebSphere eXtreme Scale:

- A **mapSet** holds a set or collection of maps that all share a common partitioning algorithm. The number of partitions and the replication settings are configured on a mapSet.
- A **grid** holds a collection of one or more mapSets.
- A **container** holds shards.
- A **(container) server** holds and manages a collection of one or more containers.
- A **JVM** or **Java process** can contain zero or one servers.

### Sample use cases

The key to success is to identify pain points and bottlenecks in your application, and then use WebSphere eXtreme scale to alleviate or eliminate them, based on the principles of data in memory, caching, and horizontal scale-out (partitioning). Here are some use cases to illustrate.

#### A. WebSphere Application Server dynamic cache service enhancement

The WebSphere Application Server dynamic cache service (often referred to as DynaCache) is an excellent example of using a side cache to improve performance. DynaCache is a WebSphere Application Server feature that originated in work that was done when IBM hosted the Web site for the 1998 Olympics. To provide extreme scaling for a worldwide audience, the idea was to cache Web page data in memory after it was generated; this is the familiar data-in-memory pattern. The cache saves not only disk I/O, but processing time to put HTML pages together.

There are a couple of things to watch out for with large dynamic caches. Some can grow to over 2 GB, even 30 or 40 GB. This exceeds the virtual memory capacity of the process; so much of the data lives on disk, which is expensive to access. Further, these caches live on each machine. The dynamic cache is also part of the application’s address space.
WebSphere eXtreme Scale provides an easy-to-use plug-in to replace the dynamic (side) cache’s in-process-memory and disk system with a distributed dynamic in-memory grid. You specify configuration parameters in an XML file, with no coding required. Benefits include:

- The cache is removed from the application's address space and placed in a WebSphere eXtreme Scale’s container’s address space.
- The cache can easily scale to 40 GB, which can be spread across multiple machines, so the whole cache can be in memory, with minimal access time.
- The (single) cache can be shared by all applications and WebSphere Application Server instances that need it.

WebSphere eXtreme Scale is a big winner here, and it is easy to use. See Related topics for more details.

**B. L2 cache for Hibernate and OpenJPA**

WebSphere eXtreme Scale includes level 2 (L2) cache plug-ins for both OpenJPA and Hibernate JPA providers. Both Hibernate and OpenJPA are supported in WebSphere Application Server. JPA enables you to configure a single side cache (shown in Figure 5) to improve performance. This is also an in-process cache, thus limited in size.

Because disks are slow, it is faster to get data from a level 2 cache than from the disk. As you have seen, WebSphere eXtreme Scale can provide very large caches. Large distributed WebSphere eXtreme Scale caches can also be shared by multiple OpenJPA- or Hibernate-using process instances. A great feature of the JPA cache is that no coding change is necessary to use them; you just need some configuration properties (like size and object types to cache) in your persistence.xml file. See Using eXtreme Scale with JPA for more details.

**C. HTTP session replication**

HTTP session data is an ideal use for WebSphere eXtreme Scale. The data is transient by nature, and there is no persistence requirement to keep it on disk. By the data in memory principle, you can dramatically increase performance by moving the data from disk to memory. The data was kept on disk (or in the database) in the first place to provide availability in the event of a failure, by permitting another Web server to access the session data if the first server failed.

WebSphere eXtreme Scale does provide failover and availability in the face of failures, and it performs and scales much better than disk solutions, so it is an ideal solution for this data. WebSphere eXtreme Scale is very easy to use for this application. No code changes to your application are required. Just put an XML file in the META-INF directory of your Web application’s .war file to configure it so that WebSphere eXtreme Scale will be the store (system of record with in-line caching) for the HTTP session data. In the event of a failure, access to the session data will be faster, since it is in memory, and the data itself remains highly available. See Scalable and robust HTTP session management with WebSphere eXtreme Scale for more details.

**D. SOA ESB cache mediation**

A retail bank with 22 million online users has customer profiles stored on a CICS Enterprise Information System (EIS) on a System z® host. Several applications access these profiles. Here,
the host is a large computer, and the profiles are stored in a database on that computer. Each application uses an SOA service over an Enterprise Service Bus (ESB) to get the profile data. Before WebSphere eXtreme Scale, each application had its own profile cache, but these caches were in each application’s address space, not shared. The cache reduces some profile fetch load from the host EIS, but a larger shared cache would reduce more load.

Here, WebSphere eXtreme Scale is used as a network-attached side cache holding around 8 GB of profiles (4 GB + 4 GB of replicas for high availability). A mediation is inserted in the ESB to memorize profile fetch service calls. The service name and parameters are used as a key and the value is the profile itself. If the profile isn’t in the cache, then the mediation gets it from the host and stores the result in the cache. An evictor removes entries older than 30 minutes to prevent staleness and limiting the size of the cache. No application code changes are required; the mediation is inserted transparently on the ESB.

Before WebSphere eXtreme Scale, logon took 700 ms with two back-end calls. With WebSphere eXtreme Scale, logon took 20 ms with profile cache access, which is 35 times faster. Further, the load on the EIS was reduced as profile fetch calls to it were eliminated, being serviced instead from the cache. Monthly costs saved as a result of this reduced load were considerable.

An SOA ESB is a great place to add caching cost effectively. It provides easy to use service invocation points, and a service mediator using a coherent cache can be easily inserted without modifying upstream (client) or downstream (original server) applications.

E. Customer profile service

Let’s return to the very first example described in this article, where the database was maxed out and the user needed to support 10 times their current load. This actual user had a peak rate of 8K page views per second. The application was using an SQL database for querying and updating profiles. Each application page performed 10 profile lookups from the database at 60 ms each, for a peak total of 80K requests per second. Each user profile is 10 KB, with 10 million users currently registered, for 100 GB of profile data. If the database goes down or if there are wide-area network access issues across the globe, the application is down. Keeping the data in memory addresses these issues.

The application was changed to write to a WebSphere eXtreme Scale in-line cache as the system of record, as opposed to the database; WebSphere eXtreme Scale was inserted as a front end to the database. The WebSphere eXtreme Scale cache was configured as a write-behind cache. In the HTTP session replication case, you saw WebSphere eXtreme Scale being used as the backing store for performance. In that case, the data was really transient. Here, you want to persist the data back to the database for long-term storage. As you have seen, WebSphere eXtreme Scale has replication for availability and transaction support. These features enable you to put a WebSphere eXtreme Scale cache in front of the database to great advantage.

As discussed earlier, with a write-behind cache, the transaction completes when the data is written to the cache, which is fast. The data is written to the database disk later. This means the write-behind cache must reliably keep the data in the cache until it can be written to disk. WebSphere
eXtreme Scale provides this reliability through replicas. You can configure this write delay in terms of number of writes or amount of time. If the back-end database fails, or if your network connection to it fails or slows, your application keeps running, and this failure is transparent to and does not affect your application. WebSphere eXtreme Scale caches all the changes and, when the database recovers, applies them -- or, applies them to a backup copy of the database if that is brought online.

This write-behind delayed-write has several advantages, even beyond the increased response time for the user on writes. The back-end database itself can be greatly offloaded and used more efficiently:

- A data field can be changed multiple times while in the cache before it is written to disk. Only the last change must be written to disk. Further, different fields in the same record can change in separate transactions. These can also be combined into a single update to the record. I/O operations are saved/eliminated. (This savings is called conflation.)
- Databases work more efficiently for batches of updates, versus updates of a couple fields in one record. See IBM Extreme Transaction Processing (XTP) Patterns: Leveraging WebSphere Extreme Scale as an in-line database buffer and Scalable Caching in a Java Enterprise Environment with WebSphere eXtreme Scale for more details.

A WebSphere eXtreme Scale distributed, network-attached cache using write-behind is installed in front of an SQL database. The cache’s server containers are deployed on ten 8-core x86 boxes serving up user profiles and accepting changes. Asynchronous replicas are used for highest performance with good availability. Each box performs 8K requests per second providing a total throughput of 80K requests per second with 6 ms response times, ten times more than before.

The solution needs to scale to 1 million requests per second, with response time remaining at 6 ms. WebSphere eXtreme Scale will be able to achieve this with linear scaling by adding about 120 more servers. Each server performs around 110 MB per second (8K requests per second with 10 KB record sizes) of total traffic (90 MB from requests, 20 MB for replication). Multiple GB ethernet cards are required per server. Multiple network cards are required now in each server given the processing power of even small servers and the performance of WebSphere eXtreme Scale. 300 32-bit, 1 GB heap JVMs are used.

**Summary**

IBM WebSphere eXtreme Scale operates as an in-memory grid that dynamically processes, partitions, replicates, and manages application data and business logic across hundreds of servers. It provides transactional integrity and transparent fail-over to ensure high availability, high reliability, and consistent response times. WebSphere eXtreme Scale is an essential distributed caching platform for elastic scalability and the next-generation cloud environments.

Elastic means the grid monitors and manages itself, enables scale-out and scale-in, and is self-healing by automatically recovering from failures. Scale-out enables memory capacity to be added while the grid is running, without requiring a restart. Conversely, scale-in enables on-the-fly removal of memory capacity.
Hopefully, this makes more sense to you now than the first time you read it! WebSphere eXtreme Scale enables applications to scale to support large volumes of users and transactions. It is also useful in smaller applications to provide improved throughput and response time.

With this insight, you can now begin to leverage the power of WebSphere eXtreme Scale to help you solve intractable scaling problems with your current vital applications. WebSphere eXtreme Scale enables many benefits, including automatic growth management, continuous availability, optimal use of existing databases, faster response times, linear scaling, and the ability to take advantage of commodity hardware. The business results include faster response to your customers, lower total cost of ownership (TCO), and improved return on investment (ROI) by more effective resource utilization and scalability.

Acknowledgements

I'd like to thank the following people for reviewing and contributing to this paper: Veronique Moses, Art Jolin, Richard Szulewski and Lan Vuong.
Related topics

- WebSphere eXtreme Scale product information
- User’s Guide to WebSphere eXtreme Scale
- ObjectMap API
- Java 2 Platform Standard Edition (J2SE) 5.0 API Specification
- EntityManager API
- Data Grid API
- WebSphere eXtreme Scale REST data service technical preview
- Using eXtreme Scale with JPA
- Scalable and robust HTTP session management with WebSphere eXtreme Scale
- Leveraging WebSphere Extreme Scale as an in-line database buffer
- Scalable Caching in a Java Enterprise Environment with WebSphere eXtreme Scale
- IBM developerWorks WebSphere

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)