White Paper

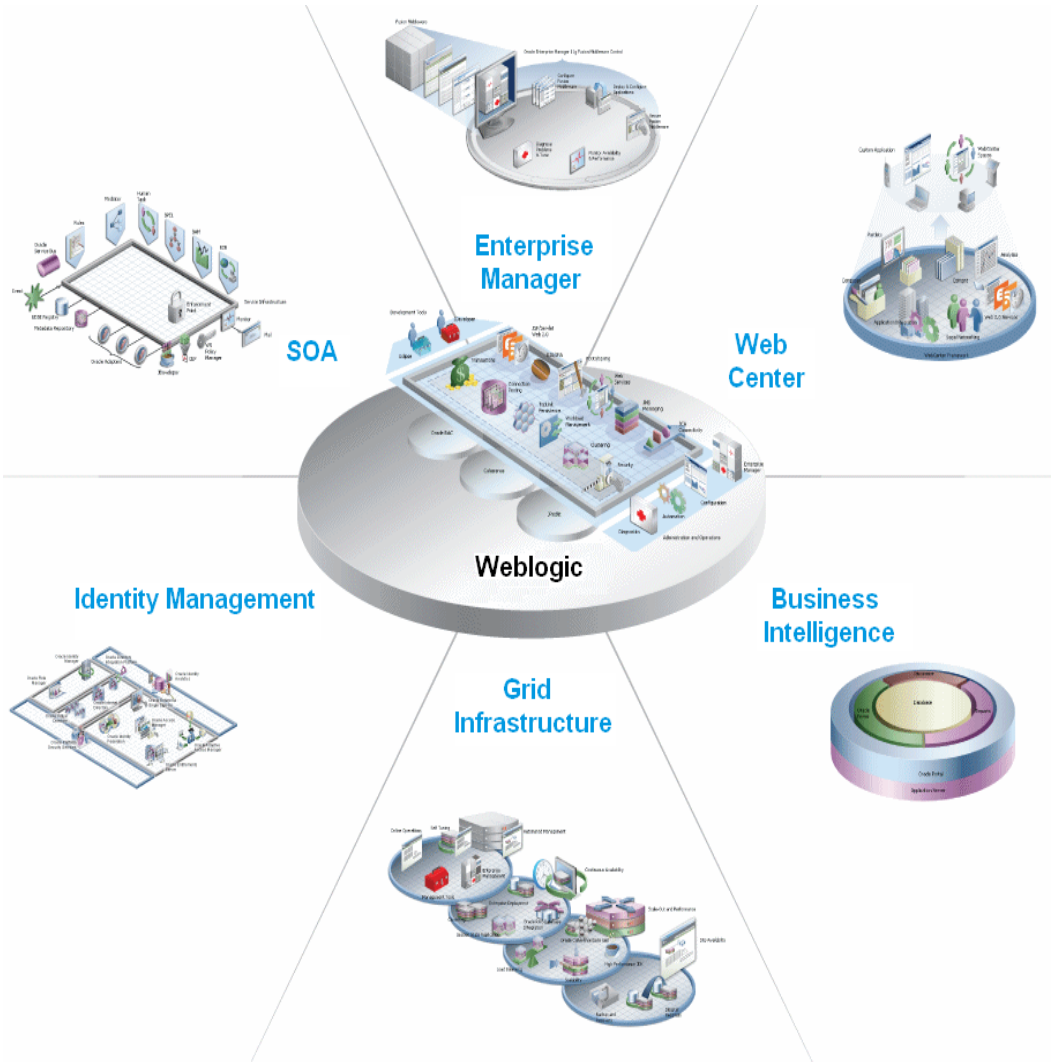
Major Performance Tuning Considerations for Weblogic Server

Table of Contents

Introduction and Background Information

Tuning WebLogic Server and the relevant applications is an iterative and complex procedure.

Oracle WebLogic Server offers mission critical infrastructure for enterprise Java applications. WebLogic Server is a scalable and enterprise ready Java Platform application server known as Java EE Application Server. The Weblogic platform takes care of the deployment of various distributed applications. The following figure illustrates how WebLogic Server fits into the Oracle Fusion Middleware stack.



The Weblogic platform is an ideal foundation for developing applications that are built on Service Oriented Architecture (SOA). SOA aims at maximizing the application service reusability. As it is built on Oracle cloud application foundation and standards based technology offerings, it is usually considered as an application server of choice for fabulous data centers building cloud platforms.

With Weblogic server, you can deploy applications in a secure, robust and scalable environment. It takes complete advantage of the latest hardware architectures including 64-bit addressable memory, high speed networks and multi-core computing systems. WebLogic Server can be configured that helps in monitoring and tuning the application throughput.

Tuning WebLogic Server and the relevant applications is an iterative and complex procedure. This white paper focuses on the major performance tuning considerations for weblogic server.

Understanding the Performance Objectives

The determination of target CPU usage should be based on the application needs and the CPU cycles for peak usages.

In order to establish the performance objectives, we need to assimilate the deployed application and the environmental constraints placed on the system. You can gather the information regarding the activity levels that application components are supposed to meet. They are:

- Anticipated number of users.
- Number of requests and the size of each request.
- Amount of data and its consistency.
- Estimating the target CPU utilization.

The target CPU utilization should not be 100%. The determination of target CPU usage should be based on the application needs and the CPU cycles for peak usages. If the CPU usage is optimized at 100% during normal loads then you can't handle peak loads. In case of latency sensitive applications and fast response systems, high CPU utilization can minimize the response times. Throughput remains constant or it may increase because of the tasks queuing up in the server. In such scenarios, 70% to 80% of CPU utilization is normally recommended. But for applications that are non-latency sensitive, it can be about 90%.

The objectives of performance are also limited by the following constraints:

- The hardware and software configurations such as the type of CPU, disk speed, disk size and memory sufficiency.
  It is not easy to predict a definite formula for evaluating the hardware requirements. The process of determining the appropriate configurations that can meet all the needs of an application is known as capacity planning. Capacity planning needs a proper assessment of the system performance goals.

- Potentiality to interoperate between the domains, using legacy systems and supporting legacy data.

- Development, implementation and maintenance costs.

With the help of above mentioned information, you can set realistic performance objectives for your environment such as response time, throughput and loading on specific hardware.

Measuring Your Performance Metrics

If you notice that the application server CPU is underutilized, confirm whether the database is bottlenecked. If the database CPU is completely utilized then check your application SQL call query plans.

After determining your performance criteria, you need to take the measurement of the metrics to quantify the objectives of performance. Here are the two steps to measure basic performance metrics:

- *Monitoring Disk and CPU Utilization*
- *Monitoring Network Data Transfers*

Monitoring Disk and CPU Utilization

It would be better to run your application under high load while monitoring the application and database server.

The primary goal is to reach a point where the application server accomplishes the expected CPU utilization. If you notice that the application server CPU is underutilized, confirm whether the database is bottlenecked. If the database CPU is completely utilized then check your application SQL call query plans. For instance, are your SQL calls either using indexes or performing linear searches? You can also confirm whether there are abundant ORDER BY clauses used in your application, which may affect the database CPU.

If you recognize that the database disk is the bottleneck (for instance, if the disk is fully utilized), you can just think of operating faster disks or

Examine the amount of data transferred between the application & the application server, as well as between the application server & the database server.

moving to a RAID (redundant array of independent disks) configuration.

Once you find the database server is not the bottleneck, you need to check whether the application server disk is the bottleneck. Some of the bottlenecks for application server disks include:

- HTTP logging
- Persistent store writing
- Server logging
- Transaction logging

The disk I/O on an application server can be optimized by:

- Employing faster disks or RAID
- Making use of JTA direct write for tlogs
- Relaxing the synchronous JMS writes
- Enhancing the HTTP log buffer.

Monitoring Network Data Transfers

Examine the amount of data transferred between the application & the application server, as well as between the application server & the database server. To avoid the bottlenecks, this amount must not exceed the bandwidth of the network.

Minimizing the Impact of Bottlenecks

If you find that neither the database server nor the network is the bottleneck, you can look at the operating system, server configuration settings and JVM. Here is the question: Is the machine hosting WebLogic Server able to achieve your estimated CPU utilization with a high client load? If your answer to this query is 'NO' then you need to verify if there is any locking taking place in the application. You must profile your application using the tools such as JProbe or OptimizeIt to locate bottlenecks and increase the application performance.

Fine-tuning your environment can reduce the effects of bottlenecks on the overall performance objectives. Tuning lets you adjust the resources and achieve all your objectives. The following section explains the top tuning considerations for weblogic server.

Tuning Weblogic Server Subsystems

Work managers can let you control the thread utilization because you have many options to define a variety of guidelines for priority based thread pools.

This section covers all the necessary details on how to tune WebLogic subsystems and services.

Thread Management

According to your defined rules and the absolute runtime performance, WebLogic Server can be tuned to optimize the application performance and manage service level agreements.

By defining a Work Manger and applying it either universally to the server domain or to a certain component of an application, you can tune the thread utilization of a server instance.

Work managers can let you control the thread utilization because you have many options to define a variety of guidelines for priority based thread pools. These guidelines can be defined as just numeric values or as the resource capacity such as a JDBC connection pool.

WebLogic Server can automatically detect when a thread becomes stuck. As the stuck thread can't complete its current task, the server logs a message immediately.

You can actually tune the detection behavior of server thread by modifying the time length before a thread is determined as stuck and also by modifying the frequency of the server checks.

Network I/O

Network channels are also known as network access points. These access points let you specify variant quality of service parameters for network communication.

The major role of a channel is to control the network traffic for a server. But you can leverage the capacity to build multiple custom channels to acknowledge a multi-threaded client to interact with server instances over multiple connections. So this helps in reducing the potential for a bottleneck. You can just consider the following two steps to configure custom multi-channel communication:

- Configuring multiple network channels by using different port and IP settings.

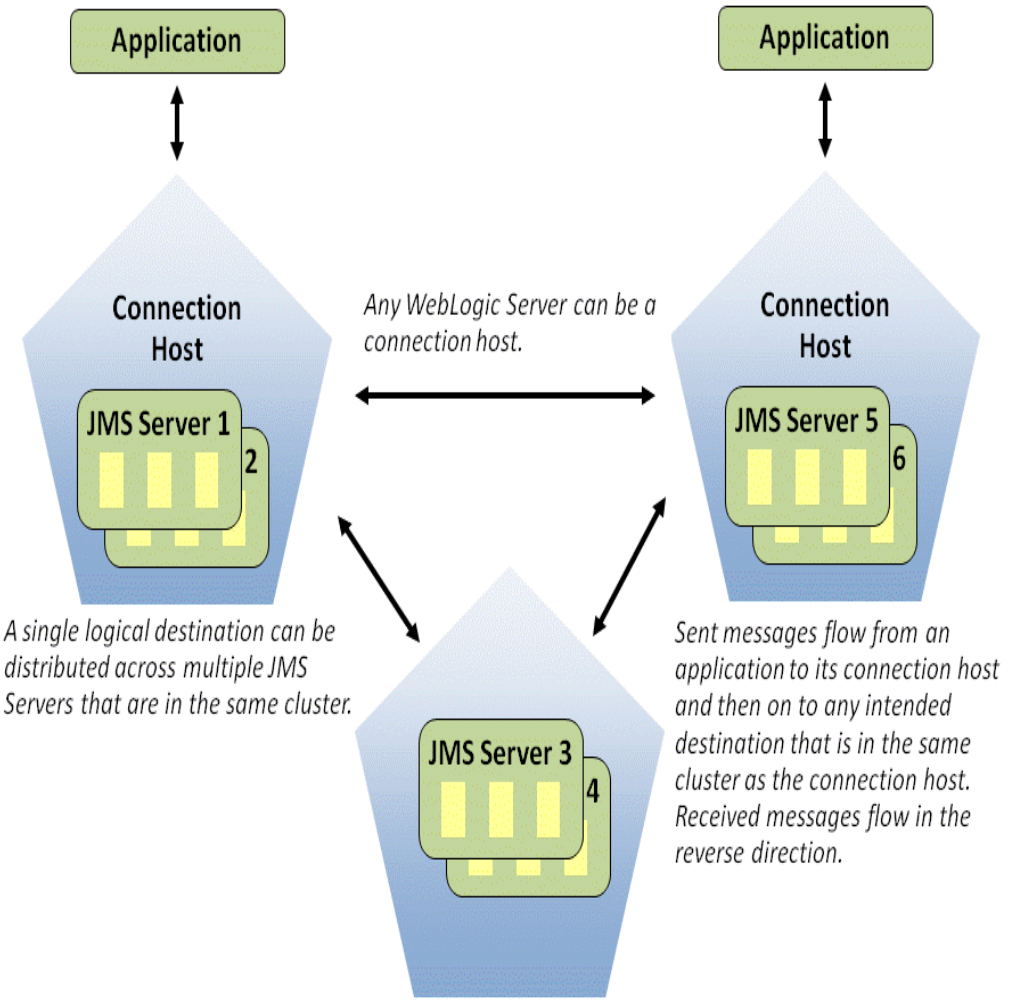
- In the client code, it is a good idea to use a JNDI URL pattern similar to that of the patterns in a clustered environment.

Weblogic JMS

You can leverage the capacity to build multiple custom channels to acknowledge a multi-threaded client to interact with server instances over multiple connections.

A JMS server can be considered as a managed container for JMS queues and topics. The thing is, a WebLogic server in a cluster can become a JMS connection host for JMS applications. JMS messages originate from an application through the connection host and flow to the appropriate destinations on JMS server.

The following figure illustrates the architecture of WebLogic JMS.



The major performance tuning considerations include:

- It is always recommended to avoid large backlogs of messages.
- Default connection factories are non-tunable. Minimize the usage of default connection factories and build custom factories. The custom factories have various options for tuning the performance.
- As JMS resources such as JMS connections, JNDI contexts and sessions are expensive, the applications should be aimed at caching and reusing the JMS resources effectively.
- It is a good practice to avoid single-threaded processing. You can use multiple producers and consumers to make sure that sufficient threads are readily available to service them.
- The server side applications must be tuned in some way to get enough instances. You need to create dedicated thread pools for such applications.
- Stay away from configuring the sorted destinations. FIFO and LIFO destinations are usually the most effective.
- In a cluster, avoid excessive message routing by properly configuring the connection factory targets.
- Execute the applications on the same servers that are also hosting the destinations. This can eliminate networking and marshalling overhead that helps in reducing the CPU usage.

WebLogic Message Bridge

Few techniques to enhance the performance of WebLogic message bridge:

The server side applications must be tuned in some way to get enough instances. You need to create dedicated thread pools for such applications.

- If remote destinations are already highly available then it is not appropriate to use messaging bridge. A messaging bridge can be used in case of distinct maintenance schedules and unreliable network.
- If ordering of messages is not essential then you can deploy multiple bridges. By using the same destinations, many instances of the bridge could be deployed.
- The performance of JMS messaging can improve significantly while handling the persistent messages.
- When the disk storage and the CPU are saturated, it is normally advised to decrease the bridge instances.

GridLink data source can utilize runtime connection load balancing (RCLB) to allocate connections to the RAC instances. This can simplify the configuration of data source and increases the performance.

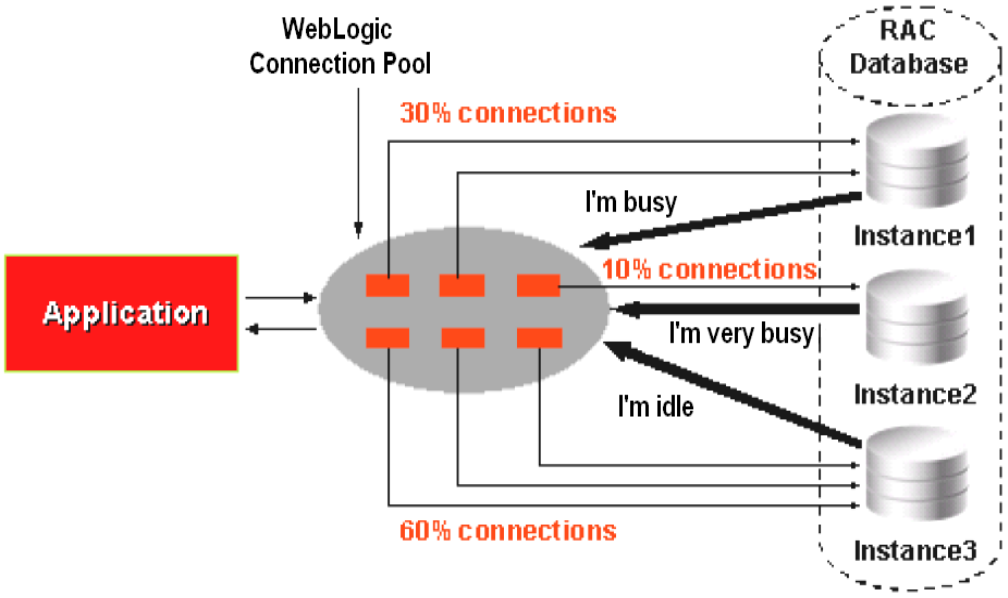
Data Sources

WebLogic Server and Oracle RAC clusters are normally used together. Oracle RAC (real application clusters) is a module that helps the users on various machines to access a single database efficiently.

A GridLink data source can be used to connect the WebLogic Server and an Oracle database service. The database may include many Oracle RAC clusters. It also enables the administrators to manage the workload effectively. As the number of services increase in the database, you can scale the GridLink data sources.

GridLink data source can utilize runtime connection load balancing (RCLB) to allocate connections to the RAC instances. This can simplify the configuration of data source and increases the performance.

The following figure illustrates the runtime connection load balancing:



The runtime connection load balancing helps the WebLogic Server in:

- Adjusting the allocation of the work based on CPU, response time and availability.
- Reacting to changes in the RAC topology.
- Managing the pooled connections and increasing the overall performance.

With ONS (oracle notification service), a GridLink data source can make use of Fast Connection Failover and reacts to all Oracle RAC events. So, a GridLink data source is very useful in:

- Detecting breakneck failures.
- Aborting and removing the worthless connections from the pool.
- Executing graceful shutdown for RAC node outages.
- Distributing all the runtime requests to active RAC instances.

Tuxedo Connector

As the WebLogic Server management system is usually based on Java Enterprise Edition and other standards, it could be easily integrated with systems that are generally utilized to manage other software and hardware entities.

WebLogic Tuxedo Connector (WTC) offers interoperability between the applications of weblogic server and tuxedo services.

Here are the tuning considerations:

- The ON_DEMAND connection policy must be avoided. The most preferred policy is ON_STARTUP and INCOMING_ONLY. This can reduce the possibility of service request failures.

- While designing the applications, the WTC features such as Link level failover, Load balancing and Service level failover should be considered.

- Parallelism can be achieved by properly configuring various servers on different tuxedo machines. While configuring tuxedo applications that may act as servers interoperating with WTC clients, parallelism could be taken into account.

- In case of Tuxedo applications, be cautious with the database access deadlocks. You can prevent deadlocks by properly configuring the tuxedo applications.

- For load balancing all the outbound requests, the imported service has to be configured with many entries using a distinct key. The imported service actually uses the composite key to figure out the uniqueness of each record.

Conclusion

This white paper throws light on the major performance tuning considerations for WebLogic Server. As the WebLogic Server management system is usually based on Java Enterprise Edition and other standards, it could be easily integrated with systems that are generally utilized to manage other software and hardware entities.