

J2EE PERFORMANCE TESTING AND TUNING PRIMER



WHITE PAPER

Gourav Suri

Mphasis Integration Architecture Team

September 2008

Executive Summary

Even though the J2EE applications being developed today are larger-than-ever, users still expect real-time data and software that can process huge amount of data at blazing speeds. With the advent of high speed internet access, users have grown accustomed to receiving responses in seconds, irrespective of how much data the server handles. Hence it is becoming increasingly important for Web applications to provide as short response times as possible.

To achieve this goal of satisfying and/or exceeding customer expectations, it is important that performance testing and tuning are given the importance they deserve in the application development lifecycle.

This document is an attempt in that direction.

Table of Contents

1. INTRODUCTION	3
2. HOW PREVALENT ARE APPLICATION PERFORMANCE ISSUES IN OUR INDUSTRY?	3
3. WHAT IS THE NEED FOR PERFORMANCE TUNING?	3
4. WHAT IS PERFORMANCE TUNING?	4
5. WHAT ARE THE PERFORMANCE METRICS USED FOR DEFINING THE PERFORMANCE OF AN ENTERPRISE APPLICATION?	4
6. HOW WILL THE BUSINESS BE IMPACTED IF APPLICATION PERFORMANCE IS NOT UP TO THE MARK?	4
7. WHAT IS PERFORMANCE TESTING?	5
8. HOW CAN THE NUMBER OF USERS USING A SYSTEM BE ACCURATELY JUDGED?	5
9. WHAT ARE PERFORMANCE TESTS?	5
10. WHAT IS THE USE OF CONDUCTING PERFORMANCE TESTS?	5
11. WHY SHOULD AN AUTOMATED TOOL BE USED FOR PERFORMANCE TESTS?	5
12. WHAT IS A LOAD TESTER?	5
13. WHAT ARE LOAD TESTS?	6
14. WHAT IS THE USE OF CONDUCTING LOAD TESTS?	7
15. WILL ALL SYSTEMS HAVE A PERFORMANCE BOTTLENECK?	7
16. WHAT IS A THROUGHPUT CURVE? WHY IS IT USEFUL?	7
17. IN AN APPLICATION DEVELOPMENT LIFE CYCLE, WHEN SHOULD A PERFORMANCE TEST BE PERFORMED?	8
18. WHAT ARE THE PRE-REQUISITES FOR PERFORMANCE TESTING?	9
19. HOW SHOULD THE TEST ENVIRONMENT BE SETUP FOR PERFORMANCE/LOAD TESTS?	10
20. WHAT IS BASELINE DATA?	10
21. WHEN ARE PERFORMANCE GOALS FOR THE SYSTEM DEFINED?	10
22. WHERE SHOULD BASELINE DATA BE ESTABLISHED?	10
23. IS THE STRATEGY FOR TUNING AN EXISTING APPLICATION DIFFERENT FROM THE STRATEGY FOR TUNING NEW APPLICATIONS?	11
24. HOW SHOULD A LOAD TEST FOR AN EXISTING APPLICATION BE DESIGNED?	11

Table of Contents

25. HOW SHOULD A LOAD TEST FOR A NEW APPLICATION BE DESIGNED?	11
26. WHAT IS RESPONSE TIME?	12
27. WHAT ARE THE ACCEPTABLE VALUES FOR RESPONSE TIME?	12
28. HOW SHOULD THE RESPONSE TIME BE MEASURED?	12
29. WHAT APPROACH SHOULD BE FOLLOWED FOR TUNING PERFORMANCE?	13
30. WHAT STEPS SHOULD THE ITERATIVE PERFORMANCE TUNING METHODOLOGY HAVE?	14
31. WHEN SHOULD THE PERFORMANCE TUNING EXERCISE BE CONCLUDED FOR AN APPLICATION?	14
32. WHAT ARE THE VARIOUS TOOLS THAT ARE USED WHILE EXECUTING PERFORMANCE/LOAD TESTS?	15
33. WHAT DATA SHOULD I CAPTURE WHILST CONDUCTING PERFORMANCE TESTS?	15
34. WHAT FACTORS INFLUENCE THE PERFORMANCE OF AN APPLICATION SERVER SOFTWARE?	16
35. WHAT TUNABLE PARAMETERS HAVE A CONSIDERABLE IMPACT ON THE PERFORMANCE OF MY APPLICATION SERVER?	19
36. HOW DOES A USER REQUEST TRAVERSE AN ENTERPRISE JAVA ENVIRONMENT?	22
37. HOW DO PERFORMANCE MONITORS HELP IN IDENTIFYING THE BOTTLENECKS?	23
38. ONCE A BOTTLENECK HAS BEEN IDENTIFIED, HOW CAN IT BE RESOLVED?	24
39. HOW DOES GARBAGE COLLECTION WORK?	29
40. WHAT ARE FAILOVER TESTS?	30
41. WHAT ARE SOAK TESTS?	30
42. WHAT ARE STRESS TESTS?	31
43. WHAT IS TARGETED INFRASTRUCTURE TEST?	31
44. WHAT ARE NETWORK SENSITIVITY TESTS?	32
45. WHAT ARE VOLUME TESTS?	33
46. FURTHER READING	34
47. REFERENCES	34

1. Introduction

J2EE Performance Testing and Tuning - An overview of what, why, when, where and how is what this document is all about.

As it is this topic is vast, add to it vendor specific tools and techniques - would just one book be sufficient forget one document in covering the length and breadth of this topic.

Load balancing, clustering, caching, scalability and reliability are topics that can be taken up as separate documents and hence, are not delved in detail in this document.



2. How prevalent are Application Performance issues in our industry?

The quality and performance of enterprise applications in our industry is astonishingly poor. Consider the following:

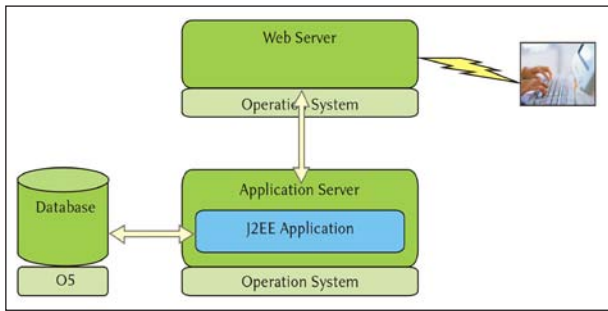
- According to Forrester Research, nearly 85% of companies with revenue of more than \$1 billion reported incidents of significant application performance problems. Survey respondents identified architecture and deployment as the primary causes of these problems. Based on this survey, it seems that no matter how much one tunes the hardware and environment, application problems will persist.
- Infonetics Research found that medium-sized businesses (101 to 1,000 employees) 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 companies \$213,000 annually.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

3. What is the need for Performance Tuning?

The most obvious and simple way to improve a website's performance is by scaling up hardware. But scaling the hardware does not work in all cases and is definitely not the most cost-effective approach. Tuning can improve performance, without extra costs for the hardware.

To get the best performance from a J2EE application, all the underlying layers must be tuned, including the application itself.



Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html>

For maximum performance, all the components in the figure above—operating system, Web server, application server, and so on—need to be optimized.

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html>

4. What is Performance Tuning?

- Performance tuning is an ongoing process. Mechanisms have to be implemented that provide performance metrics which can be compared against the performance objectives defined. It allows for a tuning phase to be undertaken before the system fails.
- Performance tuning is not a silver bullet. Simply put, good system performance depends on good design, good implementation, defined performance objectives, and performance tuning.
- The objective is to meet performance objectives, not eliminate all bottlenecks. Resources within a system are finite. By definition, at least one resource (CPU, memory, or I/O) will be a bottleneck in the system. Tuning minimizes the impact of bottlenecks on performance objectives.
- Design applications with performance in mind:
 - o Keep things simple - Avoid inappropriate use of published patterns.
 - o Apply J2EE performance patterns.
 - o Optimize Java code.
- Performance tuning is as much an art as it is a science. Changes that often result in improvement might not make any difference in some cases, or, in rare scenarios, they can result in degradation. For best results with performance tuning, take a holistic approach.

Source : <http://edocs.bea.com/wls/docs92/perform/basics.html>

5. What are the Performance Metrics used for Defining the Performance of an Enterprise Application?

Performance metrics for enterprise applications are usually defined in terms of:

- Throughput: Number of operations per unit of time
- Response time: Amount of time that it takes to process individual transactions
- Injection rate / concurrent users: Number of simultaneous requests applied to the workload

6. How will the business be impacted if Application Performance is not up to the mark?

Depending on the type of application, the impact an outage will have on business is stated below.

- Business-to-Consumer applications: Direct loss of sales revenue because of web site abandonment and loss of credibility
- Business-to-Business applications: Loss of credibility, which can eventually lead to loss of business partnerships
- Internal applications: Loss of employee productivity

Business-to-Consumer Applications: With respect to business-to-consumer applications, consider the typical online shopping pattern. Consumers usually find an item at the major retailers and then look for the best price. In the end, the prices do not usually vary much, so they choose vendor they trust within an acceptable price range. However, if the site is running slow or makes finalizing the purchase too difficult, they simply move down their lists to their next preferred vendor. By and large, retailer loyalty has been replaced by price competition and convenience. Even a minor site outage means lost business.

Business-to-Business Applications: In a business-to-business relationship, the stakes are much higher. If one of several companies that sells widgets has a relationship with a major retailer, the reliability and performance of the company's B2B application is its livelihood. If the major retailer submits an order for additional widgets and the application is unavailable or the order is lost, the company runs the risk of losing the account and the revenue source altogether. Continual application problems mean lost accounts.

Internal Applications: Poorly performing internal applications can also hurt the bottom line. Of course employee productivity suffers when an application goes down or responds slowly, but more significantly, the loss

in productivity can delay product delivery.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

7. What is Performance Testing?

In software engineering, Performance Testing is testing that is performed, from one perspective, to determine how fast some aspect of a system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage. Performance Testing is a subset of Performance engineering, an emerging computer science practice which strives to build performance into the design and architecture of a system, prior to the onset of actual coding effort.

Source : http://en.wikipedia.org/wiki/Software_performance_testing

8. How can the number of users using a system be accurately judged?

To accurately judge the number of users, distinction must be made between named, active, and concurrent users.

Named users make up the total population of individuals who can be identified by and potentially use the system. They represent the total user community, and can be active or concurrent at any time. In a real-life environment, this is the total number of individuals authorized to use the system.

Active users are logged on to the system at a given time. They include users who are simply viewing the results returned. Although this subset of active users are logged on to the system, they are not sending requests.

Concurrent users are not only logged on to the system, but represent the subset of active users who are sending a request or waiting for a response. They are the only users actually stressing the system at any given time. A good assumption is that 10 percent of active users are concurrent users.

Source : http://www.cognos.com/pdfs/whitepapers/wp_cognos_report_net_scalability_benchmark.ms_windows.pdf

9. What are Performance Tests?

Performance Tests are tests that determine end-to-end timing (benchmarking) of various time critical business processes and transactions, while the system is under low load, but with a production sized database.

Source : http://www.loadtest.com.au/types_of_tests.htm

10. What is the use of conducting Performance Tests?

They help set 'best possible' performance expectation under a given configuration of infrastructure. They also highlight very early in the testing process if changes need to be made before load testing should be undertaken.

For example, a customer search may take 15 seconds in a full sized database if indexes had not been applied correctly, or if an SQL 'hint' was incorporated in a statement that had been optimized with a much smaller database. Such performance testing would highlight a slow customer search transaction, which could be remediated prior to a full end-to-end load test.

Source : http://www.loadtest.com.au/types_of_tests.htm

11. Why should an Automated tool be used for Performance Tests?

It is 'best practice' to develop performance tests with an automated tool, such as WinRunner, so that response times from a user perspective can be measured in a repeatable manner with a high degree of precision. The same test scripts can later be re-used in a load test and the results can be compared back to the original performance tests.

A key indicator of the quality of a performance test is repeatability. Re-executing a performance test multiple times should give the same set of results each time. If the results are not the same each time, then the differences in results from one run to the next cannot be attributed to changes in the application, configuration or environment.

Source : http://www.loadtest.com.au/types_of_tests/performance_tests.htm

12. What is a Load Tester?

A load tester is an application that generates an arbitrary number of simultaneous user interactions with a system. There are several load testers on the market, but there is core functionality that each presents:

- The capability to perform user-defined transactions against a system, and in the proper frequency
- The capability to control the number of simultaneous users in the load
- The capability to fine-tune its behavior in the areas of user think-time between requests and the rate at which to ramp up to the desired number of users

Most commercial load testers also provide a “learning” engine that will allow you to manually perform your transactions while it watches and records what you have done.

The goal of a load tester, then, is to simulate the real-life use of your application once you release it into a production environment. Without a representative load on your system, you cannot accurately tune it to support real-life users.

Some of the important features to be considered before choosing a tool are:

- Support for a large number of concurrent Web users, each running through a predefined set of URL requests
- Ability to record test scripts automatically from browser
- Support for cookies, HTTP request parameters, HTTP authentication mechanisms, and HTTP over SSL (Secure Socket Layer)
- Option to simulate requests from multiple client IP addresses
- Flexible reporting module that lets you specify the log level and analyze the results long after the tests were run
- Option to specify the test duration, schedule test for a later time, and specify the total load

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html>

13. What are Load Tests?

Load Tests are end-to-end performance tests under anticipated production load.

Load Tests are major tests, requiring substantial input from the business, so that anticipated activity can be accurately simulated in a test environment. If the project has a pilot in production, then logs from the pilot can be used to generate ‘usage profiles’ that can be used as part of the testing process, and can even be used to ‘drive’ large portions of the load test.

To perform an accurate load test, quantify the projected user load and configure the load tester to generate a graduated load up to that projected user load. Each step should be graduated with enough granularity so as not to oversaturate the application if a performance problem occurs.

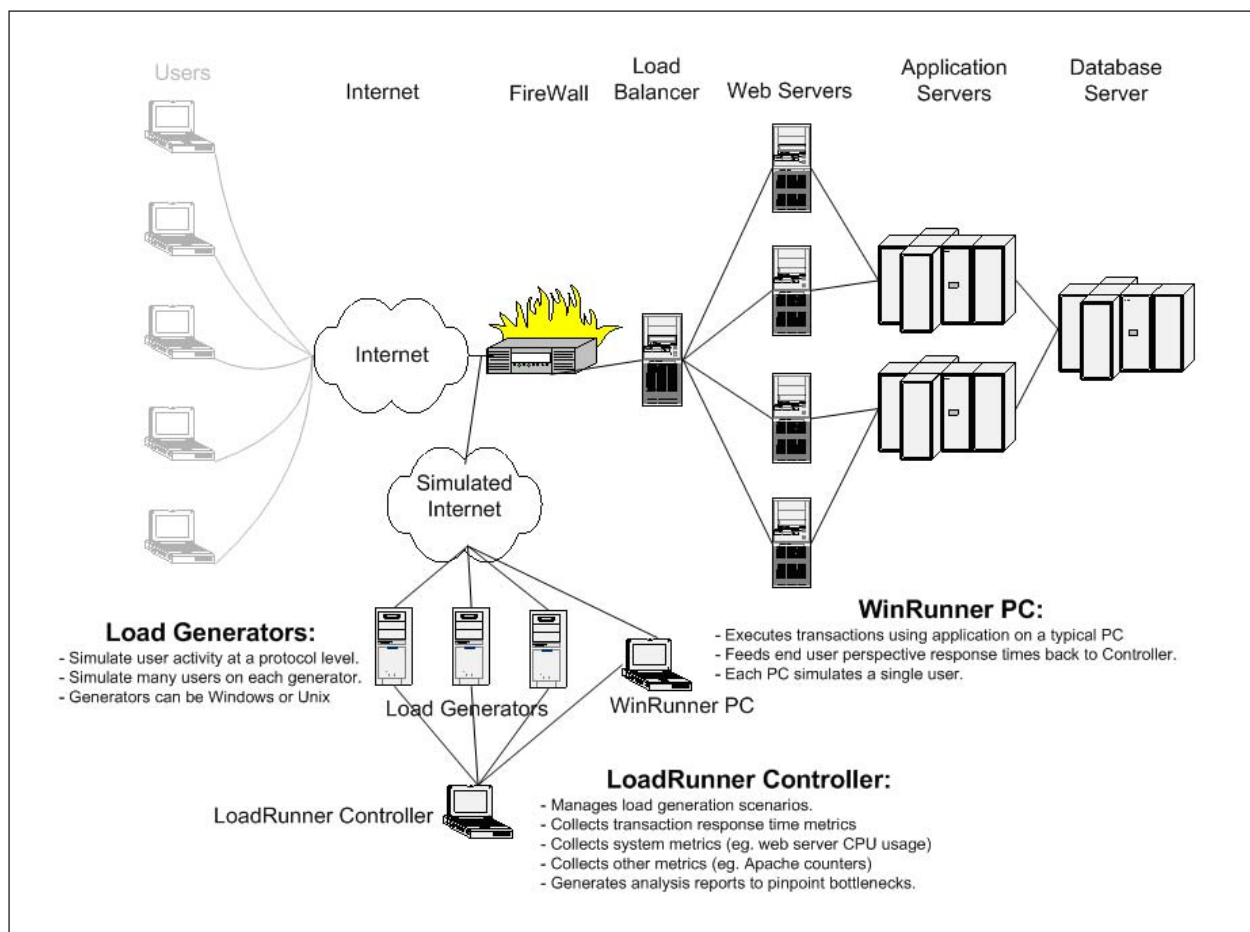
For example, if an application needs to support 1,000 concurrent users, then configure the load tester to climb up to 500 users relatively quickly, such as over a 10-minute period. Then a graduated “step” might be configured to add 20 users every minute. If the performance analysis software detects a problem at some point during the load test, then the application’s breaking point would be known within 20 users. And of course once this has been done a few times, then the initial ramp up can be set appropriately.

The point is that if 1,000 users have to be supported and the users are ramped up too quickly it might inadvertently break the application, then resource contention may obfuscate the root cause of your application’s problems.

Load testing must be executed on “today’s” production size database, and optionally with a “projected” database. If some database tables will be much larger in some months time, then Load testing should also be conducted against a projected database. It is important that such tests are repeatable, and give the same results for identical runs. They may need to be executed several times in the first year of wide scale deployment, to ensure that new releases and changes in database size do not push response times beyond prescribed SLAs.

Source : http://www.loadtest.com.au/types_of_tests/load_tests.htm

Source : http://www.loadtest.com.au/types_of_tests/load_tests.htm



14. What is the use of conducting Load Tests?

The objective of such tests are to determine the response times for various time critical transactions and business processes and ensure that they are within documented expectations or Service Level Agreements - SLAs. Load tests also measure the capability of an application to function correctly under load, by measuring transaction pass/fail/error rates.

A load test usually fits into one of the following categories:

Quantification of risk - Determine, through formal testing, the likelihood that system performance will meet the formal stated performance expectations of stakeholders, such as response time requirements under given levels of load. This is a traditional Quality Assurance (QA) type test. Note that load testing does not mitigate risk directly, but through identification and quantification of risk, presents tuning opportunities and an impetus for remediation that will mitigate risk.

Determination of minimum configuration - Determine, through formal testing, the minimum configuration that will allow the system to meet the formal stated

performance expectations of stakeholders, so that extraneous hardware, software and the associated cost of ownership can be minimized. This is a Business Technology Optimization (BTO) type test.

Source : http://www.loadtest.com.au/types_of_tests/load_tests.htm

15. Will all systems have a performance bottleneck?

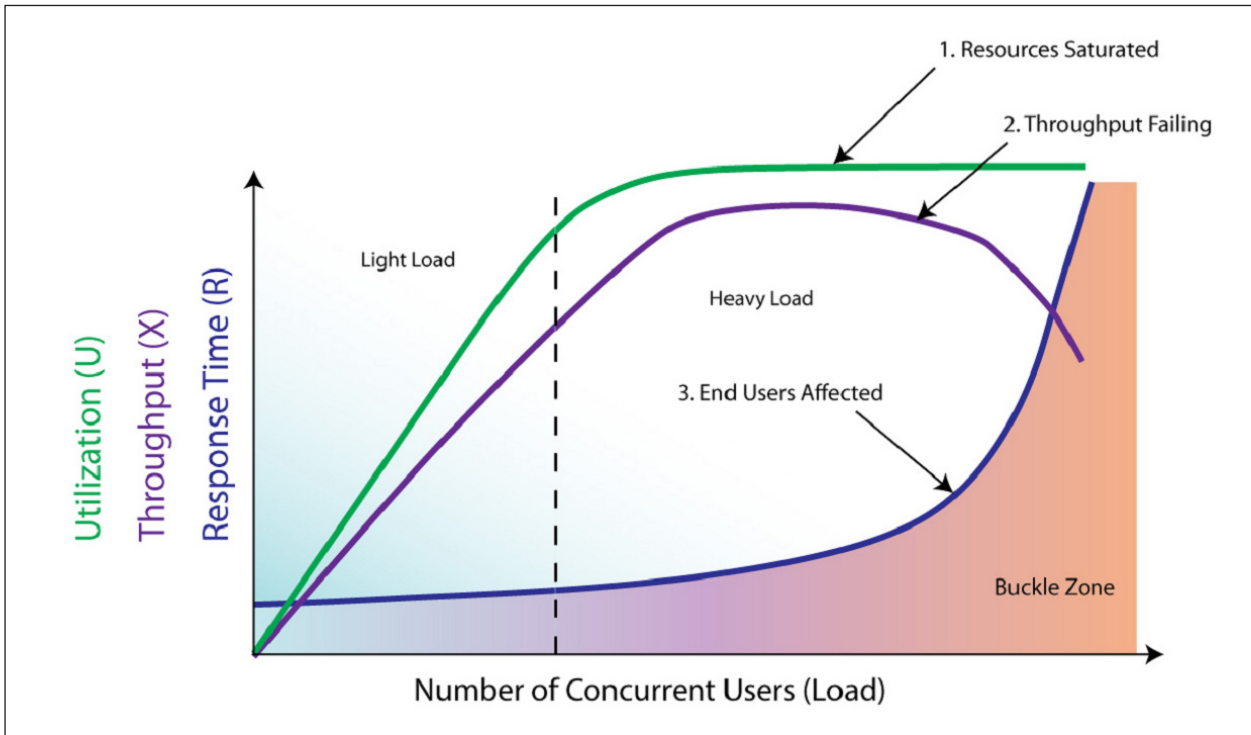
Theoretically, the only system with no performance bottleneck is designed such that every component of the system exhibits the same performance behavior and has identical capacity. In practical terms, every system has a performance bottleneck.

Source : http://download.intel.com/technology/iti/2003/volume07issue01/art03_java/vol7iss1_art03.pdf

16. What is a Throughput Curve? Why is it useful?

The figure below shows a conceptual diagram of a throughput curve, which plots system throughput, response time, and application server CPU utilization as

functions of the injection rate, i.e., the rate of requests applied to the system.



Drawing a throughput curve can be very valuable in understanding system-level bottlenecks and helping identify potential solutions.

As user load increases, both resource utilization and throughput increase because an application is being asked to do more work. Because it is doing more work, the throughput increases (where throughput is measured as work performed per period, such as requests processed per minute). And as the resource utilization increases, the application response time may also increase. The critical point comes when resources start to saturate; the application server spends so much time managing its resources (such as by context switching between threads) that throughput drops and response time increases dramatically.

Note that CPU utilization is not necessarily the only resource to monitor. For example, a common problem in initial tuning efforts is properly configuring the size of application server thread pools. If the 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; a 100% thread pool utilization on a small pool might translate to just 20% CPU utilization, so if only CPU utilization is being monitored, the thread pool resource issue would be missed.

When designing a load test, therefore, the resources should not be saturated too quickly; when resources

saturate, the entire application behaves abnormally. With a graduated load test it can be seen at a fine level of detail where the application fails.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

17. In an Application Development life cycle, when should a Performance Test be performed?

Performance tuning is typically reserved for the conclusion of an application's development life cycle. The most common argument against earlier performance tuning is that during an application's development, tuning efforts would be premature because:

- 1) The application does not yet have all of its functionality and hence tuning parameters are a moving target and
- 2) the application will most likely be deployed to a shared environment in which it will have to compete with other applications for resources—why do the work twice?

The simple reason why one needs to implement performance tuning efforts as early as QA testing is that many performance problems do not manifest themselves until an application is under load. And without properly tuning the environment one may not be able to subject the application to the proper load.

For example, if 500 virtual users are run against an application, but the Java environment is configured to use only 15 threads, then at no point during the test will more than 15 virtual users get into that application. But if the environment is configured to use 100 threads, then a whole new set of performance problems may appear and those are the problems that the users are going to see in production.

The best time to execute performance tests is at the earliest opportunity after the content of a detailed load test plan have been determined. Developing performance test scripts at such an early stage provides opportunity to identify and remediate serious performance problems and expectations before load testing commences.

For example, management expectations of response time for a new web system that replaces a block mode terminal application are often articulated as ‘sub second’. However, a web system, in a single screen, may perform the business logic of several legacy transactions and may take two seconds. Rather than waiting until the end of a load test cycle to inform the stakeholders that the test

failed to meet their formally stated expectations, a little education up front may be in order. Performance tests provide a means for this education.

Another key benefit of performance testing early in the load testing process is the opportunity to fix serious performance problems before even commencing load testing.

A common example is one or more missing indexes. When performance testing of a “customer search” screen yields response times of more than ten seconds, there may well be a missing index, or poorly constructed SQL statement. By raising such issues prior to commencing formal load testing, developers and DBAs can check that indexes have been set up properly.

Performance problems that relate to size of data transmissions also surface in performance tests when low bandwidth connections are used. For example, some data, such as images and “terms and conditions” text are not optimized for transmission over slow links.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>
http://www.loadtest.com.au/types_of_tests/performance_tests.htm

18. What are the Pre-requisites for Performance Testing?

PerformanceTest Pre-requisites	Comment	Caveats on testing where pre-requisites are not satisfied
Production Like Environment	Performance tests need to be executed on the same specification equipment as production if the results are to have integrity	Lightweight transactions that do not require significant processing can be tested, but only substantial deviations from expected transaction response times should be reported Low bandwidth performance testing of high bandwidth transactions where communications processing contributes to most of the response time can be tested
Production Like Configuration	Configuration of each component needs to be production like For example: Database configuration and Operating System Configuration	While system configuration will have less impact on performance testing than load testing, only substantial deviations from expected transaction response times should be reported
Production Like Version	The version of software to be tested should closely resemble the version to be used in production	Only major performance problems such as missing indexes and excessive communications should be reported with a version substantially different from the proposed production version
Production Like Access	If clients will access the system over a WAN, dial-up modems, DSL, ISDN, etc. then testing should be conducted using each communication access method	Only tests using production like access are valid
Production Like Data	All relevant tables in the database need to be populated with a production like quantity with a realistic mix of data e.g. Having one million customers, 999,997 of which have the name “John Smith” would produce some very unrealistic responses to customer search transactions	Low bandwidth performance testing of high bandwidth transactions where communications processing contributes to most of the response time can be tested

A performance test is not valid until the data in the system under test is realistic and the software and configuration is production like. The following table list pre-requisites for valid performance testing, along with tests that can be conducted before the pre-requisites are satisfied:

Source : http://www.loadtest.com.au/types_of_tests/performance_tests.htm

19. How should the test environment be setup for performance/load tests?

Once an enterprise application is ready for deployment, it is critical to establish a performance test environment that mimics production. It should be configured based on the estimated capacity needed to sustain the desired load, including network bandwidth and topology, processor memory sizes, disk capacity, and physical database layout.

This environment is then used to identify and remove performance and scalability barriers, using an iterative, data-driven, and top-down methodology.

If the objective is to get the tuning efforts to be as accurate as possible, then ideally the production staging environment should have the same configuration as the production environment. Unfortunately, most companies cannot justify the additional expense involved in doing so and therefore construct a production staging environment that is a scaled-down version of production.

The following are three main strategies used to scale down the production staging environment:

- Scale down the number of machines (the size of the environment), but use the same class of machines
- Scale down the class of machines
- Scale down both the number of machines and the class of machines

Unless financial resources dedicated to production staging are plentiful, scaling down the size of an environment is the most effective plan. For example, if a production environment maintains eight servers, then a production staging environment with four servers is perfectly accurate to perform scaled-down tuning against. A scaled-down environment running the same class of machines (with the same CPU, memory, and so forth) is very effective for understanding how an application should perform on a single server, and depending on the size, the percentage of performance lost in inter-server communication can be calculated (such as the overhead required to replicate stateful information across a cluster).

Scaling down the class of machine, on the other hand, can be quite problematic. But sometimes it is

necessary. If a production environment runs on a \$10 million mainframe, for example, spending an additional \$10 million on a test bed is likely out of the question. If the class of machine has to be scaled down, then the best load testing can accomplish is to identify the relative balance of resource utilizations. This information is useful; it provides information about which service requests resolve to database or external resource calls, the relative response times of each service request, relative thread pool utilization, cache utilization, and so on. Most of these values are relative to each other, but as the application is deployed to a stronger production environment, a relative scale of resources to one another can be defined, establishing best “guess” values and scaling resources appropriately.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

20. What is Baseline data?

The first set of performance data is the baseline data, as it is used for comparison with future configurations.

21. When are performance goals for the System Defined?

Prior to baseline data collection, performance goals for the system should be defined. Performance goals are usually defined in terms of desired throughput within certain response time constraints. An example of such a goal might be the system needs to be able to process 500 operations per second with 90% or more of the operations taking less than one second.

Source : <http://www.intel.com/cd/ids/developer/asmo-na/eng/178820.htm?page=3>

22. Where should Baseline data be established?

The baseline data should be established in a test environment that mimics production as closely as is practical.

In addition, the baseline configuration should incorporate basic initial configuration recommendations given by the application server, database server, JVM, and hardware-platform vendors. These recommendations should include tunable parameter settings, choice of database connectivity (JDBC) drivers, and the appropriate level of product versions, service packs, and patches.

Source : <http://www.intel.com/cd/ids/developer/asmo-na/eng/178820.htm?page=3>

23. Is the strategy for tuning an existing application different from the strategy for tuning new applications?

Yes.

24. How should a load test for an existing application be designed?

When tuning an existing application (or even a new version of an existing application), it is best to analyze the web server's access logs or employ a user experience monitor (a hardware device that records all user activities, provides reports, and can replay specific user requests in a step-by-step fashion) to determine exactly how existing users are interacting with the application.

Every time a user accesses a web application, an entry describing the request is recorded in an access log file. Sifting through this log file may be too cumbersome to perform manually, but there are many tools that can help you sort requests and determine which ones are accessed most frequently. The goal is to determine the top 80% of requests that the existing users are executing, as well as the balance between that load (for example one login, twenty reports, and one logout), and then reproduce that load in the test environment.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

25. How should a Load test for a new application be designed?

If an application has as yet not been deployed to a production environment where real end-user behavior can be observed, there is no better option than to take a best guess. "Guess" may not be the most appropriate word to describe this activity, as time has to be spent upfront in constructing detailed use cases. If a good job was done building the use cases, it would be known as to what actions to expect from the users, and the guess is based on the distribution of use-case scenarios.

When tuning a new application and environment, it is important to follow these steps;

1. Estimate,
2. Validate,
3. Reflect,
4. Periodically revalidate

Step 1: Estimate; Begin by estimating what actions to expect from users and how is the application expected to be used. This is where well-defined and thorough use cases really help. Define load scenarios for each use case

scenario and then conduct a meeting with the application business owner and application technical owner to discuss and assign relative weights with which to balance the distribution of each scenario. It is the application business owner's responsibility to spend significant time interviewing customers to understand the application functionality that users find most important. The application technical owner can then translate business functionality into the application in detailed steps that implement that functionality. Construct the test plan to exercise the production staging environment with load scripts balanced based off of the results of this meeting. The environment should then be tuned to optimally satisfy this balance of requests. Even if this production staging environment does not match production, there is still value in running a balanced load test as it helps in deriving the correlation between load and resource utilization. For example, if 500 simulated users under a balanced load use 20 database connections, then 1,000 users can be expected to use approximately 40 database connections to satisfy a similar load balance. Unfortunately, linear interpolation is not 100% accurate because increased load also affects finite resources such as CPU that degrade in performance rapidly as they approach saturation. But linear interpolation gives a ballpark estimate or best practice start values from which to tune further.

Step 2: Validate; After deploying an application to production and exposing it to end users, its usage patterns can be validated against expectations. This is the time to incorporate an access log file analyzer or an end-user experience monitor to extract end-user behavior. The first week can be used to perform a sanity check to identify any gross deviations from estimates, but depending on the user load, a month or even a quarter could be required before users become comfortable enough with an application to give application provider the confidence he/she has accurately captured their behavior. User requests that log file analysis or end-user experience monitors identify need to be correlated to use case scenarios and then compared to initial estimates. If they match, then the tuning efforts were effective, but if they are dramatically different, then there is a need to retune the application to the actual user patterns.

Step 3: Reflect; Finally, it is important to perform a postmortem analysis and reflect on how estimated user patterns mapped to actual user patterns. This step is typically overlooked, but it is only through this analysis that the estimates will become more accurate in the future. There is a need to understand where the estimates were flawed and attempt to identify why. In general, the users' behavior is not going to change significantly over time, so the estimates should become more accurate as the application evolves.

Step 4: Repeat the Validation; This procedure of end-user pattern validation should be periodically repeated to ensure that users do not change their behavior and invalidate the tuning configuration. In the early stages of an application, an application provider should perform this validation relatively frequently, such as every month, but as the application matures, these validation efforts need to be performed less frequently, such as every quarter or six months. Applications evolve over time, and new features are added to satisfy user feedback; therefore, even infrequent user pattern validation cannot be neglected. For example, once a simple Flash game deployed into the production environment subsequently crashed production servers. Other procedural issues were at the core of this problem, but the practical application here is that small modifications to a production environment can dramatically affect resource utilization and contention. And, as with this particular customer, the consequences were catastrophic.

Source : <http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

26. What is Response Time?

Response Time is the duration a user waits for server to respond to his request.

27. What are the acceptable values for Response Time?

There are three important limits for response time values:

- 0.1 second. It is an ideal response time. Users feel that the system is reacting instantaneously, and don't sense any interruption.
- 1.0 second. It is the highest acceptable response time. Users still don't feel an interruption, though they will notice the delay. Response times above one second interrupt user experience.
- 10 seconds. It is the limit after which response time becomes unacceptable. Moreover, recent studies show that if response time exceeds eight seconds, user experience is interrupted very much and most users will leave the site or system.

Normally, response times should be as fast as possible. The interval of most comfortable response times is 0.1 - 1 second. Although people can adapt to slower response times, they are generally dissatisfied with times longer than two seconds.

28. How should the response time be measured?

Traditionally, response time is often defined as the interval from when a user initiates a request to the instant at which the first part of the response is received at by the application. However, such a definition is not usually appropriate within a performance related application requirement specification. The definition of response time must incorporate the behavior, design and architecture of the system under test. While understanding the concept of response time is critical in all load and performance tests, it is probably most crucial to Load Testing, Performance Testing and Network Sensitivity Testing.

Response time measuring points must be carefully considered because in client server applications, as well as web systems, the first characters returned to the application often does not contribute to the rendering of the screen with the anticipated response, and do not represent the users impression of response time.

For example, response time in a web based booking system, that contains a banner advertising mechanism, may or may not include the time taken to download and display banner ads, depending on your interest in the project. If you are a marketing firm, you would be very interested in banner ad display time, but if you were primarily interested in the booking component, then banner ads would not be of much concern.

Also, response time measurements are typically defined at the communications layer, which is very convenient for LoadRunner / VUGen based tests, but may be quite different to what a user experiences on his or her screen. A user sees what is drawn on a screen and does not see the data transmitted down the communications line. The display is updated after the computations for rendering the screen have been performed, and those computations may be very sophisticated and take a considerable amount of time. For response time requirements that are stated in terms of what the user sees on the screen, WinRunner should be used, unless there is a reliable mathematical calculation to translate communications based response time into screen based response time.

It is important that response time is clearly defined, and the response time requirements (or expectations) are stated in such a way to ensure that unacceptable performance is flagged in the load and performance testing process.

One simple suggestion is to state an Average and a 90th percentile response time for each group of transactions that are time critical. In a set of 100 values that are sorted from best to worst, the 90th percentile simply means the 90th value in the list.

The specification is as follows:

Time to display order details	
Average time to display order details	Less than 5.0 seconds
90th percentile time to display order details	Less than 7.0 seconds

The above specification, or response time service level agreement, is a reasonably tight specification that is easy to validate against.

For example, a customer 'display order details' transaction was executed 20 times under similar conditions, with response times in seconds, sorted from best to worst, as follows -

2,2,2,2,2, 2,2,2,2,2, 3,3,3,3,3, 4,10,10,10,20 Average = 4.45 seconds, 90th Percentile = 10 seconds

The above test would fail when compared against the above stated criteria, as too many transactions were slower than seven seconds, even though the average was less than five seconds.

If the performance requirement was a simple "Average must be less than five seconds" then the test would pass, even though every fifth transaction was ten seconds or slower.

This simple approach can be easily extended to include 99th percentile and other percentiles as required for even tighter response time service level agreement specifications.

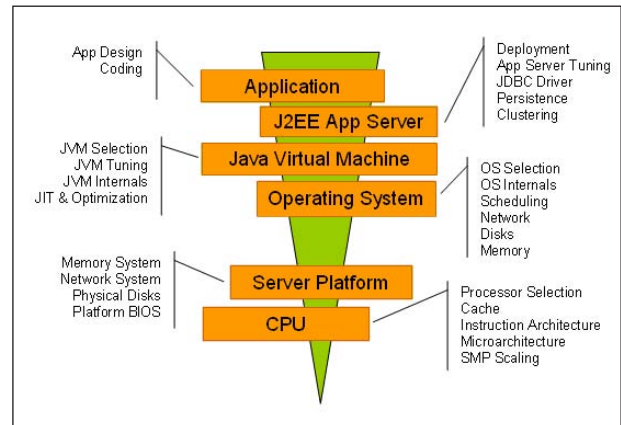
Source : <http://www.loadtest.com.au/Terminology/ResponseTime.htm>

29. What approach should be followed for tuning performance?

The first and foremost element an application implementer needs to keep in mind to achieve the desired level of performance is ensuring that the application architecture follows solid design principle. A poorly designed application, in addition to being the source of many performance-related issues, will be difficult to maintain. This compounds the problem, as resolving performance issues will often require that some code be restructured and sometimes even partially rewritten.

Application server configurations involve multiple computers interconnected over a network. Given the complexity involved, ensuring an adequate level of performance in this environment requires a systematic approach. There are many factors that may impact the overall performance and scalability of the system.

Examples of these performance and scalability factors include application design decisions, efficiency of user written application code, system topology, database configuration and tuning, disk and network input/output (I/O) activity, Operating System (OS) configuration, and application server resource throttling knobs.



Source : http://download.intel.com/technology/iti/2003/volume07issue01/art03_java/vol7iss1_art03.pdf

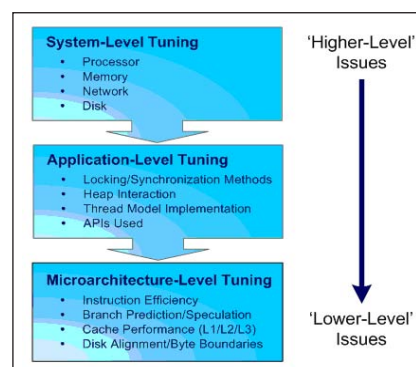
Performance optimization considerations are distributed across three levels of the top-down stack:

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

Application-level: Application design considerations and application server tuning

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

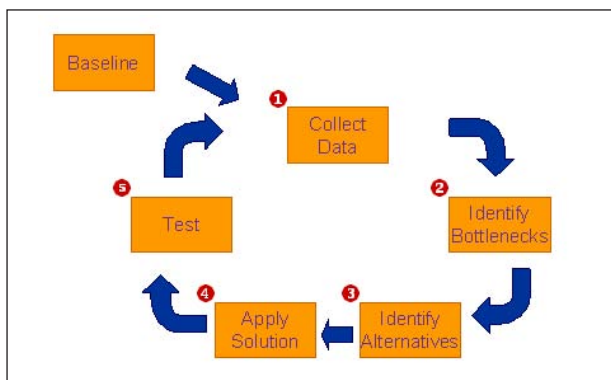
A top-down data-driven and iterative approach is the proper way to improve performance. 'Top-down' refers to addressing system-level issues first, followed by application-level issues, and finally issues at the microarchitectural level (although tuning efforts may be limited to the system level only, or to the system and application levels). 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 in figure below.



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

Source : <http://www.intel.com/cd/ids/developer/asm-na/eng/178820.htm?page=3>

30. What steps should the iterative Performance Tuning methodology have?



The steps in the iterative process, as illustrated in the above figure, are as follows:

Collect performance data: Use stress tests and performance-monitoring tools to capture performance data as the system is exercised. In the case of this workload, one should collect not only the key performance metric, but also performance data that can aid tuning and optimization.

Identify bottlenecks: Analyze the collected data to identify performance bottlenecks. Some examples of bottlenecks are data-access inefficiencies, significant disk I/O activities on the database server, and so on.

Identify alternatives: Identify, explore, and select alternatives to address the bottlenecks. For example, if disk I/O is a problem on the database back-end, consider using a high-performance disk array to overcome the bottleneck.

Apply solution: Apply the proposed solution. Sometimes applying the solution requires only a change to a single parameter, while other solutions can be as involved as reconfiguring the entire database to use a disk array and raw partitions.

Test: Evaluate the performance effect of the corresponding action. Data must be compared before and after a change has been applied. Sometimes fluctuation of the performance data for a given workload and measurement occurs. as Making make sure the change in performance is significant in such cases.

If the proposed solution does not remove the bottleneck, try a new alternative solution. Once a given bottleneck is addressed, additional bottlenecks may appear, so the process starts over again. The performance engineer must collect performance data and initiate the cycle again, until the desired level of performance is attained.

Two very important points to keep in mind during this process are letting the available data drive performance improvement actions, and making sure that only one performance improvement action is applied at a time, allowing you to associate a performance change with a specific action.

Note, however, that there are cases where one must apply multiple changes at the same time (e.g., using a new software release requires a patch in the operating system).

Source : http://download.intel.com/technology/iti/2003/volume07issue01/art03_java/vol7iss1_art03.pdf

31. When should the Performance Tuning exercise be concluded for an application?

Through system-level tuning, the main goal of performance tuning exercise should be to saturate the application server CPU (i.e., 90 - 100% utilization). Reaching maximum throughput without full saturation of the CPU is an indicator of a performance bottleneck such as I/O contention, over-synchronization, or incorrect thread pool configuration. Hitting a high response time metric with an injection rate well below CPU saturation indicates latency issues such as excessive disk I/O or improper database configuration.

Reaching application server CPU saturation indicates that there are no system-level bottlenecks outside of the application server. The throughput measured at this level would point out the maximum capacity the system has within the current application implementation and system configuration parameters. Further tuning may involve tweaking the application to address specific hotspots, adjusting garbage collection parameters, or adding application server nodes to a cluster.

Keep in mind that reaching CPU saturation is a goal for the performance tuning process, not an operational goal. An operational CPU utilization goal would be that there is sufficient capacity available to address usage surges.

Source : http://download.intel.com/technology/iti/2003/volume07issue01/art03_java/vol7iss1_art03.pdf

32. What are the various tools that are used while executing Performance/Load Tests?

Performance tools fall under the following categories:

Stress test tools: These provide the ability to script application scenarios and play them back, thereby simulating a large number of users stressing the application. Commercial examples of these types of tools are Mercury Interactive's LoadRunner and RADView's WebLoad; open-source examples include the Grinder, Apache's JMeter, and OpenSTA.

System monitoring tools: Use these to collect system-level resource utilization statistics such as CPU utilization (e.g., % processor time), disk I/O (e.g., % disk time, read/write queue lengths, I/O rates, latencies), network I/O (e.g., I/O rates, latencies). Examples of these tools are the Performance System Monitor from Microsoft's Management Console (known as perfmon), and "sar/iostat" in the Linux environment.

Application server monitoring tools: These tools gather and display key application server performance statistics such as queue depths, utilization of thread pools, and database connection pools. Examples of these tools include BEA's WebLogic Console and IBM's WebSphere Tivoli Performance Viewer.

Database monitoring tools: These tools collect database performance metrics including cache hit ratio, disk operation characteristics (e.g., sorts rates, table scan rates), SQL response times, and database table activity. Examples of these tools include Oracle's 9i Performance Manager and the DB/2 Database System Monitor.

Application profilers: These provide the ability to identify application-level hotspots and drill down to the code-level. Examples of these tools include the Intel VTune Performance Analyzer, Borland's Optimizeit Suite, and Sitraka's JProbe. A new class of application response time profilers is emerging that is based on relatively modest intrusion levels, by using bytecode instrumentation. Examples of these include the Intel VTune Enterprise Analyzer and Precise Software Solutions Precise/InDepth for J2EE.

JVM monitoring tools: Some JVMs provide the ability to monitor and report on key JVM utilization statistics such as Garbage Collection (GC) cycles and compilation/code optimization events. Examples of these tools include the "verbosegc" option, available in most JVMs, and the BEA WebLogic JRockit Console.

An important issue to keep in mind when using the above tools is that the measurement techniques employed introduce a certain level of intrusion into the system. In

some cases, the intrusion level is so great that the application characteristics are altered to the extent that they make the measurements meaningless (i.e., Heisenberg problem). For example, tools that capture and build dynamic call graphs can have an impact of one or more orders of magnitude on application performance (i.e., 10-100X). The recommended approach is to only activate the appropriate set of tools based on the level the data analysis is focused on at the time. For example, for system-level tuning, it only makes sense to engage system monitoring tools, whereas application-level tuning may require the use of an application profiler.

Source : http://download.intel.com/technology/iti/2003/volume07issue01/art03_java/vol7iss1_art03.pdf

33. What data should I capture whilst conducting Performance Tests?

There are five major categories of data that are generally useful for performance tuning. From generic to specific, they are as follows:

System Performance: The data contains system-level resource utilization statistics, such as :

- CPU utilization (% processor time),
- Disk I/O (% disk time, read/write queue lengths, I/O rates, latencies),
- Network I/O (I/O rates, latencies),
- Memory utilization (amount of virtual memory used, amount of physical memory used, amount of physical memory available),
- Context switches (number of context switches per second), and
- Interrupts (number of system interrupts per second).

Many tools are available to collect system performance data. On Windows, the Performance System Monitor from Microsoft's Management Console (known as PERFMON) is easily accessible, and a very useful freeware tool set is available from Sysinternals. Similarly, on Linux, many tools, such as sar, top, iostat and vmstat, are available.

Execution Profile: The data contains application hotspots and allows a drill down to the code-level. Hotspots are sequences of code that consume disproportionately large amounts of CPU time (e.g., measured by processor clock cycles). Hot methods or code to blocks deserve attention, as changing such code is likely give you good return on your investment of time. The Intel® VTune™ Performance Analyzer can provide details not only for clock cycles and instructions, but also for many microarchitecture performance statistics for branches, memory

references, (e.g. cache misses), and other processor-specific information.

Some JVMs, such as JRockit* also provide hotspot data. The JRockit JVM provides the JRockit Runtime Analyzer (JRA), for this purpose. While the VTune analyzer gives us the whole stack of hotspot performance data, including supporting libraries, OS and drivers, JRA gives Java users a convenient way to get the application-level hotspots through JRockit sampling.

JVM Performance: The data contains key statistics that identify performance issues when the workload is run within a JVM. Common statistics are

- Heap memory usage,
- Object life cycles,
- Garbage collections (GC),
- Synchronizations and locks,
- Optimizations, and
- Methods inlining.

A JVM typically supports some run-time parameter to collect many such statistics. JRockit provides the added convenience of also collecting such data using the JRA tool for execution-profile data.

Application-Server Performance: The data contains application-server performance statistics such as

- Queue depths,
- Utilization of thread pools, and
- Database connection pools.

Examples of these tools include BEA WebLogic Console. Enterprise Java Bean* (EJB)-specific statistics should also be collected. The key performance counters are cache utilization, activations, and passivations of EJBs. These statistics are also available from WebLogic through the use of the weblogic.admin command-line tool. As a general performance guideline, passivation and then activation of EJBs is likely to reduce the performance of the workload and should be minimized.

Application Performance: The data contains workload-specific performance metrics. For example, New Orders 90% Response Time, Manufacturing 90% Response Time, Order Cycle Times, and the gradual change of transactions over steady state are all important statistics for a sample application. Because these are application-specific statistics, creating a simple tool to parse the key statistics for the workload would help automate part of the data analysis and increase the productivity of the performance engineer.

In the case of a J2EE workload, the performance of the back-end database server is also important. The data contains performance metrics including:

- Cache hit ratio,
- Disk operation characteristics,
- SQL response times, and
- Database table activity.

Examples of these tools include Oracle's 9i* Performance Manager and the DB/2* Database.

System Monitor: For Microsoft SQL Server*, many such statistics are supported conveniently by PERFMON, as well.

Architecting the system and application for good performance goes a long way towards making the rest of the performance optimization methodology more efficient.

Source : <http://www.intel.com/cd/ids/developer/apac/zhc/dc/mrte/178820.htm?page=4>

34. What factors influence the performance of an Application Server Software?

No matter which application server software is used, its performance is a critical measurement of cost-effectiveness and the end-user experience of the application. Despite the number of popular benchmarks available, no single benchmark can be used to predict the application server's performance, which depends on a variety of factors:

1. The features of the Application Server

Application servers provide a variety of features, whose performance affect an application's response time. In general, an application server environment provides these features either within or in conjunction with the application server:

Enterprise JavaBeans (EJB) container - A container in which the business logic resides. Subcomponents include container-managed persistence (CMP) and message-driven beans (MDB).

Web container - A container in which the presentation components are run.

Java Message Service (JMS) - An underlying layer that provides the messaging backbone.

Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture - An infrastructure that connects to legacy systems through J2EE adapters.

Authentication - A subsystem that authenticates and authorizes accesses to the application.

Load balancer - A subsystem that distributes requests to various application servers and Web servers to enhance the horizontal scalability.

Java Development Kit (JDK) software - The Java virtual machine in which the J2EE containers and other components run.

Transaction manager - A component that offers transaction-related capabilities, including a two-phase commit, dynamic resource enlistment, transaction monitoring and administration, as well as automatic transaction recovery.

Java Database Connectivity (JDBC) drivers - The drivers that connect to the database; they typically support connection pooling and data caching. Application servers usually also provide connection pooling for databases.

Reverse proxy - A component that redirects requests from the Web server to the application server and back.

HTTP engine - A component that handles requests from the HTTP path to the application server.

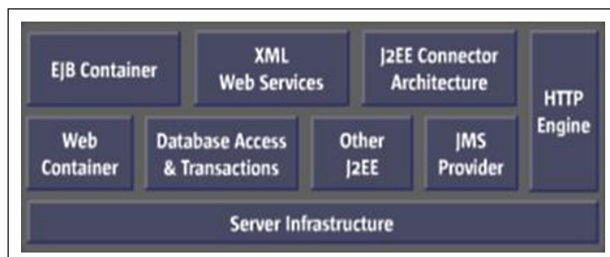
Session persistence - A component that provides session data in case of container failure.

XML and Web service runtime - A component that processes XML transactions and that transforms and executes tasks in accordance with requests from applications.

Secure socket layer (SSL) - The layer that performs encryption operations on the data exchanged with the application server.

Server infrastructure - Security, multiprocessing, multithreaded architecture, kernel threads, memory management, and such--all the components that provide virtual server support for the hosting of multiple Web sites from a single instance.

The following block diagram shows some of the core features offered by Sun ONE Application Server 7:



2. The access paths into and out of the application server

The application server resides at the heart of the data center. Applications that run on an application server can access other resources and can be accessed through a variety of paths. The response time from the server, when accessed from devices or when it access resources, impacts the application's performance. It is, therefore, important to understand the access paths:

Inbound access paths:

Hypertext Transport Protocol (HTTP/S) - Traffic is light inbound but heavy outbound. Web services usually use this path and can have different profiles. Because encryption is CPU sensitive, use of SSL impacts performance.

JMS - Traffic is bidirectional, moderate to heavy. You can also run JMS over different transports. The transport protocol that's in use affects performance.

Remote Method Invocation over Internet Inter-ORB [Object Request Broker] Protocol (RMI/IIOP) - Traffic is bidirectional, light to moderate.

Outbound access paths :

Database - Traffic is bidirectional, heavy from the database. Both the transaction type and the driver type impact performance.

J2EE connectors - Traffic is bidirectional, heavy in both directions. Enterprise systems, such as SAP and PeopleSoft, use these links.

3. The application type

Different types of applications use different components and access paths of the application server, both of which affect performance. The applications that run on an application server can be broadly classified as follows:

Bank or e-commerce type applications - These applications constitute the majority of those hosted on application servers. The main elements they rely on are JavaServer Pages (JSP) components, Java servlets, and HTTP access. Typically, security is achieved through authentication and SSL, with data requested from a database.

Web service applications - Many applications are built as Web services to enable interoperability and reuse. The main ingredients are HTTP access, XML transformations, JSP components, and Java servlets.

Wireless applications - These applications are accessed from multiple devices, typically with the underlying HTTP/S transport. Because many applications rely on alerts or notifications, JMS plays a key role in the applications. The main components are HTTP-WAP

[Wireless Access Protocol], XML transformation, JSP components, and Java servlets.

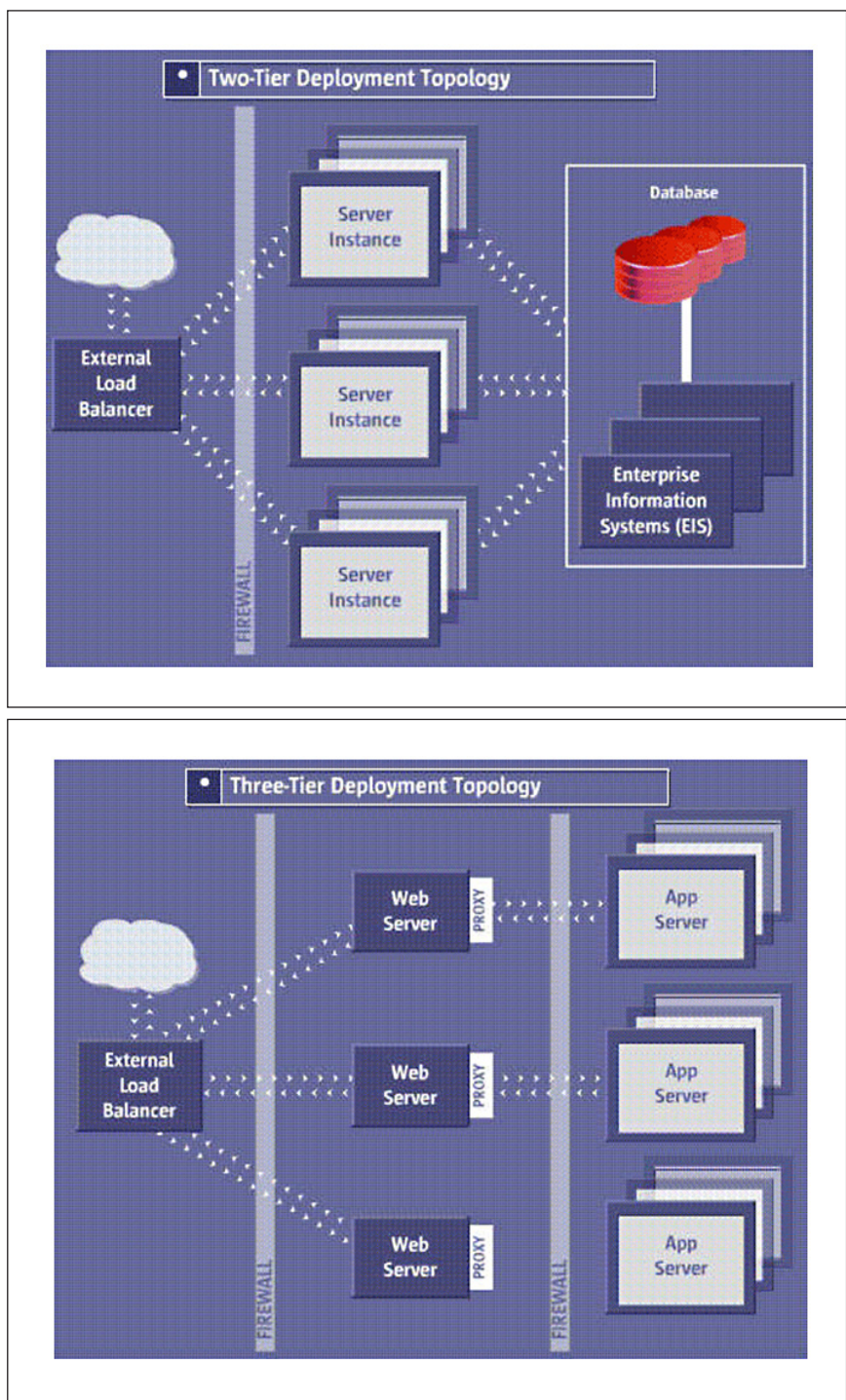
Desktop Java applications - These are thick client applications that access, through the RMI/IIOP mechanism, the business logic hosted in the EJB components.

Each type of application has its own performance profile. To benchmark an application server, you would need to rely on performance benchmarks that represent that

application. For example, you can use standardized benchmarks, such as ECPeef (now called SPECjAppServer) or the Web service measurement toolkit on the PushToTest site to understand the performance of application servers for specific types of application usage.

4. The deployment topology

The following two diagrams illustrate typical two-tier and three-tier deployment scenarios.



Source : http://java.sun.com/features/2002/11/appserver_influences.html

The multiple-tier model provides greater flexibility and tighter security. By taking advantage of its built-in SSL handling capabilities, Sun ONE Application Server 7 provides secure access to applications even when they are hosted on Tier 1. When applications are deployed with Sun ONE Application 7, one can direct the less security-sensitive requests to Tier 1 and more sensitive requests to Tier 2. In most enterprise deployments, the application server is hosted in Tier 2.

5. Scalability

Applications can be scaled either vertically (on a single machine) or horizontally (on multiple machines).

If a large system is being used, such as the Sun Enterprise 10000 (Starfire) machines, choice of application server should be based on one that scales vertically. Doing so takes advantage of the caching that is available for multiple user requests.

Similarly, if a Blades architecture is being used, such as the Sun Blade workstations, the choice of application server should be based on one that scales horizontally. Doing so results in enhanced serviceability but involves more moving parts. Also, having a larger number of servers incurs overheads in maintenance, especially if the application components are distributed on different servers. As a rule, however, the components are easier to service, and the hardware is simpler to replace.

Sun ONE Application Server scales well—either vertically or horizontally. In weighing your choice, balance all the factors described here.

35. What tunable Parameters have a considerable impact on the performance of my Application server?

1. JDBC

All application servers must provide a pooling mechanism for database connections. The creation of a database connection from within an application is an expensive operation and can take anywhere between 0.5 and 2 seconds. Thus, application servers pool database connections so applications and individual threads running inside applications can share a set of database connections.

The process is as follows: A thread of execution needs to access a database, so it requests a connection from the database connection pool, it uses the connection (some execution of a SELECT or UPDATE or DELETE statement), and then it returns the connection back to the connection pool for the next component that requests it. Because J2EE applications support many concurrent users, the size of the connection pools can greatly impact the performance of the application. If 90% of the requests

need database connections and the requirement is to be able to support 500 simultaneous users, a substantial number of connections would be needed in the connection pool. Note that when factoring in think time, the connections needed will probably be far fewer than 500 connections, but still quite a few would be needed.

Each application uses the database differently, and thus tuning the number of connections in the connection pool is application-specific. It is important to keep in mind that in practice tuning, JDBC connection pool size is one of the factors with the highest impact on the application's overall performance.

2. Enterprise JavaBeans (EJBs)

Enterprise JavaBeans (EJBs) provide the middleware layer of a J2EE application. They come in four flavors:

- Entity Beans
- Stateful Session Beans
- Stateless Session Beans
- Message Driven Beans

Both Entity Beans and Stateful Session Beans maintain some kind of stateful information. An Entity Bean may represent a row in a database or the result of some complex query, but regardless, it is treated as an object in the object-oriented model. Stateful Session Beans, on the other hand, represent temporary storage that exists beyond the context of a single request, but is not stored permanently. Typically, in a web-based application, a Stateful Session Bean's lifetime will be associated with a user's HTTP session. Because their nature is to maintain state, the application server must provide some form of caching mechanism to support them. Simple application servers may maintain a single cache that stores all Entity Beans and Stateful Session Beans, whereas more-advanced application servers provide caches on a bean-by-bean basis.

A cache has a preset size and can hold a finite number of "things." When a request is made for an item, the cache is searched for the requested item. If it is found, it is returned to the caller directly from memory; otherwise, it must be loaded from some persistent store (for example, database or file system), put into the cache, and returned to the caller. Once the cache is full, it becomes more complicated. The cache manager must select something to remove from the cache (for example, the least-recently used object) to make room for the new object. The EJB term used for removing an item from the cache to make room for the new item is passivating, and the term used for loading an item into the cache is activating. If activation and passivation are performed excessively, the result is that the cache manager spends more time reading from and writing to persistent storage than

serving requests; this is called thrashing. On the other hand, if the cache manager can locate an item in its cache and return it to the user, the performance is optimal; this is referred to as a cache hit.

When tuning a cache, the goal is to maximize the number of cache hits and minimize thrashing. This is accomplished after a thorough understanding of your application's object model and each object's usage.

Stateless Session Beans and Message Driven Beans are not allowed to maintain any stateful information; a single process may request a Stateless Session Bean for one operation and then request it again for another operation, and it has no guarantee that it will receive the same instance of that bean. This is a powerful paradigm because the bean manager does not have to manage the interactions between business processes and its bean; it exists simply to serve up beans as requested.

The size of bean pools must be large enough to service the requests from business processes; otherwise, a business process will have to wait for a bean before it can complete its work. If the pool size is too small, there are too many processes waiting for beans; if the pool size is too large, the system resources are more than actually need.

Another helpful effect of the fact that Stateless Session Beans and Message Driven Beans are stateless is that application servers can preload them to avoid the overhead of loading beans into a pool upon request; the request would have to wait for the bean to be loaded into memory before it could be used.

Two of the most influential tuning parameters of Stateless Session Bean and Message Driven Bean pools are the size of the pools that support them and the number of beans preloaded into the pools.

3. Servlets and JSPs

Although there are individual specifications for both Servlets and JavaServer Pages, the end result of both is a Servlet class loaded in memory; JSPs are translated from a JSP to a Servlet Java file, compiled to a class file, and finally loaded into memory. Servlets and JSPs do not maintain state between requests, so application servers pool them. The pool size and the number of Servlets that are preloaded into the pools can be tuned.

Because JSPs go through the translation and compilation step prior to being loaded into memory, most application servers provide a mechanism by the JSPs can be precompiled before deployment. This removes the delay that end-users would experience the first time a JSP is loaded.

Servlets (and JSPs) are required to maintain four different scopes, or areas of memory that data can be

stored in:

- **Page:** Data stored here exists for the context of a single page.
- **Request:** Data stored here exists for the duration of a request (it is passed from Servlet to Servlet, JSP to JSP, until a response is sent back to the caller).
- **Session:** Data stored here exists for the duration of a user's session (it exists through multiple requests until it is explicitly removed or it times out).
- **Application:** Data stored here is global to all Servlets and JSPs in your application until it is explicitly removed or until the Servlet container is restarted.

As a programmer, the choice of where to store data is a very important one that will impact the overall memory footprint of your application. The greatest impact, however, is the session scope. The amount of data that is stored in here is multiplied for each concurrent user. If you store 10 kilobytes of data in the session scope for each user and you have 500 users, the net impact is 5 MB. 5MB might not kill the application, but consider all 500 users going away and 500 more come. If there is no "clean up" after the users that left, 10 MB is now being used, and so on.

HTTP is a stateless protocol, meaning that the client connects to the server, makes a request, the server responds, and the connection is terminated. The application server then cannot know when a user decides to leave its site and terminate the session. The mechanism that application servers employ, therefore, is a session timeout; this defines the amount of time that a session object will live without being accessed before it is reclaimed. The session timeout that is chosen will be dependent on the application, the users, and the amount of memory set aside to maintain these sessions. A slow user should not be made to restart his transaction, but at the same time the slow user should not also drain the system resources with a timeout that is any longer than is necessary.

4. Java Messaging Service

JMS Servers facilitate asynchronous operations in the application. With the advent of EJB 2.0 came the introduction of Message Driven Beans; stateless beans representing business processes that are initiated by a JMS message. A message is put in a JMS destination, which is either a Topic or a Queue, and someone takes that message out of the destination and performs some business process based off of the contents of that message.

Because JMS Servers host messages, application servers usually define limits either to the number of messages that can be in the server at any given time or size of the

messages in bytes. These upper limits can be defined; the balance is between memory consumption and properly servicing the JMS subscribers. If the thresholds are too low, messages will be lost; if the thresholds are too high and the server is used to an excessive upper limit, it can degrade the performance of the entire system.

Along with total storage requirements, there are other aspects of JMS Servers that can be tuned, including the following:

- Message Delivery Mode: Persistent or non-persistent
- Time-to-live: Defines a expiration time on a message
- Transaction States
- Acknowledgments

Each of these aspects can be explored in detail, but the basic questions that have to be asked are these: How important are these messages to the business process? Does it matter if one or more messages are lost? Obviously, the less one care's about the messages actually reaching their destination, the better the performance—but this will be dictated by the business requirements.

5. Java Transaction API (JTA)

Application server components can be transactional, meaning that if one part of a transaction fails, the entire operation can be rolled back; and all components participating in the transaction are also rolled back. J2EE however, defines different levels of transaction participation. The more the application components participating in a transaction, the more is the overhead required, but the more reliable the business processes are. EJBs define several levels of transactional participation on a method-by-method basis for each bean's methods:

- **Not Supported:** The method does not support transactions and must be executed outside of any existing transaction.
- **Required:** A transaction is required, so if one exists, this method will use it; otherwise, you have to create a new one for it.
- **Supports:** A transaction is not required, but if one exists, this method will participate in it.
- **Requires New:** A new transaction is required, so regardless if one exists or not, a new one must be created for this method.
- **Mandatory:** A transaction is required and furthermore must be passed to this method; if a transaction does not exist when this method is invoked, it will throw an exception.

- **Never:** A transaction is forbidden; if this method is called and a transaction exists, the method will throw an exception.

The implications of each should be apparent, and the performance impact is like: Supported is the most unintrusive and yields the best performance at the cost of possible loss of data; Required is safe, yet a little more costly; and Requires New is probably the most expensive.

6. General Considerations

Besides the factors mentioned above, there are three things that dramatically affect performance that are the natural side effects of running an application server. Because application servers can service multiple simultaneous requests and because thread creation is expensive, application servers have to maintain a pool of threads that handle each request. Some application servers break this thread pool into two; one to handle the incoming requests and place those in a queue and one to take the threads from the queue and do the actual work requested by the caller.

Regardless of the implementation, the size of the thread pool limits the amount of work the application server can do; the tradeoff is that there is a point at which the context-switching (giving the CPU to each of the threads in turn) becomes so costly that performance degrades.

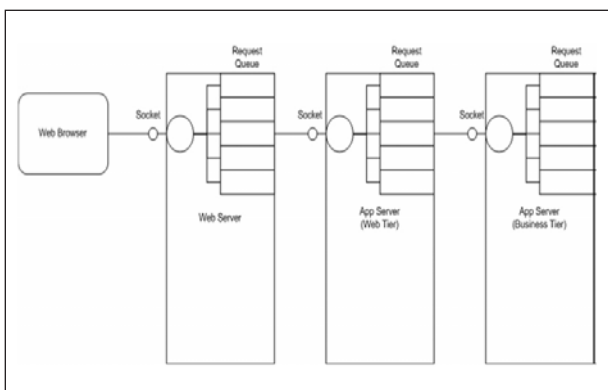
The other performance consideration is the size of the heap that the application server is running in. We already saw that it needs to maintain caches, pools, sessions, threads, and the application code, so the amount of memory you allocate for the application server has a great impact on its overall performance. The rule-of-thumb is to give the application server all the memory that can be given to it on any particular machine.

Finally, also consider tuning the garbage collection of the heap in which your application server is running.

The application server and all of its applications run inside of the JVM heap; when the heap pauses for garbage collection, everything running in the heap pauses. The Java memory model is different than traditional programming languages like C++ in that when a Java application frees an object, the object is not actually freed but rather marked as eligible for garbage collection. While the details are specific to each JVM implementation, when the heap grows to a certain point, a garbage collection process cleans up the heap by removing "dead" objects.

36. How does a user request traverse an Enterprise Java environment?

When a web browser submits a request to a web server, the web server receives the request through a listening socket and quickly moves it into a request queue. The reason for having a queue is that only one thread can listen on a single port at any given point in time. When a thread receives a request, its primary responsibility is to return to its port and receive the next connection. If the web server processes requests serially, then it would be capable of processing only one request at a time.



http://www.quest.com/Quest_Site_Assets/WhitePapers/WPA-Wait-Base-Tuning-Haines.pdf

A web server's listening process looks something like the following:

```
public class WebServer extends Thread {
...
public void run() {
ServerSocket serverSocket = new ServerSocket( 80 );
while( running ) {
Socket s = serverSocket.accept();
Request req = new Request( s );
addRequestToQueue( req );
}
}
}
```

This admittedly simplistic example demonstrates that the thread loop is very tight and acts simply as a pass-through to another thread. Each queue has an associated thread pool that waits for requests to be added to the queue to process them. When the request

is added to the queue, a thread wakes up, removes the request from the queue, and processes it. For example:

```
public synchronized void addRequestToQueue( Request req ) {
this.requests.add( req );
this.requests.notifyAll();
}
}
```

Threads waiting on the "requests" object are notified and the first one there accepts the request for processing. The actions of the thread are dependent on the request (or in the case of separation of business tiers, the request may actually be a remote method invocation.) Consider a web request against an application server; if the web server and application are separated, then the web server forwards the request to the application server (opens a socket to the application server and waits for a response) and the same process repeats. Once the request is in the application server, the application server needs to determine the appropriate resource to invoke. In this example it is going to be either a Servlet or a JSP file. For the purpose of this discussion, JSP files will be considered as Servlets.

The running thread loads the appropriate Servlet into memory and invokes its service method. This starts the Java EE application request processing as we tend to think of it. Depending on the use of Java EE components, the next step may be to invoke a Stateless Session Bean. Stateless Session Beans were created to implement the application's transactional business logic. Rather than create a new Stateless Session Bean for each request, they are pooled; the Servlet obtains one from the pool, uses it, and then returns it to the pool. If all of the beans in the pool are in use, then the processing thread must either wait for a bean to be returned to the pool or create a new one.

Most business objects make use of persistent storage, in the form of either a database or a legacy system. It is expensive for a Java application to make a query across a network for persistent storage, so for certain types of objects the persistence manager implements a cache of frequently accessed objects. The cache is queried; if the requested object is not found, it must be loaded from persistent storage. While using caches can result in much better performance than resolving all queries to persistent storage, there is danger in misusing them.

Specifically, if a cache is sized too small, the majority of requests will resolve to querying persistent storage, but with overhead; checking the cache for the requested object, selecting an object to be removed from the cache to make room for the new one (typically using a least-recently used algorithm), and adding the new object to the cache. In this case, querying persistent storage would perform much better. The final tradeoff is that a large cache requires storage space. If the need is to maintain too many objects in a cache to avoid thrashing (rapidly adding and removing objects to and from the cache), then the question to be asked is whether the object should be cached in the first place.

Establishing a connection to persistent storage is expensive. For example, establishing a database connection can take between half a second and a second and a half on average. Because it is undesirable for the pending request to absorb this overhead on each request, application servers establish these connections on startup and maintain them in connection pools. When a request needs to query persistent storage, it obtains a connection from the connection pool, uses it, and then returns it to the connection pool. If no connection is available, the request waits for a connection to be returned to the pool.

Once the request has finished processing its business logic, it needs to be forwarded to a presentation layer before returning to the caller. The most typical presentation layer implementation is to use JavaServer Pages. If JSPs are not precompiled, using JavaServer Pages can incur additional overhead of translation to Servlet code and compilation. This upfront performance hit should be addressed because it can impact the users but from a purist tuning perspective, JSP compilation does not impact the order of magnitude of the application's performance. The impact is observed once but does not have any impact as the number of users increases.

Source : http://www.quest.com/Quest_Site_Assets/WhitePapers/WPA-Wait-BasedTuning-Haines.pdf

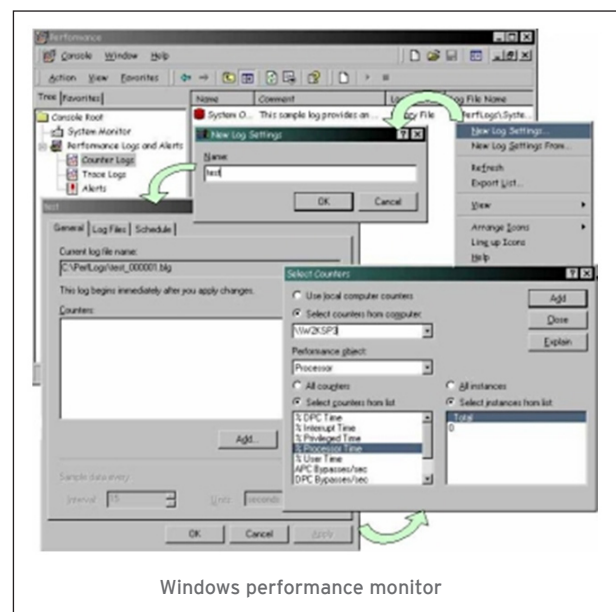
37. How do performance monitors help in identifying the bottlenecks?

Using a monitoring tool, one can collect data for various system performance indicators for all the appropriate nodes in a network topology. Many stress tools also provide monitoring tools.

Many tools are available to collect system performance data. On Windows, the Performance System Monitor from Microsoft's Management Console (known as PERFMON) is easily accessible, and a very useful freeware tool set is available from Sysinternals. Similarly, on Linux, many tools, such as sar, top, iostat and vmstat, are available

Windows performance monitor can be started from the Administrative Tools menu, accessed from the Control Panel menu, or by typing "perfmon" in the Run window (accessed from the Start menu). The performance counters data can be displayed in real time, but usually, it is required to log this data into a file so that it can be viewed later.

To log the data into a file, go to the Counter Logs selection in the left-hand side of the Performance window, right click with the mouse, and select New Log Settings as shown in Figure 2.



Windows performance monitor

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html>

The file path/name can be set for where the data should be logged, as well as a schedule of when to collect this data. It is possible to log the data and view it in real time, but the performance counters must be added twice—first, in the System Monitor link on the left-hand side and second, in Counter Logs, as shown above.

Many performance counters are available in Windows OS. The following table lists some of the important counters that you should always monitor:

The above mentioned counters should be added and any others too as appropriate in the counter log and the data should be collected while the application is being stress-tested using the stress tool. A file generated by

System resource	Performance monitor	Description
CPU	System: Processor queue length	Processor queue length indicates the number of threads in the server's processor queue waiting to be executed by the CPU. As a rule of thumb, if the processor queue remains at a value higher than 2 for an extended period of time, most likely, the CPU is a bottleneck.
	Processor: Percent of processor time	This counter provides the total overall CPU utilization for the server. A value that exceeds 70-80 percent for long periods indicates a CPU bottleneck.
	System: Percent of total privileged time	This counter measures what percentage of the total CPU execution time is used for running in kernel/privileged mode. All the I/O operations run under kernel mode, so a high value (about 20-30 percent) usually indicates problems with the network, disk, or any other I/O interface.
RAM	Memory: Pages per second	Pages per second is the number of pages read from or written to the disk for resolving hard page faults. A value that continuously exceeds 25-30 indicates a memory bottleneck.
	Memory: Available bytes	Available bytes is the amount of physical memory available to processes running on the computer, in bytes. A low value (less than 10 MB) usually means the machine requires more RAM.
Hard disk	Physical disk: Average disk queue length	The average number of requests queued for the selected disk. A sustained value above 2 indicates an I/O bottleneck.
	Physical disk: Percent of disk time	The percentage of elapsed time that the selected disk drive is busy servicing requests. A continuous value above 50 percent indicates a bottleneck with hard disks.
Network	Network interface: Total bytes per second	Shows the bytes transfer rate (sent and received) on the selected network interface. Depending on the network interface bandwidth, this counter can tell if the network is the problem.
	Network interface: Output queue length	Indicates the length of an output packet queue in packets. A sustained value higher than 2 indicates a bottleneck in the network interface.

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html?page=2>

the counter log can be opened later by clicking on the View Log File Data button on the right-hand side toolbar. Looking at these counters should give some hint as to where the problem exists—Application server, Web server, or Database server.

After identifying the bottleneck in this way, try to resolve it.

38. Once a bottleneck has been identified, how can it be resolved?

There are two different strategies - either tune the hosting environment in which the application runs or tune the application itself.

1. Environment tuning

A J2EE application environment usually consists of an application server, Web server, and a backend database.

Most application servers and Web servers provide similar kinds of configuration options though they have different mechanisms to set them. The following sections on Apache and Tomcat give examples of what can be tuned.

Apache

Probably the most important setting for Windows Apache HTTP Server is the option for number of threads. This value should be high enough to handle the maximum number of concurrent users, but not so high that it starts adding its own overhead of too many context switches. The optimum value can be determined by monitoring the number of threads in use during peak hours. To monitor the threads in use, make sure the following configuration directives are present in the Apache configuration file (httpd.conf):

```
LoadModule status_module modules/mod_status.so

<Location /server-status>
    SetHandler server-status
    Allow from all
</Location>
```

Now, from the browser, make an HTTP request to your Apache server with this URL: `http://<apache_machine>/`

server-status. It displays how many requests are being processed and their status (reading request, writing response, etc.). Monitor this page during peak load on the server to ensure the server is not running out of idle threads. After the optimum number of threads for an application has been arrived at, change the ThreadsPerChild directive in the configuration file to an appropriate value.

A few other items that improve performance in the Apache HTTP Server are:

- DNS reverse lookups are inefficient: Switch off DNS lookups by setting HostnameLookups in the configuration file to OFF.
- Do not load unnecessary modules: Apache allows dynamic modules that extend basic functionality of the Apache HTTP Server. Comment out all the LoadModule directives that are not needed.
- Try to minimize logging as much as possible: Look for directives LogLevel, CustomLog, and LogFormat in the configuration file for changing logging level.
- Minimize the JK connector's logging also by setting the JkLogLevel directive to emerg.

Tomcat

The two most important configuration options for Tomcat are its heap size and number of threads. Unfortunately there is no good way to determine the heap size needed because in most cases, the JVM doesn't start cleaning up the memory until it reaches the maximum memory allocated. One good rule of thumb is to allocate half of the total physical RAM as Tomcat heap size. If there is still an out of memory error, application designs should be modified so as to reduce the memory usage, identify any memory leaks, or try various garbage collector options in the JVM. To change the heap size, add `-Xms<size>` `-Xmx<size>` as the JVM parameter in the command line that starts Tomcat. `<size>` is the JVM heap size usually specified in megabytes by appending a suffix `m`, for example, `512m`. Initial heap size is `-Xms`, and `-Xmx` is the maximum heap size. For server applications, both should be set to the same value.

The number of threads in Tomcat can be modified by changing the values of `minProcessors` and `maxProcessors` attributes for the appropriate connector in `<Tomcat>/conf/server.xml`. If JK connector is being used, change the values of its attributes. Again, there is no simple way to decide the optimum value for these attributes. The value should be set such that enough threads are available to handle a Web application's peak load. A process's current thread count in the Windows Task Manager can be monitored, this can assist in determining the correct value of these attributes.

A few other options:

- If the latest JRE (Java Runtime Environment) is not being used, consider upgrading to the latest one. There might be up to a 30 percent performance improvement after upgrading from JRE 1.3.1 to JRE 1.4.1.
- Add the server option to the JVM options for Tomcat. This should result in better performance for server applications. Note that this option, in some cases, causes the JVM to crash for no apparent reason. In such a scenario, remove the option.
- Change the default Jasper (JavaServer Pages, or JSP, compiler) settings in `<Tomcat>/conf/web.xml` by setting `development="false"`, `reloading="false"` and `logVerbosityLevel="FATAL"`.
- Minimize logging in Tomcat by setting `debug="0"` everywhere in `<Tomcat>/conf/server.xml`.
- Remove any unnecessary resources from the Tomcat configuration file. Some examples include the Tomcat Web application and extra `<Connector>`, `<Listener>` elements.
- Set the `autodeploy` attribute of the `<Host>` tag to `false` (unless you need any of the default Tomcat applications like Tomcat Manager).
- Make sure to set `reloadable="false"` for all your Web application contexts in `<Tomcat>/conf/server.xml`.

2. Database tuning

In the case of Microsoft SQL Server, more often than not, there is no need to modify any configuration options, since it automatically tunes the database to a great degree. These settings should be changed only if the stress tests identify the database as a bottleneck. Some of the configuration options that can be tried are:

- Run the SQL Server on a dedicated server instead of a shared machine.
- Keep the application database and the temporary data base on different hard disks.
- Consider taking local backups and moving them to a different machine. The backups should complete much faster.
- Normalize the database to the third normal form. This is usually the best compromise, as the fourth and fifth forms of normalization can result in performance degradation.
- If there is more than 4 GB of physical RAM available, set the `awe` enabled configuration option to 1, which will allow SQL Server to use more than 4 GB of memory up to a maximum of 64 GB (depending on the SQL Server edition).

- In case there are many concurrent queries executing and enough memory is available, the value of the min memory per query option can be increased (default is 1,024 KB).
- Change the value of the max worker threads option, which indicates the maximum number of user connections allowed. Once this limit is reached, any new user requests will wait until one of the existing worker threads finishes its current task. The default value for this option is 255.
- Set the priority boost option to 1. This will allow SQL Server to run with a higher priority as compared to the other applications running on the same server. If the SQL Server is running on a dedicated server, it is usually safe to set this option.

If none of the configuration options resolve the bottleneck, consider scaling up the database server. Horizontal scaling is not possible in SQL Server, as it does not support true clustering, so usually, the only option is vertical scaling.

3. Application tuning

After tuning the hosting environment, optimize the application source code and database schema. In this section, one of the many possible ways to tune your Java code and your SQL queries are looked at.

Java code optimization

The most popular way to optimize Java code is by using a profiler. Sun's JVM has built-in support for profiling (Java Virtual Machine Profiler Interface, or JVMPPI) that can be switched at execution time by passing the right JVM parameters. Many commercial profilers are available; some rely on JVMPPI, others provide their own custom hooks into Java applications (using bytecode instrumentation or some other method). But be aware that all these profilers add significant overhead. Thus, the application cannot be profiled at a realistic load level. Use these profilers with a single user or a limited number of users. It is still a good idea to run the application through the profiler and analyze the results for any obvious bottlenecks.

To identify an application's slowest areas in a full-fledged deployed environment, add custom timing logs to the application, which can be switched off easily in the production environment. A logging API, such as log4j or J2SE 1.4's Java Logging API, is handy for this purpose. The code below shows a sample utility class that can be used for adding timing logs for your application:

```
import java.util.HashMap;
//Import org.apache.log4j.Logger;
public class LogTimeStamp
{
```

```
    private static HashMap ht = new HashMap();
    // Preferably we should use log4j instead of System.
    out
    // private static Logger logger = Logger.
    getLogger("LogTimeStamp");
    private static class ThreadExecInfo {
        long timestamp;
        int stepno;
    }
    public static void LogMsg(String Msg) {
        LogMsg(Msg, false);
    }
    /*
    * Passing true in the second parameter of this function
    * resets the counter for
    * the current thread. Otherwise it keeps track of the
    * last invocation and prints
    * the current counter value and the time difference
    * between the two invocations.
    */
    public static void LogMsg(String Msg, boolean flag) {
        LogTimeStamp.ThreadExecInfo thr;
        long timestamp = System.currentTimeMillis();
        synchronized (ht) {
            thr = (LogTimeStamp.ThreadExecInfo)
                ht.get(Thread.currentThread().getName());
            if (thr == null) {
                thr = new LogTimeStamp.ThreadExecInfo();
                ht.put(Thread.currentThread().getName(), thr);
            }
        }
        if (flag == true) {
            thr.stepno = 0;
        }
        if (thr.stepno != 0) {
            // logger.debug(Thread.currentThread().get
            Name() + ":" + thr.stepno + ":" +
            // Msg + ":" + (timestamp - thr.timestamp));

            System.out.println(Thread.currentThread().get
            Name() + ":" + thr.stepno + ":" +
            Msg + ":" + (timestamp - thr.timestamp));
        }
        thr.stepno = thr.stepno + 1;
        thr.timestamp = timestamp;
    }
}
```

After adding the above class in the application, the method `LogTimeStamp.LogMsg()` must be invoked at various checkpoints in the code. This method prints the time (in milliseconds) it took for one thread to get from one checkpoint to the next one. First, call `LogTimeStamp.LogMsg("Your Msg", true)` at one place in the code that is the start of a user request. Then insert the following invocations in the code:

```
public void startingMethod() {
```

```

...
    LogTimeStamp.LogMsg("This is a test message",
true); //This is starting point
...
    LogTimeStamp.LogMsg("One more test message");
//This will become check point 1
    method1();
...
}
public void method1() {
...
    LogTimeStamp.LogMsg("Yet another test message");
//This will become check point 2
    method2();
...
    LogTimeStamp.LogMsg("Oh no another test mes-
sage"); //This will become check point 4
}
public void method2() {
...
    LogTimeStamp.LogMsg("Wow! another test mes-
sage"); //This will become check point 3
...
}

```

A Perl script can take the output of the above log messages as input and print the results in the format below. From these results, it can be inferred as to which part of the code requires the most time and can concentrate on optimizing that part:

Transactions Time	Avg. Time	Max Time	Min	
[This is a ...] to [One more t...] 7500	14410	20937		
[One more t...] to [Yet anothe...]	16	62	0	
[Yet anothe...] to [Wow! anoth...] 27703	39860	50844		
[Wow! anoth...] to [Oh no anot...]	711	1844	94	
[Oh no anot...] to [OK thats e...] 19718	68089	228452		

The above approach represents just one of the ways to tune the Java code. You can use whatever methodology works for you.

Some general suggestions one should be aware of while developing a J2EE application are:

- Avoid using synchronized blocks in the code as much as possible. That does not mean that one should abdicate handling synchronization for the code's multithreaded parts, but should try to limit its usage. Synchronized blocks can severely impair an application's scalability.

- Proper logging proves necessary in serious software development. Try and use a logging mechanism (like log4j) that allows switching off logging in the production environment to reduce logging overhead.
- Instead of creating and destroying resources every time they are needed, use a resource pool for every resource that is costly to create. One obvious choice for this is JDBC (Java Database Connectivity) Connection objects. Threads are also usually good candidates for pooling. Many free APIs are available for pooling various resources.
- Try to minimize the objects stored in HttpSession. Extra objects in HttpSession not only lead to more memory usage, they also add additional overhead for serialization/deserialization in case of persistent sessions.
- Wherever possible, use RequestDispatcher.forward() instead of HttpServletResponse.sendRedirect(), as the latter involves a trip to the browser.
- Minimize the use of SingleThreadModel in servlets so that the servlet container does not have to create many instances of your servlet.
- Java stream objects perform better than reader/writer objects because they do not have to deal with string conversion to bytes. Use OutputStream in place of PrintWriter.
- Reduce the default session timeout either by changing the servlet container configuration or by calling HttpSession.setMaxInactiveInterval() in the code.
- Just as the DNS lookup in the Web server configuration is not used, try not to use ServletRequest.getRemoteHost(), which involves a reverse DNS lookup.
- Always add directive <%@ page session="false"%> to JSP pages where a session is not needed.
- Excessive use of custom tags also may result in poor performance. Keep performance in mind while designing a custom tag library.

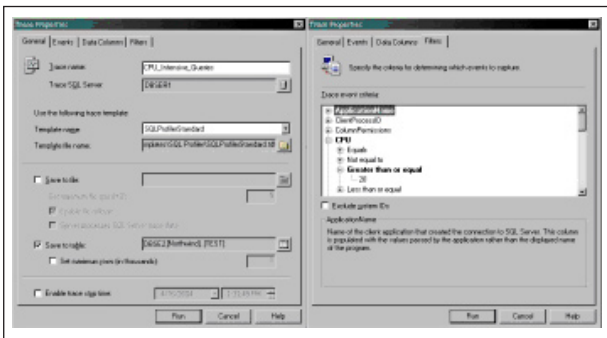
SQL query optimization

The optimization of SQL queries is a vast subject in itself, and many books cover only this topic. SQL query running times can vary by many orders of magnitude even if they return the same results in all cases. Find below a way to identify slow queries and a few suggestions as to how to fix some of the most common mistakes.

First of all, to identify slow queries, SQL Profiler can be used, a tool from Microsoft that comes standard with SQL Server 2000. This tool should be run on a machine other than where the SQL Server database server is running and the results should be stored in a different database

as well. Storing results in a database allows all kinds of reports to be generated using standard SQL queries. Profiling any application inherently adds a lot of overhead, so try to use appropriate filters that can reduce the total amount of data collected.

To start the profiling, from SQL Profiler's File menu, select New, then Trace, and give the connection information and the appropriate credentials to connect to the database that has to be profiled. A Trace Properties windows will open, enter a meaningful name so as to recognize it later. Select Save To Table option and also give the connection information and credentials for the database server (this should differ from the server that is being profiled) where the data collected by the profiler has to be stored. Next, provide the database and the table name where the results will be stored. Usually, a filter can also be added by going to the Filters tab and adding the appropriate filters (for example "duration greater than or equal to 500 milliseconds" or "CPU greater than or equal to 20" as shown in Figure below). Now click on the Run button and the profiling will start.



SQL Profiler

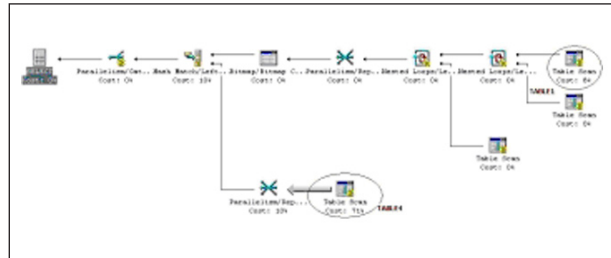
Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html?page=4>

Let's say, based on SQL Profile's results, the following query has been identified as the most time consuming:

```
SELECT [TABLE1].[T1COL1], [TABLE1].[T1COL2],
       [TABLE1].[T1COL3], [TABLE1].[T1COL4]
FROM ((([TABLE1] LEFT JOIN [TABLE4] ON
[TABLE1].[T1COL4] = [TABLE4].[T4COL4])
LEFT JOIN [TABLE3] ON [TABLE1].[T1COL3] =
[TABLE3].[T3COL3])
LEFT JOIN [TABLE2] ON [TABLE1].[T1COL2] =
[TABLE2].[T2COL2])
WHERE [TABLE1].[T1COL5] = 'VALUE1'
```

Now the next task is to optimize this query to improve performance. The TABLE1 has 700,000 records, TABLE2

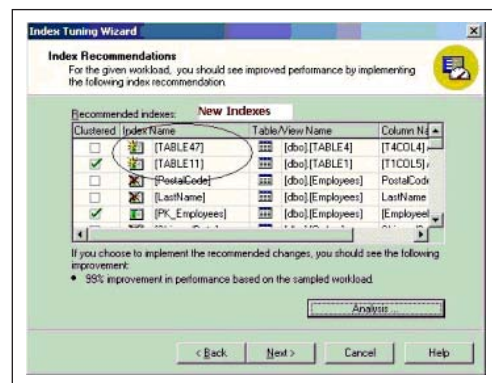
has 16, TABLE3 has 100, and TABLE4 happens to have more than 4 million records. The first step is to understand the cost of the this query, and Query Analyzer comes in handy for this task. Select Show Execution Plan in the Query menu and execute this query in Query Analyzer. Figure below shows the resulting execution plan.



Execution plan before indexes

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html?page=4>

From this plan, it can be seen that the SQL Server is clearly doing full-table scans for all four tables, and together, they make up around 80 percent of the total query cost. Luckily, another feature in Query Analyzer can analyze a query and recommend appropriate indexes. Run Index Tuning Wizard from the Query menu again. This wizard analyzes the query and gives recommendations for indexes. As shown in Figure below, it recommends two indexes to be created ([TABLE4].[T4COL4] & [TABLE1].[T1COL5]) and also indicates performance will improve 99 percent!



Index Tuning Wizard

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html?page=4>

After creating the indexes, the execution time declines from 4,125 milliseconds to 110 milliseconds, and the new execution plan shown in Figure 6 shows only two table scans (not a problem as TABLE2 and TABLE3 both have limited records).

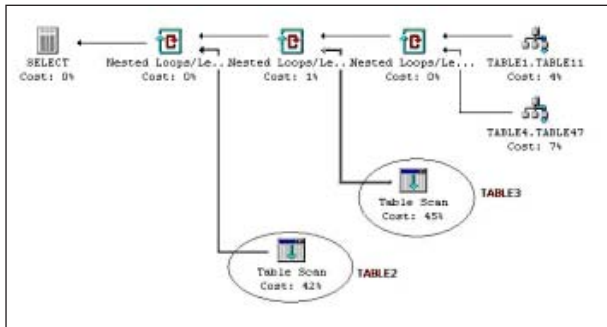


Figure 6. Execution plan after indexes. Click on thumbnail to view full-sized image.

Source : <http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html?page=4>

This was just an example of what proper tuning can achieve in terms of performance. In general, SQL Server's auto-tuning features automatically handle many tasks. For example, reordering WHERE clauses will never yield any benefit, as SQL Server internally handles that. Still, here are a few things one should keep in mind while writing SQL queries:

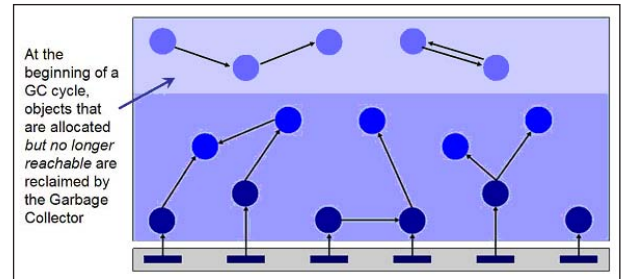
- **Keep the transactions as short as possible.** The longer a transaction is open, the longer it holds the locks on the resources acquired, and every other transaction must wait.
- **Do not use DISTINCT clauses unnecessarily.** If you know the rows will be unique, do not add DISTINCT in the SELECT clause.
- **When possible, avoid using SELECT *.** Select only the columns from which the data is needed.
- **Consider adding indexes to those columns causing full-table scans for your queries.** Indexes can result in a big performance gain, as shown above, even though they consume extra disk space.
- **Avoid using too many string functions or operators in your queries.** Functions like SUBSTRING, LOWER, UPPER, and LIKE result in poor performance.

39. How does garbage collection work?

There are three phases to garbage collection (GC):

- Mark
- Sweep
- Compact (not performed during each garbage collection)

The mark phase is implemented through a process called the reachability test, which is illustrated in the figure below :



Source : http://www.quest.com/Quest_Site_Assets/WhitePapers/WPA-Wait-BasedTuning-Haines.pdf

The garbage collector starts by identifying all objects that are directly visible in each thread; these objects comprise the root set. It traverses through all the objects in the root set to determine what objects they can see. It repeats this process until it has identified all visible objects. Internally the JVM may maintain an array of bits for all the memory location in the heap and "mark" the memory location in its array for each live object it finds.

After the reachability test, the garbage collector "sweeps" away all memory locations that are not marked (that is, all the dead objects). Finally, if the heap is sufficiently fragmented, the garbage collector may "compact" the heap to create enough contiguous blocks of memory to hold new objects.

As things are loaded into memory (and unload them), the heap will soon fill up—requiring that garbage collection frees memory for the application to continue. In the Sun JDK version 1.3.x, which ships with most production application servers, it defines two types of garbage collection: minor and major.

Minor collections are performed by a process called copying that is very efficient, whereas major collections are performed by a process called mark compact, which is very burdensome to the Virtual Machine. The heap is broken down into two partitions: the young generation, in which objects are created and hopefully destroyed; and the old generation, in which objects are retired to a more permanent place in memory. The young generation uses copying to perform minor collections, whereas the old generation uses mark compact to perform major collections, so the goal when tuning garbage collection is to size the generations to maximize minor collections and minimize major collections.

The impact of garbage collection on the performance of applications can be subtle or significant. For example, the Sun JVM uses a generational garbage collection algorithm that can operate in one of two modes: minor and major. A minor collection is relatively quick (usually

in the order of a few tenths of a second) and does not require that operations be suspended while it runs; a major collection is longer running and freezes all operations while it runs. This JVM freeze is called a “pause” and in extreme circumstances can take on the order of several seconds to run. During this time your application ceases processing, and response times reflect this pause.

Source : <http://www.informit.com/articles/printerfriendly.aspx?p=31441>

40. What are Failover Tests?

Failover Tests verify of redundancy mechanisms while the system is under load. This is in contrast to Load Tests which are conducted under anticipated load with no component failure during the course of a test.

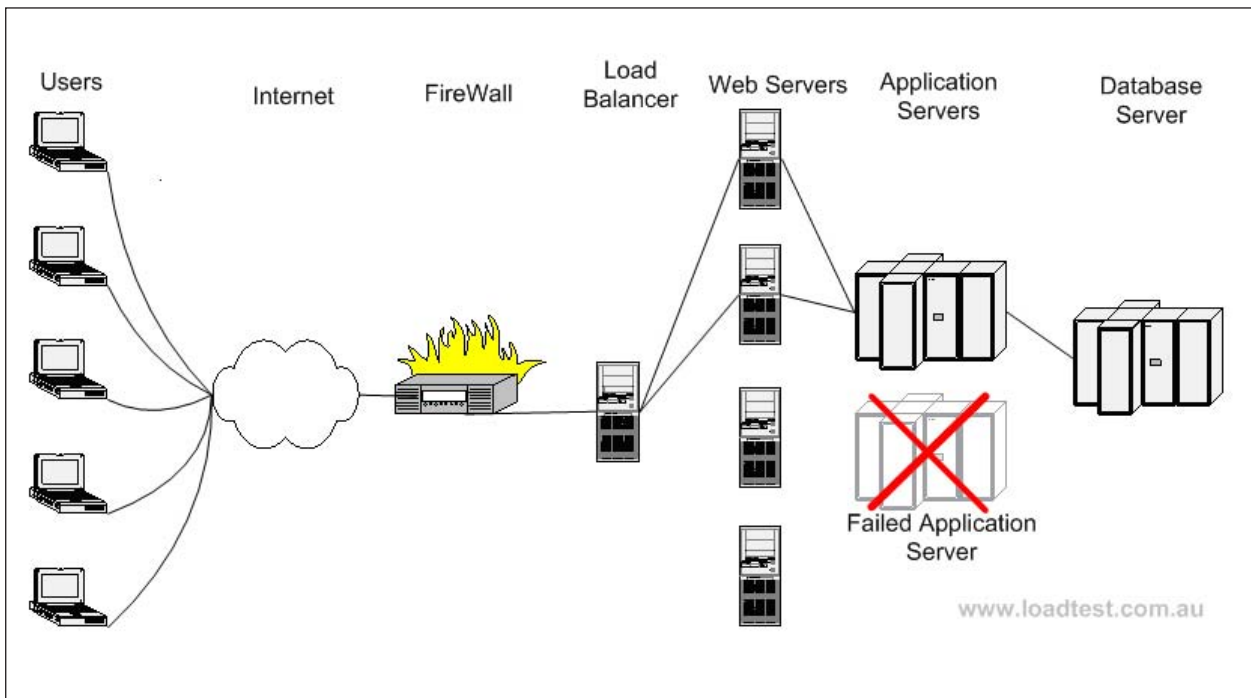
For example, in a web environment, failover testing determines what will happen if multiple web servers are being used under peak anticipated load, and one of them dies.

Does the load balancer react quickly enough?

Can the other web servers handle the sudden dumping of extra load?

Failover testing allows technicians to address problems in advance, in the comfort of a testing situation, rather than in the heat of a production outage. It also provides a baseline of failover capability so that a ‘sick’ server can be shutdown with confidence, in the knowledge that the remaining infrastructure will cope with the surge of failover load.

Source : http://www.loadtest.com.au/types_of_tests/failover_tests.htm



41. What are soak tests?

Soak testing is running a system at high levels of load for prolonged periods of time. A soak test would normally execute several times more transactions in an entire day (or night) than would be expected in a busy day, to identify any performance problems that appear after a large number of transactions have been executed.

Also, it is possible that a system may ‘stop’ working after a certain number of transactions have been processed due to memory leaks or other defects. Soak tests provide an opportunity to identify such defects, whereas load tests and stress tests may not find such problems due to their relatively short duration.

Some typical problems identified during soak tests are listed below :

- Serious memory leaks that would eventually result in a memory crisis
- Failure to close connections between tiers of a multi-tiered system under some circumstances which could stall some or all modules of the system
- Failure to close database cursors under some conditions which would eventually result in the entire system stalling
- Gradual degradation of response time of some functions as internal data-structures become less efficient during a long test

Apart from monitoring response time, it is also important to measure CPU usage and available memory. If a server process needs to be available for the application to operate, it is often worthwhile to record its memory usage at the start and end of a soak test. It is also important to monitor internal memory usages of facilities such as Java Virtual Machines, if applicable.

Source : http://www.loadtest.com.au/types_of_tests/soak_tests.htm

42. What are stress tests?

Stress Tests determine the load under which a system fails, and how it fails. This is in contrast to Load Testing, which attempts to simulate anticipated load. It is important to know in advance if a 'stress' situation will result in a catastrophic system failure, or if everything just "goes really slow". There are various varieties of Stress Tests, including spike, stepped and gradual ramp-up tests. Catastrophic failures require restarting various infrastructure and contribute to downtime, a stress-full environment for support staff and managers, as well as possible financial losses. If a major performance bottleneck is reached, then the system performance will usually degrade to a point that is unsatisfactory, but performance should return to normal when the excessive load is removed.

Before conducting a Stress Test, it is usually advisable to conduct targeted infrastructure tests on each of the key components in the system. A variation on targeted infrastructure tests would be to execute each one as a mini stress test.

In a stress event, it is most likely that many more connections will be requested per minute than under normal levels of expected peak activity. In many stress situations, the actions of each connected user will not be typical of actions observed under normal operating conditions. This is partly due to the slow response and partly due to the root cause of the stress event.

Let's take an example of a large holiday resort web site. Normal activity will be characterized by browsing, room searches and bookings. If a national online news service posted a sensational article about the resort and included a URL in the article, then the site may be subjected to a huge number of hits, but most of the visits would probably be a quick browse. It is unlikely that many of the additional visitors would search for rooms and it would be even less likely that they would make bookings. However, if instead of a news article, a national newspaper advertisement erroneously understated the price of accommodation, then there may well be an influx of visitors who clamour to book a room, only to find that the price did not match their expectations.

In both of the above situations, the normal traffic would be increased with traffic of a different usage profile. So a stress test design would incorporate a Load Test as well as additional virtual users running a special series of 'stress' navigations and transactions.

For the sake of simplicity, one can just increase the number of users using the business processes and functions coded in the Load Test. However, one must then keep in mind that a system failure with that type of activity may be different to the type of failure that may occur if a special series of 'stress' navigations were utilized for stress testing.

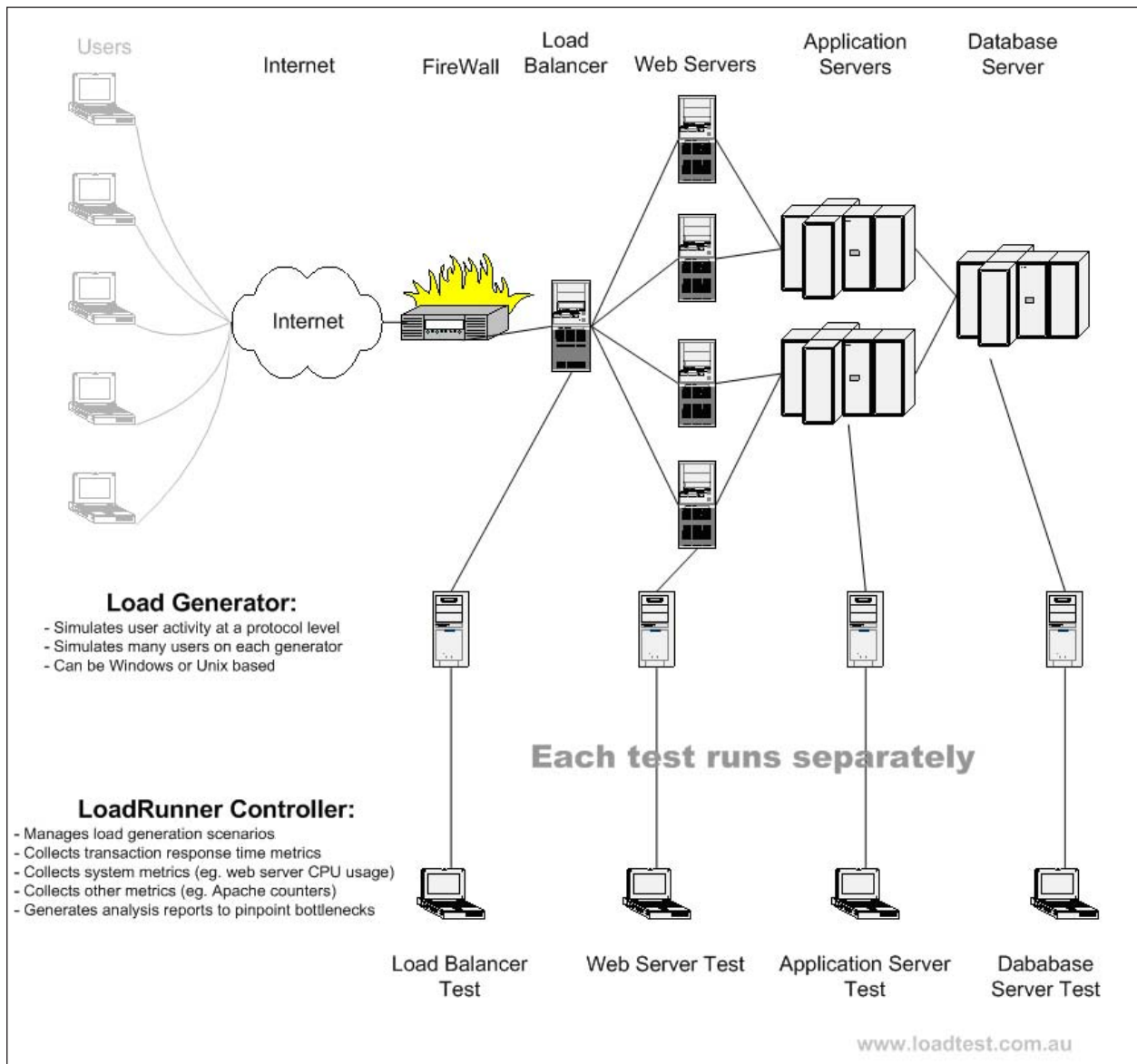
Source : http://www.loadtest.com.au/types_of_tests/stress_tests.htm

43. What is Targeted Infrastructure Test?

Targeted Infrastructure Tests are Isolated tests of each layer and or component in an end-to-end application configuration. It includes communications infrastructure, Load Balancers, Web Servers, Application Servers, Crypto cards, Citrix Servers, Database etc., allowing for identification of any performance issues that would fundamentally limit the overall ability of a system to deliver at a given performance level.

Each test can be quite simple. For example, a test ensuring that 500 concurrent (idle) sessions can be maintained by Web Servers and related equipment, should be executed prior to a full 500 user end-to-end performance test, as a configuration file somewhere in the system may limit the number of users to less than 500. It is much easier to identify such a configuration issue in a Targeted Infrastructure test than in a full end-to-end test.

Source : http://www.loadtest.com.au/types_of_tests/targeted_infrastructure_tests.htm



44. What are Network Sensitivity Tests ?

Network Sensitivity tests are variations on Load Tests and Performance Tests that focus on the Wide Area Network (WAN) limitations and network activity (eg. traffic, latency, error rates, etc.) Network Sensitivity tests can be used to predict the impact of a given WAN segment or traffic profile on various applications that are bandwidth dependant. Network issues often arise at low levels of concurrency over low bandwidth WAN segments. Very 'chatty' applications can appear to be more prone to response time degradation under certain conditions than other applications that actually use more bandwidth. For example, some applications may degrade to unacceptable levels of response time when a certain pattern of network traffic uses 50% of available bandwidth, while other applications are virtually

unchanged in response time even with 85% of available bandwidth consumed elsewhere.

This is a particularly important test for deployment of a time critical application over a WAN.

Also, some front-end systems such as web servers, need to work much harder with 'dirty' communications compared with the clean communications encountered on a high speed LAN in an isolated load and performance testing environment.

The three principle reasons for executing Network Sensitivity tests are as follows:

- Determine the impact on response time of a WAN link. (Variation of a Performance Test)

- Determine the capacity of a system based on a given WAN link. (Variation of a Load Test)
- Determine the impact on the system under test that is under 'dirty' communications load. (Variation of a Load Test)

Source : http://www.loadtest.com.au/types_of_tests/network_sensitivity_tests.htm

45. What are Volume Tests?

Volume Tests are often most appropriate to Messaging, Batch and Conversion processing type situations. In a Volume Test, there is often no such measure as Response time. Instead, there is usually a concept of Throughput.

A key to effective volume testing is the identification of the relevant capacity drivers. A capacity driver is something that directly impacts on the total processing capacity. For a messaging system, a capacity driver may well be the size of messages being processed.

Volume Testing of Messaging Systems

Most messaging systems do not interrogate the body of the messages they are processing, so varying the content of the test messages may not impact the total message throughput capacity, but changing the size of the messages may have a significant effect. However, the message header may include indicators that have a very significant impact on processing efficiency. For example, a flag saying that the message need not be delivered under certain circumstances is much easier to deal with than a message with a flag saying that it must be held for delivery for as long as necessary to deliver the message, and the message must not be lost. In the former example, the message may be held in memory, but in the later example, the message must be physically written to disk multiple times (normal disk write and another write to a journal mechanism of some sort plus possible mirroring writes and remote failover system writes!)

Before conducting a meaningful test on a messaging system, the following must be known:

- The capacity drivers for the messages (as discussed above)
- The peak rate of messages that need to be processed, grouped by capacity driver
- The duration of peak message activity that needs to be replicated
- The required message processing rates

A test can then be designed to measure the throughput of a messaging system as well as the internal messaging system metrics while that throughput rate is being

processed. Such measures would typically include CPU utilization and disk activity.

It is important that a test be run, at peak load, for a period of time equal to or greater than the expected production duration of peak load. To run the test for less time would be like trying to test a freeway system with peak hour vehicular traffic, but limiting the test to five minutes. The traffic would be absorbed into the system easily, and you would not be able to determine a realistic forecast of the peak hour capacity of the freeway. You would intuitively know that a reasonable test of a freeway system must include entire 'morning peak' and 'evening peak' of traffic profiles, as both peaks are very different. (Morning traffic generally converges on a city, whereas evening traffic is dispersed into the suburbs.)

Volume Testing of Batch Processing Systems

Capacity drivers in batch processing systems are also critical as certain record types may require significant CPU processing, while other record types may invoke substantial database and disk activity. Some batch processes also contain substantial aggregation processing, and the mix of transactions can significantly impact the processing requirements of the aggregation phase.

In addition to the contents of any batch file, the total amount of processing effort may also depend on the size and makeup of the database that the batch process interacts with. Also, some details in the database may be used to validate batch records, so the test database must 'match' test batch files.

Before conducting a meaningful test on a batch system, the following must be known:

- The capacity drivers for the batch records (as discussed above)
- The mix of batch records to be processed, grouped by capacity driver
- Peak expected batch sizes (check end of month, quarter & year batch sizes)
- Similarity of production database and test database
- Performance Requirements (eg. records per second)

Batch runs can be analysed and the capacity drivers can be identified, so that large batches can be generated for validation of processing within batch windows. Volume tests are also executed to ensure that the anticipated numbers of transactions are able to be processed and that they satisfy the stated performance requirements.

Source : http://www.loadtest.com.au/types_of_tests/network_sensitivity_tests.htm

46. Further reading

<http://www.javaperformancetuning.com/tips/appservers.shtml>

47. References

http://publib.boulder.ibm.com/infocenter/wsphelp/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rprf_javamemory.html

http://www.adobe.com/livedocs/coldfusion/5.0/AdvancedColdFusion_Administration/overview2.htm

<http://cnx.org/content/m13409/latest/>

<http://www.informit.com/blogs/blog.aspx?uk=New-Performance-Tuning-Methodology-White-Paper>

http://download.intel.com/technology/itj/2003/volume07issue01/art03_java/vol7iss1_art03.pdf

http://www.quest.com/Quest_Site_Assets/WhitePapers/WPA-Wait-BasedTuning-Haines.pdf

<http://www.javaworld.com/javaworld/jw-05-2004/jw-0517-optimization.html>

