White Paper

J2EE Performance Tuning

Table of Contents

Introduction and Background Information.....2

Issues Related to J2EE Performance Tuning.....2

Need for Performance Tuning 3

Typical Approach for Tuning Performance..... 4

A Holistic Approach..... 6

When Can You Conclude the Tuning?..... 8

Conclusion.....10

Introduction and Background Information

As the complexity and the scope of enterprise operations increase, so does the hazards of failure. Various studies show that serious application issues occur in 15% of enterprises every day and around 53% of enterprises every week.

J2EE represents a complex environment often found within business-critical and multi-tier applications.

Most J2EE applications use well defined interfaces that are part of the J2EE specification. These interfaces include Servlets, JSPs (Java Server Pages), JDBC (Java Database Connectivity) drivers and EJBs (Enterprise JavaBeans). Usually, the byte code instrumentation of such interfaces brings mechanized recognition of the response time contributions pertaining to the usage of relational databases (RDBMS), enterprise beans, and other user interfaces.

J2EE applications normally spend significant amounts of time performing operations outside the scope of common interfaces such as using third party packages, communicating with legacy applications and executing the business logic that is unique to the application.

Issues related to J2EE Performance Tuning

Addressing the performance issues of these complex J2EE applications may consume an increasing amount of IT staff time. As the complexity and the scope of enterprise operations increase, so does the hazards of failure. Various studies show that serious application issues occur in 15% of enterprises every day and around 53% of enterprises every week.

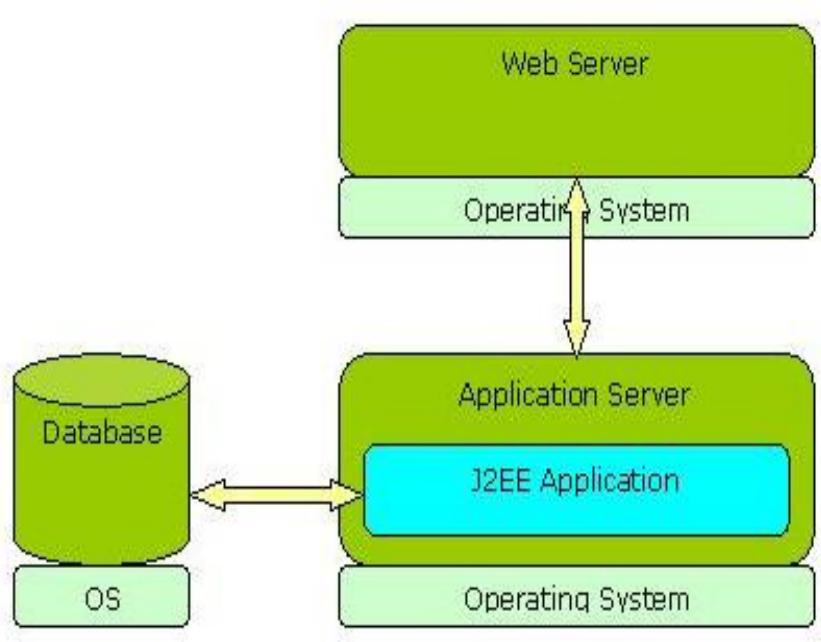
According to a recent Forrester Research survey, many new applications are actually the revamping or front-ending of old applications using J2EE application servers, for instance. The introduction of these technologies and the trends toward using service oriented architecture and web services increase the complexity of applications. Nearly 85% of enterprises with revenue of more than $1 billion reported incidents of significant application performance issues. Survey respondents identified architecture and deployment as the primary causes of these issues. Based on this survey, it seems that no matter how much one tunes the hardware and environment, application issues will persist.

Infonetics Research survey found that medium sized enterprises are losing an average of 1% of their annual revenue, or $867,000, to downtime. Application outages and degradations are the biggest sources of downtime, costing these enterprises $213,000 annually.

Need for Performance Tuning

According to a recent Forrester Research survey, many new applications are actually the revamping or front-ending of old applications using J2EE application servers, for instance.

The obvious way to improve the performance of an application is by scaling up the hardware. But hardware scaling may not work in all scenarios and is certainly not the best cost effective method. Tuning can improve the performance without extra costs for the hardware. In order to obtain the best J2EE application performance, all the underlying layers should be properly tuned.



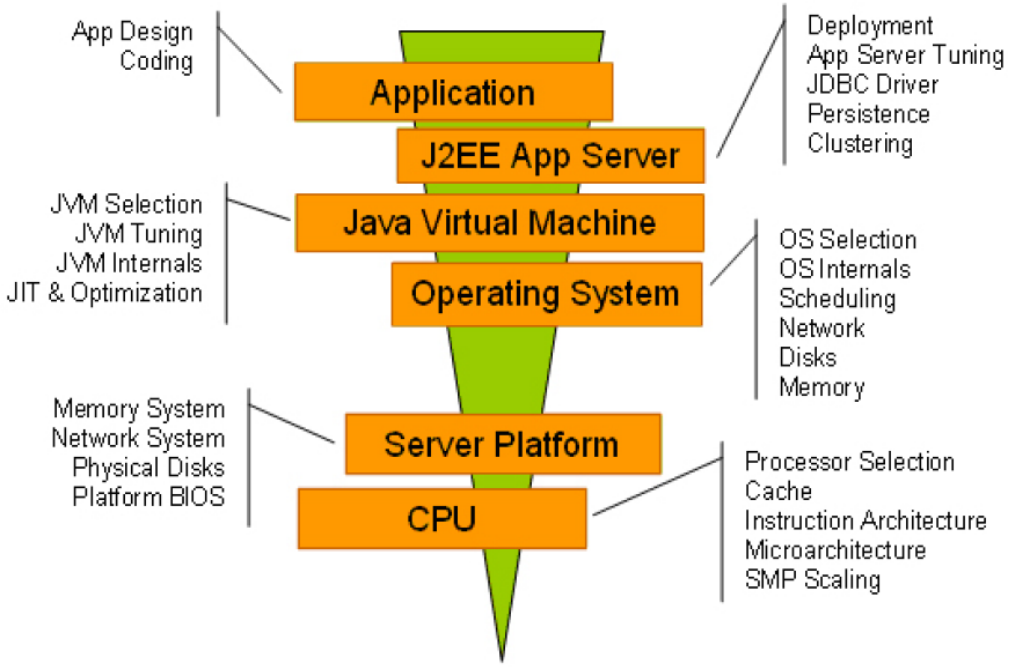
For maximum performance, all the components such as operating system, web server and application server need to be optimized.

Typical Approach for Tuning Performance

Infonetics Research survey found that medium sized enterprises are losing an average of 1% of their annual revenue, or $867,000, to downtime.

In order to achieve the desired performance level, an application implementer has to do all the necessary things to ensure that the application architecture follows precise design principle. An awfully configured application is the source of many performance related issues and it is difficult to maintain.

The configuration of application server involves multiple computers interconnected over a network. So ensuring an adequate level of performance in such an environment requires a systematic approach. There are many factors that may impact the overall performance and scalability of the system. Few examples of these performance and scalability related factors include system design decisions, application code efficiency, configuration and tuning of the database, network I/O activity, system topology, disk I/O activity and Operating System (OS) configuration.



Performance optimization considerations are distributed across various levels of the top-down stack:

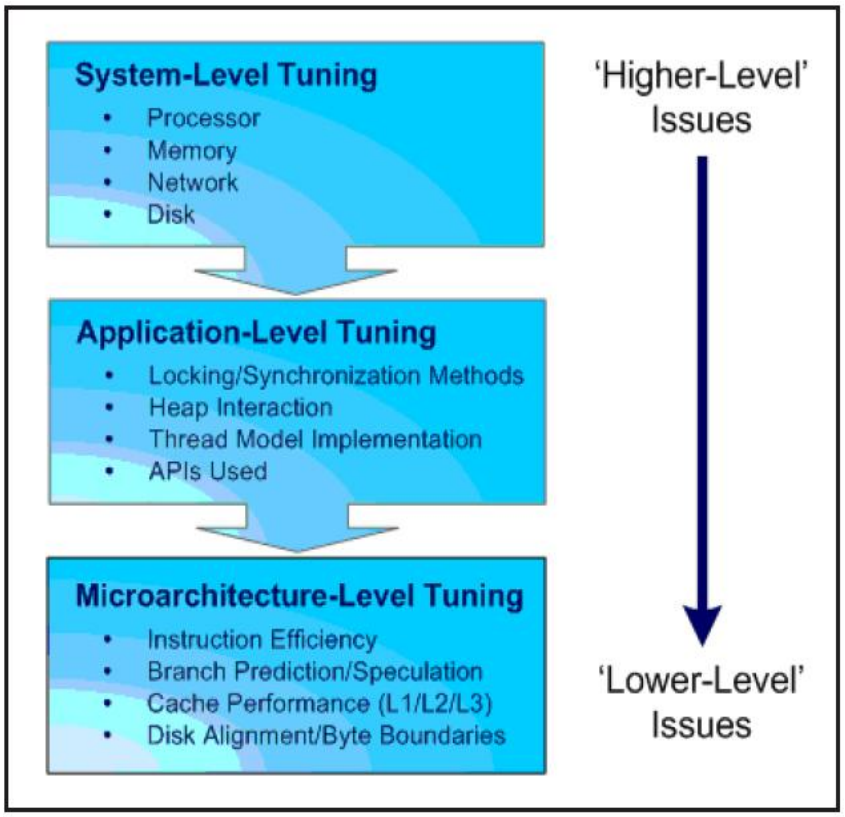
System level: Performance barriers such as input/output (I/O), operating system and database bottlenecks.

A holistic approach deals with the interaction view of J2EE application with different modules or components.

Application level: Application design considerations and application server tuning.

Machine level: Hardware level performance considerations and JVM implementations such as processor frequency, cache sizes, and multi-processor scaling.

A top-down, data driven and iterative approach is the proper way to improve performance. Here top-down refers to addressing system level issues first, followed by application level issues, and finally issues at the micro-architectural level. The reason for addressing issues in this order is that higher level issues may mask issues that originate at lower levels. The top-down approach is illustrated as below.



Data driven denotes that the performance data must be measured, and iterative means the process is repeated multiple times until the desired level of performance is reached.

A Holistic Approach

Cache objects in frequently used methods so that they will persist between calls. Make sure that the cached object's state is set properly before subsequent use.

A performance tuning expert has to consider the holistic view of any J2EE application. A holistic view means the interaction view of J2EE application with different modules or components. This section focuses on the tuning of various J2EE application layers.

Application Layer Tuning

Here we can tune and optimize our code as explained below.

Proper garbage collection

- You are supposed to reuse the existing objects.
- Creation of unnecessary objects should be avoided.
- You need to avoid explicit garbage collection i.e. avoid calling System.gc().

Use static methods whenever there is a scope

As static methods can not be overridden, you can use them in the following cases:

- If there is some code that can be shared easily by all the instance methods, extract that code into a static method (Or)
- If any operation is not dependent on instance creation, for example: utility class methods.

Using correct parser

- SAX programs are much faster than their DOM equivalents and almost use less memory.
- SAX can be used only for a sequential processing of an XML file whereas DOM can be used for random processing of XML files.
- For parsing large XML files, it is a good idea to go for SAX parser.

Data caching

- Cache objects in frequently used methods so that they will persist between calls. Make sure that the cached object's state is set properly before subsequent use.
- You can utilize browser caching.

Optimized coding

- Writing an intelligent and optimized Java code is one of the most important aspects of performance tuning.

You may compress and decompress the serialized objects and save disk space. This can be done on the fly to improve the performance of client-server applications.

- You can always change the storage location of data based on the frequency of access.
- Import selectively and reuse existing code.
- Write modular code and promote loose coupling.
- Try to use mutable objects. Know when to use immutable objects as they consume memory.
- Do not create unnecessary threads and temporary variables.
- Avoid excessive writing to the Java console. It's not a good idea and it degrades the performance.

Using Data Types

- Primitive types are faster than classes encapsulating the types. It is better to avoid the costs of creating and manipulating the objects by utilizing primitive variable types when prudent. Memory could be minimized and the access time of variables could be enhanced accordingly.
- You must use 'int' whenever appropriate on 32-bit systems ('long' is 64-bit data type).
- Casting in Java is not done at compile time. So avoid unnecessary recasting of variables.
- Use static final keyword while creating constants. You can declare the data as static and final when it is invariant.
- If possible, declare methods as final. Validate all input data at client side.
- Static include is always faster than dynamic include.
- Try to minimize the data stored in HTTP session.
- Disable the session creation in JSP if not required.

Efficient Data Storage

- You may compress and decompress the serialized objects and save disk space. This can be done on the fly to improve the performance of client-server applications (zipping and unzipping the data).
- Try to encode and decode the data by using different classes such as base32 and base64 classes.

Database Layer Tuning

- Whenever you are using select query, try to mention only the required columns. You can just use the columns that you wish to fetch and this will improve the performance.
- You can always optimize your query by using appropriate tools.
- Try to use prepared statements as they are precompiled.
- Whenever you create any statement object or a connection object in your DAO layer, it would be better to close or flush them in a finally block.

Reduce the complexity of J2EE applications. For instance, try to separate the controller logic from the business logic.

- Try to minimize the number of JDBC calls. You can achieve this by using JDBC connection pool or various design patterns such as singleton pattern.

Database Server Tuning

- Create index over the table whenever it is required.
- Whenever your select query involves more than 6 tables, try to denormalize the table. To improve the performance, you need to reduce the number of tables.
- In case of batch operations, it is not a good idea to create SQL queries. SQL query will be compiled every time and then it is executed. You can use stored procedures. Stored procedures are compiled only once when they are executed for the first time.
- Avoid using views unless it is really needed. Querying from view takes more time than directly querying from the table.

Application Server Tuning

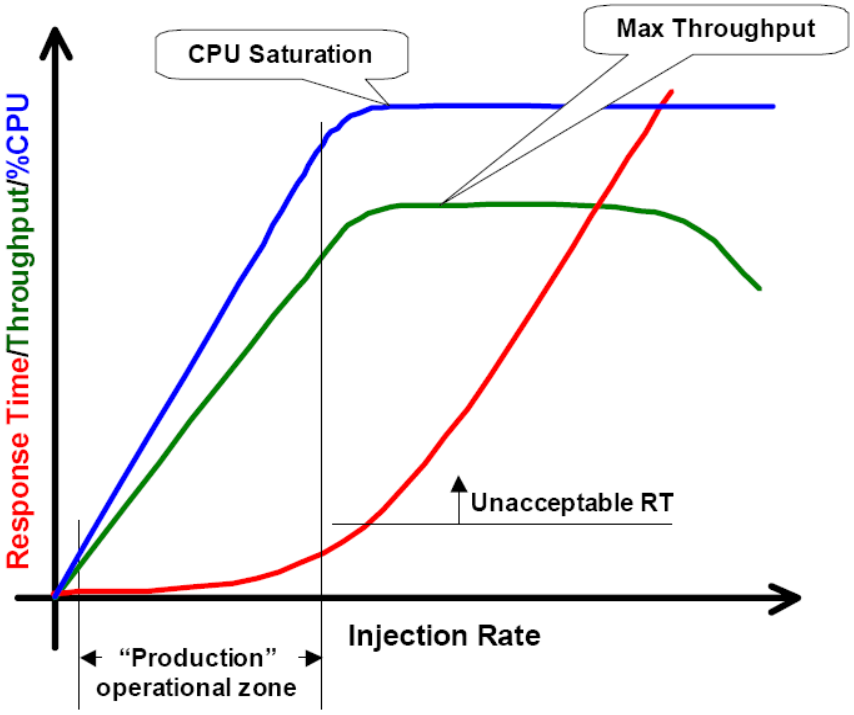
- Here you need to know where to tune and also the extent of tuning.
- Increase the JVM size of the application server. For instance, in Weblogic Admin server, the size of the JVM can be extended by altering the values of Xmx and Xms in setDomainEnv.sh shell script.
- Session attributes must implement java.io.Serializable interface.
- Make your applications distributable.
- Group all related search queries in a single JDBC call. The basic idea of performance tuning in DAO layer is to reduce the number of JDBC calls.
- Optimize the methods that are used most often.
- Reduce the complexity of J2EE application. For example, try to separate the controller logic from the business logic.

When Can You Conclude Performance Tuning?

The primary goal of system level tuning is to saturate the application server CPU i.e. around 90% to 100% utilization. Achieving highest throughput without a fully saturated CPU is an indication of performance bottleneck such as over-synchronization, I/O contention and improper configuration of thread pools. Reaching high response time metrics with an injection rate below the CPU saturation levels may indicate some latency issues such as improper database configuration or excessive disk I/O.

An Operational goal is to make sure that there is adequate capacity available to address all the usage surges.

The application server CPU saturation level denotes that there are no system level bottlenecks outside of the server. The throughput measured at these levels would point out the maximum capacity the system has within the current application implementation. Further tuning includes adjusting the parameters of garbage collection and increasing application server nodes to the cluster.



Note: CPU utilization is not necessarily the only resource to monitor. For instance, a common issue in initial tuning efforts is properly configuring the size of application server thread pools. If thread pools are too small, requests may be forced to wait in an execution queue, increasing response time dramatically. But small thread pools do not exercise the CPU.

As 100% thread pool utilization on a small pool might translate to just 20% CPU utilization, the thread pool resource issue would be missed if only CPU utilization is being monitored.

Hitting CPU saturation level may be a goal for the performance tuning process but need not be an operational goal. An operational goal is to make sure that there is adequate capacity available to address all the usage surges.

Conclusion

For effective results with performance tuning, you need to consider the Holistic approach.

Performance tuning is an ongoing process. Simply put, good system performance depends on good design, good implementation, defined performance objectives and performance tuning. Mechanisms have to be implemented that provide performance metrics which can be compared against the performance objectives defined.

Design applications with performance in mind. You need to keep things simple, avoid inappropriate use of published patterns, apply J2EE performance patterns efficiently and optimize code fragments. For effective results with performance tuning, you need to consider the holistic approach as discussed in the previous sections of this white paper.