Scaling Java EE Applications with Coherence*Web - a Technical Deep Dive

David Felcey, Michael Bräuer
Oracle Corporation UK Limited, Oracle Deutschland B.V. & Co. KG

Keywords:

Coherence*Web, Oracle Coherence, HTTP Session Management, Session Attributes, Scalability, Elasticity, Reliability

Introduction

Many Web applications use built-in HTTP session management provided by Java Application Servers, like Oracle WebLogic Server, Glassfish, or Tomcat. An HTTP session contains user-specific information collected during a users interactions and is maintained by the application server until it is invalidated, either by user interaction or a timeout. It solves the problem of how to keep state over a sequence of stateless HTTP requests/responses.

For fast access most application servers store the session state in memory. Resilience is usually provided by creating multiple copies on different servers or persisting the state to a database. But these strategies have some drawbacks, making the infrastructure inflexible or impacting on the performance or scalability of the application. Coherence*Web addresses these issues. It enables application servers to hold their session state in memory, for fast access, provides redundancy in case of failures and most importantly allows web applications to flexibly scale to handle traffic spikes or a growing user base. This document will outline how Coherence*Web works and how it can be used to scale Web applications.

What is Coherence*Web?

Coherence*Web is an HTTP session management module dedicated to managing session state in clustered environments. Built on top of Oracle Coherence, Coherence*Web:

- enables session sharing and management across different Web applications, domains and heterogeneous application servers.
- brings Coherence data grid's data scalability, availability, reliability, and performance to in-memory session management and storage.
- supports all of the mainstream application servers such as Oracle WebLogic Server, IBM WebSphere, Tomcat, and others – including Microsoft IIS.
- supports numerous portal containers, including Oracle WebLogic Portal.
- allows for session state to be managed in the various caching topologies available in Coherence (that is, Replicated, Partitioned, Near Caching, and so on).
- allows storage of session data outside of the Java EE application server, freeing application server heap space and enabling server restarts without session data loss.

- supports multiple advanced session models (that is, Monolithic, Traditional, and Split Session) which define how the session state is physically managed and serialised/de-serialised in the cluster.

- provides advanced session distribution and lifecycle controls by way of the session distribution controller and reaper.

- supports fine-grained session and session attribute scoping by way of pluggable policies.

What Impacts on Web Application Scalability?

A number of factors can impact on the scalability of a web application, such as the scalability of dependent resources, like a database, LDAP server or messaging system. Also insufficient physical resources, CPU, memory or network bandwidth, or the design of the application itself can be limiting factors. The architectures of most Java applications servers and the J2EE framework aim to mitigate against this last problems but their the scalability can be limited too, by the JVM they run in and the way it works.

Storing user session information in the same JVM, where the Java applications server runs, can take up a lot of memory that cannot then be used to process user requests. Because of this, the memory for the application servers JVM usually needs to be increased as the number of requests increase. The problem with this is that the more memory the JVM has to manage the more likely it is to stall or pause when it needs to clean up unused memory - or perform a GC (Garbage Collection). From a user perspective these pauses make the Web application seem unresponsive and the site slow. Even a pause of a couple of seconds can give users the perception that the site is difficult to use and lead them to look elsewhere. So this problem can have real business impact.

To balance these competing concerns, servicing an increasing number of users and preventing long pauses, Coherence*Web can cache the user session state outside of the Java application server JVM. This means that application server JVM requires less memory, so reducing the impact of any GC operations. The application server tier and session caching tier can also be scaled independently of each other, depending on the resource bottleneck.

In a recent test using a mixed session (both large and small session objects – 5k and 100k) with a mixed read/write workload, a WebLogic Server using Coherence*Web was able to more than quadruple the number of concurrent sessions it was able to support - compared to its built-in session management capabilities. Now this not a poor reflection on Oracle WebLogic Server, this is likely to be the case with any other Java application server.

Furthermore, many web applications have natural peaks and troughs in their usage. By dynamically and intelligently scaling your application server and caching tiers, you can not only accommodate these different usage patterns but also fully utilise of your hardware resources.

How Does Coherence*Web Work?

Caching Information Efficiently: Session Models

Internally Coherence*Web can store session attributes using different object models. The *Monolithic Session Model* defines one “big” cache entry holding the complete session that is serialized/de-

serialized all together. The *Traditional Session Model* is similar but (de-)serialization is done per attribute. The *Split Session Model* groups all attributes lower than a certain defined size in one entry and attributes that are larger than this threshold size are treated as individual cache entries. In either case all session information must be serializable.

Optimising Access: Consistency and Locking

In order to achieve consistency of session data when accessing them concurrently by multiple threads from within one application or even from different application servers, different locking approaches are configurable, optimistic locking, last-write-wins locking, application locking, member locking, and thread locking.

For instance, optimistic locking allows concurrent access to a session by multiple threads in a single JVM or multiple JVMs, while prohibiting concurrent modification. It detects and prevents concurrent updates upon completion of an HTTP request that modifies the session. It is the default locking mode.

Scale-Out on Demand: Deployment Models

There are two deployment models for Coherence*Web either *In-Process* or *Out-of-Process* – as shown in figure 1. As you can see, in both cases the Java application server is part of the Coherence "cluster".

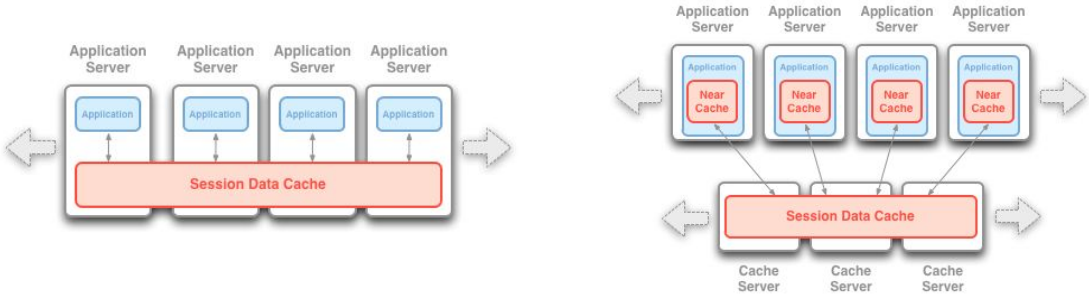

Figure 1: In-Process vs. Out-of-Process Model

With the In-Process model the Java application server nodes are also cache servers and all hold a portion of the session data in the JVM, so they are "storage-enabled" Coherence cluster members.

In an Out-of-Process model the Coherence cache application runs in additional JVMs – they form a tier of additional cache servers. These are members of the same Coherence cluster as the Java application servers. But these separate Coherence JVMs hold the session data, so the Java application servers are called "storage disabled". Additional storage capacity can be dynamically added or removed by adding more Coherence cache servers in the cache server tier. This can be done without any application outage, downtime or re-configuration.

To further optimise the Out-of-Process model a local or "near cache" can be introduced on the Java application servers. This will hold a size limited subset of the most frequently requested session data and will significantly reduce the number of requests that the Java application server must make across the network to the cache servers. The "near cache" is also synchronized with the cache server data, so if a session is modified on a cache server stale copies in any "near caches" will be removed.

The Out-Of-Process model is very powerful when used in conjunction with the Split Session Model because only those attributes being modified are loaded into the Java applications server JVM, saving memory and minimising the network overhead.

Container Integration and Packaging

There are several options when deploying and packaging Coherence*Web. It can be scoped globally, by placing it into the Java application server's CLASSPATH, or just for use by an application, by adding it to a WAR or EAR file. Each packaging options provides a different level of cluster and cache isolation. When the Web application using Coherence*Web starts-up it will join the Coherence cluster. Coherence itself is a standalone Java program that can be started from the command line – when using an Out-Of-Process model.

Additional Benefits of Oracle Coherence*Web

As well as enabling Web applications to scale more easily Coherence*Web can also:

- *Dynamically add capacity to handle traffic spikes.*

  Coherence*Web allows additional capacity to be dynamically provisioned by just starting more cache servers when using the Out-of-Process model.

- *Share session state across applications to create more modular application architectures or to unify disparate web applications.*

  These features have been utilised by a number of customers. One is using Coherence to link separate web applications with different business functions, like ticketing, reservations and customer rewards, allowing the development of each to progress and be enhanced at their own pace using a unified session state to link them together. Another has used Coherence*Web to share session state between different web applications from different divisions, to provide a common checkout for a better customer experience.

- *Provide seamless state replication and recovery.*

  This is being used by some customers to create a stateless application server tier that allows them to create an infrastructure with zero downtime. Any application server can be taken out of service for maintenance without impacting on customers using their site.

- *Cache Web application specific information.*

  Additional product or reference data can be cached in Coherence to accelerate performance and improve the user experience.

- *Cache more data than there is memory.*

  Coherence has an *Elastic Data* RAM/Disk overflow feature. This allows better resource utilisation as all the session state does not need to be held in memory.

Conclusion

Coherence*Web is a proven standards based, HTTP session management solution that doesn't have the limitations of traditional approaches. It seamlessly integrates with a large number of Java Application Servers (and Microsoft IIS) and does not require any changes to application. Scaling Web applications to support increasingly large user sessions and unpredictable usage spikes can be a real challenge. Coherence*Web can help developers, architects and support staff meet this challenge by providing a scalable data platform to optimizes both software and hardware resources.

Contact addresses:

David Felcey
Oracle Coherence Product Management

Oracle Corporation UK Limited,
Southgate Centre Two, 321 Wilmslow Road,
Heald Green, Cheadle,Cheshire SK8 3PW, GB

E-Mail: david.felcey@oracle.com
Internet: www.oracle.de

Michael Bräuer
Systemberatung

Oracle Deutschland B.V. & Co. KG
Schiffbauergasse 14
D-14467 Potsdam

E-Mail michael.braeuer@oracle.com
Internet: www.oracle.de

Additional Resources:

Oracle Coherence*Web Documentation
Download Oracle Coherence (and Coherence*Web)