BEA WebLogic® Integration 10.2

Capacity Planning and Performance Tuning

Version 10.2
Document Revised: April, 2008

Contents

1. Introduction

2. Capacity Planning Process

A. Capacity Planning Example

B. WLI Tuning

CHAPTER 1

Introduction

This guide describes capacity planning, which is the process of determining the hardware and software resources required for running WebLogic Integration (WLI) applications.

It also provides information about tuning the environment to maintain and improve performance of WLI applications.

Note: The tests described in this guide were conducted in a controlled environment; the numbers presented here may not match the results you get when you run the tests in your environment. The numbers are meant to illustrate the capacity planning process.

This guide does not provide a ready-reckoner that you can use to find out the hardware resources required for a given scenario. It describes the methodology that you can follow to estimate the resource requirements based on certain tests.

Target Audience

This document is intended for the following audience groups:

- Administrators and others in the IT department who are responsible for determining the IT resources required for running WLI applications in the organization.
- Administrators who are responsible for tuning the environment to maintain and improve performance of WLI applications.
- Software architects, programmers, administrators who are involved in designing, developing, and deploying WLI applications.

It is assumed that readers are familiar with the features of WLI. For more information, see the product documentation at http://edocs.bea.com/wli/docs102/index.html.

The tests that are part of the capacity planning process must be performed by people who are experienced in conducting performance tests and familiar with techniques for statistical analysis.

About this Document

This information in this guide is organized as follows:

- Introduction (the chapter that you are currently reading): Contains information about the target audience, the definition of capacity planning, and purposes of capacity planning.
- Capacity Planning Process: Describes the activities involved in the capacity planning process.
- Capacity Planning Example: Illustrates the capacity planning process with an example.
- WLI Tuning: Provides information about tuning the environment to maintain and improve performance of WLI applications.

What Is Capacity Planning?

Capacity planning is the process of determining the hardware and software resources required for running an application with a given user load at a satisfactory level of service. It is a continuous process that ensures that the required computing resources (primarily, hardware) for an application are available at the right time.

While planning for capacity, consider future growth requirements. The number of WLI applications and their scope might increase with time. It is important to determine future requirements, well in advance, so that necessary action can be taken to plan for, procure, and install the required resources. Besides ensuring that adequate resources are available for meeting the performance and scalability requirements, capacity planning can save considerable money and resources for the organization.

Purpose of Capacity Planning

Capacity planning is necessary for the following purposes:

- **To achieve predictable and consistent performance**

  The load on a system may not always be uniform; it may vary from low, high, to very high. Capacity planning helps you provide users a satisfactory level of service at varying server loads. Even at peak loads, performance can be expected to remain within acceptable limits. If capacity is not planned, and if adequate resources are not available, users may experience significant variation in performance as the load varies. At very high loads, the system may run so slow that users cannot even use the system.

- **To determine scalability characteristics**

  Capacity planning helps in determining the scalability characteristics of the application. If the available hardware resources are not adequate for the current load or if the load is expected to increase in future, system administrators can estimate the additional hardware resources required to maintain or improve performance.

- **To identify current capacity**

  Even in situations where the available hardware resources are adequate for the current load, capacity planning helps in identifying limitations of the existing hardware.

- **To estimate the additional hardware required to support increased demand**

  If the load on the servers is expected to increase due to factors such as business expansion and changes in marketing strategy, capacity planning helps in estimating the additional hardware resources necessary for the increased load.

Related Information

- *Best Practices for WebLogic Integration*
- *Database Tuning Guide*
- *WebLogic Server Performance and Tuning*
- *Capacity Planning for Web Services: Metrics, Models, and Methods* – Menasce-Almeida
- *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services* – Neil J. Gunther
- *The Practical Performance Analyst* – Neil J. Gunther

CHAPTER 2

Capacity Planning Process

The capacity planning process involves several activities. The following sections describe these activities:

Note: The tests described in this guide were conducted in a controlled environment; the numbers presented here may not match the results that you get when you run the tests in your environment. The numbers are meant to illustrate the capacity planning process.

Design the WLI Application

The following are some of the performance-related design issues that architects and developers should keep in mind while designing WLI applications:

- **Invoking JPDs**

  You must use only process controls (not service controls) to invoke subprocesses. Service controls are recommended for invoking only web services and JPDs in a different server or cluster. Using process control does not incur the overhead of a web service stack, unlike service control which has more CPU and input/output cost.

- **Process control callbacks vs. message broker subscriptions**

  Process control callbacks are faster than message broker subscriptions because process control callbacks are routed directly to the JPD instance. Message broker subscriptions with a filter involve database access to map the filter value to the process instance.

  Note: Dynamic subscriptions offer loose coupling. So you can use dynamic subscription instead of process control callback in design scenarios where loose coupling is required.

- **Persistence flags**

  By design, if your process becomes stateful and the operation does not require the state to be persisted in the database, consider changing the persistence flag to **Never** or **Overflow**.

  Note: Persistence set to **Never** or **Overflow** might not work properly in a cluster.

- **Accessing worklists through worklist APIs versus WLI JPD controls**

  Accessing worklists through worklist APIs is faster than accessing them through WLI JPD controls. However, controls are easier to use and program. Accessing the worklist directly through the API avoids the overhead of control runtime as controls internally use worklist APIs.

  – Accessing worklists through JPD controls is recommended in scenarios where process orchestration requires worklist access.

  – For plain worklist manipulations, worklist APIs work faster than JPD controls.

- **JPD state management**

  Stateless JPDs are executed in memory and the states are not persisted; therefore, they provide better performance than stateful JPDs. In a scenario where you do not need information about the previous state of a process, use stateless JPDs. There is no

input/output cost for stateless JPDs as the state is not persisted. Stateless JPDs are inherently more scalable.

- **Callbacks over JMS**

  For an asynchronous process, if the callback location is a WLS JMS queue and is same for all instances of the process, WLI performance is affected under high load conditions.

- **Synchronous vs. asynchronous process**

  An asynchronous process has some latency cost. Consider a task that requires a few hundred milliseconds for completion.

  – If this task is designed as an asynchronous JPD, the overhead of the asynchronous processing infrastructure could increase processing time significantly.

  – If, on the other hand, it is designed as a synchronous JPD, the impact on processing time would not be significant.

In general, asynchronous processing provides tremendous value for lengthy processes. Synchronous processing, on the other hand, is better suited for tasks that are expected to take less time. However, there could be applications in which a synchronous process is blocking, which could be a significant bottle neck, that in turn causes the server to use many more threads than neccessary.

Note: For more information about design considerations that may affect performance, see *Best Practices for WebLogic Integration* and "WLI Tuning" on page B-1.

Tune the Environment

Performance of a WLI application depends not just on the design of the application, but also on the environment in which it runs.

The environment includes the WLI server, the database, the operating system and network, and the JVM. All of these components should be tuned appropriately to extract good performance from the system.

- Tuning WLI

  Appropriate settings should be made for parameters such as JDBC data sources, `weblogic.wli.DocumentMaxInlinSize`, process tracking level, B2B message tracking level, and Log4j. For more information, see "WLI Tuning" on page B-1.

- Tuning the Database

This includes defining settings for initialization parameters, generation of statistics, disk I/O, indexing, and so on. For more information about tuning the database, see *Database Tuning Guide*.

- Tuning the Operating System and Network

  Proper tuning of the OS and the network improves system performance by preventing the occurrence of error conditions. For more information, see "Operating System Tuning" in *WebLogic Server Performance and Tuning*.

- Tuning the JVM

  The JVM heap size should be tuned to minimize the time that the JVM takes to perform garbage collection and maximize the number of clients that the server can handle at any given time. For more information, see "Tuning Java Virtual Machines" in *WebLogic Server Performance and Tuning*.

Prepare the Application for Performance Testing

Certain minor changes may need to be made in the application for running the performance tests and for invoking the application through load generator scripts.

The extent of change depends on the nature of the application, capability of the load generator, and the outcome that is expected from the capacity planning process.

Following are examples of the changes that may be required:

- If you want to measure end-to-end performance (from process invocation to completion) of an asynchronous JPD without writing callback handlers in the load generator script, you can generate a SyncAsync WSDL for the process and use it in the load generator. This would make the load generator script simple, leaving all the complexity in the server; the focus remains on the server rather than on the load generator script.
- If the JPD interacts with a third-party software, and if, you do not want to or cannot use that software (to avoid accounting for delay in calling the third-party software), you can use a simulator in place of the third-party software. This ensures that the test results give reliable information about the true performance and capacity of the WLI application.

Design the Workload

The quality of the result of any performance test depends on the workload that is used.

Workload is the amount of processing that the system is expected to complete. It consists of certain applications running in the system with a certain number of users connecting to and interacting with the system.

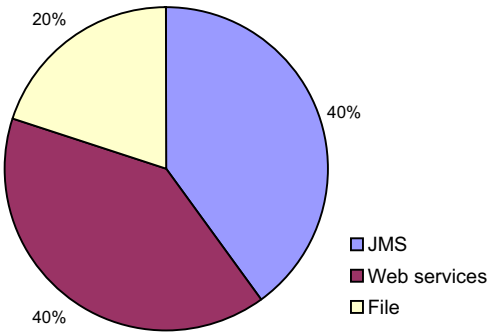
The workload should be designed so that it is as close to the production environment as possible.

- In a production environment with many concurrent users, not all users may perform the same operation; the operations would vary depending on the users' profile and interest.

  In addition, users may require think time, which is the time required by users to think about possible alternatives and take decisions before triggering an action in the system.

  A WLI application that has three types of clients – web services, JMS, and file – may, for example, have a user profile as shown in the following figure.

Figure 2-1 Sample User Profile



- User behavior varies depending on the type of application. A user of a long-running process may not execute operations in quick succession. The behavior of a batch processing user, on the other hand, may be completely different.

The following parameters must be considered while designing the workload:

- Number of users at different load conditions (low, moderate, and peak)
- Average number of concurrent users
- Think time (time that users take to think before performing any action in the system)
- Profile of the users (operations that they are expected to execute)
- Peak message-arrival rate

- Message size

The next step is to define the unit of work and SLA.

Define the Unit of Work and SLA

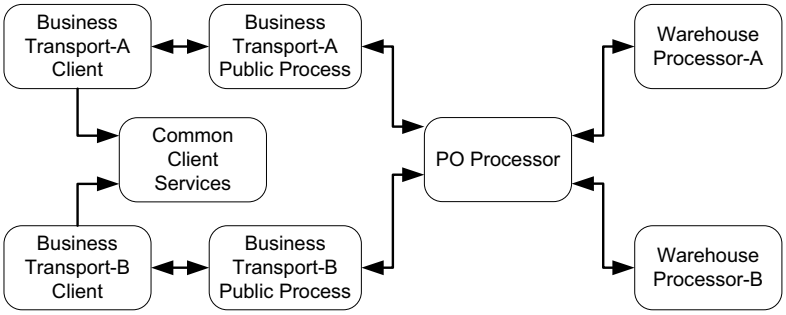
A Service Level Agreement (SLA) is a contract – between the service provider and service consumer – that defines acceptable (and unacceptable) levels of service. The SLA is typically defined in terms of response time or throughput (transactions per second).

For systems of synchronous nature, the aim is to tune the system that achieves highest throughput while meeting the response time (realistically, for about 95% of the transactions). The response time thus becomes the SLA. For systems of asynchronous nature, throughput or messages per second is the SLA.

For capacity planning purposes, it is important to define the unit of work (that is, the set of activities included in each transaction), before using it to define the SLA.

Consider the purchase order application shown in the following figure.

Figure 2-2 Unit of Work: Purchase Order Application



Each node is a JPD. All of these JPDs are required for processing the purchase order. In this scenario, the unit of work (transaction) can be defined as either of the following:

- Each JPD that should be executed to process the purchase order.
- The entire flow of operations, starting from the business transport client to receiving a response from the warehouse processor.

It is recommended that the entire flow of business operations, rather than each JPD, be considered a single unit of work.

The next step is to design the load generation script.

Design the Load Generation Script

A load generation script is required to load the server with the designed workload while running the tests.

Note: For information about running the tests, see"Run Benchmark Tests" on page 2-8 and "Run Scalability Tests" on page 2-14.

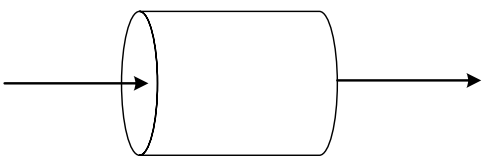
While writing the load generation script, you should keep the following points in mind:

- Ensure that each user sends a new request only after the previous request is served and a response is received.
- If the system processes requests in batches, ensure that a new batch of requests is sent only after the previous batch is processed.

The load level becomes easier to understand and manage, when you limit each simulated user to a single in-flight request,. If the rate at which requests are sent is not controlled, requests may continue to arrive at the system even beyond the flow-balance rate, leading to issues such as queue overflow.

The following figure depicts a single user sending the next request only after the previous request is processed by the server.

Figure 2-3 Balanced Load Generation Script



With this approach, the arrival rate (load) on the system can be increased by increasing the number of concurrent users, without affecting the system adversely; therefore, the capacity of the system can be measured accurately.

The following figure depicts a single user sending new requests without waiting for the server to finish processing previous requests.

Figure 2-4 Non-blocking Script With Arrival Rate More Than Throughput

This approach could cause issues such as queue overflow and lead to misinterpretation of capacity.

A balanced load generation script is recommended.

Configure the Test Environment

The test environment should be configured as described in this section to ensure that the results of the tests are reliable and not affected by external factors.

- When the tests are conducted, no other process must run in any of the test machines – load generator, WLI server, and database – apart from the processes that need to be tested. In addition, no other machine or external program must access or use the database machine.
- OS tasks, such as automatic system update and scheduled jobs, must not interfere with the tests.
- To prevent network-related issues (slow network or network traffic) from affecting test results, it is recommended that a VLAN be set up with all the machines involved in the tests. This ensures that the test machines are isolated from network traffic originating in or intended for machines that are not involved in the tests.
- A network speed of 1000 Mbps is recommended for running the tests. The bandwidth must be monitored while the tests are running.

Run Benchmark Tests

Benchmark tests help in identifying system bottlenecks and tuning the system appropriately.

The tests involve increasing the load on the system, in gradual steps, till the throughput does not increase any further.

Note: For the purpose of benchmark tests, load is any aspect of the WLI application under test – number of concurrent users, document size, and so on – that demands system resources.

The load should be increased gradually to ensure that the system has adequate warm-up time.

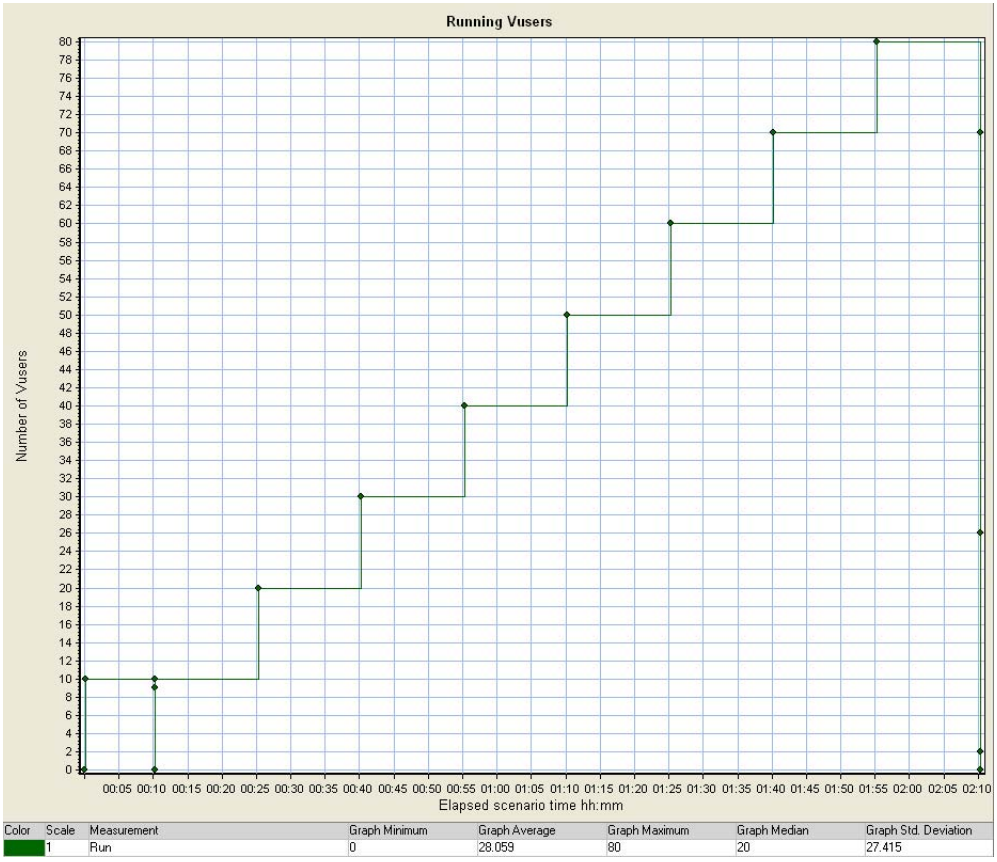
Benchmark tests are run with no think time and with a single WLI machine.

When the throughput stops increasing, one of the following may have occurred:

- Utilization of one of the hardware resources has reached 100%, indicating that the particular resource has become a bottleneck.
- Utilization has not reached 100% for any of the hardware resources, but the throughput has peaked, indicating that the system requires further tuning to make better use of the available hardware resources.

The following figure depicts a Mercury LoadRunner ramp-up schedule in which the initial 10 minutes are for warm-up tests with 10 concurrent users. Subsequently, the load is increased at the rate of 10 additional users every 15 minutes.

Figure 2-5 Test Ramp-up Schedule



The following data must be recorded while running the tests:

- **Application behavior**
  - Concurrent user load
  - Response time
  - Throughput for each unit of work

  Use tools such as Mercury LoadRunner and The Grinder for emulating users and capturing metrics.

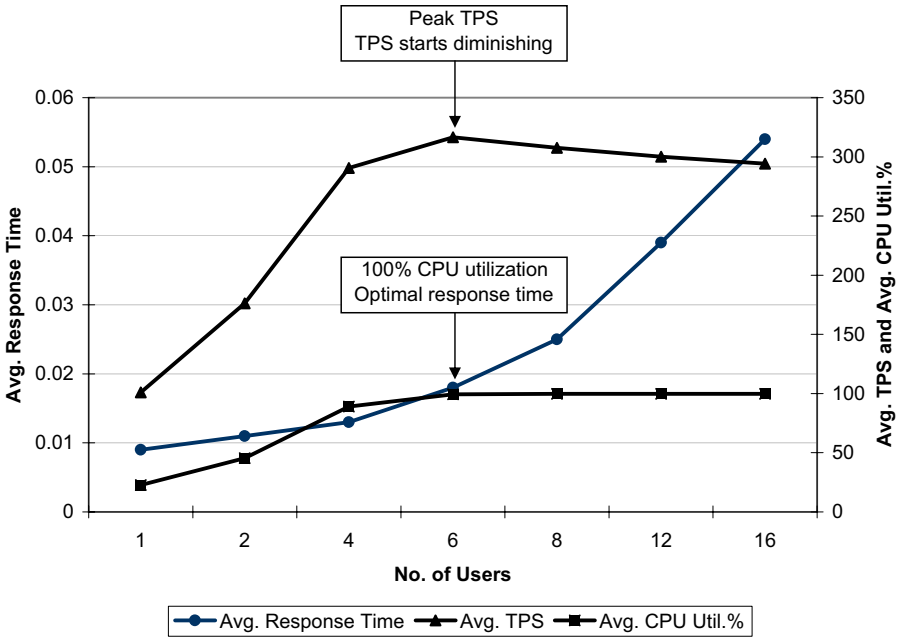
- **Resource utilization**

- CPU utilization
- Memory footprint
- Network utilization
- I/O utilization

If the load generation tool does not capture all of this data, use OS-specific utilities such as vmstat, iostat, and perfmon.

The following figure shows the result of a benchmark test.

Figure 2-6 Results of Benchmark Test



As users are added, the average TPS increases. When utilization of one of the hardware resources (in this case, CPU) reaches 100%, the average TPS peaks. The response time at this point is the optimal result. When further users are added to the system, the TPS starts diminishing. TPS rises in a near linear fashion as an increasing load is applied until the system is saturated due to a CPU or input/output constraint, then it levels off and begins to fall. Response times rise in a near linear fashion until the saturation point is reached, and then increases in a non-linear fashion.

This pattern of results indicates a system and typical behavior where resources are utilized to the maximum.

The next activity in the capacity planning process is to validate the results of the benchmark tests.

Validating the Results Using Little's Law

Before analyzing the test results, you must validate them using Little's Law, to identify bottlenecks in the test setup. The test results should not deviate significantly from the result that is obtained when Little's Law is applied.

The response-time formula for a multi-user system can be derived by using Little's Law. Consider **n** users with an average think time of **z** connected to an arbitrary system with response time **r**. Each user cycles between thinking and waiting-for-response; so the total number of jobs in the meta-system (consisting of users and the computer system) is fixed at **n**.

n = x (z + r)

r = n/x - z

n is the load, expressed as the number of users; **z + r** is the average response time, and **x** is the throughput.

Note: Maintain consistency in defining units. For example, if throughput is expressed in TPS, response time should be expressed in seconds.

Tips for Thorough Validation of Results

- It is recommended that you run benchmarks for at least thirty minutes, and preferably longer.
- Discard at least the first fifteen minutes of data obtained under load to allow dynamic parts of the system a one time warm up, and applications, database caches, and connection pools to be populated. For example, allow:
  - The JVM to compile hot spots to native code (the Sun JVM), or to optimize the native co de at hot spots (JRockit).
  - The JVM to tune the garbage collector (JRockit dynamically picks the GC strategy, and dynamically tunes serveral key parameters).
  - Application and database caches, connection pools to be populated.
- Watch for the variability of individual response times and ensure that irregular effects are not skewing the results, for example, if an occasional operating system job is running in

the middle of a test run. A recommended quality metric is the standard deviation of response times divided by the mean response time. Monitor this value for repeated runs, and check that it is reasonably stable.

Interpreting the Results

While interpreting the results, take care to consider only the steady-state values of the system. Do not include ramp-up and ramp-down time in the performance metrics.

When the throughput saturates, utilization of a resource – CPU, memory, hard disk, or network – must have peaked. If utilization has not peaked for any of the resources, analyze the system for bottlenecks and tune it appropriately.

Tips for Analyzing Bottlenecks and Tuning

- To identify potential bottlenecks related to disk I/O, check whether the disk idle time percentage is low or average disk queue length is consistently high.
  - If this is the case, check for I/O activities caused by the application or WLI, and try to reduce the number of such activities.
  - If this is not the case, then faster hard disks may be required.
- Monitor I/O activities in the database machine.
- To identify memory-related bottlenecks, monitor the heap utilization and garbage collection time.
- In addition, look for the following:
  - Long run-queue for the processor: If this is the case, try reducing the load.
  - High utilization of network bandwidth: If this is the case, upgrade the network speed.
- If bottlenecks are observed in the load generator machine, try using multiple load generator machines.

If no resource bottlenecks exist at the point when throughput saturates, bottlenecks could exist in the application and system parameters. These bottlenecks could be caused by any of the following:

- WLI
  - Check whether any of the tuning parameters described in WLI Tuning could make a difference.

  - Ensure that the application is designed according to the best practices. For more information, see *Best Practices for WebLogic Integration*.
  - Use profiling tools such as ej-technologies' JProfiler, and Quest Software's jProbe to detect CPU or memory issues related to the WLI application. It is recommended that you use JProfiler as jProbe is generally considered too heavy to use under load.

- Database
  - Ensure that the database is tuned appropriately. For more information, see *Database Tuning Guide*.
  - Look for database performance issues by using tools such as statspack and Oracle Performance Manager.

- OS and network

  Ensure that the operating system and network parameters are tuned appropriately. For more information, see *WebLogic Server Performance and Tuning*.

- JVM options
  - JVM parameters can have a significant impact on performance. For more information, see "Tuning Java Virtual Machines" in *WebLogic Server Performance and Tuning*.
  - Use tools such as Sun Java JConsole and BEA JRockit Runtime Analyzer to detect issues related to the JVM.

Run Scalability Tests

A system can be considered scalable, when adding additional hardware resources consistently provides a commensurate increase in performance. Such a system can handle increased load without degradation in performance. To handle the increased load, hardware resources may need to be added.

Applications can be scaled horizontally by adding machines and vertically by adding resources (such as CPUs) to the same machine.

Horizontal and Vertical Scaling

The following table compares the relative advantages of horizontal and vertical scaling:

Table 2-1 Relative Advantages of Horizontal and Vertical Scaling

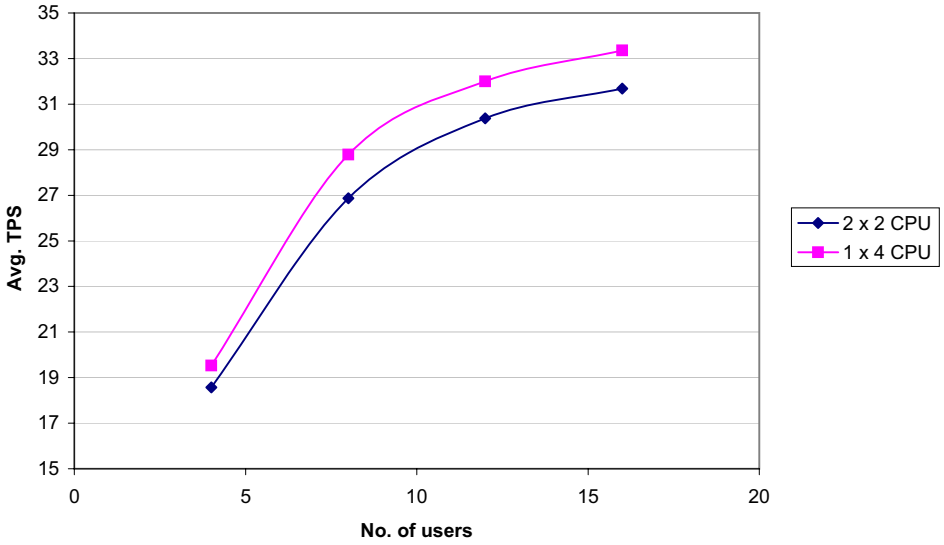
Vertical Scaling (More resources in a single machine)	Horizontal Scaling (More machines)
• Facilitates easy administration. • Improves manageability. • Provides more effective interconnection between system resources.	• Offers high availability. • No scalability ceiling.

When an application needs to be scaled, you may opt for horizontal scaling, vertical scaling, or a combination, depending on your requirements.

- Horizontal scaling is often appropriate for organizations that decide to invest in low-cost servers initially and acquire more machines as the load increases. However, it involves additional costs for load balancing and system administration. An additional advantage of Horizontal scaling is that it suffers less from scalability ceilings that are expensive to address. With Vertical scaling, you cannot scale past the point that your machine is fully loaded without discarding it for another, larger machine.
- Vertical scaling requires a high-end machine that allows easy addition of resources when the need arises.
- A combination of horizontal and vertical scaling is sometimes the best solution; it helps organizations benefit from the relative advantages of both approaches.

The following figure shows a comparison between WLI running on a single non-clustered 4-CPU machine (vertical scaling) and on two clustered 2-CPU machines (horizontal scaling).

Figure 2-7 Horizontal and Vertical Scaling



Performance in the horizontal scaling scenario (two 2-CPU machines) is slightly lower than in the vertical scaling scenario (single 4-CPU machine) due to additional load balancing and clustering overhead in the horizontal scaling scenario. However, you can add additional machines to increase the capacity of the horizontally scaled system. This is not possible with a vertically scaled system.

Conducting Scalability Tests

Scalability tests help you find out how the application scales when additional resources are added in the system – horizontally and vertically. This information is useful for estimating the additional hardware resources required for a given scenario.

The scalability test involves increasing the load, in gradual steps, till the SLA is achieved or the target resource utilization is reached, whichever occurs first.

Note: In contrast, benchmark tests involve increasing the load till the throughput stops increasing.

For running scalability tests, the workload should be designed to emulate, as closely as possible, the production scenario. If no human user interaction is necessary and if the process invocations

happen programmatically, it is recommended that you use a zero-think-time approach, similar to the approach for benchmark tests.

If the target resource utilization level is reached before the SLA is achieved, additional resources must be added to the system. The additional resources (vertical scaling) or machines (horizontal scaling) must be added in the order 1, 2, 4, 8, and so on.

Note: A minimum of three data points must be used to derive the equation for estimating capacity.

All the data that was recorded while running benchmark tests must be captured while running the scalability test. For more information, see "Run Benchmark Tests" on page 2-8.

Note: Only the data that is recorded when the resource utilization is closest to the target level must be used to estimate the additional resource requirement.

After running the test, validate and analyze them as described for benchmark tests, and then, if required, estimate the additional resource requirement as described in the next section.

Estimate Resource Requirement

A Capacity Plan helps estimate the current and future resource requirements for the current SLA and for future loads. You need to create the load model of the system in order to create a capacity plan.

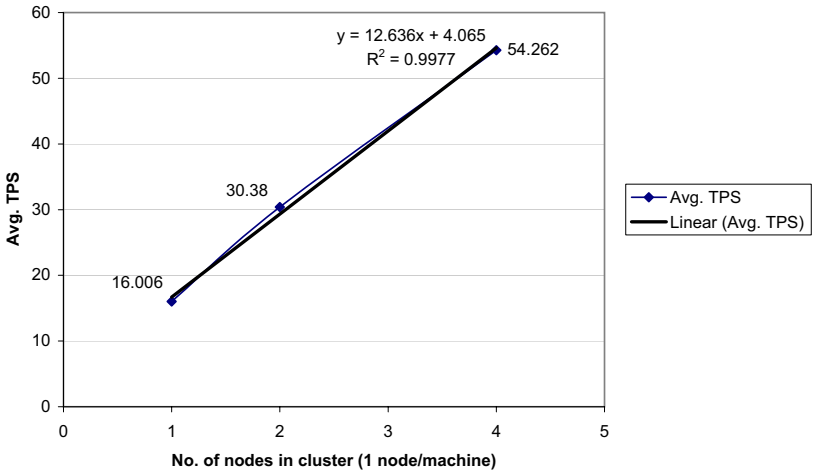
The test results provide the data points to create this load model. You can derive an equation for the curve obtained from the test results, and use it to estimate the additional hardware resources that are required. Use techniques such as linear regression and curve fitting to predict the required resources. You can implement these techniques using spreadsheet applications such as Microsoft Excel.

Note: The accuracy of the prediction depends on the correctness of the load model. The load model should be based on each relevant resource for your application such as CPU.

You can also validate the model against the available historical performance data.

The following figure shows the results of a horizontal scalability test.

Figure 2-8 Capacity Estimation: Horizontal Scaling



The graph shows the average number of transactions per second (TPS) at 70% CPU utilization for clusters with varying number of nodes.

For the results of this scalability test, a linear equation is the best fit. In a best fit curve, $\mathbf{R^2}$ must approach unity (the value, 1).

The equation is **y = 12.636x + 4.065**, where **y** is the average TPS and **x** is the number of nodes.

Note: Though adding additional resources horizontally or vertically can result in a higher TPS, this may not be useful if the objective is to achieve a certain response time. In such cases, consider using faster CPUs.

Based on the results of the scalability tests and the tuning that is necessary for achieving the required results, you should configure the application for deployment to the production environment.

Note: If the resources that you decide to purchase for the production environment are not of the same type or model as those used for the scalability tests, you can estimate the resource requirement by using the following formula:

E x **T1** / **T2**

E = Estimation from scalability tests

T1 = SPECint rate of the machine on which the test was executed

T2 = SPECint rate of the machine that you want to purchase

This formula is applicable only if the scaling is based on number of CPUs.

The extrapolation using SPECint rate is only an approximation. The Capacity Planning exercise is best conducted with the same hardware and configuration as the production environment. For more information about SPECint rates, see http://www.spec.org.

APPENDIX A

Capacity Planning Example

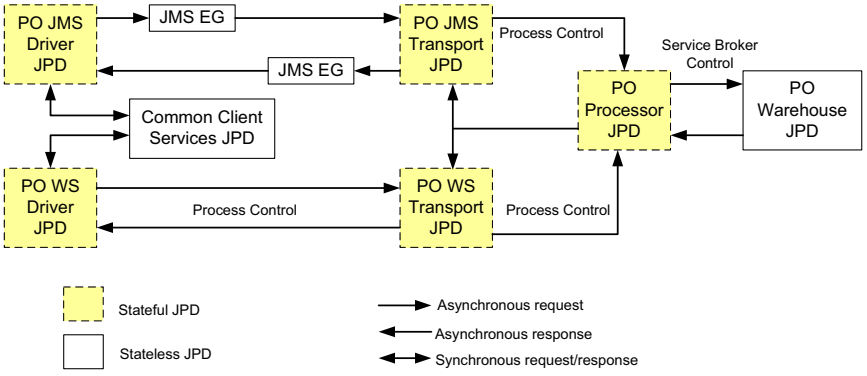
This example is based on a simple Purchase Order application that processes the following Business Object Documents (BODs) defined according to Open Applications Group (OAG) standards:

- A purchase order (OAG 003_process_po_007)
- An acknowledgement (004_acknowledge_po_008)

Note: The tests described in this guide were conducted in a controlled environment; so the numbers presented here may not match the results that you get when you run the tests in your environment. The numbers are meant to illustrate the capacity planning process.

The following figure shows the configuration of the application.

Figure A-1 Configuration of Sample Purchase Order Application



The application has the following components:

- Two business transport clients:
  - PO JMS Driver JPD
  - PO WS Driver JPD
- Two business transport public processes:
  - PO JMS Transport JPD
  - PO WS Transport JPD
- A purchase order processor (PO Processor JPD), which accepts a purchase order, iterates through the line items in it, and returns a callback.
- A warehouse processor (PO Warehouse JPD).

  Each line item in the purchase order has a description, which is mapped to a specific warehouse by using dynamic control properties.
- Common services for logging, business-level document validation, and so on.

Workload and Load Generation Script

In this example, the workload is designed with zero think time because the process is automated (no human user interaction) and the goal is in terms of transactions processed per second.

For this example, a mix of 50% PO WS Driver JPD users and 50% PO JMS Driver JPD users is considered. A Mercury LoadRunner-based Java program is used as a load generation client for the tests. The client triggers the Driver JPDs by using SOAP/HTTP and waits for a callback before sending the next request.

Unit of Work and SLA

The entire flow of operations for processing the purchase order – invoking the driver JPD, sending the purchase order document, and receiving an acknowledgement document – is considered a single unit of work.

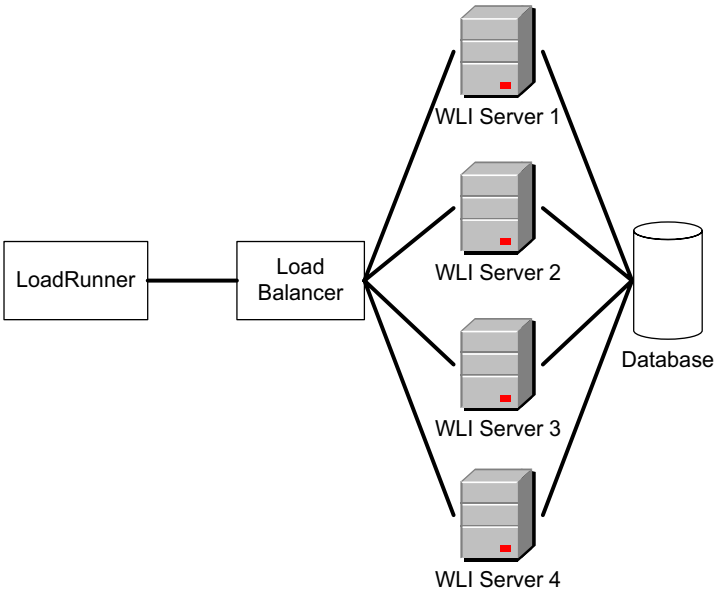
The target SLA for this example is 50 purchase orders processed per second, with a maximum average CPU utilization of 50%.

Test Setup

The following sections describe the hardware and software configurations for executing the scalability tests.

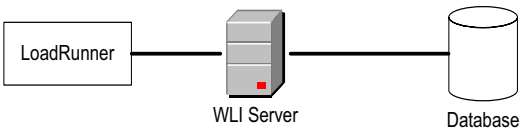
Horizontal Scalability

Figure A-2 Test Setup: Horizontal Scalability



Vertical Scalability

Figure A-3 Test Setup: Vertical Scalability



Hardware Configuration

Table A-1 Hardware Configuration

Purpose	Model	OS	CPU	RAM	Hard Drive	Network Interface
WLI (Horizontal Scalability)	Dell 6850	Windows® Server 2003, Enterprise Edition	2 x 3.4 GHz Intel Xeon Dual Core	16 GB	5 x 146 GB; 10K RPM SCSI	1000 Mbps Ethernet
WLI (Vertical Scalability)	Dell 6850	Windows® Server 2003, Enterprise Edition	4 x 3.4 GHz Intel Xeon Dual Core	16 GB	5 x 146 GB; 10K RPM SCSI	1000 Mbps Ethernet
Database	HP DL 580 G4	Windows® Server 2003, Enterprise Edition	4 x 3.4 GHz Intel Xeon Dual Core	16 GB	8 x 72 GB 15K RPM SCSI with 512 MB battery-backed write cache	1000 Mbps Ethernet
Load Generator	Dell 1850	Windows® Server 2003, Enterprise Edition	2 x 3.0 GHz Intel Xeon HT	4 GB	2 x 146 GB SCSI	1000 Mbps Ethernet
	HP DL 360	Windows® Server 2003, Enterprise Edition	2 x 3.0 GHz Intel Xeon HT	4 GB	1 x 136 GB SCSI	1000 Mbps Ethernet
Load Balancer (Horizontal Scalability)	F5 BIG IP 1500	--	--	--	--	1000 Mbps Ethernet

Software Configuration: Horizontal Scalability

- **WLI**
  - HTTP logging: Off
  - Log4j tuning: Log level 'Error'
  - Server log levels: Error (or Warn, if that is the highest level)

  - JDBC drivers
    - Oracle Thin Driver for `cgDataSource-nonXA` and `cgDataSource`
    - Oracle Thin XA for all other data sources
  - JVM: BEA JRockit 1.5.0_06-b05
  - JVM parameters: `-Xms 1536m -Xmx 1536m -Xgc:parallel`
  - cgDataSource:
    - InitialCapacity = MaxCapacity = 75
    - LLR enabled
  - System configuration:
    - Reporting data stream: Disabled
    - Default tracking level: None
    - Default reporting data policy: Off
    - Default variable tracking level: Off
    - Reliable tracking: Off
    - Reliable reporting: Off
  - WLI JMS control: **send-with-xa** = True
  - JMS persistence: Off (Non-persistent queues used for AsyncDispatcher and `wli.internal.instance.info.buffer`)
- **Load generator**: Mercury LoadRunner 8.1 FP4
- **Big IP load balancer**: Server configured to type 'Performance (Layer 4)'

Software Configuration: Vertical Scalability

The software configuration for vertical scalability tests is the same as for horizontal scalability tests.

Database Configuration

- Database: Oracle 9i - 9.2.0.6.0.
- Archive log disabled.

- Separate tablespace for WLI, WLI archive, worklist, indexes, and JPD BLOB in different hard disks.
- Database machine started with the `/3 GB` switch.
- `Init.ora` parameters:
  - `*.aq_tm_processes=1`
  - `*.background_dump_dest='D:\oracle\admin\PERFDB2\bdump'`
  - `*.compatible='9.2.0.0.0'`
  - `*.control_files='D:\oracle\oradata\PERFDB2\CONTROL01.CTL','D:\oracle\oradata\PERFDB2\CONTROL02.CTL','D:\oracle\oradata\PERFDB2\CONTROL03.CTL'`
  - `*.core_dump_dest='D:\oracle\admin\PERFDB2\cdump'`
  - `*.db_block_size=8192`
  - `*.db_cache_size=25165824`
  - `*.db_domain=''`
  - `*.db_file_multiblock_read_count=16`
  - `*.db_name='PERFDB2'`
  - `*.dispatchers='(PROTOCOL=TCP) (SERVICE=PERFDB2XDB)'`
  - `*.fast_start_mttr_target=300`
  - `*.hash_join_enabled=TRUE`
  - `*.instance_name='PERFDB2'`
  - `*.java_pool_size=33554432`
  - `*.job_queue_processes=10`
  - `*.large_pool_size=8388608`
  - `*.open_cursors=300`
  - `*.pga_aggregate_target=55165824`
  - `*.processes=1000`
  - `*.query_rewrite_enabled='FALSE'`
  - `*.remote_login_passwordfile='EXCLUSIVE'`
  - `*.shared_pool_size=90331648`
  - `*.sort_area_size=524288`
  - `*.star_transformation_enabled='FALSE'`
  - `*.timed_statistics=TRUE`

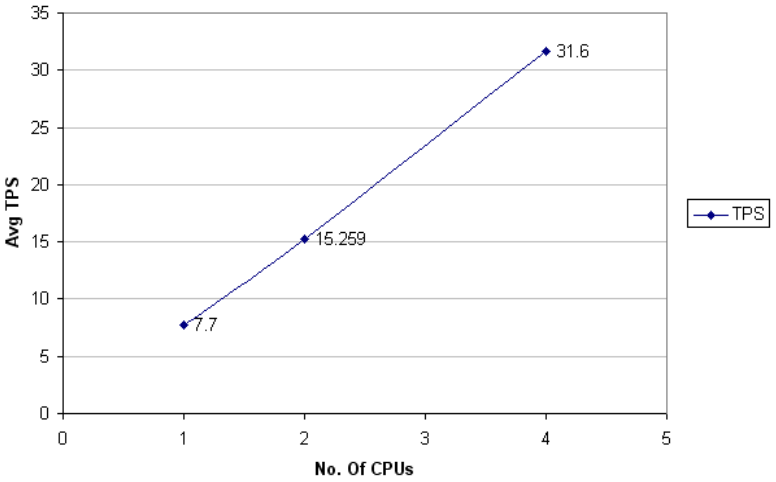
```
- *.undo_management='AUTO'
- *.undo_retention=10800
- *.undo_tablespace='UNDOTBS1'
- *.user_dump_dest='D:\oracle\admin\PERFDB2\udump'
```

Scalability Test Results

The following sections describe the results of the scalability tests.

The following figure shows the results of the vertical scalability tests.

Figure A-4 Results of Vertical Scalability Tests

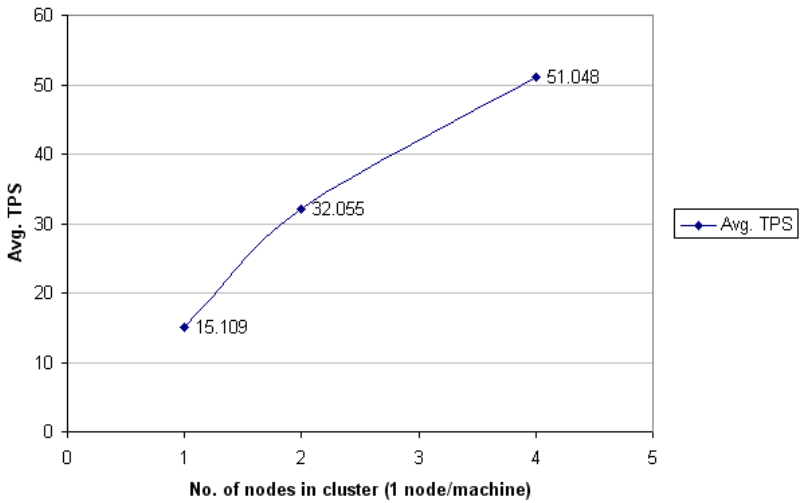


The tests were executed with 1, 2, and 4 CPUs for WLI. The load was increased gradually till the CPU utilization reached 50%. The graph shows that the SLA of 50 purchase orders processed per second with a maximum average CPU utilization of 50% cannot be achieved with four CPUs.

You can fit a curve for the results obtained from the tests, derive an equation for the curve, and use it to estimate the additional hardware resources required, as described in "Capacity Estimation" on page A-10.

The following figure shows the results of the horizontal scalability tests.

Figure A-5 Results of Horizontal Scalability Tests



The tests were executed with 1, 2, and 4 machines, with one managed server in each machine. The graph shows the average TPS at 50% CPU utilization. The graph shows that a 4-node WLI cluster (with one machine per node) can process 51.048 purchase orders at 50% CPU utilization.

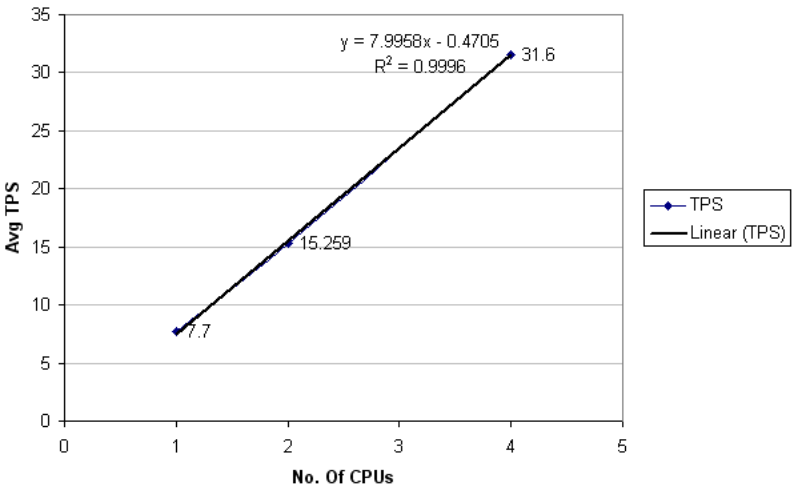
Capacity Estimation

In the example described here, the required SLA was achieved with the available hardware resources in the horizontal scaling scenario, whereas it was not achieved in the vertical scaling scenario.

If the required SLA is not achieved, you can fit a curve for the results obtained from the tests, derive an equation for the curve, and use it to estimate the additional hardware resources required.

The following figure shows capacity estimation in the vertical scaling scenario.

Figure A-6 Capacity Estimation: Vertical Scaling

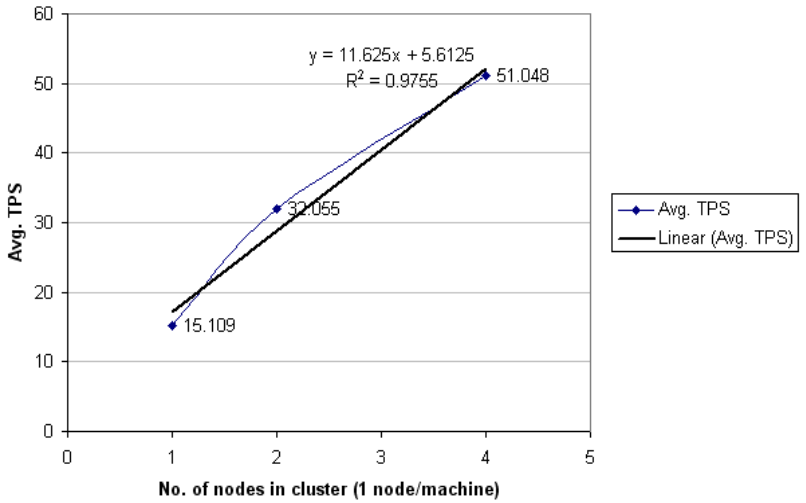


For this example, the equation is **y = 7.9958x - 0.4705**, where **y** is the average TPS and **x** is the number of CPUs. The curve indicates that WLI applications scale linearly.

Note: This equation is based on the test setup described in this guide. The equation that you must use to estimate capacity would depend on your test setup, nature of your application, and the SLA. In addition, the tests described in this document assume an environment where database- and database I/O-related bottlenecks do not exist.

The following figure shows capacity estimation in the horizontal scaling scenario.

Figure A-7 Capacity Estimation: Horizontal Scaling



The equation is **y = 11.625x + 5.6125**, where **y** is the average TPS and **x** is the number of nodes in the cluster.

Note: This equation is based on the test setup described in this guide. The equation that you must use to estimate capacity would depend on your test setup, nature of your application, and the SLA. In addition, the tests described in this document assume an environment where database- and database I/O-related bottlenecks do not exist.

APPENDIX B

WLI Tuning

This section describes the WLI parameters that you can tune in a production environment. This section discusses the following topics:

BPM Logging Levels

The `ApacheLog4jCfg.xml` file located in the WLI domain directory specifies the WLI BPM logging configuration and the logging levels. You must set priority levels to ‘debug’ or ‘info’ only if such messages are required for debugging and testing purposes. In a production environment, you must set priority values to ‘warn’ or ‘error’ for all log appenders.

You must set the debug levels in the WebLogic Server console as the Worklist application creates a log on WebLogic Server as well as Log4j. See JMS Server: Logging for more information.

JDBC Data Source Tuning

WLI uses the following JDBC data sources:

- cgDataSource — A data source used for storing JPD conversation state, process tracking information, and logging last resource information.
- cgDataSource-nonXA — A non-XA data source used for persistent store of internal WLI queues.
- p13nDataSource — A data source used by worklist.
- bpmArchDataSource — A data source for archiving database tables.

You must monitor these JDBC data sources and ensure that an adequate number of connections are available in the connection pool by modifying the values of Initial Capacity, Maximum Capacity, and Capacity Increment. It is recommended that InitialCapacity be set to MaxCapacity for your connection pool, so there is no time spent in growing or shrinking the pool.

If your application is heavily database driven, consider the following:

- Pinned-To-Thread property

  To minimize the time that an application takes to reserve a database connection from a data source and to eliminate contention between threads for a database connection, you can enable the `Pinned-To-Thread` property for the data source. However, when this property is set, the connection remains with the execute thread and is not returned to the connection pool. If the pool does not contain any connection, more connections are created. These connections are more than what you specified in MaxCapacity. Therefore, use the PinnedToThread property only if it improves performance.

  For more information, see Using Pinned-To-Thread Property to Increase Performance in *WebLogic Server Performance and Tuning*.

- Statement cache

  When you use a prepared statement or callable statement in an application or EJB, there is considerable processing overhead for the communication between the application server and the database server and on the database server itself. To minimize the processing costs, WLS can cache the prepared and callable statements used in your applications. It is recommended that you use Least Recently Used (LRU) algorithms and set cache size to the number of statements executed in your application. Determining the exact cache size is a trial-and-error process.

  For more information, see Increasing Performance with the Statement Cache and Tuning Data Source Connection Pool Options in *Configuring and Managing WebLogic JDBC*.

AsyncDispatcher MDB Tuning

The default WebLogic Server work manager configuration constrains the number of AsyncDispatcher requests being processed for a particular JPD project to a maximum of 16. Consequently, a maximum number of 16 threads can handle asynchronous requests for the project. Your system may not use this number of threads for CPU bound processing, as WebLogic Server tunes the number of threads dynamically to optimise throughput. However, for input/output bound processing, which is typical for WLI applications, you can remove the constraint by configuring a separate work manager for the project.

You can assign more threads by adding a separate workmanager to AsyncDispatcher, so more AsyncDispatcher MDB instances are created.

The same AsyncDispatcher MDB is used by all JPDs in a JPD project. In the following scenarios, consider splitting the project into smaller projects:

- You start multiple JPDs of the same project concurrently.
- AsyncDispatcher is under high load conditions.

For more information, see Determining the Number of Concurrent MDBs in *Tuning Message-Driven Beans*.

HTTP Control

The default HTTP-pooled connections per host is 2. To increase the pool size per host, you can use the following system property:

```
wli.httpcontrolmaxconnectionsperhost
```

The default HTTP-pooled connections for all hosts is 20. To increase the pool size for all hosts, you can use the following system property:

`wli.httpcontrolmaxtotalconnections`

Internal Queues

WLI uses many internal queues in the conversational-jms module. By default, most of these queues use a persistent JDBC store.

If the queues do not require JDBC persistence, you can change it to 'Non-Persistent' or use the File Persistence parameter. The `wli.internal.instance.info.buffer` queue used for process instance information can significantly improve the performance of WLI when the queue is changed to Non-Persistent.

Note: Changing the persistence of a queue can cause data loss if the server or the machine crashes unexpectedly.

For more information, see Tuning the WebLogic Persistent Store and Tuning WebLogic JMS in *WebLogic Server Performance and Tuning*.

JMS EG Tuning

By default, the JMS EG application created in the WLI domain directory has 16 MDB instances due to self-tuning.

You can try to increase the pool size to improve performance. If you want more than 16 MDB instances, you can assign a workmanager with a higher number of threads in the minimum and maximum thread constraints as the dispatch-policy of the EG.

For more information, see Determining the Number of Concurrent MDBs in *Tuning Message-Driven Beans*.

Document Size

The `weblogic.wli.DocumentMaxInlinSize` parameter in `wli-config.properties` in the WLI domain directory specifies the maximum size of a document stored in the SQL Document Store. If a document (incoming message) is greater than the value specified in `weblogic.wli.DocumentMaxInlinSize`, only document references are included in the message. Otherwise, the document is passed as a copy. If the message is large, the document store is contacted more frequently, thereby increasing database I/Os and decreasing memory usage. Therefore, the value of the threshold is important.

You can determine the size of your message, using Message Tracking. Tune the threshold value to improve performance. If a document is exchanged among many JPDs, you can use a document store to pass the relevant document reference on to the JPDs instead of passing the same data on to multiple JPDs.

Process Tracking

Table B-1 lists the different levels of process tracking and the events generated. From performance perspective, the system performance degrades as we move from none to full level of process tracking.

Table B-1 Levels of Process Tracking

Level	Description
NONE	No event is generated.
MINIMUM	Only global events, such as process start/end, suspend, and resume are recorded.
NODE	Each executed node generates a start node event and an end/abort node event.
FULL	NODE and events that result from the execution of JpdContext.log() are generated.

If you change NONE to FULL, many more events are generated. These events lead to more database operations towards process instance information and process tracking data. You can partition the WLI_PROCESS_EVENT table, which is affected by higher levels of process tracking.

For more information about partitioning the WLI_PROCESS_EVENT table in *Oracle9i Database Tuning Guide*. For more information about process tracking, see Process Tracking Data in *Using the WebLogic Integration Administration Console*.

B2B Process Tracking

For trading partner integration, the following levels of message tracking are provided:

- None
- Metadata
- All

When you change the message tracking level from None to All, the database operations can have an impact on performance. For more information, see Configuring Trading Partner Management in *Using the WebLogic Integration Administration Console.*

Worklist Reporting and Querying

In a scenario where database load is high, the worklist-reporting feature can have an impact on performance. This feature requires additional database operations for reporting. It is recommended that you move the reporting store to a different database machine by pointing a separate reporting datasource to a database machine other than p13ndatasource.

For more information about the default reporting store, see Default Reporting Store in *Using Worklist*.

Do not try to fetch all worklist task data at once. Your system slows down when the database contains a lot of tasks. You can use `WorklistTaskQuery.getTaskDataCursor` to fetch a maximum of 1000 tasks from the database. `WorklistTaskDataCursor` fetches tasks in batches, as specified in `batchSize`.

JMS Control Tuning

If you are using WLI JMS control, you can set the `send-with-xa` annotation of the WLI JMS control to `true` for better performance with WLI.

Test Console

The test console stores all logs in the JVM heap. You must clear the logs so that system performance is not affected.

By default, the test console (`testConsoleFlag` in the `setDomainEnv` file) is enabled for a development domain and disabled for a production domain. Do not enable the test console for production systems unless required for debugging purposes.

JPD Meta Data Cache Tuning

To control the behaviour of the JPD process meta-data cache you can do either of the following:

- Set the `wlw.softDispReferences` system property to allow caching the meta-data using soft-references so that these references are relinquished when there is a memory crunch.

The only drawback is that soft-references can significantly increase garbage collection times.

- Set the following optional `com.bea.wli.dispatcher.cacheSize` property in the JAVA_PROPERTIES of `setDomainEnv.cmd/sh` variable. This is a numeric value representing the boundary limits of the cache, which is interpreted as follows:
  - A value of -1 represents an unbounded cache. All the meta-data of the JPDs is cached.
  - A value of 0 disables the cache completely. None of the meta-data of the JPDs is cached.
  - A positive non-zero value represents a bounded cache. If the value is 'n', then the meta-data of 'n' last accessed JPDs is cached.