

Software Performance Testing Handbook

A Comprehensive Guide for Beginners

Ramya Ramalinga Moorthy

**I dedicate to
my family and friends**

Contents

Preface.....	7
Chapter 1: Introduction.....	8
What is Performance Testing	8
How Performance Testing is different from Functional Testing	9
What is an E-Business Application	9
Components of a Web Site	10
Why Performance Testing is Crucial	11
Performance Testing Jargons	11
Chapter 2: Performance Testing – A Quick Overview	16
Performance Management	16
Performance Testing Approach	17
Dynamism in Performance Testing	19
Application Performance Management	20
Myths about Performance Testing	20
Performance Process Maturity Model	23
Chapter 3: Deriving Performance Test Goals	26
Why to set the Performance Test Goals	26
Know your customer	26
Deriving Quantitative Goals	27
Industrial Standards for Benchmarks	28
User behavior Modeling	28
Importance of automation in Performance Testing	30
Performance Test Estimation	31
Chapter 4: Workload Identification.....	35
What is Workload?	35
Types of Workload	35
Customer Behavior Modeling	38
Customer Behavior Model Graph (CBMG)	38

User Community Modeling Language (UCML)	38
Web log Analysis	38
Web Log Analysis Tools Overview	39
Web Log Analyzers Summary	40
Web Log Analysis Metrics	44
Web Site Traffic Monitoring	48
Overview of Statistical distributions	48
Chapter 5: Overview of Performance Testing Tools	59
Performance Test Tool Requirements	59
Market Tools	60
Open Vs Licensed Performance test tools	60
Criteria for choosing the Performance test tool	60
The most important tool	61
Performance Testing Tools Overview	61
Chapter 6: Setting up Performance Test Environment and Performance Test Scripts development best practices	64
Know the Test and Production environment	64
Test environment Isolation	64
Network Isolation	65
Load Generators	65
Test Data Generators	66
Test Execution Environment	66
How to choose the test scenarios	67
Tips for writing successful performance test scripts	68
Real time versus Virtual user Mapping	71
Chapter 7: Application Benchmarking	72
What is benchmarking?	72
Why should we benchmark the applications?	72
Software Benchmarking	72
Hardware Benchmarking	73

Industry Standards for Benchmarking	73
Transaction processing Performance Council (TPC)	73
Standard Performance Evaluation Council	74
Mistakes in Benchmarking	75
Chapter 8: Performance Test Execution.....	76
Quality of Service of web application	76
Virtual User Simulation	77
Pre-requisites for Performance Tests	77
Types of Performance Tests	78
Performance Test Execution Approach.....	81
Tips for Load & Stress Test Execution	84
Test Execution Factors	85
Chapter 9: Performance Test Monitoring	89
Introduction to Performance Monitoring.....	89
What Counters needs to be monitored?	89
Key Performance Counters.....	90
Performance Monitoring in Windows platform	93
Using Perfmon.....	93
Post production Monitoring.....	95
Benefits of Performance Monitoring	95
Chapter 10: Performance Bottleneck Analysis.....	96
Scott's Recommendation on Scatter Charts	96
Scatter Chart Types.....	97
Performance Test Analysis	98
Best Practices on Performance Bottleneck Analysis and Isolation.....	100
Performance Testing Vs Profiling	102
HP Diagnostics.....	102
SQL Profiler	106
Know your Limits	107
Typical bottlenecks on DB server	108

Chapter 11: Performance Test Reporting	110
Why is it Critical	110
How to report the results of performance testing	110
Components of a good Test Report	111
Components of a Good Visual	112
Response Time Reporting	112
Best Practices of Test Reporting recommended by Scott Barber	113
Chapter 12: Road Ahead – Moving towards advanced Performance Engineering techniques ..	115
Moving towards Performance Engineering	115
What is Queuing Theory?	115
Types of Queuing System	116
What can be modeled as Queues?	117
Limitations of Queuing Theory	118
Operational Laws	118
a. Utilization Law	118
b. Little’s Law:	119
c. Interactive Response Time Law	120
d. Forced Flow Law	120
e. Flow Balance Assumption	121
f. Bounding Law of Service Demand	121
Capacity Planning	122
Why Capacity Planning is required	122
When is it required?	123

Preface

The main objective of compiling this material is to help beginners to quickly march forward into Performance Testing and also to enrich the knowledge for intermediate level performance testers. There is no copyright specifically available for this material. I have decided to pile up the possible information which I have learnt in the past 3+ years of my life as a Performance Tester which might help fellow performance testers.

Also, I would like to thank all the viewers who provided feedbacks and appreciations for sharing performance testing related information in my blog. I decided to compile this material as I realized that lot of beginners are getting benefited from the small articles published in my blog. This material is the outcome based on the interest level shown by various people in my blog <http://ramya-moorthy.blogspot.com>.

The contents of this material is based on what I have learnt from different pioneers in Performance testing field which benefited me to perform performance testing in a effective way along with my personal experiences and best practices which I learnt while testing more than 30-40 different web applications. This is a performance test tool agnostic material, though few test tools are cited for understanding at few places within this material. The audiences are requested to note that all the information provided in this material is more related to web application performance testing.

Information provided in this document about companies, domains, products, etc are fictional and does not correspond to any real organization. The contents provided in this material are purely based on my personal experience and recommendation and the entire risk arising out of the use of this documentation remains with the recipient.

For any queries, contact

Ramya Ramalinga Moorthy B.E, M.S

Bangalore, India.

Ramya.rmoorthy@gmail.com

<http://ramya-moorthy.blogspot.com>

Chapter 1: Introduction

What is Performance Testing

In today's world, we have reached a point where we are expecting things to get done in fraction of seconds. Everyone is interested in developing a very high performance system in order to meet the customer demand. The demand for the performance is slowly gaining more importance.

Performance Testing is the act of testing or evaluating software or a component or hardware for its conformance with the performance testing goals and optimizing its performance. It's about testing the application with the intent to identify the application scalability. The Performance Testing seems to be very vague if approached in an adhoc way. It needs specific process steps and work principles in place in order to completely achieve the goal of conducting this activity.

One should realize that Performance Testing is a subset of Performance Engineering, which is an art of building performance into the design and architecture of a system, prior to the actual coding effort. Performance Engineering helps in building software with an aid to meet the performance requirements right from the requirements phase.

The context of Performance Testing remains the same though it is carried out for achieving different kind of goals like comparing the performance of the system deployed in two different environments, to benchmark the system performance for certain load, to measure the system breakpoint, etc.

In earlier days, wherein the importance of performance testing is not known, organizations end up in purchasing high capacity hardware to solve the performance problems though it is the not the right way to meet the customer demand. Nowadays, organizations are slowly realizing the importance of Performance Testing even from the early life cycle phase of the system development. But even now, many people think that Performance Testing is all about using a test tool to simulate high load and identify the response time of various transactions. It is not simple as they think!!

How Performance Testing is different from Functional Testing

Performance Testing is altogether a different way of looking at the system. The important pre-requisite for conducting the performance testing is that the system should be functionally stable. Unless the code changes are freezed and the system functionality is baselined, the performance testing does not have a value on a system.

I could remember one of my projects wherein a performance test engineer was forced to check a functional issue as part of conducting load testing. The objective of the functional testing is to verify and validate the functional requirements whereas the objective of performance testing is to verify and validate the nonfunctional requirements (NFR) of the system related to performance.

Most of the time, Performance testing is often compared with functional testing. Many a time, I could remember the days wherein we need to explain and convince the management why we need more man – days than a week's time mentioned in the project plan. It is not like functional testing where there is a possibility to conduct high-level smoke tests on the system before the actual test execution. It is always better not to conduct performance testing, rather than conducting it without meeting its pre-requisites.

What is an E-Business Application

The business which uses electronic media like computer, wireless devices etc is referred as E-Business. The examples of e-business include Internet Banking, Tele-shopping, etc. E-Business is widely used for servicing the customers and for collaborating with the business partners. Nowadays, the awareness of using these kinds of e-business applications has increased and because of which the criticality of building applications with much of ease and performance is also becoming increasingly important. Because of the market competition, service providers are competing to develop solutions which provide the best to the customers not only in terms of the functional features but also in terms of security and performance.

The four different types of e-business include

- Business-to-Business (B2B)
- Business-to-Customer (B2C)
- Business-to-public administration
- Customer-to-public administration

Business-to-Business (B2B) includes the exchange of services, information, etc that happens between businesses (Example: Company Website, Brokerage Site).

Business-to-Customer (B2C) includes the sale of products or services by the company to the customer (Example: Online Banking, Railway charts). It forms the major contributor to the e-businesses in Internet.

Business-to-Public administration includes the transactions between the companies and the regional or local government organizations (Example: Income tax payments)

Customer-to-public administration includes the transaction between the customers and the regional or local government organizations (Example: Online Voting, Online Passport application).

Components of a Web Site

A typical E-business application will have the end users accessing their application using a web browser (Example: Internet Explorer, Mozilla, Netscape, Opera, etc) through Internet. The e-business application runs in an environment with different kind of servers like application server, web server and database server.

Web Server

It translates the web url path to the system resource and responds with the requested system resource. i.e. It accepts the HTTP requests from the clients (via web browsers) and serves it if the requests are pertaining to the static contents (like images or html document).

The key performance metrics include server throughput and number of requests served at any point of time.

Application Server

It handles most of the business logic of the web application. The client requests (via web browser) pertaining to the dynamic contents (like jsp page which needs to render and fetch data from the database server).

The key performance metrics include server throughput, CPU utilization and memory utilization.

Database Server

It is a data storage and retrieval system which manages the data queries. It provides data to the application server when requested.

The key performance metrics include Query response times, CPU utilization and memory utilization.

Why Performance Testing is Crucial

Performance Testing is an essential activity for developing optimal software solutions. It is highly important to avoid last minute surprises that arise after deploying the application in the production environment. The failures experienced by the end users are vulnerable to the competitors.

Performance Testing helps to answer the sample of following questions:

- What is the response time of system during the expected load condition?
- How is my system behaving during unexpected load condition?
- Is my system scalable to specific user load?
- Which environment provides the best performance for my system?
- Is my system performing well after adding the new NIC card?
- Will my system handle the spike user loads? Or will it crash?
- Does my system need a dedicated database server to meet the performance goals?

The performance being the essential quality of software application, it is required to evaluate the software systems for its performance.

Performance Testing Jargons

The following are the some of the main performance testing terminologies used in day today life. A good understanding of the below jargons are critical to get the most out of this book.

Business Transaction: It refers to a sequence of request-response and the related information exchange between the client and the server in order to accomplish a business requirement. For example, an online banking customer transferring money from one account to other is a business transaction.

Test Scenario: It refers to a high critical business transaction or high traffic workflow identified for the performance testing. Normally all the business transactions cannot be tested for its performance. The Pareto analysis helps to identify those specific 20% transactions which is often exercised by the end users rather than testing the remaining 80% non-critical transactions.

Think Time: It refers to the time taken by the user for thinking or clicking on any of the web page links, buttons, etc while navigating through the web site. Think time is a very important parameter which needs to be set while scripting the scenarios using the performance testing tools. The business analysts of the application or web site management team or sometimes even the end user survey might give a realistic picture about the think time requirements of a transaction.

Virtual User: The Performance testing tool simulates the real user traffic by creating virtual users during the performance test. A virtual user is configured in the performance test tool to run the script simulating the real user behavior. Though in real time, users access the application from a unique IP address, more than one virtual user can be configured to access the application from the same IP address.

Simultaneous User load: The simultaneous users have the active session on the server at any point of time wherein each user will be executing different transactions. For example, if we say 100 simultaneous users load, then there will be 100 active sessions opened up in the server, wherein each user will be performing different set of transactions – one logging in, another viewing reports, another navigating to the next page, etc. The simultaneous user load of a web site would be always greater than the concurrent user load of a web site.

Concurrent User load: The concurrent users connect to the server and perform the same operation at any point of time. For example, if we say 100 concurrent user load, all 100 users would be logging in at the same point of time, view the reports at the same point of time, etc. For example, an online banking web site might have 10,000 – 20,000 simultaneous user load, but 1000 to 1500 concurrent user load.

Performance Requirements/Goals: It refers to the quantitative way of putting forth the criteria for claiming the system performance is good. The performance requirements of an application are often expressed in response time, hits per second, transactions per second, etc. The performance requirements of the System Under Test (SUT) need to be quantitatively expressed in the performance test plan.

Workload: It refers to the user load subjected by a web application under real time user access or during the performance test and the way the users are distributed between various transaction flows. Normally, the web server log files of a web site can be analyzed to learn about the workload of the site (if the web site is already in production). For web sites which are yet to go alive for the first time, a workload model needs to be developed based on the discussion with business analysts, application experts, etc. It is very important to know the workload of the application before conducting the performance test. Conducting the performance test for a system without proper analysis on the workload might lead to misleading results.

User Abandonment: It refers to the situation wherein the end users exit from the web site because of the slow performance of the site. The user abandonment rate varies from web site to web site. A low priority web site might experience a high user abandonment rate compared to the payroll web site. Analysis on the abandonment rate of the site needs to be done before concluding on the load on the site.

Baseline Test: It refers to the test conducted to measure the application performance for 1 virtual user load. The baseline test is often conducted to collect the metrics about the system performance for 1 user load and thereby it could be used for comparing the system performance at a high load condition. It also helps to check the validity of the test script.

Benchmark Test: It refers to the test conducted to measure the application performance for a low load condition. Normally for a benchmark test, 15 -20% of the target load can be considered. The objective of the benchmark test is to validate the correctness of the test scripts and to check the readiness of the system before subjecting it to a high load. It helps to know whether various components of the system collaborate as per the design and meet the performance test SLA for 20% of target load.

Hit: It refers to the server request for accessing a web page or a file or an image from a web server. For example, if a web page contains 5 images, then a user visit to that page creates 6 hits on the web server (5 hits for fetching each of the images and 1 hit for fetching the web page). For

a web site, the performance requirement can be derived in terms of hits per unit time like 'the system should support the load of 10 hits per second with a response time of below 5 seconds'.

Response Time: It refers to the servicing time or processing time in order to serve the request. It is measured from the time the web browser sends the request to the web server to the time when the first byte of response is received by the requesting user after server processing. The response time of a transaction say login refers to the time taken by the server to respond to the request by logging into the system and displaying the home page (it includes web server processing time + application server processing time + database server processing time + network latency).

Throughput: It refers to the amount of data (in bytes) transferred by the server in order to the serve the client requests. It is a good indicator of server performance as it refers to the amount of work done by the server. Throughput also refers to the number of requests or transactions processed by the server at any point of time. For example, the server throughput can be expressed as 2.5Mbps or 35 Hits/sec or 8 Transactions/sec.

Page Views: It refers to the number of pages transferred by the server in order to serve the client requests. i.e. number of times the web site is completely loaded or refreshed. A page is a html page or script page or plain text documents. It is also a good indicator of server performance. If a user visits a web site and navigates to 5 pages and then logs off then the Page Views could be indicated as 5.

Performance Bottleneck: It refers to the slow spot, the effects of which are widely felt. It refers to the situation/areas which do not allow the application to perform as per its ideal specifications. For example, the response time increase for the load of 100 virtual users because of improper setting of HTTP connections parameter in the IIS server, CPU utilization reaching 95% during 100 users load are typical performance bottlenecks. A bottleneck might lead to a failure if mitigation actions are not taken.

Load Test: It refers to the test conducted on the system simulating the actual usage pattern or a representative load in order to identify the system behavior and diagnose the performance bottlenecks in the system. The load test should have the objective of checking how the system performs for the specific load, say 1000 simultaneous users. It is carried out by ramping up the user load from zero to the maximum count with the specific ramp up rate depending upon the

kind of application and the user access pattern. Sometimes load testing is conducted to identify the maximum acceptable user load or the highest arrival rate the system can handle.

Stress Test: It refers to the test conducted by subjecting the system to an unreasonable load to identify the system breakpoint. It is generally considered as a negative testing as the system is subjected to an unrealistic load. It is usually conducted to know the importance of potential harm created during unrealistic load and to act on it proactively. It also sometime helps the web site owners to inform the user base about the scalability limit of the application.

Spike Test: It refers to test conducted by subjecting the system to a short burst of concurrent load to the system. This test might be essential while conducting performance tests on an auction site wherein a sudden load is expected.

Volume Test: It refers to the tests designed to measure the throughput of the system more in a batch processing, messaging kind of environment. The objective of this test is to identify the processing capacity. Also, in database server perspective , a volume test can be conducted to test the server performance by subjecting it to huge data (say , 75,000 rows of employee information in EMPLOYEE table) keeping in mind the data available by end of 2 years of web site usage.

Stability / Longevity / Endurance Test: It refers to the tests conducted to check whether the system is able to withstand the load continuously for a longer duration say 48 hours, etc. The system is subjected to reasonable representative load and tested for its continued performance. These tests are helpful in identifying memory problems (memory leaks, buffer overflow, etc) which arise after continuous usage of the application for a longer duration.

Bottleneck Isolation Test: It refers to the specific tests conducted on the system or a specific component of a system to isolate or confirm the performance problem. It is often conducted during or after the load tests to identify the root cause / confirm a specific performance issue. Mostly these tests are decided dynamically to debug the performance problems.

Chapter 2: Performance Testing – A Quick Overview

Performance Management

Reactive Approach

The Performance testing activity is often considered as a reactive way of performance management. In most of the cases, the system performance is never thought of during the early phases of Software Development Life Cycle phases (SDLC). Performance Testing is often thought of only as a last activity after the system testing phase.

Also, if the performance bottlenecks are related to the system architecture or the system design, then it becomes highly impossible to fix the issues due to the high COPQ (Cost of Poor Quality) and in certain cases, the system is put into trash because of the huge deviations in the performance constraints.

Basically waiting for the performance problems to appear and then dealing with it at the end is always not a better approach. Hence performance testing is considered as a reactive approach as there is not much importance given to the system during early life cycle phases. It is more a 'fix-it-later' approach which is not that effective.

Proactive Approach

The Proactive approach anticipates the performance problems well in advance and adopts techniques to mitigate them. The importance of performance is thought about during all the SDLC phases right from the requirement analysis phase and various performance engineering activities are identified for the system.

The disadvantages of 'fix-it-later' approach are well understood and engineering practices are adopted to analyze the system design in performance angle. As the system is evaluated for the performance right from the design phase, the chances of last minute surprises is very less in proactive approach.

Performance Testing Approach

Though Performance Testing is considered as more of a reactive approach, there should be a process in place to carry out performance testing in a strategic way. The strategy detailed in Figure 2.1 represents the Performance Testing approach which has shown success in lot of projects.

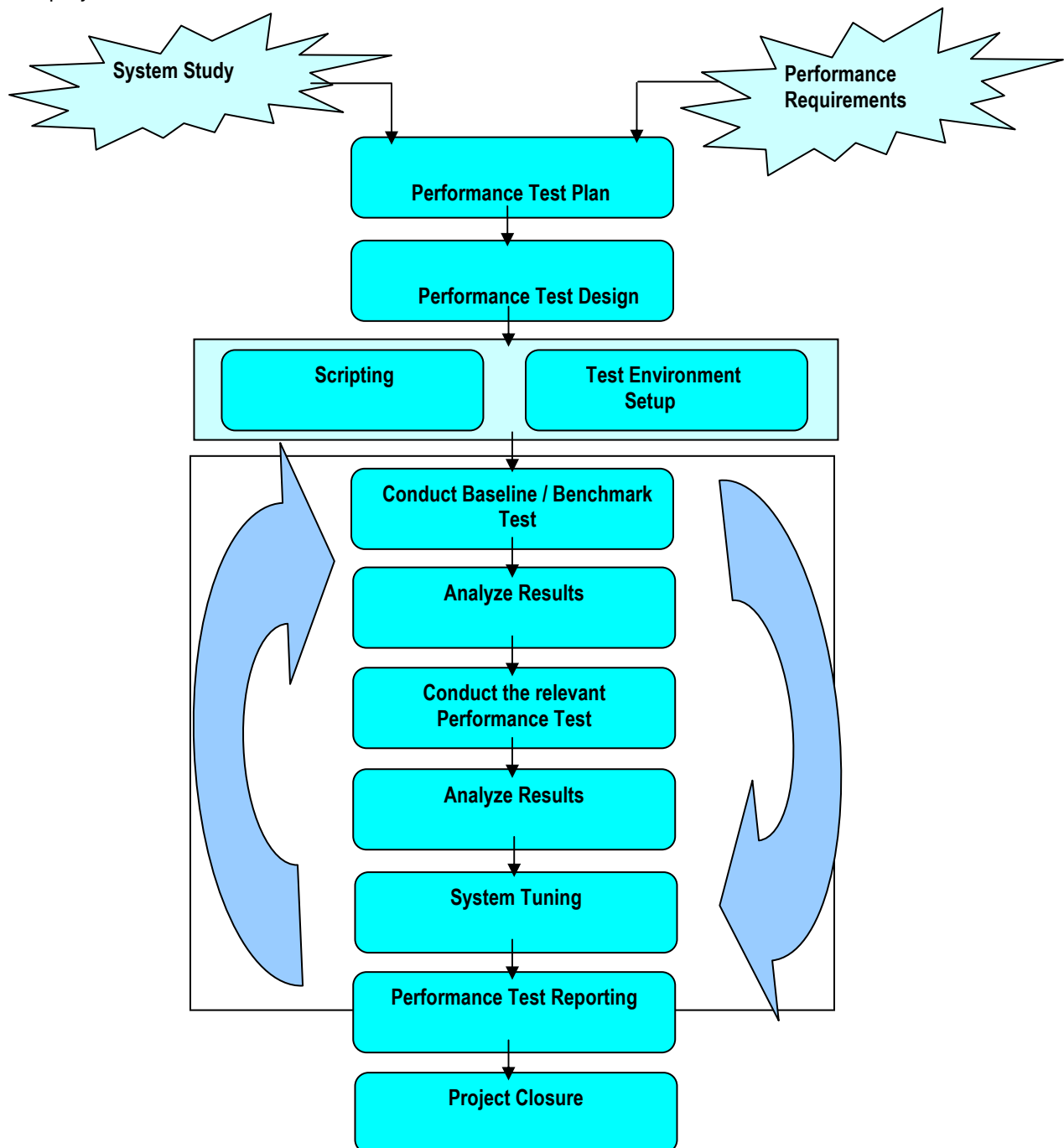


Figure 1: Performance Test Activities

The Performance Test lifecycle phases includes: Planning, Scripting, Test Execution and Test Reporting.

Performance Test Planning: In this phase, the performance test engineer understands the purpose and various functional flows of the system. The Performance test objective of the system needs to be understood by the performance test engineer (if specified by the customer) or the performance requirements or goals needs to be derived based on the customer expectations. The Performance Test plan including the testing tool to be used, types of tests to be conducted, business scenarios to be included during the performance test, effort required for the performance test, entry and exit criteria for conducting the test, etc needs to be created by the performance tester and delivered to the stakeholders. The business stakeholders should help the performance tester to understand the application usage details in order to arrive at the right test strategy.

Scripting: The performance test scenarios identified during the planning phase are scripted during this phase. The test scripts are generated using the performance test tool. The script simulates the real time user access to the system. The scripts should be prepared with sufficient test data in order to simulate multiple user load on the system.

Performance Test Execution: The planned tests are executed during this phase and the system performance is certified during this phase. Before starting the actual performance tests planned for the system, it is always recommended to conduct a baseline test for 1 virtual user in order to validate the test scripts and to collect metrics about the system performance for 1 user load. The application failures / script failures noticed during the baseline test needs to be resolved before proceeding with the benchmark test which is conducted with 15 – 20% of the target load. The benchmark test helps in confirming the readiness of the system before subjecting it to the actual target load.

The system is then subjected to various tests (load tests, stress tests, volume tests, isolation tests and longevity tests, etc) depending upon the test strategy prepared during the planning phase. The system performance behavior and bottlenecks are then analyzed. The performance bottleneck analysis activity is basically a team effort wherein the performance tester, system administrators, technical experts and DBA play a vital role in identifying the root cause of the bottleneck.

The test execution and bottleneck analysis activities are cyclic in nature. After the system tuning, again the tests are conducted to recheck the impact on the system performance. Sometimes, it might lead to generation of few other scripts or redefining the performance goals.

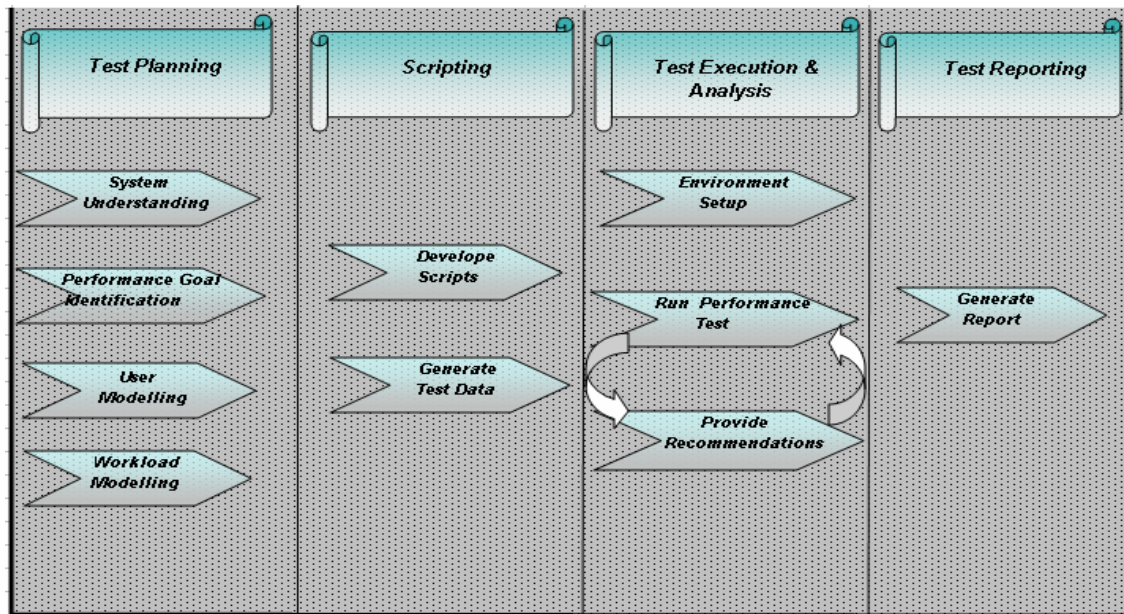


Figure2 : Performance Test Lifecycle

Performance Test Reporting: During this phase, the performance test analysis and recommendations are documented in an understandable format and presented to the stakeholders. It is very important that the performance test report should be presented in a detailed manner with the explanations for the bottlenecks identified during the test cycle and the recommendations.

Dynamism in Performance Testing

The degree of dynamism in performance testing is very high. Scott Barber rightly compares the performance testing to crime investigation. As it is more of an investigation activity which may take any direction, it is very tough to estimate the performance test efforts.

Most of the Organizations use ROM (Rough Order of Magnitude) estimation based on the historical data & technical risk factors. For them, the test estimation effort is validated through the following breakup.

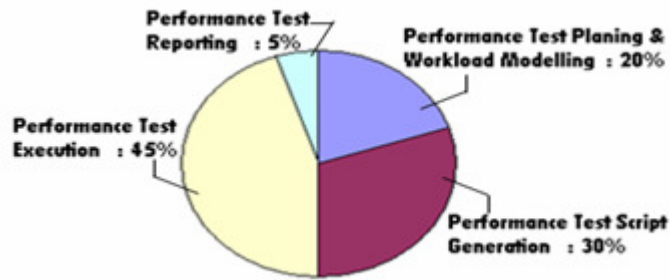


Figure 3: Performance Test Efforts breakup

At a high level, the performance test planning phase contributes to 20% of the total performance testing effort. The Test script development phase contributes to 30% of the total performance testing effort. The Test execution phase contributes to 45% of the total performance testing effort. The Test reporting phase contributes to 5% of the total performance testing effort. The importance of the activities carried out in each of the phases depends on the scope of the project. In most of the projects, replanning might be required during the test execution phase because of its dynamic nature. The projects planned for specific tests may end up having multiple rounds of unplanned tests because of the unexpected performance bottlenecks.

Application Performance Management

Application Performance Management (APM) is about managing the performance of the application throughout its lifecycle to ensure availability and better performance to the end users of the system. It forms a closed loop of performance management by managing the performance in development, testing and production stages of the application. Lot of production monitoring and diagnostics tools are available in the market to identify the performance problems of the production system and to provide quick diagnostics to resolve the performance problems. It also provides lot of inputs to carry out the pre-production performance testing. Some tools provide flexibility to use the same test assets (scripts, etc) for both pre-production and post-production performance testing.

Myths about Performance Testing

Organizations generally like to make the headlines with their success stories and not about the failure web sites due to unexpected user load. Industry study reveals that poor performing IT applications cost industrialized nations almost \$45 billion annually. Most of the organizations understand the importance of performance testing and the impact of bypassing it. *'How much time would it take to do the performance testing of an application?'* is a million dollar question for

all organizations wherein the Software Performance Engineering (SPE) is not in place from the start of the project. Most of the organizations plan for the performance testing of the applications towards the end of the project, though it is of great importance to decide whether the performance bottlenecks are addressed before the go-live dates in order to meet the end user demand on the production systems.

The following are the some of the popular myths about Performance Testing.

MYTH 1 : Performance testing is the last activity thought of only based on time availability.

FACT : The importance to the system performance should be thought from the requirements identification phase and performance engineering activities needs to be practiced during each phase of SDLC.

MYTH 2 : Conducting Performance testing would increase the system performance irrespective of implementing the recommendations.

FACT : Conducting Performance test alone will not improve the system performance. It helps to identify whether the system meets the performance test goals and identify the performance bottlenecks of the system.

MYTH 3 : Performance Testing is just doing code profiling or memory profiling to tune the code.

FACT : Performance Testing is about evaluating the system for its conformance to the performance test goals and thereby identifying the performance bottlenecks in the software and hardware of the system.

MYTH 4 : Performance Testing needs to be done on all the functional flows of the application to identify performance issues.

FACT : Performance Testing needs to be done for specific scenarios based on Pareto analysis. The scenarios that are used often, which are of stakeholder's concern, high critical scenarios, scenarios which are considered error prone are the right candidate for conducting performance testing.

MYTH 5 : The response time goal should be per the industry standards.

FACT : There is no industry standard for response time. The response time goal needs to be derived based on the hardware and software capacity available for the system considering the end users tolerance limit.

MYTH 6 : Instead of investing in performance testing activities, it is better to procure high capacity hardware as it is cheap.

FACT : Industry studies show that the hardware price is improving at around 40% per annum whereas the demand for IT resources is increasing at around 60% per annum. The reason for the actual performance issue needs to be identified through performance testing activity.

MYTH 7 : Performance Testing can be planned in parallel during the functional testing.

FACT : Performance Testing needs to be planned on a stable system only after the completion of system testing phase.

MYTH 8 : An application needs performance testing once in its life time irrespective of how many modules of the application are revamped over a period of time.

FACT : Performance testing needs to be best done as a continuous process of measuring, analyzing and tuning the system performance. The system performance needs to be revalidated whenever the code is changed or added in the system or if there is a hardware change.

MYTH 9 : The Performance bottlenecks can be identified by conducting one test run.

FACT : The Performance bottlenecks cannot be identified by conducting one round of test. The isolation of bottlenecks becomes very tough in complex systems. Multiple rounds of isolation tests need to be run in order to identify the performance bottlenecks depending upon the type of bottleneck.

MYTH 10 : Performance Testing can be done only if the system is properly tuned.

FACT : Performance Testing can be done on any system to identify the bottlenecks. Irrespective of the system condition, performance testing can be conducted.

MYTH 11 : The Performance Tester is solely responsible for detecting and tuning the system for performance.

FACT : It is often a joint effort to detect the performance bottleneck and tune the system to meet its performance goals. Advices from Subject matter experts, DBAs and System administrators become a value add to identify/isolate the bottleneck easily.

MYTH 12 : Performance Testing can be carried out only on the development / test environment.

FACT : Performance Testing needs to be planned on an isolated environment in order to isolate the performance bottlenecks easily.

Performance Process Maturity Model

Michael Maddox, a pioneer in Performance Testing, proposes five levels of the maturity model for the performance test process. All the levels have the following common characteristics:

1. The levels are cumulative. The performance activities and processes practiced at level 2 are retained and enhanced at level 3 and so on through higher levels.
2. Different applications may exhibit different maturity levels.
3. Some level of learning and feedback is applied as work progresses. Organizations at higher levels of maturity apply more effective, more strategic feedback.

Maturity level 1: Fire Fighting

In this level, a strong emphasis is on the reactive mode of behavior. The developers create systems with little awareness of operational considerations, including performance. The performance issues identified before deployment are addressed by minor adjustments to program logic yielding only incremental improvements. The systems delivered for production are effectively 'thrown over the wall'. A very little understanding about the servers and quantitative science hardly applied to their sizing leading to unexpected outages followed by emergency upgrades and high rework. Hardly server monitoring activities are planned to generate strategic value.

Maturity Level 2: Monitoring

In this level, some level of automation to collect performance data from the systems in production are taken up. There is some effort to systematically deal with the server resource measurements. The monitoring system would catch the key out-of-threshold measurements. Application systems are still subjected to little performance scrutiny prior to deployment. As a result, unexpected levels of user population can still disrupt the stability of the system. Capacity Planning is limited to systems for which there exists performance data already collected. There is little formal capacity planning expertise, modeling or other rigorous performance methodology applied. Workload measurements such as transaction counts may be largely ignored, as the emphasis is more on the resource measurements.

Maturity Level 3: Performance Optimizing

In this level, performance evaluation and planning are part of the development process. Emphasis on 'building performance into systems rather than trying to add it later' is practiced in this level. Capacity Planning is formalized and uses a well-refined methodology, supported by a specialized capacity planning tool. The data collected on a defined workload is feed to the capacity planning activity. High degree of automation is used to publish the performance reports. There exists awareness of and process for dealing with business changes and how they impact system performance and resource requirements. Systems that are developed at maturity level 3 perform predictably within the performance limits specified in the requirements.

Maturity Level 4: Business Optimizing

In this level, primary improvement made is the understanding of the effectiveness of the system more fully in the areas of user productivity with the application, business value of the system, proposed business changes to the system impacting the resource utilization or user productivity, complete costs of the system well documented from end-to-end. A wide range of coordinated skills is required to reach this level of maturity, because it encompasses performance, human factors, management and system analysis domains, etc. A process-oriented culture is emerging in this level and enterprise goals are visible to all to measure the design and performance decisions.

Maturity Level 5: Process Optimizing

In this level, executive management understands the benefits of performance and process optimization fully. The focus in this level is to extend the benefits still further by closely examining the costs versus profit possibilities where computer systems are involved and rationalizing the benefits to be gained from each potential optimization against the costs of achieving that optimization, i.e. looking at the ROI (return on investment). A process culture ensures visibility of enterprise goals to all. Everyone measures his or her efforts against process effectiveness, elevating process concerns to the management when necessary. Enterprise architecture has been rationalized and performance optimization is being achieved and improved across broad sectors of the enterprise.

Five Steps for Solving Software Performance Problems

By Connie.U.Smith & Lloyd.D.Williams

© Copyright 2002, Software Engineering Research and Performance Engineering Services.

Step 1: Figure out Where You Need to Be

You should define precise, quantitative, measurable performance objectives. You can express performance objectives in several ways, including response time, throughput, or constraints on resource usage. When defining performance objectives, don't forget that your needs may change over the product's lifetime. It is a good idea to consider future uses of your software so that you can anticipate these changes and build in the necessary scalability.

Step 2: Determine Where You Are Now

Begin by quantifying the problems and identifying their causes. Identify the workloads (transaction or customer-call volumes, event arrival rates, and so on) that are causing problems. We have found that starting with an understanding and assessment of the software architecture offers better opportunities for achieving performance and scalability objectives. We have found that it is best to understand the software's purpose, architecture, and processing steps.

Step 3: Decide Whether You Can Achieve Your Objectives

Before you dive into tuning your software, it's a good idea to stop and see if you can actually achieve your objectives by tuning. Once you've identified the hot spots, some simple calculations will help you decide if you can meet your performance objectives by tuning the software.

Step 4: Develop a Plan for Achieving Your Objectives

Once you have identified potential remedies, you can rank them based on their payoff. Then apply them starting with those that have the highest payoff.

You determine the quantitative performance improvement by estimating the new resource requirements.

Step 5: Conduct an Economic Analysis of the Project

Upon completion, gather data on the time and cost for the performance analysis, time and for software changes (include coding and testing), hardware costs if applicable, software distribution costs when applicable, and all other costs of the project. Then gather data on the effect of the improvements. It is also a good idea to review the changes made to see if they could have been predicted earlier in the software's life.

Chapter 3: Deriving Performance Test Goals

Why to set the Performance Test Goals

It is always required to decide what we are trying to achieve and then take quality steps in an organized way to achieve it. The performance test goals are important to check whether the system meets its expectations. In order to claim that the system performance is good or bad, one needs performance test goals to be in place to justify the claim quantitatively.

The Performance Test Engineer needs to certify the application for performance based on the goals set by the stakeholders / derived with the help of stakeholder's inputs.

Know your customer

How to get the performance requirements from non-technical customers is a valuable question that everyone has. In most cases, customer expectations are either incompletely defined or more conceptually defined. Customers are not expected to be the performance test experts. (If so, then we are jobless!). We need to better understand their expectations by asking the right set of questions and translating their conceptual requirements into more quantitative goals. The performance tester needs to play the role of a modem translating between the novice user language and performance testing terminologies. It is the performance tester's responsibility to get the essence from the customer.

Talk in Customer's Language: One needs to quickly understand the customer and should start talking in their language. Make a note that you don't simply ask questions like what is the Hits/sec goal, response times expected out of various transactions and make the customer to feel uncomfortable. In one of my projects, when I asked the above set of questions, I got an unexpected answer from them, 'Hey, in fact we also wanted to know answers for these questions. That's why we have planned for the Performance Testing'. Though their answer surprised me; then I slowly understood that I am not talking in their language.

Look for Customer's Expectation: Some customers may be clear in their objective; some may have conceptual expectations on the performance test results. Try to understand their language and thereby their expectation. Don't talk using performance testing jargons and make the customer feel that they are ignorant, which is not the case. Don't expect that your customer will give you the performance goals in a single shot. Rather, start the discussion with an overview of important terminologies like what response time actually means, what is a Hit, what is a transaction, what is a performance goal, why is it required before starting the performance tests, etc so that the customer feels comfortable to explain what he/she wants. Try to educate the customer and explain with an example to understand what they are expecting from the system.

Gain Customer's Confidence: Try to build a good rapport with your customer. Try to make them realize that you care for them. Try to briefly explain about how you have solved the issues in some other projects in your past. Try to make them feel that you have achieved great things in your past projects. That's the first impression you get from them. Once you get that, you will have the full liberty to state your expectations from them in order to provide successful results. Don't scare the customer talking too much technical rather make them to understand the results in a very polite way. More than results, your way of expressing the result and your communication style impresses the customer.

Deriving Quantitative Goals

Here are the few examples of conceptual goals provided by the customers in few of my projects.

- The system should be very fast.
- The response time of the system should be as per the industry standard.
- The web server performance should be as high as possible.
- The system should support large number of users.
- The system should run without any failures for a long duration.

In most of the cases, the customers end up giving conceptual goals. It is the responsibility of the performance tester to understand the application and analyze the relevance of those conceptual goals to derive quantitative goals.

The goals need to be simple and quantitative in order to check whether it is met during the performance test. Examples of quantitative goals are

- The system should have the response time of less than 10 seconds for all reports.
- The web server should support 10 hits per second without single failure.
- The application server throughput should support 2Mbps with the response time of less than 15 seconds for all transactions.
- The database server CPU utilization should not go beyond 75% on the average during 500 users load.

Industrial Standards for Benchmarks

Benchmarking refers to the method of comparing the performance of two or more systems across different hardware or software or architectural components.

Standard Performance Evaluation Corporation (SPEC) and Transaction Processing Performance Council (TPC) forms the popular Industrial Standards. These are non-profit corporations formed in 1980s and they are widely used today in evaluating the system performance. They establish benchmarks that can be used to compare the performance of different computer systems.

For example, SPEC CPU2006 provides the comparative performance measure that can be used to compare compute-intensive workloads on different computer systems. TPC-App is an application Server and web services benchmark which helps to showcase the performance capabilities of application server systems.

It is often misinterpreted that there is an industry standard for response time measure. The “8-second rule” became very popular in 1990s, which states that most web application should have 8 seconds as the page response time. But there is no rule of thumb against this rule. The response time goal needs to be derived or decided based on the server capacity or by identifying the tolerance limit of the corresponding end users rather than looking for an industrial standard.

User behavior Modeling

The User behavior model describes the user behavior in terms of the navigational patterns of the end users in the web site, the frequency of access to various functionalities of the system and how long the end user uses the web site. It forms the input for the workload modeling which deals with modeling the intensity of the system usage.

Most of the performance testers think that it is very tough to identify the user behavior pattern. But it isn't that tough. It is more of understanding the purpose of the system and how the users tend to use the system.

For example, let us consider an Employee Appraisal System. If we want to develop the user behavior model for this application, then the foremost thing is that we should understand the application features and navigational flows. Some of the important flows of the application include

- Employee enters the goals
- Employee deletes the goals
- Employee submit the goals to the manager for approval

There might be lot of other functionalities in the application, but for performance testing only important paths/flows which are often used or which are expected to error prone. It is more of a Pareto analysis to determine the specific 20% of the functional flows to conduct the performance testing leaving the remaining 80% of the functional flows which are less frequently used. Performance optimization of those flows would not bring in a huge improvement in the overall system performance.

Pareto Analysis

Wikipedia says that Pareto analysis is a formal statistical technique in decision making that is used for selection of a limited number of tasks that produce significant overall effect. Pareto analysis uses the Pareto principle, the idea of doing 20% of work by which you can generate 80% of the advantage of doing the entire job. Or in terms of quality improvement, a large majority of problems (80%) are produced by a few key causes (20%). In essence, the problem-solver estimates the benefit delivered by each action, then selects a number of the most effective actions that deliver a total benefit reasonably close to the maximal possible one.

CBMG, CVM and UCML

Customer Behavior Modeling Graph (CBMG), Customer Visit Model (CVM) and User Community Modeling Language (UCML) are the representation languages that can be used to achieve the user modeling.

Customer Behavior Modeling Graph considers the entire system as a set of states seen from the server side and the transitions between the states happen as user navigates from one page to the other. CBMG can be constructed if the transition probability matrix is known. Each flow is associated with the rank which is determined by the frequency in accessing a particular state. After constructing the model, a number of metrics like session length, number of visits, etc could be derived from it.

Customer Visit Model provides the visit ratios to each state of the CBMG model. The visit ratios represent the number of visits made to a state during a session. More details on CBMG and CVM can be found in Daniel Menace's book on performance testing, '*Scaling for E-Business*'. These representations help in identifying the average number of access to various business scenarios during each user visit, which helps in setting the performance test goals.

User Community Modeling Language comprises of set of symbols used to visualize the system usage pictorially. This allows an ease of conversation and results in more accurate user models than the complex spreadsheets of data. More details on UCML can be found in Scott Barber's articles available at perftestplus.com.

If the web application is already in production, then the web log files or the real time traffic monitoring systems like Webtrends should have the user access pattern information. If the web application is going alive for the first time, then the user access information needs to be obtained through discussion with business analysts or the business owners of the system.

Importance of automation in Performance Testing

Performance testing cannot be done manually without use of any tools. May be if the requirement is to check the system performance for 1 or 2 users accessing concurrently then it can be done manually. But it is inefficient because of the issues in repeatability and accuracy of test runs. Also the performance bottleneck isolation is a very tedious task using manual performance testing approach. These issues paved the way for generation of lot of in-house tools where organizations started developing their own homegrown solutions depending on the project requirements and budget constraints. Slowly a lot of open source performance testing tools and licensed tools were developed by different vendors.

The performance test tool choice needs to be decided during the planning phase. A feasibility analysis can be conducted to check whether the application testing is supported by the tool

before deciding on the tool. Organizations prefer to use open source tools like OpenSTA, JMeter, etc due to budget constraints. But there are few compromises in using open source tools in terms of ease and readymade analysis reports.

Performance Test Estimation

Performance Testing forms an investigation activity and hence it is very difficult to predict the efforts required for performance testing. But by experience I have come across various factors which impact the effort requirement for all the test activities during performance testing. As a best practice, every performance test project should get started with the project kick off meeting where the high level performance test objectives and application overview is discussed. The discussion inputs need to be considered for the effort estimation plan. The effort estimation needs to be approved by the stakeholders before continuing with the performance test plan development activity.

The performance test effort requirement depends on the performance test scope of the project. The effort required for each life cycle phase of performance testing depends on the project category. The broad classification of the performance test projects are as follows:

- **Verification / ReTest:** Mostly this category projects will be simple in nature and it requires a recheck on the application performance due to minor changes in the application / infrastructure.
- **Application Benchmarking:** Mostly this category projects will be of simple to medium complexity and it requires certification of the application for the supported load levels without any detailed bottleneck analysis or tuning activity. In most of the cases, the application is tested for the first time.
- **Regression Test:** Mostly this category projects will be of medium to complex category and it requires redoing of the entire performance testing for the application which was performance certified earlier due to major functional changes or fixes.
- **Performance Testing:** Mostly this category project will be of medium to complex category projects and it requires complete effort for all the test phases and it includes bottleneck analysis and tuning based on the project requirements. In most of the cases, the application is tested for the first time.

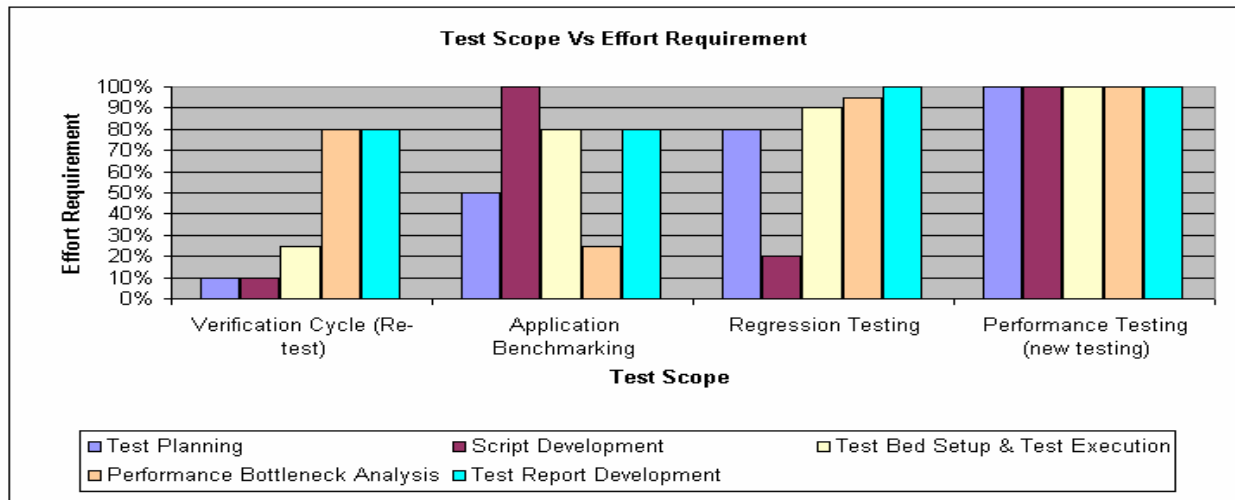


Figure 4: Test Scope vs Test Effort Requirement

The factors affecting the effort requirement for various test activities carried out in each phase of the performance testing (planning, scripting, execution and reporting) are discussed below.

Test Planning

Normally test planning effort varies from 3 to 8 days approximately based on the system complexity. The effort requirement needs to be estimated based on the below factors.

- Knowledge on Application Under Test (AUT)
- Performance Test Requirements Baseline
 - Review and Baseline (based on business targets) and/or
 - Analysis and Baseline (based on Server Workload Modelling)
 - Performance Test Scenarios Baseline (provided by business experts) and/or
 - Performance Test Scenarios Baseline (based on Server Workload Modelling)
- Performance Test Tool Feasibility Analysis
- Performance Test Approach
 - Number of Cycles
 - Types of Tests in each test cycle

Scripting

Normally the scripting effort varies from 5 to 15 days approximately based on the system complexity. The effort requirement needs to be estimated based on the below factors.

- Protocol complexity
- Number of business processes
- Scripting complexity
- Number of steps within business processes
- Test data creation and analysis

A sample customized template used for defining the scripting effort is provided below.

Factors	Classification	Approximate Effort Requirement based on the classification (in hours)		
Protocol complexity	Simple / Medium / Complex	4	16	32
Number of Business Processes	Less than 4 / Greater than 4 & Less than 6 / Greater than 8 & Less than 10	24	40	56
Scripting complexity	Simple / Medium / Complex	4	12	16
Number of steps within Business Processes	Less than 10 / Greater than 10 & Less than 20 / Greater than 20 & Less than 30	4	8	16
Test data creation & analysis	Simple / Medium / Complex	4	8	12

Figure 5: Scripting Effort Calculation - Factors

Test Execution

The Test execution effort varies based on the project scope, number of test cycles and number of tests planned per cycle. Effort requirement varies case to case and needs to be estimated only based on the below factors.

- Number of test cycles
- Number of tests per test cycle
- Bottleneck analysis scope
 - Simple (Bottleneck analysis and reporting)
 - Complex (Bottleneck analysis and tuning)

Test Reporting

The Test reporting effort varies based on the stakeholder expectation. Normally it varies from 2 to 4 days approximately based on the audience of the test report. Also as a best practice, it is required to provide intermediate test reports during the end of each type of the test i.e. after the completion of all load tests, after the completion of stress test, etc, to avoid sharing last minute surprises about the system performance at the end of test cycle.

Chapter 4: Workload Identification

What is Workload?

The Workload refers to the user load created on the server by the real time user access or during the performance tests. As per G.Kotis words, “a workload can be defined as all set of inputs (programs, commands, etc) from the users to the system from the environment”. For example, for a UNIX terminal, the way how the end user enters the system commands in the terminal is the workload. The workload of a specific system varies from the other.

The workload could be either natural or synthetic. The natural or real workload is the natural realistic actual load in the real time production environment. The synthetic workload mimics the real time behavior of the system and hence it models the natural workload. The characteristics of a synthetic workload could be controlled.

The workload selected for carrying out the tests should closely resemble the real time workload pattern of a web application. If there is more deviation from the test workload and the real time production system workload, then the performance test results will not give accurate results of the system performance in production environment.

Types of Workload

It is very important that the workload of the system should be properly analyzed and designed. This can be done by understanding the end users of the system and the usage pattern of the end users. If the server workload is not analyzed properly and understood, then performance tests might create a different workload on the server leading to unrealistic tests. While conducting performance tests any of the following workloads can be used.

Steady-State Workload

A steady-state workload represents the constant load of simultaneous or concurrent users accessing the system. The number of users accessing the application at any point of time is constant. Though it does not represent the real time usage, this workload is widely used to carry out stability tests (which last for a longer duration of say 32 hours, etc).

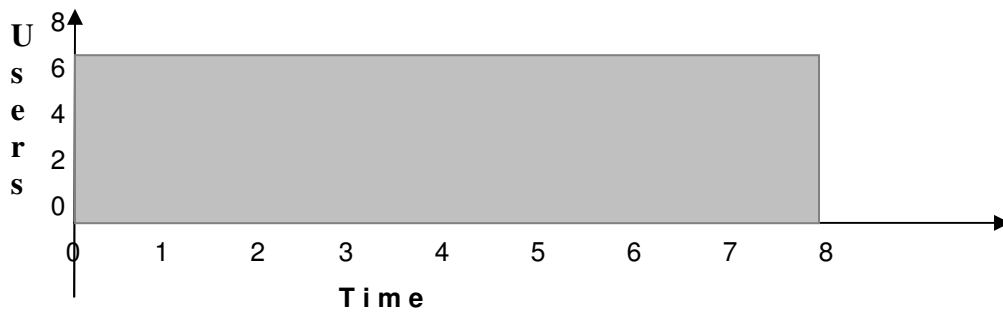


Figure 6: Steady-State Workload

Inclined Workload

An inclined workload represents the increasing load of simultaneous or concurrent users accessing the system. The test starts with minimum number of users and the virtual users are ramped up step by step in a periodic fashion. This workload represents the real time usage of the web application. Load tests uses this workload in order to identify the maximum users supported by the system under test.

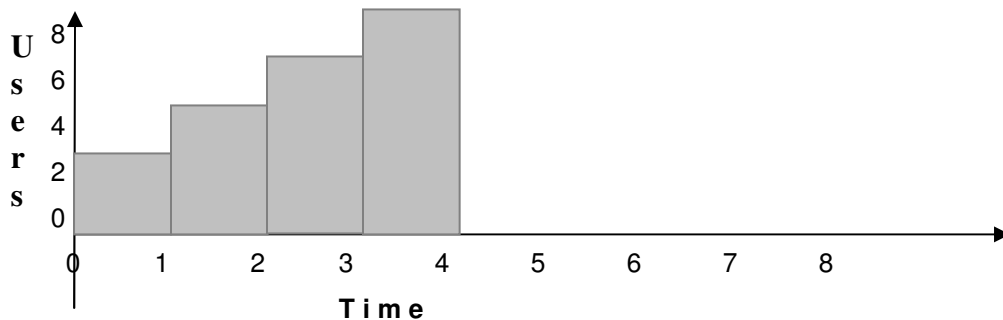


Figure 7: Inclined Workload

Bursty Workload

A bursty workload represents a sudden peak load accessing the system. This test is usually carried out to check the system performance for sudden load condition. Sometimes during stress tests, bursty workload would be used in order to identify the system performance for sudden spikes.

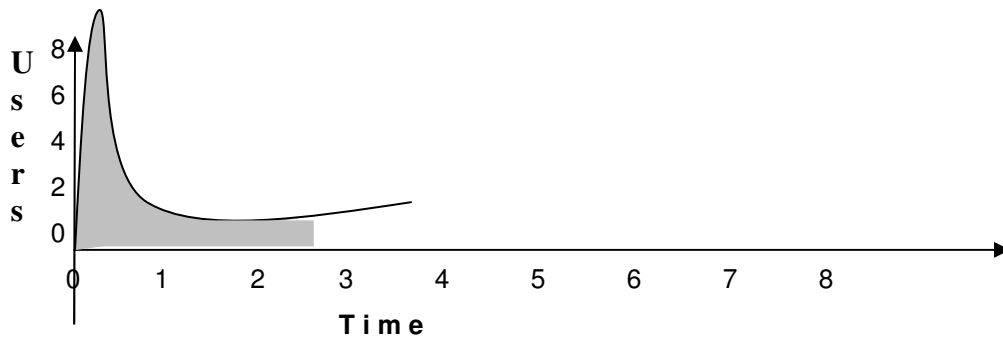


Figure 8: Bursty Workload

Time Specific Workload

A time specific workload is very tough to achieve. Normally when the web applications are accessed by geographically distributed users, at any point of the time there will be users accessing the application from various regions with varying time zones. Only for a very less duration say an hour or two the load on the server will be less because of the non-overlapping time zones. For such applications, the user patterns will have repetitive highs & lows throughout the day and extreme load during overlapping time zones at certain point of time. In order to simulate such conditions, time specific workload can be used.

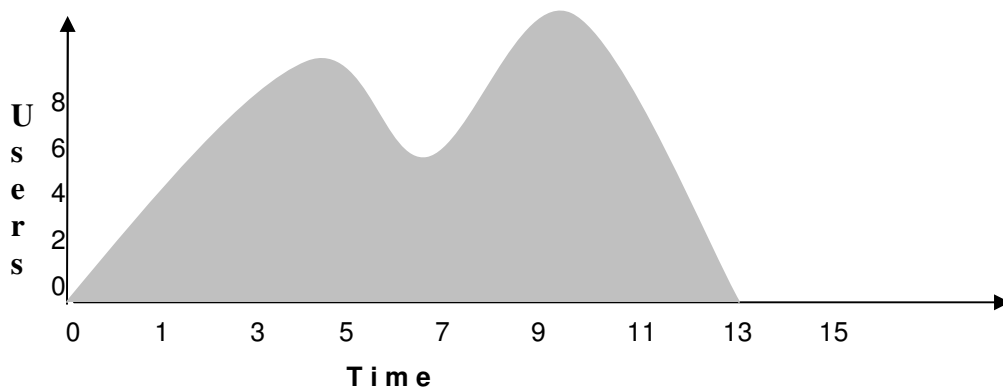


Figure 9: Time Specific Workload

Customer Behavior Modeling

More than knowing that quantitative amount of the server load, it is very important to know the kind of requests that comprises the server load. The end users of an e-business site can send various kinds of requests during each visit to the site. The time taken by the end users i.e session time, during each visit to the site would be different based on their objective. Different users might have different navigation pattern. Some users would use the site extensively whereas some users would use the site for once. Hence, it becomes very important to characterize the customers of the site before adopting the performance test strategy. The customer behavior model would provide inputs to the system workload model.

Customer Behavior Model Graph (CBMG)

The CBMG is in the form of a graph which visually depicts the various states of the customers using the site. It consists of nodes which represent the states and arrows connecting the states which represent the transition between the states. CBMG always have 1 Entry state and 1 Exit state and other states depends upon the navigational pattern of each site. CBMG is used to depict how the customers interact with the web site. Based on the CBMG model, the session duration, number of users, frequently used navigation flows and periodicity of usage by the customers can be identified.

User Community Modeling Language (UCML)

The UCML is a set of symbols that can be used to create visual system user models and depict associated parameters. It is used in the performance testing to determine what functional flows needs to be included in a test and with what frequency they will occur. It includes various symbols including Quantity Circle, Description Line, Synchronization Point, Optional Box, Condition, Loop, Exit Path, and Branch.

Web log Analysis

The web log analysis refers to the analysis of the log files of a web server (Example: IIS or Apache web server) and derive metrics about the access pattern of the real time users. There are various formats used for logging the information in a log file. Some of the ASCII file formats include W3C Extended File format, IIS log file format, NCSA common log file format. Different

format uses different time zones to represent the activities. (W3C Extended file format uses Coordinated Universal Time (UTC) same as Greenwich Mean Time (GMT)).

The sample W3C Extended File format and IIS log file format are provided below.

#Fields: time c-ip cs-method cs-uri-stem sc-status cs-version

Example:

17:42:15 172.16.255.255 GET /default.htm 200 HTTP/1.0

#Fields: date time c-ip cs-username s-ip s-port cs-method cs-uri-stem cs-uri-query sc-status sc-bytes cs-bytes time-taken cs (User-Agent)

Example:

2005-12-11 00:00:22 131.127.217.67 - 131.127.217.69 80 GET /scripts/lightyearisapi.dll Test 200
282 270 0 Mozilla/4.06+ [en] + (WinNT; +I)

Web Log Analysis Tools Overview

As the web server log files cannot be easily interpreted, analyzer tool needs to be used to analyze the access requests and report statistics. There are many web log analytics tools available in the market to analyze the log file contents and to report statistics graphically. The choice of the web log analysis tool depends on the type of information required and budget constraints.

Some of criteria for selecting the log analysis tool are

- What do you need to measure?
- Is the interface easy to use?
- How fast are reports produced?
- Are reports and terminology in plain English?
- What is the maximum log file capacity that can be analyzed?
- How accurate and reliable are the reports?
- How often do you require log analysis reports?
- Is it possible to access the tool via web?
- Is it possible to open the reports without any thick client?
- Is it possible to do customization on the reports?
- Can we apply filter on specific reports?

- Does the tool provide online help? Is the documentation easily understandable?
- Does the tool support a variety of web servers?
- What the users of the tool say about the tool? Is any comparative rating available which appreciates the usage of the tool?
- Is it possible to export the reports to a MS Word/Excel or PDF format?

Web Log Analyzers Summary

The following are the few of the web analysis tools which have been investigated during the study* of web analysis tools. This information has been found from various search engine searches & software developer's company sites.

* The below details are true as on February 2007.

S.No	Product	Vendor	Download page	Pricing
1.	Analog	Analog	http://www.analog.cx/download.html . Latest Version : 6.0 OS Supported : Windows, Mac	Free (Open Source)
2.	Webalizer	Mrunix	http://www.mrunix.net/webalizer/ Latest Version : 2.0.1-10 OS Supported : Solaris, Linux , Mac , OS/2 , Unix	Free (Open Source)
3.	Summary	Summary.Net	http://www.summary.net/download.html Latest Version : 3.0.2 OS Supported : Windows, Macintosh OS X , Linux X86 , Free BSD	Commercial (30 days trail version) \$59 - \$ 695
4.	WebTrends	WebTrends Corp	http://www.webtrends.com Latest Version : 8.0 OS Supported : Windows	Commercial (15 days trail version) \$500 - \$10000
5.	AWStats	AWStats	http://awstats.sourceforge.net/?seenIEPage=1#DOWNLOAD Latest Version : 6.6 OS Supported : All platforms	Free (Open Source)
6.	Affinium NetInsight	Unica	http://www.unica.com/product/product.cfm?pw=wahomeng	Commercial (15 days trail

Software Performance Testing Handbook
A Comprehensive Guide for Beginners

			Latest Version : 7.1 OS Supported : All platforms	version)
7.	AWF Full	Dee & Steve	http://www.stedee.id.au/taxonomy/term/14#downloads Latest Version : 3.7.3 OS Supported : Unix , Linux , Windows(development in progress)	Free
8.	Clicktracks Analyzer	Clicktracks	http://www.clicktracks.com/downloads/ Latest Version : 6.1.3 OS Supported : Windows	Commercial(15 days Trail version) \$495
9.	FastStats	Mach5	http://www.mach5.com/download/analyze/index.php Latest Version : 4.1.6 OS Supported : Windows	Commercial (30 days trail version) \$99 - \$199
10.	SmarterStats	Smartertools	http://www.smartertools.com/Products/SmarterStats/Free.aspx Latest Version : 3.x OS Supported : Windows	Free Edition - can be used for only one site Commercial(f or using more than 1 site) From \$199
11.	NetTracker	Unica	http://www.unica.com/product/product.cfm?pw=wahomeng Latest Version : 7.1 OS Supported : All platforms	Commercial (15 days trail version)
12.	Urichin	Uricin	http://www.google.com/analytics/urchin_downloads.html Latest Version : 5.1 OS Supported : All Platforms	Commercial (15 days trail version) \$895
13.	WebAbacus	WebAbacus	www.webabacus.co.uk/ Latest Version : 5	Commercial (Free Online Demo)
14.	SurfStats	SurfStats Software	http://www.surfstats.com/download.asp	Commercial

Software Performance Testing Handbook
A Comprehensive Guide for Beginners

			Latest Version : 8.3.0.3 OS Supported : Windows	(30 days trail version) \$95 to \$995
15.	W3perl	W3perl	http://www.w3perl.com/softs/ Latest Version : 2.97 OS Supported : Unix & Windows	Free
16.	WebLog expert	Alentum Software	http://www.weblogexpert.com/download.htm Latest Version : 3.0 OS Supported : Windows	Commercial(30 days trail version) \$74
17.	AlterWind Log Analyzer	AlterWind	http://www.alterwind.com/download.html Latest Version : 3.1 OS Supported : Windows	Commercial (30 days trail version) \$99
18.	123LogAnalyzer	ZY Computing	http://www.123logalyzer.com/download.htm Latest Version : 3.10 OS Supported : Windows	Commercial (25 days trail version) \$130
19.	Deep Log Analyzer	Deep Software	http://www.deep-software.com/download.asp Latest Version : 3.1 OS Supported : Windows	Commercial (25 days trail version) \$300
20.	EWebLog Analyzer	eSoftys	http://www.esoftys.com/download.html Latest Version : 2.30 OS Supported : Windows	Commercial (25 days trail version) \$79
21.	Sawmill	Flowerfire	http://www.sawmill.net/downloads.html Latest Version : 7.2.8 OS Supported : Windows , MacOS , UNIX	Commercial (30 days trail version) \$99
22.	Download Analyzer	Download Analyzer	http://www.downloadanalyzer.com/download.html Latest Version : 1.6 OS Supported : Windows Platform	Free
23.	AnalyseSpider	Jgsoft Associates	http://www.analysespider.com/download.html Latest Version : 3.01	Commercial (15 days trail version)

Software Performance Testing Handbook
A Comprehensive Guide for Beginners

			OS Supported : Windows Platform	\$99.95
24.	Geo Log Analyzer	Alstone Software	http://www.altstone.com/downloads.php Latest Version : 1.48 OS Supported : Windows Platforms	Commercial (30 days trail version) \$50
25.	Statomatic	Alex ivanov	http://www.statomatic.com/trial.htm Latest Version : 3.3 OS Supported : Windows , SunOS, Linux, FreeBSD	Commercial (30 days trail version) \$75-99
26.	WUsage	Boutell	http://www.boutell.com/wusage/ Latest Version : 8 OS Supported : Windows ,MacOS , Linux ,FreeBSD , Solaris, OS/2 , AIX, SunOS, etc	Commercial (30 days trail version) \$295
27.	VisitorVille	VisitorVille	http://www.visitorville.com/download.html Latest Version : 3.1.2 OS Supported : Windows Platform	Commercial (30 days trail version) \$14 - \$169
28.	Absolute Log Analyzer	Bitstrike	http://www.bitstrike.com/download.php Latest Version : 2.3.93 OS Supported : Windows Platforms	Commercial (21 days trail version) \$59 - \$259
29.	10-Strike Log-Analyzer	10-Strike software	http://www.10-strike.com/logan/ Latest Version : 1.5 OS Supported : Windows Platform	Commercial (30 days trail version) \$69.5
30.	RedWood	Bensberg, Frank et al.	http://sourceforge.net/project/downloading.php?group_id=37657&use_mirror=umn&filename=redwood.zip&97909022 OS Supported : All Platforms	Free
31.	Visitors	Salvatore Sanfilippo	http://www.hping.org/visitors/ Latest Version : OS Supported : Windows , Unix & Linux	Free
32.	HTTP-Analyze	Rent-A-Guru	http://www.http-analyze.org/download.html Latest Version : 2.5	Free

			OS Supported : Windows , Unix	
33.	FlashStats	Maximized Software	http://www.maximized.com/download/products/ Latest Version : 1.5 , FlashStats 2006 OS Supported : Windows , UNIX, Linux, Solaris , MAC OS	Commercial(30/15 days Trail Version) \$99-\$249 \$49-\$79

Web Log Analysis Metrics

The web server log file analysis provides a lot of information about the visitor behavior which helps in identifying the exact load on the system during different time periods. The historic traffic trends and deviations in expected traffic trends can be identified and compared, which helps in business decisions. A lot of people with varying background like Business owners, Domain managers, Site administrators, Performance Engineers, Capacity Planners, etc, use the log analyzer tools to arrive at server load and for user navigation trend analysis.

The following are the some of the data which could be identified from the web log analysis tool.

Number of Visitors

It represents the count of visitors accessing the site at any point of time. A visitor can make multiple visits. The visitors are identified uniquely through the IP address. By default, a visitor session is terminated when a user is inactive for more than 30 minutes. So a **unique visitor** (with unique IP address) may visit the web site twice and be reported as two visits. The tool should provide information about the visits and the unique visitors at any point of time.

Number of Hits

It represents the count of requests (hits) for any resource (example: image or html page) on a web server at any point of time. The hits are not equal to the number of pages accessed. For example, if a web page contains 5 images, a visit to this page generates 6 hits on the web server i.e. one hit for the web page and 5 hits for the image files on the html page. Identifying the peak hits per second during the peak rush hour will provide a realistic picture of the peak traffic. But most of the tools do not provide the drill down capability to find the peak hits per second data.

Number of Page Views

It represents the count of requests for the web pages (example: .jsp / .html file) on a web server at any point of time. The page views are equal to the number of pages accessed. If a visitor views

3 pages on a web site, then it generates 3 page views on the web server. Each page view might consist of multiple hits, as the page view includes the images files on an html page.

Authenticated Users Report

It represents the list of users whose user names are authenticated as required by the web site. The information including the number of hits generated by the authenticated users and the region or country from where the authenticated users have accessed the web site is also provided.

Peak Traffic Day → Peak traffic Hour

It represents the daily, weekly, monthly server load statistics and drill down capabilities to find the peak traffic day and peak traffic hour in the entire period under analysis. This is identified by referring the number of visitors accessing the site and the hits generated during the access period. Identifying the peak traffic day and peak traffic hour helps to know the maximum traffic supported by the web site.

Throughput / Bytes Transferred

It represents the total number of bytes transferred by the server to the client(s) at any point of time. It is an important metric which portrays the server performance at any point of time. The peak throughput metric helps to know the maximum data that is transferred by the server to resolve the client requests during the peak hours.

Files Downloaded Report

It represents the popularity list of all the files including the web pages, images, media files, etc downloaded by the users. Files are ranked by the number of times they are requested by the visitors (number of hits). This report includes files with all extensions.

Top Downloads Report

It represents the popular files downloaded from the web site with the extensions .zip, .exe, .tar, etc. It does not include the image files, html pages, etc. It also provides the bytes transferred information of how many total bytes of data were transferred by the web server to the visitors for each downloaded file.

HTTP Errors

It represents the HTTP response errors sent by the server during the access period. The summary of HTTP error codes and the time of occurrence is a useful metric to identify the server

behavior. The typical errors reported are page not found errors, incomplete download errors, server errors, etc. The following are the five classes of response error codes.

1XX (Informational)	: Request received, continuing process.
2XX (Success)	: The action was successfully received, understood and accepted.
3XX (Redirection)	: The client must take additional action to complete the request.
4XX (Client error)	: The request contains bad syntax or cannot be fulfilled.
5XX (Server error)	: The server failed to fulfill an apparently valid request.

Most significant Entry & Exit Pages

It represents the top entry and exit pages used by the users to login/logout to/from the web site. Some web sites have more than one option to login (For example: through registered user's login page, by Google search, by download page, etc) or to logout in different ways. The predominantly used user option can be identified by using this metric.

Browsers

It represents the information about the web browsers used by the users to connect to the server. The set of various browsers used, the user agents and the percentage usage by the user are provided.

Platforms

It represents the information about the operating system used by the users to connect to the server. The set of various platforms used and the usage percentage are provided.

Geographical regions → Top Countries → Top Cities

It represents the top geographical regions, top countries and top cities from which the users accessed the server. It provides the IP details, bytes transferred, hits generated, etc per region, country and city breakup.

Visitor Path Analysis

The visitor arrival rate can be used to identify the statistical distribution pattern (Self Similar distribution, Poisson distribution, Exponential distribution, etc) of the application. This metric is usually not provided by the log analysis tool. This is a user derived metric which needs to be calculated by the performance tester based on the mean arrival rate of the request and the peak server load details.

Visitor average session length

It represents the average visitor session duration in minutes. The session length of visitor groups is provided in a sorted order.

Search Engines and Robots

The Search engines are the information retrieval systems which allow the users to search for specific web sites based on the keyword search. The popular search engines are Google, Yahoo, MSN, ASK.com etc.

A **web crawler** (also known as a **Web spider** or **Web robot**) is a program or automated script which browses WWW in a methodical, automated manner. It is an automated Web browser which follows every link it sees.

The Log Analyzer provides the list of search engine's name and key phrase that referred visitor to the website. The **Top Search Engines Report** is a list of search engines used by the visitors to find out the website, ranked by the number of referrals from each search engine. The **Referring Sites Report** shows the referrer websites that drive visitors to your site ranked by the number of hits received from that referrer. The **Spider History Report** illustrates day-by-day history of search spiders visits.

Some of the metrics like peak number of unique visitors in a high traffic day, average session length of a visitor, maximum hits reported per sec during the peak hour of the peak day, maximum bytes transferred per sec during the peak hour of the peak day, maximum page views per sec during the peak hour of the peak day needs to be derived (if not provided directly in the log analysis tool) as the performance test goals needs to be set based on these parameters.

Log Analysis Tools	Star Ratings	Price
WebTrends	*****	\$500 - \$10000
Deep Log Analyzer	*****	\$300
Absolute Log Analyzer	*****	\$59 - \$259
EWebLog Analyzer	****	\$79
AWStats	****	Free
SurfStats	****	\$95 to \$995
FastStats	****	\$99 - \$199
Summary	****	\$59-\$695
Statomatic	****	\$75-99

Clicktracks, Smarter stats, Urichin, Analog, AwfFull,123LogAnalyzer, Sawmill, Alterwind, FlashStats	***	\$495, \$199, \$895,Free,Free, \$130, \$99,\$99 , \$49- \$79 (respectively)
--	-----	---

Web Site Traffic Monitoring

The web server monitoring helps to monitor the web server's traffic, trends, quality, value, and stability of internet, intranet, and extranet web sites, proxy servers, services and IP devices. Also it helps in real time log file analysis. One such popular tool is WebTrends. Through WebTrends monitoring of the web site, we could have a detailed real-time insight about what is happening on the server during real time user access. The details like whether the web server is fully functional at a glance, finding out problems in the website before the visitors do, to know the network traffic in real-time makes it more useful. In general, it allows the organization to easily track Key Performance Indicators (KPI) that is critical for measuring the success of the online business.

Overview of Statistical distributions

The Performance Test Engineer needs to identify the underlying statistical distribution of the application's user request arrival pattern. This helps to extrapolate the server load (requests handled per unit time) for high user load. For example, if two applications (A1 and A2) have the similar target load objective (say 1000 users) but they fall into different statistical distribution (say A1 – Poisson and A2 – Self similar), then the applications A1 and A2 needs to be performance tested for different maximum server loads in order to certify the applications for 99.9% availability. The maximum load point differs based on the type of the distribution which is very vital to identify the system performance for that maximum load point. The statistical distribution pattern analysis helps in deciding appropriate load requirement during performance tests.

The probability distributions are a fundamental concept in statistics. Discrete probability functions are referred to as probability mass functions and continuous probability functions are referred to as probability density functions. The term probability functions cover both discrete and continuous distributions. When we are referring to probability functions in generic terms, we may use the term probability density functions (PDFs) to mean both discrete and continuous probability functions.

The arrival pattern analysis needs to be studied in order to understand the worst case peak load situation and to simulate the calculated maximum load while conducting the performance tests.

Many probability distributions are not a single distribution, but are in fact a family of distributions. This is due to the distribution having one or more shape parameters.

The shape parameters allow a distribution to take on a variety of shapes, depending on the value of the shape parameter. These distributions are particularly useful in modeling applications since they are flexible enough to model a variety of data sets.

A probability distribution is characterized by location and scale parameters. Location and scale parameters are typically used in modeling applications.

For example, the following graph is the probability density function for the standard normal distribution, which has the location parameter equal to zero (vertical center) and scale parameter equal to one.

The effect of a scale parameter greater than one is to stretch the PDF. The greater the magnitude, the greater is the stretching. The effect of a scale parameter less than one is to compress the PDF.

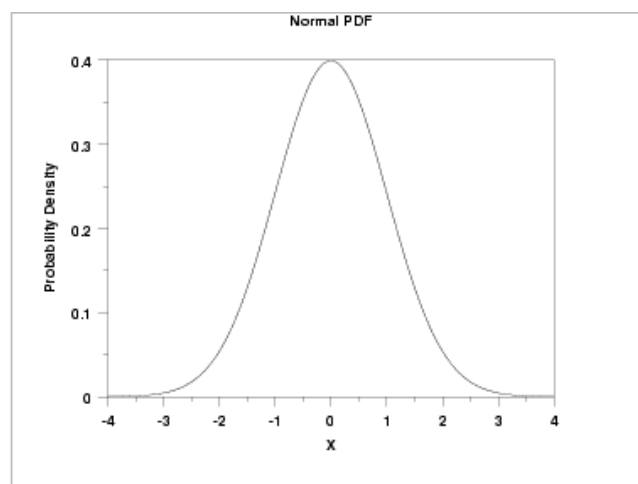


Figure 10: PDF of Normal Distribution

One common application of probability distributions is modeling data with a specific probability distribution. This involves the following two steps:

1. Determination of the "best-fitting" distribution.
2. Estimation of the parameters (shape, location, and scale parameters) for that distribution.

Most of the usage pattern of a client-server or web applications falls into any of the following distributions discussed below. A Performance Test Engineer needs to know the shape of the important statistical distributions which are very prevalent for web systems which helps to decide the worst case peak load on the server.

Uniform Distribution

The general formula for the probability density function of the uniform distribution is

$$f(x) = \frac{1}{b-a} \quad \text{for } a < x < b$$

where A is the location parameter and $(b - a)$ is the scale parameter. The case where $a = 0$ and $b = 1$ is called the **standard uniform distribution**.

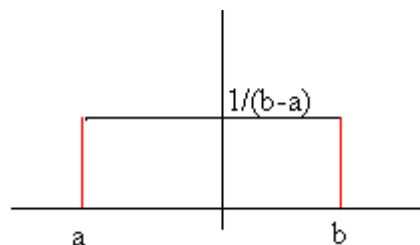


Figure 11: Uniform Distribution

Normal Distribution

The general formula for the probability density function of the normal distribution is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the location parameter and σ is the scale parameter. The case where $\mu = 0$ and $\sigma = 1$ is called the **standard normal distribution**. The parameters of the distribution are μ and σ^2 , where μ is the mean of the distribution and σ^2 is the variance.

The normal distribution is symmetrical about its mean:

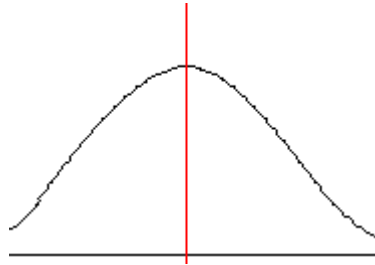


Figure 12: Normal Distribution

Poisson Distribution

The Poisson distribution is used to model the number of events occurring within a given time interval.

The formula for the Poisson probability distribution function is

$$P(X = x) = \frac{(e^{-\lambda}) \lambda^x}{x!}$$

where λ is the shape parameter which indicates the average number of events in the given time interval.

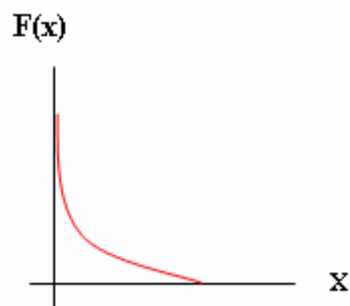


Figure 13: Poisson Distribution

Exponential Distribution

The formula for Probability density function (PDF) is given by $P(x)$, where:

$$P(x) = \lambda e^{-\lambda x}$$

where $x \geq 0$. Mean & Variance is given as

$$\mu = \frac{1}{\lambda} \quad \sigma^2 = \frac{1}{\lambda^2}$$

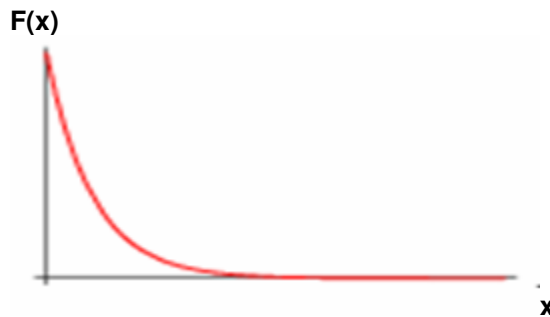


Figure 14: Exponential Distribution

Heavy-Tailed (Pareto) Distribution

Unlike the exponential distribution, the heavy-tailed distribution (also called Pareto distribution) follows power law decay. The PDF is given by

$P(X > x) \sim x^{-a}$, where $1 < a < 2$, with a maximum likelihood estimator (MLE) value for the mean is

$$\mu = \frac{a}{a-1} \quad \sigma^2 = \frac{a}{(a-1)^2(a-2)}$$

$P(x)$

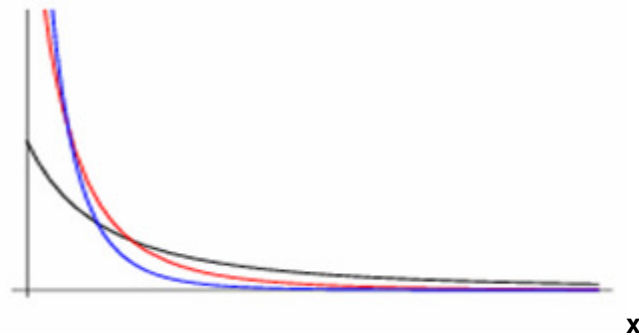


Figure 15: Heavy Tailed Distribution

Workload Modelling Approach

This approach provides the sequence of steps that needs to be followed for the workload modeling of the ebusiness application.

- Baseline your objective for developing the workload model.
- Understand the web site's workload pattern
- Analyze the workload trace & Identify the request arrival pattern
- Identify the underlying statistical distribution (which is mostly Poisson or Self-Similar) and conduct tests to confirm.
- Gather hardware resource consumption metrics
- Construct the model

Baseline the objective for developing the workload model

The basic objective of building the workload model needs to be baselined. The workload characterization required for the performance tuning activity would be different from the workload characterization required for the capacity planning activity. For example, if we are interested in identifying the web server workload in order to validate whether the system can handle business forecast of 10% user increase by end of the year, then the workload needs to be characterized to measure the number of clients, number of transactions handled per second, CPU and IO utilization for the current load, request arrival rate and the user think time. If the objective of the

workload model is to provide the response time estimates for the business forecast of 20% user load increase, then the workload needs to be characterized with different set of parameters. A slight tweaking in the previous parameters needs to be done.

Understand the web site's workload pattern

Unless we understand the web site's real workload pattern, the workload model cannot be developed. Understand the business solution provided by the web site and study the types of customers who can access the application. Identify the objective of the end users and the total user base of the application, if applicable. Look for the situations where the user load on the system would increase. Also, look for situations where there are possibilities that the user access pattern would differ considerably. Based on discussion with the business groups or end users of the application and/or put yourself into the end users of the application to collect metrics like frequently accessed pages, expected end user response time, business peak and off hours, possibility of any user projections expected, etc.

Analyze the workload trace & identify the request arrival pattern

As the web site actual workload is analyzed, it will be easier to interpret the workload trace of the application. Start with the user perceived metrics, move towards the functional details and then end up in identifying the hardware resource consumption. The workload trace file from the web server would be right start to gather the inputs for building the workload model. The Log files would have the information about each of the HTTP requests received from the client including the time of the request along with the Client IP details and response HTTP code sent by the server. Analyzing this log files provide the load details. Based on your business, collect relevant user load metrics like

- ✓ The user load during the weekdays along with the peak hour traffic details
- ✓ The user load during the weekends along with the peak hour traffic details
- ✓ Specific day (say, there is a new product is released in an auction site) wherein the user load is high.
- ✓ Also look for the errors reported and session duration information.

After identifying user load pattern, start looking for the functional details of the web site like frequently accessed urls, files, images and downloads. Sometimes correlate the business necessity with the accessed pages data available from these logs in order to make necessary assumptions. Arrive at the correct transaction mix of transactions expected during the peak hour

traffic. This transaction mix could be a small subset of various functional flows available in the system. The server resource utilization data needs to be identified by subjecting the system to the above derived load. In some cases, if the production monitoring is enabled using market available tools, we can get the utilization data.

Analyze the user requests arrival pattern. Watch the inter-arrival time between the user requests. Based on the business understanding, identify the representative log file sample (which falls into peak traffic hour category) for further analysis. From the representative log, plot the users requests arrived per second value as Y-axis against the time variable as X-axis similar to the samples provided in the below figure.

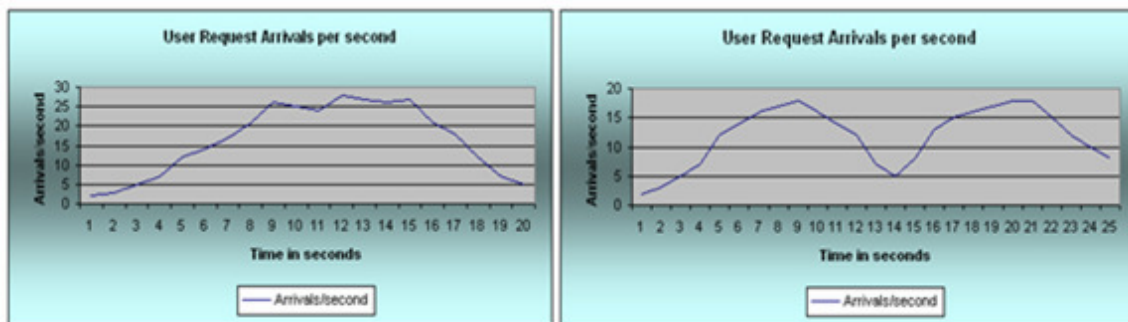


Figure 16: User Request Arrival Pattern Graph

Identify the underlying statistical distribution (which is mostly Poisson or Self-Similar) and conduct tests to confirm

From the representative log data, say if the data collected is from the peak hour traffic, then calculate the total user requests/hour and identify the maximum arrivals/sec value. Calculate the mean arrival rate for the peak hour traffic data. By analyzing the mean and maximum arrival rate during the peak hour, identify the distribution of the requests. It is not advisable to just go by the mean arrival rate value to conduct performance tests. It is important to know the underlying statistical distribution of the requests arrival pattern as the maximum value (99th percentile concurrency) would have a huge variation as shown in the below figure.

Arrival Pattern	99 percentile concurrency
Uniform	10
Normal	17
Poisson	22
Pareto	315
Self-Similar	363

Figure 17: Arrival Pattern Vs 99th Percentile Value

The Open systems (Internet facing applications), would mostly fit into Poisson or Self-similar category of the distributions. Understand the basic difference in these two distributions. If the distribution is Poisson, it can't be self-similar and vice-versa. In Poisson, the inter-arrival time between the requests is always exponential. There is no dependence between the samples of the specific time intervals. In Self-similar, the samples are not independent –they are correlated to each other. It seems to have long range dependence. Perform an Aggregation for varying time scales to check whether the burstiness gets cancels out. In Poisson, the burstiness is not noticed for higher scales, but in Self-Similar distribution, the burstiness could be visible & similar because of its dependence with other samples. Confirm the occurrence of underlying statistical distribution by conducting Goodness Of Fit (GOF) tests using the Dataplot statistical software, which is a free, downloadable statistical software package maintained by the National Institute of Standards and Technology. Conduct these tests to compare 2 distributions & confirm whether the data sample we have & the model are one and the same. The Q-Q plot & Kolmogorov-Smirnov tests can be used to confirm the occurrence of the standard distributions. To check whether the process is Self-Similar distribution, conduct Rescaled Range method or Time-variance plot.

Q-Q plots are a simple graphical means to compare distributions by finding the percentiles of the two distributions, and plot one set as a function of the other set. If the distributions match, the percentiles should come out at the same distances, leading to a straight line with slope 1.

The Kolmogorov-Smirnov test is based on calculating the *maximal* distance between the Cumulative distribution functions of the theoretical distribution and the sample's empirical distribution, over all the samples.

The Rescaled Range method is based on calculating the Hurst value. If the Hurst value lies between, $0.5 < \text{Hurst value} < 1$, then it is a self-similar pattern. It measures the range covered after n steps, and check the exponent that relates it to n.

The Time variance plot method is based on calculating the rate in which the variance decays as observations are aggregated.

Gather hardware resource consumption metrics

Based on the transaction mix we identified from the logs, subject the system to a low load of the same transaction mix and identify the service demand of CPU and I/O subsystems. The service demand is not a monitored metric. Based on the Utilization law, $U = X.S$, by monitoring the system throughput (in terms of completions/second) and % utilization, the Service Demand of each of the subsystems can be calculated.

Construct the model

Having identified the transaction mix and the hardware resource requirement, decide on the abstraction level based on the accuracy you are interested in. Model abstraction depends on the accuracy we are looking in. If we need more accuracy, the partition the collected data into multiple classes based on resource usage or geographic distribution or query performance, etc similar to the table provided below.

Transactions	Frequency	CPU Time (in sec)	I/O Time (in sec)
Light	10%	6	120
Medium	70%	28	260
Heavy	20%	350	900

Figure 18: Workload Partitioning into multiple classes

The Forgotten Facts

The following facts are most of the time forgotten while modeling the application workload.

1. The statistical distribution fitting of the application's user requests arrival pattern is often missed. The user load details along with the frequently accessed pages information from the log file analysis are considered to be enough by mistake.
2. A thorough analysis to confirm the occurrence of the statistical distribution is missed most of the time. The usage of GOF tests or Variance Time plot test is forgotten for most of the time.

3. Certain burstiness needs to be considered as Outliers & ignored. Aggregation technique needs to be done for different time scales in order to confirm whether the burstiness gets canceled out.
4. Based on the expected accuracy of the model, the Workload needs to be classified further into multiple classes based on the application context for more accuracy. Most of the time, classes are formed with respect to the total number of requests per geographical distribution, Size of the files accessed, clustering the functional components based on their service requirements, clustering the functional components based on the query response time, etc.

Chapter 5: Overview of Performance Testing Tools

Performance Test Tool Requirements

There are several important requirements that need to be analyzed before procuring the performance test tool. Daniel Menasce provides a four layered hierarchical framework for the e-business application.

The top level is the *Business Level* - This level describes the business type such as B2B, B2C, or C2C and the product type such as physical goods, digital goods, or services. At this level, load-testing tools are required to understand how business decisions affect the IT infrastructure.

The second level is the *Functional Level* – This level deals with e-business functions that implement the site's business model. The e-business functions might include browse, search, select, and add to shopping cart. At this level, load testing tools are required to perform load test on applications supported by many technologies like Flash, JavaScript, ActiveX, Cookies and SSL.

The third level is the *Customer Behavior Model* – This level deals with users' navigation patterns through a site. Customers interact with web sites through sessions, which are basically sequence of consecutive requests that form a navigational pattern that you can capture in graphs, such as the customer behavior model graph (CBMG). At this level, tools are required to flexibly and easily develop the test scripts represented in the CBMG model.

The fourth level is the *model of IT resources* – This level is required to support site activities, such as processors, storage devices, networks and software components. At this level, tools to conduct various types of test (including Load test, Stress test, Volume test, etc) on the web application and to identify the performance problems.

Market Tools

There are lot of market wide, popular open source and licensed tools available to carryout performance testing. There are hardly few tools available in the market to carryout capacity planning, workload modeling and performance modeling. Also, these tools are not universally accepted ones. Usually, these kinds of tools are built in-house with the help of subject matter experts in certain organizations. There are few organizations available where they provide consultancy and support for capacity planning and performance management of systems. These organizations have come up with lot of excel based tools to support workload modeling, performance modeling and capacity planning activities.

Open Vs Licensed Performance test tools

There are lot of open source and licensed tools available in the market to do performance testing. Usually for small organizations where there are budget constraints, open source tools would be the only choice. There are good open source performance test tools like JMeter, OpenSTA available in the market. One should be very careful in using custom build or open source tools as the tool implementation logic including thread handling, inter-arrival time between the requests spawned, etc would have a huge impact in the way performance tests are conducted.

When it comes to organizations which have performance critical applications, due to expectation on high accuracy and reliability on the test results, they prefer using licensed tools in the market like HP Load Runner, HP Performance Center, Segue's SilkPerformer, Empirix E-Test, Radview Webload, Compuware QALoad, etc.

Criteria for choosing the Performance test tool

There are certain things which need to be analyzed before choosing the performance test tool for your application. Look for the following before you choose the tool

1. Look for the protocols supported by the test tool. Does the tool support web applications and client-server applications?
2. Check whether the tool provides the editor to develop the test scripts.
3. Check whether the tool provides features to identify the transactions uniquely, to provide user think time, to provide the test data, to handle server side dynamic variables and to see the script replay with the application pages.

4. Check whether the tool has options to configure the ramp up, ramp down and steady state load duration for running the test.
5. Check whether there are provisions to monitor the system resources during the test.
6. Check whether monitors could be used across different operating systems.
7. Check whether the tool provides the test results in a readable format (which can be opened as html or word format)
8. Check whether the tool provides enough information about the Hits/sec, Throughput (in transactions/sec and bytes/sec), Running users/sec, System resource monitors and errors/sec details.
9. The tool should not consume much of the resources of the client machine resulting in load generation issues.

The most important tool

Choosing the correct automation test tool is not the challenge in conducting the performance testing. The performance test tool could only give the details on the server performance, but it's the Performance Tester who needs to interpret the observations clearly and analyze the root cause of the issues, confirm the bottlenecks by rerunning the tests, provide the report of observations and recommendations in user understandable format. The Performance tester's brain is the most important tool to make the performance testing a success.

The Performance tester should be strong in the concepts rather than just being a performance test tool expert. Though expertise in the test tool is an added advantage, the tester should possess right set of capabilities to pinpoint the performance issues of the application.

Performance Testing Tools Overview

Some of the popular performance test tools include

- HP's Load Runner / Performance Center (licensed)
- Radview's Webload (licensed)
- Compuware's QALoad (licensed)
- Borland's SilkPerformer (licensed)
- Empirix's e-Load (licensed)

- OpenSTA (freeware)
- JMeter (freeware)
- Grinder (freeware)

HP Loadrunner

It supports a variety of protocols including HTTP/HTTPS, SAP, PeopleSoft, Citrix, Oracle Apps, RTE, Winsock, COM/DCOM, SOAP, Tuxedo, etc. Scripting is very easy and it uses TSL language for creating scripts which has the syntax of C and allows addition of C libraries. Scripting is icon based and does not require any programming background to generate scripts, thus making it ideal for testing community. For load generation, it provides features to control multiple load agents/generators and collect results. It provides monitors for various web servers, application and database servers with a very attractive UI for easy use of monitors. It provides many graphs like Transactions per second, Hits per second, server resource utilization, performance under load, server throughput, etc and also provides features to customize the graphs and export it to html or word formats. It also provides an analysis tool with the dynamic graph creation. It supports advanced inbuilt features like IP spoofing, WAN/LAN emulation, auto-correlation, browser cache management and network bandwidth emulation. Though it is the very popular tool in the market with wide audience, the very high cost of the tool creates a barrier for using the tool in all type of organizations. The limitation of this tool is on the support for monitoring a non-windows environment. Vendor provides good amount of support services but the cost is very high.

OpenSTA

It is a free open source tool which supports only HTTP/HTTPS protocols. It uses SCL for generating the scripts which is a proprietary BASIC like language and it supports DOM addressing. It facilitates real time monitoring using Windows NT/2000 perfmon SMTP collectors. For load generation, it provides features to control multiple load generators and collect results on single controller. It does not provide inbuilt support for IP Spoofing, auto-correlation, WAN/LAN emulation and network bandwidth emulation. Inbuilt result analysis is very limited and it provides simple few charts and graphs which can be exported to MS excel. The user needs to use macros and external sources to create customized graphs. For support and consultancy various independent sources are available at affordable cost.

JMeter

It is written in Java and works well on Windows and Linux/Solaris. It supports HTTP, FTP, JNDI and JDBC, though HTTP is by far the most mature part of JMeter. It provides easy UI based script development features and script management features but the stability of the tool is the big limitation in JMeter. It has no built in monitoring support. Thus, wrapper scripts are required to synchronize test data with external performance monitoring data. It provides very limited transaction monitors. For load generation, it provides features to run tests for any number of users and controls multiple load agents/generators. JMeter does not gather any server-side performance metrics. But it can generate a limited number of client-side graphs while the test is in progress. No features are provided for test result analysis and the log files needs to be manually exported and analyzed to identify the system performance. Features like IP Spoofing (alternative feature provided but still not god enough), WAN/LAN emulation and network bandwidth emulation are not supported. For support, discussion forums and documentations are available.

Grinder

It is a free tool developed based on Java, Swing and Jython which supports only HTTP protocol. It uses Jython language for creating the scripts and it provides good script management features. This tool is mainly target for developers as it needs more understanding of the object oriented programming, hence scripting needs considerable amount of programming skills. It has no built in monitoring support. Thus, wrapper scripts are required to synchronize test data with external performance monitoring data. It provides very limited transaction monitors. For load generation, it provides features to run tests for any number of users, but the scripts needs to be manually deployed each load agent. No features are provided for test result analysis. It offers lots of flexibility for loading and executing third party libraries. There is no test scheduling feature or command line execution feature available in the tool and it does not support IP Spoofing, WAN/LAN emulation and network bandwidth emulation. The tool does not provide options to dynamically increase or decrease the user load while the test is running.

Chapter 6: Setting up Performance Test Environment and Performance Test Scripts development best practices

Know the Test and Production environment

The actual target production environment of the system under test needs to be studied by the Performance Test Engineer. The performance test strategy should contain the details about the target system environment on production. The Performance Test Engineer should know the deployment architecture of the application in the target production environment and must emphasize the development team to setup a similar test environment to run the performance tests. There could be a huge difference in the system performance on test and production environment as both environment uses different hardware platform to deploy the application. For example, if a three tier application deployed in a test environment gets deployed using two tier architecture in the production environment, then the system performance would have a major difference in both the environments.

Always identify the system configuration details of the server machines in the target environment like the number of CPUs, CPU capacity (clock speed), RAM capacity, disk capacity, free space available in the disk, NIC card capacity and network bandwidth. These details needs to be identified before scheduling the performance test and should be documented in the test plan document for future reference.

Test environment Isolation

If the performance test bed is setup locally, then the test bed should be isolated from the other test beds used for functional testing, UAT (user acceptance testing), etc. The Performance test should be conducted in an isolated environment, so that it is easy to isolate the bottleneck and avoid unnecessary extra load during the performance test. Also there is a high probability that servers could create major failures during load conditions which might affect other activities if planned on the same environment.

Network Isolation

Isolating the network used for the performance test is very important if the objective of the test is to identify the bandwidth requirement to support the target users. In that case, if the network is heavily loaded with other traffic, then it's obvious that amount of data sent to/from client and server would have an impact. Hence, in this case the performance test should be conducted in a separate LAN wherein all the server and test machines (used for client traffic simulation) are located within the same dedicated LAN.

Load Generators

The load testing tools are normally used to create the required load on the server. This can be done using the Load Generators. When the performance test tool is configured to simulate 10 Virtual users, then the tool generates 10 threads or process (depending on the configuration setting of the tool) which sends the client requests to the server as per the recorded think time intervals available in the test script. The system should have enough hardware resources (CPU and Memory) to handle the running threads. The number of virtual users to be spawned from a machine depends on the hardware capability of the machine. If a low configuration desktop is used for load generation, then not more than 15-20 virtual users should be spawned. If a high end server is used for load generation, then about 1000 virtual users could also be spawned as long as enough hardware resources are available in the system. In case of testing of SAP applications, the virtual users that can be spawned from an individual machine becomes very limited.

Most of the load testing tool provides a utility called Load Generator, which can be installed in any other machine other than the one where the test is scheduled. A Load Generator can create the configured number of users load on the machine where the utility is installed and the test script gets auto loaded and run for specified number of times during the test. For example, in order to run a 1000 users test, say load generators can be installed on 5 machines and can be configured to create 200 virtual users load per machine. The user load created on 5 different IPs is sent to the server, as the load generators are configured to run the test scripts that are allocated in the scheduler (controller), from where the test is run. From the test scheduler, the load generated by each agent could be viewed separately or collated (depending upon features supported by the test tool).

In order to simulate the load generation from a large number of IP addresses, the performance test tool provides a feature called IP Spoofing. Using this feature, the load generated from a load

generator can be spoofed as though the load is created from a list of IP addresses. This would be really useful while conducting any tests on a load balanced environment. If the load balancer is configured to have stickiness based on client IP (the load generated from an IP address will be always routed to 1 of the load balanced server which handles the first request), then it would be required to use the IP spoofing in order to conduct performance tests on the load balanced environment. If not, the entire user load created from the load generator machine will be sent to one of the servers in the load balanced environment, thus creating an unrealistic load simulation. If the load balancer's routing algorithm is round robin based on sticky sessions, then running load test with single load generator will not create any issues.

Test Data Generators

During the performance tests, most of the time the local test bed might not have the required database records as in the target production database. The target production database volume or projected data volume needs to be studied and accordingly the local test bed needs to be populated with the required number of records. Depending upon the complexity of the test data, it could be done either by the Performance Test Engineer or DBA. Many tools are available in the market in order to generate the test data. Alternatively, database dump taken from the target production system can be used to create the required database records in the local test bed. But in cases where there are security concerns about using the production data in the test environment, depersonalized version of production data can be used if the application is able to handle the depersonalized data without any issues. Conducting the performance test with appropriate database volume is very essential. The query execution time of a SQL could be very less when the database records are around 1000, but when the database volume becomes 10000 records, the SQL execution time could definitely have an impact to fetch all the records and perform the processing as per the business logic. There is a high possibility that some database issues would be hidden if the application is not tested for the actual expected database volume. The way the records are arranged in the table and indexed might create a huge performance impact on the system.

Test Execution Environment

Most of the time, performance testing is conducted on a test environment which is of lower configuration than production server configuration. Though it is always recommended to conduct performance tests on the environment which is similar to production server configuration, it is practically impossible to setup a similar environment for conducting tests due to financial reasons.

The Performance Test Engineer needs to understand the deployment architecture of the application and ensure correct software versions (application version, server version, operating system, internal libraries if any, etc) are used for deployment in the test environment which is similar to the production environment. Though Performance Test Engineer is not directly responsible for performing these activities, performance tests should be conducted only after receiving the confirmation from deployment team that the software used in the test environment resembles the production environment. The Software (web server or application server or database server software version, license conditions, log levels, other processes, etc), Hardware (CPU, Memory and Disk configurations), Network (traffic and network bandwidth details) difference on both test and production environment needs to be identified and documented for future references.

If performance engineering techniques to map the test environment performance to the production environment performance is not practiced, stakeholder needs to be made aware of the differences in the test and production environment and the risks involved in the performance assumptions on production environment. The system performance needs to be benchmarked and certified for the test environment configuration and stakeholder needs to be informed about the differences in environment capacities and the possible differences expected on the system performance.

How to choose the test scenarios

Often Performance testing is compared with the functional testing approach. Unlike functional testing, where the interest is in testing the system for all possible positive and negative flows, in performance testing the interest is to identify only few specific scenarios which are more frequently used by the end users. Improving the performance of these selected scenarios will bring a major performance impact on the system.

Always performance test scenarios selection needs a pareto analysis wherein only 20% of the scenarios which are most frequently used in the application need to be tuned for high performance. Improving the performance of the remaining 80% scenarios will not bring considerable performance improvement. Along with the identified scenarios, scenarios of stakeholder's concern and scenarios of technical concern or error prone scenarios can also be used.

The user workload modeling languages like UCML or CBMG can be used to visually present the usage pattern of the site and tests needs to be conducted based on the inputs from the server workload model.

Tips for writing successful performance test scripts

The Test scripts are generated to mimic the user behavior during the performance tests. The Test scripts needs to be modeled in such a way that it exactly simulates the user actions and user think times. The scripts are generated by record and playback feature provided by the test tool and a series of below customizations needs to be done to make the scripts more realistic.

- It is recommended to script or configure (depending upon the feature provided by the tool) in such a way that login and logout transactions are executed only once and rest of the transactions. Incase of Load Runner tool, always place the login transaction inside the vuser_init() and logout transaction inside vuser_end(). This helps to achieve more realistic usage pattern and also helps to close the user session (by enforcing execution of logout transaction) even if the script is stopped in the middle of execution.
- Provide test data variables in the script to enable the script to supply different combinations of the data for different users as in real time.
- Configure appropriate test data usage options (Unique or Random or Sequential).
- Provide appropriate naming convention for each user action to uniquely identify the response time metric for all the transactions of business interest.
- Provide wait times between each of the user action in order to simulate the real time end user behavior. The wait time or think time refers to the user time for thinking or navigation time.
- Handle the unique dynamic values generated by the server i.e. session values, flow id, descriptors, etc to make the script robust enough for simulating multiple users.
- Provide customization in the script navigation logic to handle the user behavior.
- Add logic to verify the correctness of the page responses.

The Test scripts should have the transaction names for those requests for which the response time and throughput metrics needs to be collected and reported. These requests should be

named with proper naming convention so that it becomes easy to refer to a specific transaction by its unique name.

Running the script containing the right set of transaction mix alone doesn't mimic the real time user behavior. Each transaction should be always coupled with another important metric called think time, which is the time taken by the user to think and to navigate between various pages. This wait time which is placed between each transaction determines the accuracy of the scripts simulating the user behavior. Analyzing the think time between each transaction is very critical as it determines the load on the server at any point of time. By conducting discussions or surveys with business analysts and end users of the application, the think time for each transaction can be identified. For example, assume you have 2 transactions in your script, Login transaction followed by the think time of about 10 seconds followed by the Logout transaction. If the think time is changed to 5 seconds, then the frequency in which login and logout requests are sent to the server increases, hence the load (requests/sec) on the server is high. If the think time is changed to 20 seconds, then the server load (requests/sec) decreases as the login and logout requests are sent in 20 seconds interval. Therefore, providing high think times decreases the load on the server whereas reducing the think time increases the server load. This is a very important fact to be known by the Performance Test Engineer. For a similar user load, say for example 100 users, the server load (requests per second) can be increased or reduced depending upon the think time configuration. If the think time settings are not correct then there is a high probability that the server load created during the performance test can go wrong though the test is conducted for the required number of users.

The Think times should be studied for each of the transaction based on the end user surveys or from log file analysis. The think times should be randomized with the tolerance limit of 20-25% to accommodate fast and slow user. Also care should be taken while designing the test scenario that the business actions are performed for appropriate number of times as expected in the real time usage.

The test data parameterization is another important activity which needs to be taken care in the scripts. Care should be taken so that enough test data is provided in the script so that various combinations of test data are used during the test which is likely to happen during real time access. For example, the user login credentials needs to be parameterized in the script. If the test is planned to run for 100 users, then the script should have 100 login credentials available in the script. If more than 1 user is allowed to use the same test data, then the response time metrics would not be appropriate as the application might be caching the web pages or the database

query and there is a probability that the response times are not realistic. It is always recommended to make sure that the script is supplied with enough test data to be used uniquely for each user login. By using parameterization, we can make sure that the correct formatted unique data is sent from the client-side for each user during the course of the test run.

From the server side, there might be some dynamic data generated for each user which may be unique for each user. For example the session id of each user would be unique. If the server provides the session id of the user while returning the response for a specific client side request, then this part of the script cannot replay correctly if it is run for a different user id hence it should be handled in the script. These dynamic values can be handled in the script by identifying the corresponding left and right boundaries to capture the dynamic value, which is then substituted against the variable which is placed in the place of the dynamic value. This process is called as correlation.

Some performance test tools provide auto correlation features to identify and handle these dynamic values. If the tool does not provide a built-in feature, then the dynamic values generated by the server needs to be identified manually and handled in the script. Sometimes the recorded script cannot replay because of some specific dynamic values which are generated by the server. This kind of dynamic values needs to be first studied whether it is unique with respect to each logged in user or unique for each run even for the same user. Sometimes the user action gets completed even if the dynamic values are not handled. Hence, firstly the logic of why and where the dynamic value is getting generated from the server end needs to be analyzed.

Depending upon the business constraints, the test data usage in the script needs to be planned accordingly. In some cases, the test data usage should be unique wherein each test data could be used only once during the test. In some cases, the test data needs to be sequentially used wherein each test data used in a sequential manner. If the test tool does not provide sufficient features to configure the test data usage, then the test data selection logic should be programmed in the test script.

Last but not least, make sure the script is very realistic in terms of the user actions. Try to have a single script which covers all the flows identified for the application and use random number generator function to distribute the user load across various functional flows. Usage of randomization function to distribute the user load between the business scenarios scripted as functions in the same single script aids in achieving appropriate throughput mix which cannot be

guaranteed if multiple individual scripts are used with user distributed across various business scenarios scripted as individual scripts.

Real time versus Virtual user Mapping

Though Performance testing is conducted for the target number of user load, it's Performance Test Engineer's responsibility to assure that each simulated virtual user is equivalent to the real time user, only then the load generated on the server could be realistic. For applications which have the real time usage history (moving to production environment for second time or later), the server load (requests handled per second) during the peak traffic hour needs to be compared with the server load created on the server during the performance test. This helps to confirm the load simulated during performance test is realistic. The think times provided in the test scripts needs to be adjusted so as to create the appropriate load on the server. Analyzing the server load in terms of requests per second is recommended as it is more granular and appropriate unit to measure the load handled by the server rather than the user load measured in number of users. The number of users handled by the server is a user perceived metric which needs to be reported to the business stakeholders, whereas Performance Test Engineer's focus should be on the number of requests handled by the server.

Chapter 7: Application Benchmarking

What is benchmarking?

In general, Benchmarking is the process of determining who is very best, who sets the standard, and what that standard is. When it comes to the Performance Testing context, most of the time we often use the word benchmark. Benchmarking is the process of the determining the relative performance of the software program by running the standard set of tests. We can benchmark the hardware or software performance by comparing its relative performance figures. For example, we can benchmark the software application across various application servers by deploying it in Websphere server, Weblogic server and JBoss server to compare the performance of the application in each server. We can benchmark the hardware requirement for the software by running the tests on Dell server, HP Proliant server and IBM servers to compare the CPU, Disk and Memory utilization details.

Why should we benchmark the applications?

Unless we compare ourselves with others against a common measure (for example say height) which is a measure applicable for both entities, we cannot say who is best. It becomes very tough if we don't know what is the standard measure that can be used to compare oneself against the other. The same is applicable for the software systems. One always needs to know where their competitors stand. In order to compare different software or hardware performance, we need to have the common standard measurements.

Software Benchmarking

Mostly during the architecture and design phase, software benchmarking activities might occur. In order to baseline the application architecture, one might need to identify whether the application running in JBoss server is better or the application running on Websphere server is better.

Performance tests carried out during this phase would not have the entire application in place, but it would have the prototype or the POC (Proof of Concept) ready. Testing the POC and benchmarking the system performance for specific software is the cost effective solution.

Hardware Benchmarking

In order to benchmark the hardware requirement of the application, you might need to identify whether IBM machine with dual core CPU or IBM machine with quad CPU is required or sometimes you might need to know whether to go for IBM servers or HP servers or Dell servers for achieving best performance in a cost effective way.

Industry Standards for Benchmarking

It becomes impossible for the application owners to test the application performance against variety of the server machines available in the market and to choose among them due to cost factor. There are organizations available in the market which does this benchmarking. The application owners could refer to these industrial benchmarks to decide on their infrastructure requirements. Transactions Processing Performance Council (TPC), Standard Performance Evaluation Council (SPEC), Synchromesh benchmarks, etc are the industry standard benchmarks available. There are other open source and vendor specific benchmarks available in the market. These organizations perform the testing on different servers with varied hardware configurations and provide the performance figures. As we all know we need to have some common measure to compare with the competitors, these industry standards provide the list of measures which is common across server platforms.

Transaction processing Performance Council (TPC)

The TPC is a non-profit corporation founded in 1980's to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. The TPC benchmarks are widely used today in evaluating the performance of computer systems.

The TPC benchmarks involve the measurement and evaluation of computer functions and operations through transaction as it is commonly understood in the business world. A typical transaction, as defined by the TPC, would include updating to a database system, a set of operations including disk read/writes, operating system calls, or some form of data transfer from

one subsystem to another. There are different types of benchmarks available in TPC. It includes TPC-App, TPC-C, TPC-E and TPC-H. The following information is taken from the TPC site - <http://www.tpc.org/>.

TPC-App is an application server and web services benchmark. TPC-App showcases the performance capabilities of application server systems. It measures performance using Service Invocation per second (SIPS), SIPS per application server, the associated price per SIPS and the availability date of the priced configuration.

TPC-C simulates a complete computing environment where a population of users executes transactions against a database. It involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. The benchmark is centered on the principal activities (transactions) of an order-entry environment. TPC-C performance is measured in new-order transactions per minute. The primary metrics are the transaction rate (tpmC), the associated price per transaction (\$/tpmC), and the availability date of the priced configuration.

TPC-E is a new On-Line Transaction Processing (OLTP) workload developed by the TPC. The benchmark defines the required mix of transactions the benchmark must maintain. The TPC-E metric is given in transactions per second (tps). It specifically refers to the number of Trade-Result transactions the server can sustain over a period of time.

TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. The TPC-H Price/Performance metric is expressed as \$/QphH@Size.

Standard Performance Evaluation Council

The **SPEC** is a non-profit organization that aims to produce fair, impartial and meaningful benchmarks for the computers. SPEC was founded in 1988 and the goal is to ensure that the marketplace has a fair and useful set of metrics to differentiate candidate systems. Its member organizations include leading computer and software manufacturers. SPEC benchmarks are widely used today in evaluating the performance of computer systems; the results are published on the SPEC web site - <http://www.spec.org/>.

The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications that has already been ported to a wide variety of platforms by its membership. The benchmarker then takes this source code, compiles it for the system in question and then can tune the system for the best results. The use of already accepted and ported source code greatly reduces the problem of making apples-to-oranges comparisons.

They provide benchmarks for CPU, Graphics Applications, High performance computing, Java client/server, Network file system, Mail servers, SIP and Web servers.

Mistakes in Benchmarking

One needs to understand the fact that benchmarks provided by organizations like TPC, SPEC are based on the workload created by a typical application used for the performance testing. If the application owners develop applications with a specific workload which is totally different from the typical application workload used for benchmarking, then it is not an apple to apple comparison. But we don't have any choice as it is impossible to test our product across different configurations to choose the best one. Hence we can refer to the benchmarks and add appropriate risk buffer if the workload of the application under test is completely different from the one used for benchmarking.

Chapter 8: Performance Test Execution

In the Internet, the competitors are only a mouse click away. Hence the web sites should provide high performing, time effective application thereby increasing the satisfaction level of the end users. Dissatisfied users would abandon the site and never come back. Bad mouthing would spoil other user's interest towards the web site. Hence there is a high necessity to design applications with high performance to meet the expectations of the end user.

There are certain activities that need to be performed before we plan for the performance tests for a web site. By ensuring these activities, we could improve the accuracy of the performance tests and would run tests simulating the real time usage.

Quality of Service of web application

The QOS (Quality Of Service) of a web application is normally measured using the system response time and system throughput. It is about 'How fast is the system able to respond to the customer requests?' and 'How much load the server can withstand by providing quick response to the customers?' There is a high chance for losing the business when the customers are not satisfied with the QOS of the web site.

Providing QOS to the end users of the system is a big challenge faced by the application designers. The objective of the system testing is to identify the functional defects before the end user finds it. The system needs to be assured not only for its functional stability but also for its availability and responsiveness at all times of the user load. The end users will not be happy just by having a functionally stable system; the system should be very responsive and available enough at all times of user accesses.

It is very important for a web application to respond quickly with minimal processing time irrespective of number of users accessing the system. Though the application might be

functionally stable for 1 user, but there could be inherent design issues or code level issues which might create show stopper issues during multi user load. Hence it is highly required to test the system performance before releasing the system to the open community.

Virtual User Simulation

In order to conduct performance tests, the system needs to be subjected to the real time load, which cannot be handled manually. A virtual user refers to the single execution of test script which runs simulating the end user navigation pattern on a web site with realistic wait times. The server performance could be monitored while the virtual users are creating the required load on the server. There are possibilities that, for higher user load, the system response time may go high and in the worst case, it might lead to server crash.

Pre-requisites for Performance Tests

The first step is to understand the web site's actual usage pattern. Mostly this information can be collected from the web server log files which are considered as the 'gold mines' to know every possible detail about the server load. The 'log analyzers', tools which helps to analyze the web log files and provide the analysis report in terms of number of user sessions, number of unique users, number of errors, peak day traffic, peak hour traffic, user request arrival pattern, frequently accessed pages, user navigation pattern, etc. From this analysis, the actual usage of the server can be identified.

The second step is to design the performance test approach. Based on the business forecast, service level agreement (SLAs) needs to be derived to be met by the application in order to handle the user load in the target environment. The analysis done in the first step helps to set quantitative SLA for the application. Design the test cases that needs to be tested along with the parameters like ramp up strategy, test duration, think time settings, test scenarios to be tested, different load conditions and types of tests to be conducted along with the pass/fail criteria. Baseline the test approach and use this as the reference for planning the rest of the test activities.

The third step is to identify the performance test tool to help in simulating the user load on the server. Look for performance test tools available in the market (licensed or freeware tools) based on the budget constraints and technologies supported by the test tool. Using the test tool of choice, develop the test script which simulates the actual user access pattern. Have enough test

data available in the test script in order to simulate different users using different data available in the script.

The fourth step is to look for the readiness of the test environment. The test environment includes the application's server infrastructure and also the load test environment. Enough care should be taken to ensure that no other users access the application during the test run and the servers are isolated from any other usage. In the case of load test environment, the following activities need to be carried out before starting the tests:

- Ensure that load generators are in place to generate the required load,
- Ensure the availability of test tool license to support the load to be simulated,
- Ensure valid test scripts with the required test data and think time is ready,
- Ensure appropriate server monitors with the required performance counters are configured,
- Ensure the server administrators are informed for their assistance in case of server failures.

Types of Performance Tests

There are various types of tests that could be performed on an application during the performance certification activity. The objective and importance of various types of tests are discussed below.

1. Baseline test
2. Benchmark test
3. Load test
4. Stress test
5. Soak test
6. Endurance or Longevity test

But it is not required that all application needs to undergo all of these tests during performance testing. It purely depends on the application context and its usage pattern. It is always recommended to carry out any special tests which are different from those mentioned above, if the application usage pattern demands one.

Baseline Test

The Baseline test is performed to check the correctness of the test script developed for conducting the performance test. This test is conducted for 1 user and the system response time and the server utilization metrics are identified. This forms the basis for any future comparisons.

Mostly if the baseline test result doesn't meet the response time targets set for the application under target load, then further performance tests will not be conducted. This means that the application is not performing well even during the non-load conditions. Also, baseline test can be conducted to verify the performance improvement done in the code. Conducting the baseline test before and after code optimization activities would help in identifying the system performance improvement for 1 user load.

The test scripts could be generated with realistic think time and other settings which closely resemble the real time usage can be made. The response time profile and system throughput profile can be created and documented for the 1 user load which can be used as the baseline for any further tests.

Benchmark Test

A Benchmark test is performed to check the correctness of the script for multiple user loads. This test could be conducted with 15%-20% of the target load. This test helps in identifying the issues in the script, issues in the test data and to check the readiness of the target environment to support high user load.

The system response time and server utilization metrics would help in analyzing whether the system can withstand high user load tests. The metrics collected during benchmark tests would refer to system's best performance results. Any degradation in system performance can be identified by comparing the system performance with the benchmark test metrics.

Load Test

A Load test is performed to identify the system performance for the real time user load. It is conducted with the objective of finding the system performance for the anticipated target load. This test is required to measure the system performance for various system load conditions. The

sole purpose of load tests is to verify the system performance for normal and peak load conditions.

For conducting the load tests, the system should be tested with the realistic server load. A perfect load test should aim at generating the load with the right transaction mix and right load on the server at any point of the time. Either based on discussion with the business analysts or based on the web server log files, the peak user load and the usage profile needs to be characterized. Based on this analysis, load test needs to be strategized before starting the actual test execution activity.

The test scripts that are generated with correct think time which would closely resemble the end user usage pattern needs to be used for the load tests. The test should subject the server to right transaction mix expected on the real time usage.

The typical benefits of load test include identifying the system throughput, server hardware issues in web server, application server and database server during normal and peak load conditions, identifying application related errors during concurrent load, to benchmark the system performance for specific user loads.

Stress Test

The stress test is performed to validate the application behavior when it is subjected to the load beyond the peak load conditions. It is the negative test conducted on the system to identify the system performance beyond the expected load level. This test helps in identifying the performance bottlenecks in the application and with respect to the hardware resource saturation issues which is expected to occur only during high loads or during beyond expected load levels. Hence by identifying application's behavior during extreme loads, the mitigation actions can be planned proactively.

Basically the stress test focuses to overload the system and thereby to check how long the system can withstand higher loads beyond the target load. It also helps in identifying the server hardware issues and plan for the server capacity.

The test scripts that are generated for the stress test need not be realistic. It need not have the think times and the cache could be disabled, so that high server load could be quickly created and the server behavior for high loads can be identified quickly.

Spike Test

The Spike test is a very similar to the stress test but the system is subjected to the high extreme loads for short period of time. This test helps in validating the system performance for sudden spikes in the user loads.

The test scripts that are generated for the spike test need not be realistic. The think time can be ignored and the cache can be disabled in the script.

Endurance Test

The Endurance test focuses on validating the system performance for the anticipated user loads for prolonged time duration. This test helps in identifying the memory issues and buffer overflows that occurs during continuous usage of the system. Also, this test helps to comment the system availability based on the test duration window. Usually endurance test is run with 70-80% of the target load with realistic think time settings.

Performance Test Execution Approach

Let us recollect the basic objective of running a performance test. The Performance test is conducted to identify and certify how fast the application responds and at what point of load the application performance starts degrading. The very important expectation on the performance test activity is the ability to run more realistic tests. By injecting the required number of users load during the test, realistic end user usage pattern cannot be achieved. The server workload needs to be analyzed which is purely based on the expected accuracy level in the test results. For a performance critical application, often high accuracy is expected combined with high level of confidence about the system performance.

Normally, for performance testing, pareto's 80-20 rule should be followed. After identifying the most frequently used business flows, user distribution among the identified business flows needs to be done carefully. Though 100% real time simulation is impossible, lot of analysis should go into this analysis to come up with more accurate realistic numbers. Usually for this analysis, web log analysis provides lot of inputs in case of applications where usage history exists. A high accuracy level can be expected in case of the availability of real time usage data. For applications, going to production for the first time, expected accuracy level should be less due to the non-availability of real time data.

It is always better to neglect those business flows which are seen or expected to be occasionally used during peak business hours. Rather during performance tests, subject the server to a constant load for 15-30 minutes approximately (also depends on application usage pattern) to get more samples of server behavior (with respect to response time and resource utilization levels) during loaded condition and then certify the application for the subjected load. A test could be called realistic only when the user load and the transaction mix are appropriate along with the associated confidence about the server performance behavior.

It is required to verify the system performance for more number of samples and then conclude the performance metrics for response time, server utilization levels, etc. The test should always have sufficient time for user ramp up, before subjecting the system to the expected load period. All the measurements about the server performance need to be taken only during the stable load period. The test should be configured in such a way that all users are actively accessing the system during the stable load period. If the peak expected server load is for example 200 requests per second, then the test should be scheduled in such a way that that running users should create a constant load of 200 requests per second atleast for 10 to 15 minutes duration. Though the peak expected user load (200 requests per second) occurs for less number of times in the real time with sufficient time interval between the spikes, the performance test should subject the system for a constant load to get more samples on the server performance. This helps in assuring the server performance behavior for a peak traffic situations. Hence the Performance Test Engineer should come up with a proper test strategy in order to test the system for realistic load and also at the same time, the test should be able to guarantee the system performance behavior for the peak load situations.

Let us discuss an example to validate the load test scenario created for the business requirements. The application needs to be validated for the 100 user load which consists of following 3 business scenarios.

- Business Scenario A -60% of target load
- Business Scenario B -30% of target load
- Business Scenario C -10% of target load

Based on the inputs from business analysis, it was found that scenario A will be executed 6 times in an hour, scenario B will be executed 2 times in an hour and scenario C will be executed 1 time in an hour. Also, it was known that the peak server load was 200 requests per second during the peak hour.

The following load test is scheduled in order to meet the above test requirements.

Business Scenarios	100 Users Load Test : Duration (60 minutes)					
	Ramp up Duration (10 minutes)	Stable Load Duration (40 minutes)				Ramp Down Duration (10 minutes)
A	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6
B	Iteration 1		idle time	Iteration 2		idle time
C	idle time	idle time	Iteration 1	idle time	Iteration 2	idle time

Figure 19: Load Test Execution Strategy

This load test strategy meets the user load targets (100 users accessing the system for 40 minutes of stable load period) and the number of executions of each business scenarios (6 iterations of scenario A, 2 iterations of scenario B and 1 iteration of scenario C). During the stable load period, when all three business scenarios were running (40:00 to 50:00 mm:ss), the server load created on the server was about 200 requests per second. Hence the test seems to be realistic to the production load.

The above test scenario meets the performance test requirements of the applications and it provides the system performance figures for the realistic system usage. But what is missing in the above load test scenario is the confidence level of server being able to handle the 200 requests per second, as there was only 1 sample collected during the interval (40:00 to 50:00 mm:ss) during which all three business scenarios were running. A load test should aim for assessing the server performance for realistic usage pattern and also create a high confidence by analyzing the system performance for atleast 3 to 4 samples. The confidence level increases only when more number of samples is made available and analyzed for server behavior. Also, the server resource utilization levels needs to be assured only by subjecting the system to a stable constant load. Hence it is recommended to run all the three business scenarios together for atleast 20 to 30 minutes to collect enough samples on the response time metrics and server resource utilization metrics by maintaining the load at 200 requests per second. This can be combined in the same load test or can be run as a separate test depending the application performance requirements and performance behavior of the system.

The moment we talk about realistic tests, think time should be the high priority one worth spending time for detailed analysis. In spite of having correct business flows and user distributions between the business flows, the moment think time is not realistic, the entire test goes for a toss leading to unrealistic tests. The think time decides the user request arrival pattern of an application which decides the load on the server at any point of time. Usually it is always

recommended to use realistic think time between transactions and randomize it while running the test (say 80% to 120% of configured think time). This randomization helps in simulating the think time of inexperienced user of the web site and the frequent user of the web site.

Tips for Load & Stress Test Execution

At least 3 cycles of load tests need to be planned before subjecting the system to the target load. The low load test should test the system for 30% of the target load followed by medium load test which tests the system for 60% of the target load followed by high load test which subjects 100% of the target load on the system.

Every load test should have 3 regions – ramp up, stress and ramp down regions. In the Ramp up regions, users need to be scheduled to start accessing the application with specific periodic intervals (say 5 users per minute). This value needs to be decided based on the application usage pattern. If your application is an auction site where there is a possibility that 50 users can login per second during a peak business hour, and then consider having high ramp up rate accordingly. Hence depending upon the ramp up rate the ramp up duration will change for each test.

In the stress region, the system is subjected to a constant load of injected users. No new user starts accessing the application in this region and the users continue to access the application throughout the stable load period. The server load (number of users) is kept constant and server behavior during the steady load can be identified. The server resource utilization levels need to be picked up from the steady state period and not during the ramp up period. As long as you get required number of samples to comment on the server resource utilization levels and system response times, the objective of conducting the test can be met.

In the ramp down region, users need to be scheduled to logout with specific periodic intervals (say 20 users per minute). Usually for ramp down, it does not really need any application usage pattern analysis. In real time, users would logout and leave the application in a random fashion. Just considering that, it is advisable to have a ramp down after the stress period. But other than that there is no specific requirement about ramp down intensity.

Unlike Load tests, usually one stress test needs to be conducted which targets to stress the system to find the system breakpoint and its possible causes. The system performance behavior identified from the high load test forms a major input to the stress test user load. Usually about

10-20% extra load can be subjected to system to identify the system behavior and possible failure causes. It is always advisable to slowly increase the system load till the break point is identified. Though only one test can be planned, sometimes, to check for the consistent behavior, the same test can be repeated again during some other time slot and the system behavior could be analyzed with high confidence level.

After Load and Stress tests, depending upon the objective other tests like endurance test, spike test, etc could be executed to confirm the system behavior.

Test Execution Factors

Knowledge on the following facts would help the Performance Test Engineer for appropriate planning.

Number of Tests

The approximate number of tests required for performance testing based on the target user load SLAs of an ideal AUT is provided below. In case of presence of performance bottlenecks, this approximation does not holds good. As it is always recommended to have a series of load tests planned for various increasing load levels, the number of tests to be conducted slightly increases if the load targets are higher.

It is recommended to have benchmark test conducted for 20% of the target load and at least 3-4 load tests to be conducted for various loads of 40%, 60%, and 80% of target load before subjecting the system to the target load. But if the target load is too high, then more than 3-4 load tests will be required to ensure the system is subjected to increasing load levels without creating a huge difference in the load subjected between consecutive tests.

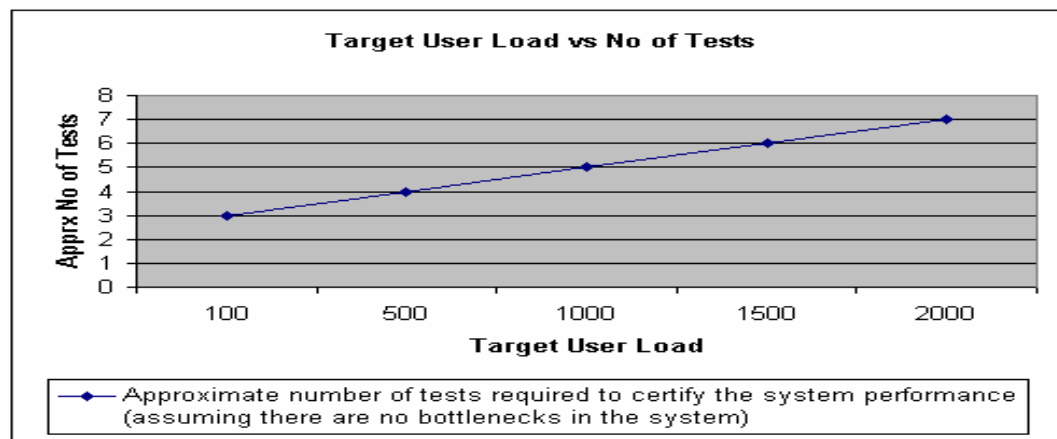


Figure 20: Target User Load vs No of Tests

Response Time SLA

A typical Response time graph of the Application Under Test (AUT) is provided below. The horizontal line (in red) represents the Service Level Agreement (SLA for response time). The graph provides the response time details for various user loads.

An ideal Performance Test activity should identify the 'knee point' within the scope of performance testing.

In the below graph, the AUT meets the SLA (response times less than 8 seconds) until the load of 350 users, though the System Throughput SLA target (System should support 500 users load) is not met.

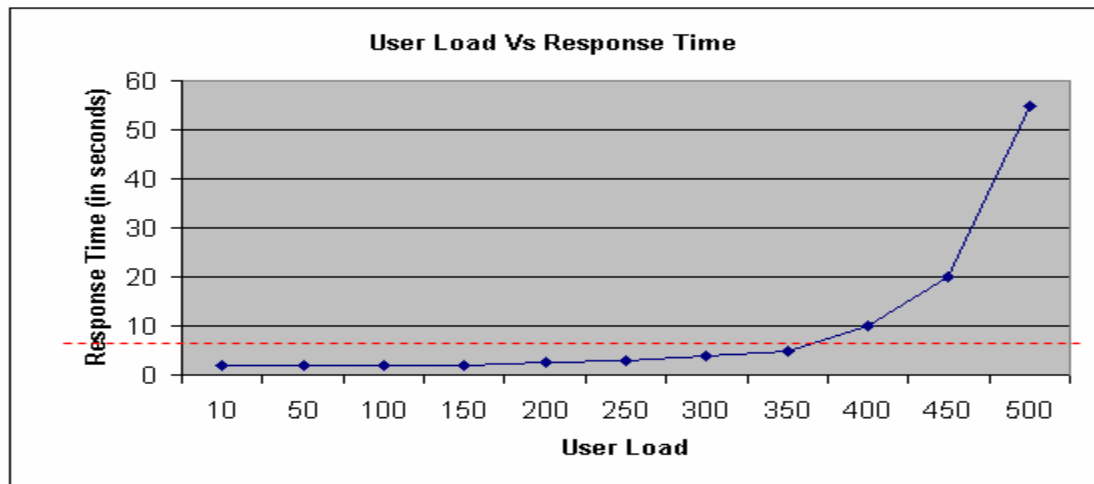


Figure 21: User Load vs Response Time

System Throughput SLA

A typical System Throughput graph of the Application Under Test (AUT) is provided below. The horizontal line (in red) represents the Service Level Agreement (SLA for System Throughput). The graph provides the system throughput (transactions / requests per unit time) details for various user loads.

An ideal Performance Test activity should identify the 'knee point' within the scope of performance testing.

In the below graph, the AUT does not meet the Throughput SLA (System should support 500 users load) after 370 users load. The system is not capable to handle more than 26 transactions or requests per second and hence the throughput does not linearly increase with the increasing user load levels.

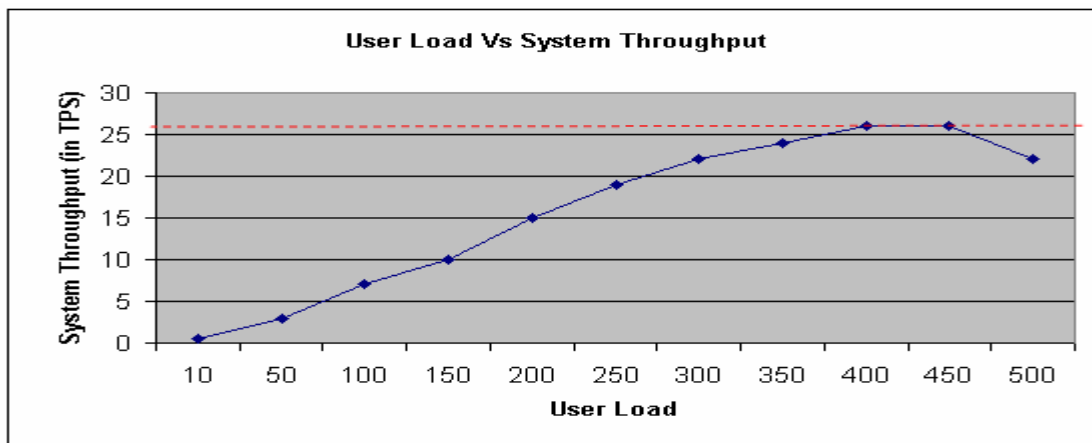


Figure 22: User Load vs Throughput

System Performance Bottleneck Severity

A typical System Performance Bottlenecks view of the Application Under Test (AUT) is provided below. The horizontal line (in red) represents the SLA for fixing only the high severity performance bottleneck to meet the Response time and Throughput SLAs of AUT. The graph provides the severity of various bottlenecks identified during the performance test activity for various user load levels.

Any application will have more than one performance bottleneck, but the ideal performance test activity should identify and isolate those high severity bottlenecks in order to meet the agreed upon SLAs. In the below graph, fixing up of only the Application Server Bottleneck (represented as blue line) can help in achieving the agreed SLA and hence fixing up of next high priority bottleneck (in this case, it is web server bottlenecks) is under stakeholder's discretion.

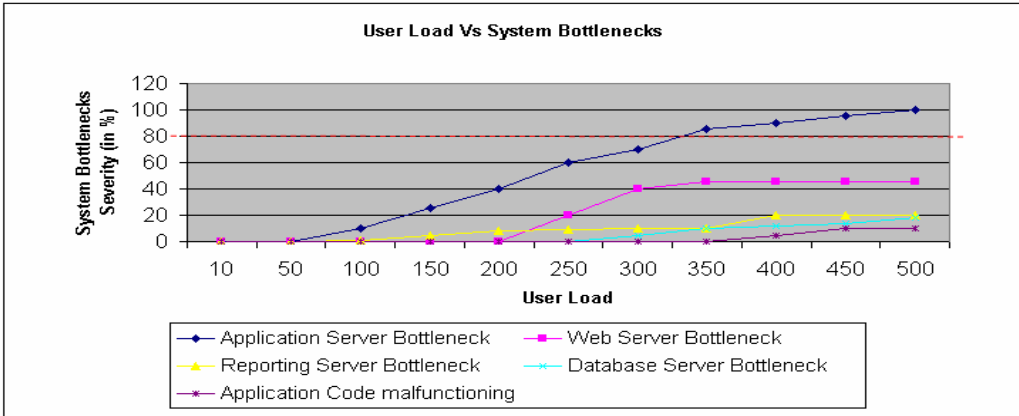


Figure 23: User Load vs System Bottlenecks

Chapter 9: Performance Test Monitoring

Introduction to Performance Monitoring

Performance monitoring is the process of collecting and analyzing the server data to compare the server statistics against the expected values. It helps the performance tester to have the health check of the server during the test. By monitoring the servers during the test, one can identify the server behavior for load condition and take steps to change the server behavior by adopting software or hardware performance tuning activities.

Each performance counter helps in identifying a specific value about the server performance. For example, % CPU Utilization is a performance counter which helps in identifying the utilization level of the CPU.

In a nutshell, server monitoring should provide information on four parameters of any system: Latency, Throughput, Utilization and Efficiency, which helps in answering the following questions.

- Is your server available?
- How busy is your CPU?
- Is there enough Primary Memory (RAM)?
- Is the disk fast enough?
- Is there any other hardware issues?
- Is the hardware issue result of software malfunctioning?

What Counters needs to be monitored?

With about thousands of performance counters available in the server, it is always confronting what can be chosen among them. There are no standards that specific counters are important which needs to be monitored on a regular basis and what others which can be ignored. It purely depends on the objective of the performance monitoring. End of the day, it depends on what are

you going to do with the collected data. Will that be helpful for you in performance bottleneck analysis and troubleshooting, capacity planning, ensure system availability, etc?

Never plan for performance monitoring without having a clear objective. Let us assume that you want to monitor your server performance in order to troubleshoot the performance problems at load conditions. You need not monitor all the performance counters of CPU, Memory, Cache, Disk, Network, etc. Always make a practice to start with minimal number of counters which are good enough to provide you the symptoms of the performance bottlenecks. Just about 8-10 counters are good enough for you to get to know some hints on where is the problem. For instance, let us assume there are some symptoms on memory issues are noticed. You are noticing high page faults/sec. Now you add few more memory counters (like Available Mbytes, Pages/sec, Page reads/sec, Page writes/sec, Pages Input/sec, Pages Output/sec) which will provide more details on the memory statistics which will help you to drill down to the problem.

Key Performance Counters

Always start with few counters and once you notice a specific problem, start adding few more counters related to the symptom. Start monitoring the performance of the major resources like CPU, Memory, Disk or Network. This section provides you the details about key counters from each of above mentioned 4 areas which are very important for a Performance Tester to know.

Processor Bottlenecks

The bottlenecks related to processor (CPU) are comparatively easy to identify. The important performance counters that helps in identifying the processor bottleneck includes

% Processor Utilization (Processor_Total: % Processor Time) – This counter helps in knowing how busy the system is. It indicates the processor activity. It is the average percentage of elapsed time that the processor spends to execute a productive (non-idle) thread. A consistent level of more than 80% utilization (in case of single CPU machines) indicates that there is not enough CPU capacity. It is worth further investigation using other processor counters.

% User time (Processor_Total: % User Time) – This refers to the processor's time spent in handling the application related processes. A high percentage indicates that the application is consuming high CPU. The process level counters needs to be monitored to understand which user process consumes more CPU.

% Privileged time (Processor_Total: % Privilege Time) – This refers to the processor's time spent in handling the kernel mode processes. A high value indicates that the processor is too busy in handling other operating system related activities. It needs immediate attention from the system administrator to check the system configuration or service.

% I/O Wait (%wio - in case of UNIX platforms) – This refers to the percentage wait for completion of I/O activity. It is a good indication to confirm whether the threads are waiting for the I/O completion.

Processor Queue Length (System: Processor Queue Length) – This counter helps in identifying how many threads are waiting in queue for execution. A consistent queue length of more than 2 indicates bottleneck and it is worth investigation. Generally if the queue length is more than the number of CPUs available in the system, then it might reduce the system performance. A high value of % usr time coupled with high processor queue length indicates the processor bottleneck.

Other counters of interest:

Other counters like **Processor: Interrupts per second**, **System: Context Switches per second** can be used in case of any specific issues. Interrupts per second refers to the number of interrupts that the hardware devices sends to the processor. A consistent value of above 1000 in Interrupts per second indicates hardware failure or driver configuration issues. Context Switches refers to the switching of the processor from a lower priority thread to a high priority thread. A consistent value of above 15000 per second per processor indicates the presence of too many threads of same priority and possibility of having blocked threads.

Memory Counters

Available Memory (Memory: Available Mbytes) – This indicates the amount of free available memory. A consistent value of less than 20% of RAM indicates the memory problem as per Microsoft standards.

Pages per second (Memory: Pages/sec) – This counter is the best indicator memory shortage problems. It indicates the number of pages stored/retrieved from the secondary memory (disk) due to paging. A consistent value of more than 5 indicates the memory problem.

Page Faults per second (Memory: Page Faults/sec) – This counter provides the count of how many times the information was not available in the primary memory location where it was expected to be. This includes the count of soft page faults/sec (available at different location in the primary memory) and hard page faults/sec (available in the disk). A consistent high Hard Page faults/sec value indicates the memory shortage problem. A high Page Faults/sec value coupled with high Memory: Pages/sec indicates the shortage of memory.

Process: (Process: Working Set) – This counter provides the instance level (process) memory information of which process takes more primary memory (RAM). The working set refers to the number of pages that a process can address in the primary memory without causing a paging.

Other counters of interest:

There are other memory counters to drill down the possible memory issue. **Memory Page Reads/sec, Memory Page Writes/sec** can be used to confirm specific issues related to memory read or memory write.

Disk Counters

It might be required to know the performance of entire disk system and also the specific logical drive level impact. Look for Physical Disk counters when evaluating the performance of overall disk system and look for Logical Disk counters when you are interested in understanding how much is the impact caused by a specific application running in a logical disk (say, D:\ drive).

% Disk Utilization: (Physical Disk: % Disk Time) – This counter provides a clear indication how much busy is disk.

Disk Queue Length: (Physical Disk: Average Disk Queue length) – This counter is a clear indicator of disk issues. A consistent value of more than the number of spindles plus two indicates waiting threads for disk.

Other counters of interest:

There are other counters like **Physical Disk: Average Disk Read queue length, Physical Disk: Average Disk Write queue length, Physical Disk: Average Disk sec/Transfer, Logical Disk: Disk Reads/sec** which needs to be referred to drill down the disk bottleneck.

Network Counters

Total Bytes (Network Interface: Bytes Total/sec) – This counter refers to the total number of bytes sent and received at any point of time. A consistent value of more than 80% of the network bandwidth indicates the problem.

Bytes Sent (Network Interface: Bytes Sent/sec) – This refers to the total number of bytes sent to the NIC at any point of time.

Bytes Received (Network Interface: Bytes Received/sec) – This refers to the total number of bytes of incoming traffic received by the NIC at any point of time.

Performance Monitoring in Windows platform

Windows operating system comes with a performance monitoring tool called Perfmon. This monitoring tool can be used to collect the server statistics during the performance test run. Normally, most of the performance testing tools have its own monitors to monitor the system resources of the system under test. In this case, it becomes easy to compare the system load related metrics with the system resource utilization metrics to arrive at a conclusion. Infact, any performance testing tool that monitors windows machine internally talks to perfmon to collect the system resource utilization details.

Using Perfmon

During the performance tests, it is always recommended to monitor the web, application and DB server resource utilization levels. Server monitoring helps to identify and isolate hardware related issues.

Run perfmon from Start Menu → Run. Type perfmon in the Run window and click enter key. It opens the performance monitor window as shown below. By default, the following three counters will be available : % processor time , Avg Disk Queue length and Pages/sec. Add more counters by clicking on the Plus icon (+) available in the top tool bar.

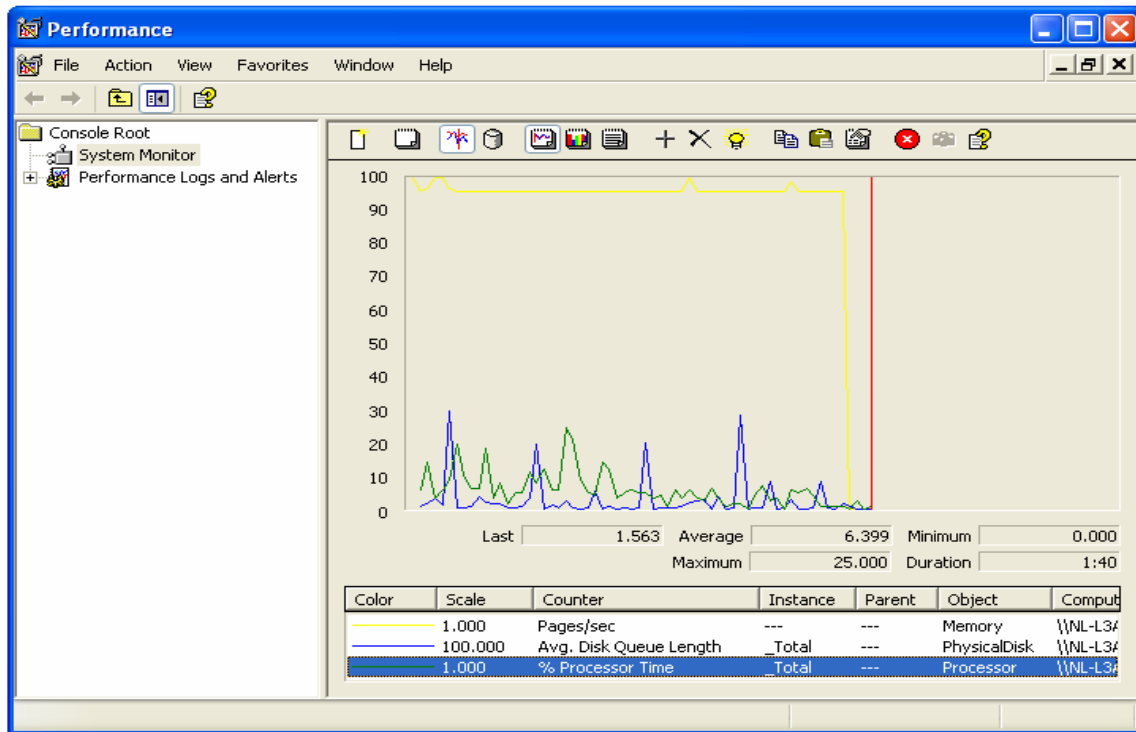


Figure 24: Perfmon Monitor

While running the test, perfmon can be scheduled to monitor any number of counters and logs could be created so that the counter information is available for offline analysis. Also alerts can be created to alarm when specific counter level reaches a threshold value. By default, the monitoring interval is every 15 seconds, if required this interval can be reduced, but be aware that it will increase logging frequency thereby increasing the log file capacity. To schedule perfmon to create a log file, click on the counter logs and create a new file and edit the existing default *System Overview* file as shown below. Click on Add Counters button to add relevant counters and schedule it to run during specific time interval or by manually clicking on start icon (right arrow icon in top tool bar as in the below figure).

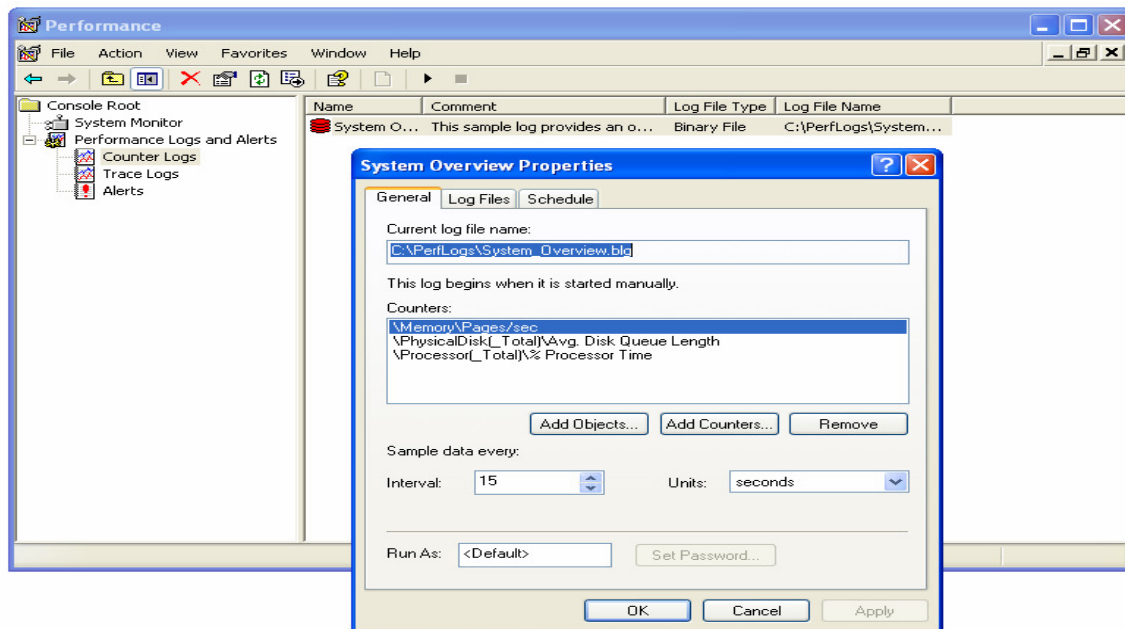


Figure 25: Setting up perfmon Counters

Post production Monitoring

There are lots of licensed tools available in the market which is used for post production monitoring. These tools are used to capture the web server traffic details and provide online traffic details. A very popular tool of this category is WebTrends. Many organizations use this tool to get to the traffic trends of an application running in production environment. There are other tools like HP OpenView tools which run in production servers and monitor the server resource utilization levels. It provides alarming mechanism to indicate the heavy usage and provides easy bottleneck isolation capabilities. But due to the cost involved with these kinds of tools, most of small organizations don't opt for them. But post production monitoring data would be of great use for designing realistic performance tests.

Benefits of Performance Monitoring

- Allows you to analyze and isolate the performance problem.
- Understand the resource utilization of the server and make best use of them.
- Plan for Capacity Planning activity based on the resource utilization level.
- Provides the server performance details offline (by creating an alert of sending a mail/message when resource utilization reaches the threshold value).

Chapter 10: Performance Bottleneck Analysis

Often Performance Test Engineers feel performance bottleneck analysis is a complex task. Unless you experience bottlenecks and experiment few things, one cannot become an expert in bottleneck analysis by reading articles or books. I agree that it is more of an art than science. But bottleneck analysis and isolation can be made easy when systematically approached and fundamental concepts in Queueing Theory are understood. This chapter explains few things which every Performance Test Engineer needs to have in mind during performance bottleneck analysis.

Scott's Recommendation on Scatter Charts

Scatter Charts are one of the powerful tools for bottleneck analysis. It provides a quick view of the bottleneck and it is easy for a Performance Test Engineer to explain it to non-technical stakeholders.

As part of the bottleneck analysis, the first chart which every Performance Test Engineer needs to be look for is the response time chart. Look for the trend in the server processing time for each of the transaction or timer provided in the script during the span of the test. Always ignore the metrics collected during ramp up and ramp down as any conclusion should not be made while the server load is changing. Any metric that is collected during the stable constant load period can only be considered for the analysis.

A Scatter Chart can be created by having test time plotted in seconds on X-axis and the response time measure plotted in seconds on Y-axis. Each measurement represents the server response time for a specific transaction or timer provided in the script during the span of the test. A simple scatter chart is provided below. The blue dots represent the response time for loading the system home page and the red dots represent the response time for login transaction.

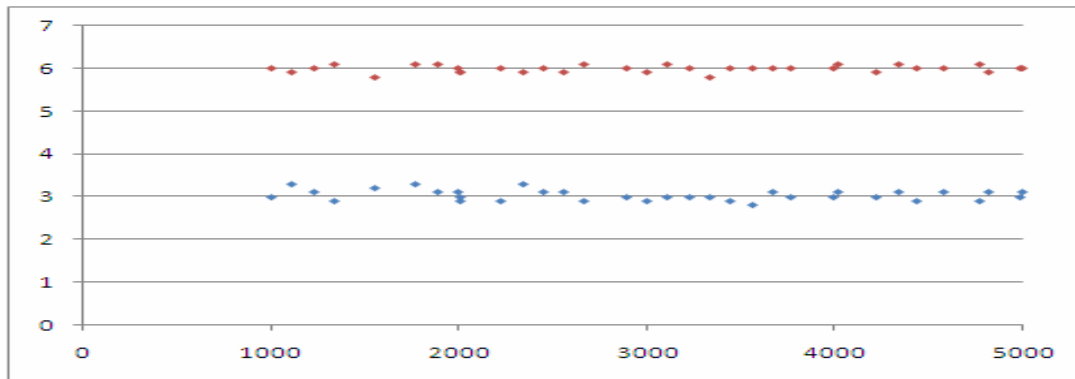


Figure 26: Scatter Chart

Scatter Chart Types

The Scatter Chart Pattern of the applications might fall into any of the below categories - Good Pattern, Banding Pattern, Outlier Pattern, Caching Pattern, Classic slowdown Pattern and Stacking Pattern. Analyzing the scatter chart pattern would bring in the right start to go about with the bottleneck analysis.

Good Pattern - A very small standard deviation will be noticed between response times. This pattern can be expected for an ideal system without any bottlenecks which validates the test conducted on the system.

Banding Pattern - Obvious horizontal bands will be noticed for each of the transactions. The above provided figure represents banding pattern. Easy to isolate the transaction that does not meets the expected levels and go about the further analysis on the high response time transaction.

Outlier Pattern - Along with obvious horizontal bands of response time, few instances of high deviation on response time will be noticed. Further analysis might be required to interpret the reason for those outliers. It can be ignored if it is due to test data related errors.

Caching Pattern - The response time data collected during the beginning of the test will be high compared to the one collected during rest of the test. As long as the reason for this pattern is server restart which added extra time for JSP recompilation, it can be ignored. Else, it might need further analysis for the reason.

Classic Slowdown Pattern - The response time shows a slow increase as time progresses. This is a typical behavior. As the response time starts increasing after a point, the number of requests handled by the server comes down in spite of constant user load injected by the test tool. Hence the total load on the server decreases which increases server's ability to handle the requests resulting in decrease in the response time. The reason for increase in response time needs to be analyzed by looking at server resource (CPU, Memory, and Disk) utilization levels.

Stacking Pattern – This pattern is little complicated which needs more analysis. Mostly there is a high probability that the test is invalid if this pattern is observed. Constant and resembling ups and downs are noticed in the response time values. The user abandonment and think time modeling used in the script needs to be verified.

Compound Pattern – This pattern forms the combination of multiple other patterns which mostly needs overlying several others system metrics to identify the bottleneck.

Performance Test Analysis

Though the performance testing activity is to simulate high loads on the system and to identify the system break points, practically not all the projects will have the objective to find bottlenecks on the system due to time or cost constraints. Many a time it so happens that project stakeholders expect the Performance Engineers to do just an application benchmarking (run the planned tests and report the system behavior), though there is a planned production move after the performance test. In my experience, many a time it happened that I end up convincing the project stakeholders about the business risk in doing application benchmarking without planning for bottleneck analysis and isolation. I have to accept that sometimes I have failed in my attempt and ended up doing benchmarking in spite of project being planned for production move.

It is very essential that every Performance Test Engineer should understand what is required for the application and suggest it to the stakeholders irrespective of limiting to time or cost constraints. However, acceptance is under stakeholder's discretion.

Whenever any performance tests are run (benchmark tests, load tests, stress tests, etc), the first metric which needs to be looked at is the response time metric. This is the basic metric which needs to be checked to know the server's processing time, time taken by the server to respond to user requests. Always adopt the practice of merging response time and running users graphs

both for analysis and reporting purpose. Then look for the scatter chart pattern and start looking at the suspects.

The second important metric to look at is the server throughput, server's processing capability in terms of requests handled per unit time or transaction handled per unit time or amount of bytes returned to the users per unit time. For this metric, there are several graphs available in the test tool like Hits per second, Transactions per second, Throughput (in Bytes/second).

I am sure you will not accept why I am talking about hits per second metric as this metric refers to the input load provided to the system and not the server's output. To understand it better, consider this example. There is a small tank which has an inlet and outlet pipe of same capacity at the top and bottom of the tank respectively. Assume the tank doesn't have any other leakage; Fill half of the tank by water having outlet pipe closed and then open the outlet pipe and observe the water inflow and outflow. We can observe that the rate of water flowing into the inlet pipe will be the same as the rate of water flowing out of the outlet pipe. This is true as long as the tank is in good stable condition. The same applies to the server during stable state. The incoming flow rate, A (arrivals per unit time) and out coming flow rate, C (completions per unit time) will be equal during the stable state (as long as there is no saturation in the server).

$$A = C \text{ where } A \text{ is the arrival rate \& } C \text{ is the completion rate}$$

The Hits per second graph provides the details on the requests handled by the server during unit time. For increasing loads, the hits per second should linearly increase and during the constant load period, the hits on the server should be constant. If there is a drop in the hits per second value, then it represents an issue on the server. This represents that the server is not able to handle the incoming requests which represents instability in the server. This issue needs to be cross verified with the server monitoring graphs to look for saturation issues of various service centers.

The errors observed during the tests needs to be analyzed and test data related issues needs to be isolated from the application errors. If test data related issues exist, it needs to be fixed and the test needs to be rerun to check for consistency in the issues.

Best Practices on Performance Bottleneck Analysis and Isolation

The bottleneck analysis can be made easy if systematically approached. Irrespective of implemented technology or platforms, analysis can be planned as per the below steps.

1. The system resource utilization levels of all infrastructure components needs to be monitored during the performance test which might be required for the analysis.
2. The response time (in seconds) and system throughput (in transactions per second) needs to be verified to confirm whether the system is able to support the injected load. A linearly increasing curve will be noticed in case of an ideal system. If any discrepancies are noticed then the system needs to be analyzed for its behavior.
3. Firstly, the presence of bottleneck needs to be reconfirmed by running another test run. Sometimes due to other unknown batch processes running in the server or network failures or issues with specific test data, etc there might be a drastic negative impact on the server performance. Hence it is very important to confirm the consistency of the issue.
4. After confirming the consistency of the bottleneck, the bottleneck needs to be classified as software related or hardware related. Note: Often hardware configuration setting could create a bottleneck which is tough to differentiate from the software related issue. All the hardware configuration settings like thread pool, JDBC pool, DB connections, etc needs to be monitored to confirm the hardware configuration related issues.
5. If the transaction response times of all the transactions meet the response time SLA during baseline and benchmark test, without any server failures then it seems the software is not an issue during these tests. A specific transaction not meeting the response time targets or server failures (http 500 errors, application errors, etc) noticed during the test is an example for typical software failures. A slow increase in response time during load tests is a typical behavior of any application. Note: Specific software issues, memory overflow after certain load, concurrency issues after certain load period, etc cannot be identified during benchmark tests, but still the initial software issues can be reported during these tests.

Load tests needs to be conducted only after baseline and benchmark tests are conducted. By correlating the resource utilization parameters with the response time graphs, we can analyze whether there is any relation between response time increase and resource usage levels. If there are any server errors (HTTP 5XX) then there could be a potential software bottleneck. Also check if the maximum throughput achieved during the test reaches a flat curve during increasing load level. If the flat curve is not seen, but

- there server errors are seen, it means the server has enough capacity to handle the load, but the software is not scalable to handle more load. This helps to confirm the presence of issues in the software during concurrent access. The application logs of the application need to be analyzed to verify whether any exceptions or errors are logged during the tests which also help to confirm the presence of software bottlenecks. If for example, the response time of most of the transactions are high and all transactions related to static request like home page request has less response times during high load condition then it indicates that the web server performance is up to the expectation and there are some bottlenecks in the application or DB server.
6. If the response time of all the transactions seem to increase after a specific load level, then it clearly indicates the saturation of a specific service center (Memory, CPU or Disk) of the web server or the application server or the db tier (in a typical web application context) or a hardware configuration related error.
 7. Look for certain mandatory performance metrics in each of the tier to confirm the presence of bottleneck.
 - a. On the web tier, check whether the number of HTTP connections is maxing out, check whether the CPU, memory and disk utilization levels are below the tolerance limits.
 - b. On the application tier, check for the thread pool size (Web or EJB container threads), db connection pool size, CPU, Memory and Disk utilization levels.
 - c. On the database tier, look for queries for which execution time is high and for presence of any deadlocks, CPU, Memory and Disk utilization levels. SQL profiler for SQL database, Statspack reports for Oracle database would provide the required information.
 8. On each of the tier, always start with few performance counters (one or two) for each service center and add more counters only when there is a suspect and drill down to the bottleneck.
 9. Always note down all the observations collected during the test and consult with various subject matter experts like server administrators, DBAs, etc. Sometimes it so happens that figuring out appropriate configuration settings for some of the settings needs to be arrived after several runs by using a trial and error approach.
 10. Maintain a detailed test log or run book which documents all the changes in the configuration settings and objective of each tests else after a period of time after executing several tests, there is a high chance to get lost and it will become very tough to compare the system performance or to bring back the system to its previous state.

Performance Testing Vs Profiling

The performance test tools have the capability to pinpoint the user actions which have high response time. By configuring right set of monitoring on the server infrastructure, hardware related bottlenecks could be effectively identified during performance tests. In the case of software bottlenecks, the performance test tools in general does not provide the details like which tier or software component consumes more processing time. Also, the performance test tools do not have the feature to drill down to the method call or component which contributes to high response time. Any issues related to the application code cannot be identified from the performance test tool.

Profiling is a dynamic program analysis which helps in understanding the program behavior by inserting specific code into the program code (instrumentation) to capture the call times, call stack, frequency of method calls, call performance, thread concurrency issues, memory usage limits, garbage collection details and heap behavior during the program execution. The profiling tools hooks into the JVM of the application (Java) and instruments the application classes to monitor its performance hence provide capabilities to show the call chains and helps in identifying the line of code which leads to high processing time for 1 user load. Profilers are considered as unit test tools as it helps in identifying code performance and it would be a best practice to do profiling and address the software issues before subjecting the system to performance tests.

HP Diagnostics

HP Diagnostics tool facilitates drill down capabilities to identify the root cause for high response time transactions of the system. It helps to identify slow performing components, memory leaks, thread contentions, slow SQL query, slow layers and concurrency issues during the time of the load tests. The ability to provide the server performance behavior during the loaded condition makes HP Diagnostics tool special and different from other profiler tools like Dev Partner, Glow code, etc. The Diagnostics tool can be integrated with the HP performance test tools (Load Runner, Performance Center), the same performance test scripts used for performance testing could be used to create the required load on the server and the server performance characteristics could be monitored and diagnosed.

The tool provides solutions for J2EE applications, .Net applications, SAP NetWeaver applications and oracle 10g database. It consists of several components – Probe, Collector, Server (operates in Mediator and Commander modes). The Diagnostics Probes are responsible for capturing the

events from the application and sends the aggregated information to the mediator. Probes can be installed as a standalone profiler or with the Diagnostics server. The standalone probe works similar to any profiler tool and has capabilities to provide application performance metrics until 5 users load. The Diagnostics Collectors are responsible for gathering data from external ERP/CRM environments (SAP R/3 system and Oracle 10g database). The Diagnostics Server is responsible for working with the Probes and with other Mercury products to capture, process, and present the performance metrics of the application. The deployment may consist of one or many Diagnostics Servers. If there is only one Diagnostics Server in the deployment, it is configured in Commander mode and must perform both the Commander and Mediator roles. If there is more than one Diagnostics Server in a deployment, one of them must be configured in Commander mode and all the rest in Mediator mode. One or more probes communicate with the Diagnostics Server configured in Mediator mode which in turn communicates to the central Diagnostics Server configured in Commander mode. The main benefit in using the Diagnostics Probes along with the other components in the Diagnostics environment is the possibility to merge the load test results and diagnosis information to have the drill down information available for high response time transactions identified during performance tests.

Database Diagnostics Tools

Majority of the web based applications use either Oracle or SQL as the database server. J2EE applications use Oracle as the database server and .NET applications uses SQL as the database server. Other than database server resource monitoring, issues related to poor indexes, inefficient SQLs, deadlocks, etc needs to be monitored and tuned to improve the overall system performance. The database servers come up with their own performance monitoring utilities which needs to be used during the performance test. Setting up monitors for Oracle and SQL server might provide the list of performance counters, but it would be very tedious to interpret the reported data and judge whether the values are abnormal. The Performance Test Engineer should know how to use the database performance diagnostic tools and how to interpret the results. It is very important that Performance Test Engineer should at least be able to understand the fundamentals in order to interpret the performance issues and speak to database experts in appropriate language.

Oracle Statspack Monitoring

Statspack is a set of SQL, PL/SQL, and SQL*Plus scripts that allow the collection, automation, storage, and viewing of performance data which can be used for performance diagnosis. It has

the features of Oracle's `utlbstat.sql` and `utlstat.sql` utilities (used in previous versions of Oracle before v8i). It has about 25 tables which help to maintain the performance information of the server which can be used for reporting and analysis. It provides features to take snapshot of the server statistics during the requested timeframe and maintains this information in a table, so that the server performance between multiple snapshots can be analyzed and server performance deviation can be measured. Each snapshot is identified by a snapshot id, `SNAP_ID`. The server statistics captured during the second snapshot will include the accumulated values of first snapshot along with the server statistics captured after the first snapshot. When a snapshot is executed, `STATSPACK` will sample the memory structures inside SGA and capture them in respective tables, which can be queried later at any point of time for analysis.

It is always a good practice to have snapshot intervals in minutes, so that it provides more precise, concise and reasonable information about the server statistics. It would be ideal to take snapshots of 15 or 30 minutes interval and compare the server performance between the snapshots. For performing this performance analysis, the initialization parameter `TIMED_STATISTICS` needs to be set to `TRUE`. The following is the syntax to create a snapshot. The level of details that needs to be captured in the snapshot is defined by a parameter which can be set while creating the snapshots. It should be set to any of the following levels.

- | | |
|----------|---|
| Level 0 | This level captures general statistics, including rollback segment, row cache, SGA, system events, background events, session events, system statistics, wait statistics, lock statistics, and Latch information. |
| Level 5 | This level includes capturing high resource usage SQL Statements, along with all data captured by lower levels. |
| Level 6 | This level includes capturing SQL plan and SQL plan usage information for high resource usage SQL Statements, along with all data captured by lower levels. |
| Level 7 | This level captures segment level statistics, including logical and physical reads, row lock, itl and buffer busy waits, along with all data captured by lower levels. |
| Level 10 | This level includes capturing Child Latch statistics, along with all data captured by lower levels. |

SQL> exec statspack.snap (i_snap_level => 6, i_modify_parameter => 'true');
where i_snap_level parameter refers to the level of details to be captured in the snapshot and i_modify_parameter refers to whether the snap level is permanent for future snapshots.

The snapshots created could be compared by running the spreport.sql. It prompts for beginning SNAP_ID, ending SNAP_ID and report name.

SQL> @?/rdbms/admin/spreport.sql

The statspack report provides the server statistics information which could be analyzed using online analysis web sites like oraperf.com, statspackanalyzer.com or licensed tools like spanalyser can be used. These resources help in analyzing the statspack file and provide user friendly information along with possible recommendation on the database server tuning.

Some of the important information included in the statspack report includes slow performing SQLs sorted based on various parameters like query executions, Parse calls, Buffer Gets (CPU time), etc, wait events, hit ratio, cache usage levels, transactions per second, rollbacks per transaction, physical and logical reads, count of various scans (table scans, full scans, range scans, index scans), lock information, etc.

The Performance Test Engineer should have the knowledge on interpreting the statspack analysis report and need to work with DBA to resolve the performance issues.

SQL Trace and TKPROF

The SQL Trace facility and TKPROF are two basic performance diagnostic tools that help in tuning applications running against the oracle server. The SQL Trace facility and TKPROF helps to assess the efficiency of the SQL statements used by the instance or session.

The SQL Trace facility provides performance information on individual SQL statements. It generates the following statistics for each statement:

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed

- Misses on the library cache
- Username under which each parse occurred
- Each commit and rollback

TKPROF program takes SQL trace file as the input, formats the contents of the trace file and places the output into a readable output file. Optionally, TKPROF can also determine the execution plans of SQL statements.

SQL Profiler

Microsoft SQL Profiler is a graphical user interface to SQL Trace for monitoring an instance of the Database Engine or Analysis Services. SQL Server Profiler shows how SQL Server resolves queries internally. It captures SQL Server events from a server which can be saved in a trace file that can be later analyzed or used to replay a specific series of steps when trying to diagnose a problem.

SQL Profiler can be used for:

- SQL Server performance monitoring
- Measure the execution time for Transact-SQL statements and stored procedures
- Debugging and Diagnostics
- Fine tuning indexes and queries
- Troubleshoot problems in SQL Server

In order to monitor server performance, create a new template or use an existing template that defines the data to be collected. An event is an action generated by the SQL Server engine, such as a login connection or the execution of a T-SQL statement. The details about the events that need to be monitored during the run can be defined in the template. During the run, profiler displays the *event classes* and *data columns* that describe the event data being collected.

The following are the event classes available in the SQL Profiler which can be selected during the run.

- Locks
- Objects

- Performance
- Scans
- Security Audit
- Server
- Sessions
- Stored Procedures
- Transactions
- User Configurable

For all these event classes, the data columns could be configured and filters could be set in order to narrow down the trace.

Know your Limits

Performance bottleneck investigation is a group activity and even if any one of the participant is missing, there is a chance that the result might go wrong. In overall, Performance Test Engineer is primarily responsible for coordinating various members like server administrator, DBAs, architects, network administrator, etc but during the bottleneck analysis and tuning phase, other's participation and team work drives the show and a series of tests are planned to isolate the bottleneck and arrive at appropriate configuration setting based on the subject matter experts advise.

As Scott Barber says, the Performance Test Engineer needs to play the role of Business Analysts, Systems Analyst, Usability Analyst, Test Designer, Test Manager, Functional Tester and Programmer. The role of a Performance Test Engineer is completely different from Functional Test Engineer, which is not understood most of the time. Other than reporting the bottleneck, a Performance Test Engineer should deeply get involved in drilling down to the problem, but by experience one will notice that this is only possible by close interaction with development team.

Every Performance Test Engineer needs to understand the fact that this is the stage where you need development team or architects or infrastructure administrators support to interpret the test results. At the time of bottleneck isolation tests, best guesses are expected to come from these teams. This is the stage where development team takes lead and the Performance Test Engineer end up supporting the development team by planning and executing dynamic number of tests.

Typical bottlenecks on DB server

The DB server would tend to have problems either due to slow SQL or bad indexes most of the times. Check for the following metrics on the DB server:

Cache Hit Ratio	: This value should be high about 90-95% range; else data cache size is too low and might lead to too much physical IO.
Deadlocks	: This value should be zero for the target load, else it refers to design problem.
Table Scan blocks per sec	: This Value should be low for normal transactions (can be high for reporting functions); else it indicates poor or missing indexes.
Parse to Execute Ratio	: This value should be low (<20%) else it could indicate under-sized query cache or flawed query model.
Open Cursors	: This value should be low else it indicates the inefficient query model.
SQL *Net bytes recd/sent from/to client	: This value indicates the data-intensiveness of queries. A high value indicates high amount of network roundtrip calls between the client to/from the db server. The read bytes should be <50% of sent bytes, else it indicates complex application queries should become stored procedures.

Performance Testing Versus Performance Tuning

Performance Testing is often confused with Performance Tuning activity. Though both these activities are related and should go hand in hand, most of the time stakeholders expect a better performing system at the end of Performance Testing. It is assumed that Performance Test Engineer is solely responsible to provide recommendations on improving the system performance, which is not possible. The Performance Testing provides the system performance metrics during the load condition and often it should be combined with the performance tuning activities performed by the respective subject matter experts. Performance Test Engineer should have the foundational knowledge on various web, application and database servers, various technologies and server hardware which aid in identifying the bottleneck. It is the responsibility of the Performance Test Engineer to apply appropriate test strategy to identify and isolate performance bottlenecks and report the observations and possible recommendations. Based on the conclusions from the performance test, possible performance tuning options needs to be arrived at joint consensus along with respective subject matter experts. It is impossible to gain the system wide knowledge to resolve the performance issues effectively unless there is a cooperation of cross functional team.

Chapter 11: Performance Test Reporting

The Performance Test Reporting approach is totally different from the functional test reporting way. In the Performance Testing world, it is not easy to provide the pass/fail criteria as in the case of functional test report. The performance test report development is not as easy as everyone thinks because the test report should contain the symptoms and observations of the performance bottlenecks along with the recommendations. More than that, the information should be abstracted based on the audience of the test report.

Why is it Critical

The Performance Test Report is the document or presentation which contains the details of the performance test outcome. In case of business critical systems, the Performance Test Report would have a direct impact on the business decisions regarding the production release of the application. In some cases, the decisions related to infrastructure upgrades depend upon the test recommendations provided in the Performance Test Report. In case of products, few sections of the Test Report can also be shared with all the clients as the performance benchmark. Though we should accept the fact the test report contents purely depend on the test objective, there are certain basic fundamental things which need to be followed to develop a good test report.

How to report the results of performance testing

The foremost information to be provided in the Performance Test Report should be the overall objective of the performance testing. It should also provide the overview of the application and the business scenarios identified for testing and the user distribution model used for the testing. Though these details are provided in the test plan document, it is a good practice to include these information in the test report.

The main objective of the Performance Testing is to identify the server capacity (Throughput) to handle user load. The business might also be interested in the response time identification for the critical transactions. The Performance Testing activity provides the insight of SPEED,

SCALABILITY and STABILITY of the application. The Performance Test Engineer should have this in mind while writing the test report.

The important information that should be available on the Performance Test Report includes the response time details in seconds, server throughput details in terms of transactions per second or Bytes per second , application errors or other errors encountered during the test and an indication about the hardware resource utilization levels.

Mere copying of few graphs from the performance testing tool doesn't meet the objective of test report creation. The Performance Test Engineer should explain the test results through plain sentences in business English about what is the inference made from each of the graphs provided in the test report. In case of specific bottlenecks, symptoms needs to be explained as all the readers of the test report may not understand from looking at the graph. Each of the graphs should have its purpose detailed along with the clear details of x-axis and y-axis.

The Performance Test Engineer should ensure that right statistics are used and the report is customized to the intended audience. Also the performance test report is expected to have more visuals rather than providing the test results in text format.

Components of a good Test Report

A Performance test report will have different kind of readers like business people (management level people), technical members (architects, designers) and infra members (server admin and DBA). A good test report should meet the expectations of different kinds of readers and should make them understand the test results by adopting right level of abstraction.

It is always a best practice to have a separate section in the test report for providing the test results summary with a high level abstraction of information where technical details needs to be presented at high level but enough information on the test conclusions should be available as major business decisions are made based on this summary.

At a high-level, a performance test report should have the following information readily available.

- Performance Test objectives / goals (Service Level Agreements)
- Test environment and the server configurations
- Tests conducted (along with the deviations from the plan)
- Performance Test Result summary

- System Performance details with enough visuals
- System Performance Bottlenecks details with enough visuals
- Performance test conclusion and recommendations

Components of a Good Visual

The Performance tester should understand the fact that the visuals (various graphs) provided in the test report should contain the following components.

- Graph (Type of chart, X and Y axis , scale, Title and appropriate color)
- Performance boundaries (best , good and worst areas highlighted)
- Legend (supporting the graph)
- Annotations (callouts and highlight marks)
- Interpretation note (symptoms and inferences)
- Conclusion

Response Time Reporting

In order to comment on the system response time metrics, always refer to 90th percentile value instead of average response time. The 90th percentile response time refers to the maximum response time 90% of the users experienced during the test run. The below example represents how the average response time metric reporting becomes a risk. In the below examples, reporting of average response time might lead to the assumption that response time targets (response time SLA is 20 seconds) are met but is not the actual case. In Case1 and Case2, the average and 90th percentile response times are calculated from five samples of response time.

Case 1:

Samples	Response Time in seconds
Sample 1	4
Sample 2	1
Sample 3	30
Sample 4	35
Sample 5	15
Avg Response Time	17
90th Perentile Response time	32.5

Figure 27: Response Time Reporting

Case 2:

Samples	Response Time in seconds
Sample 1	1
Sample 2	1.8
Sample 3	2.5
Sample 4	4
Sample 5	7
Avg Response Time	3.26
90th Perentile Response time	5.5

Figure 28: Response Time Reporting

Most of the times, reporting of 90th percentile response time provides more realistic picture than average response time. The standard deviation between the samples considered for response time calculation should be less 1, which indicates all the response time samples are closer to the mean with less deviation. In such cases, both average and 90th percentile response time values will have almost similar value.

Best Practices of Test Reporting recommended by Scott Barber

In this section, we will discuss on the best practices of performance test reporting. First and foremost will be the usage of response time metrics table recommended by Scott Barber in his user experience series.

The use of Response time comparison table (95th percentile value which represents the maximum response time that 95% of the users experienced during the test run) for all the tests conducted provides a quick summary of the system performance. The rows represent the transactions and columns represent the different load conditions.

95th Percentile Response Time Comparison Table				
Transactions / Users	50 users	100 users	200 users	300 users
Transaction 1	1.2	1.9	2.1	21
Transaction 2	0.7	1.2	1.8	19
Transaction 3	0.9	1.6	2	17

Figure 29: Response Time Comparison Table

Second best practice is to always use a degradation curve to show the user experience at various load condition. It is drawn with various load levels plotted against x-axis and the transaction response time against y-axis. Also use a data table along with the graph to make the graph more readable. The degradation curve consists of 4 regions – single user region, performance plateau region, stress region and knee in performance.

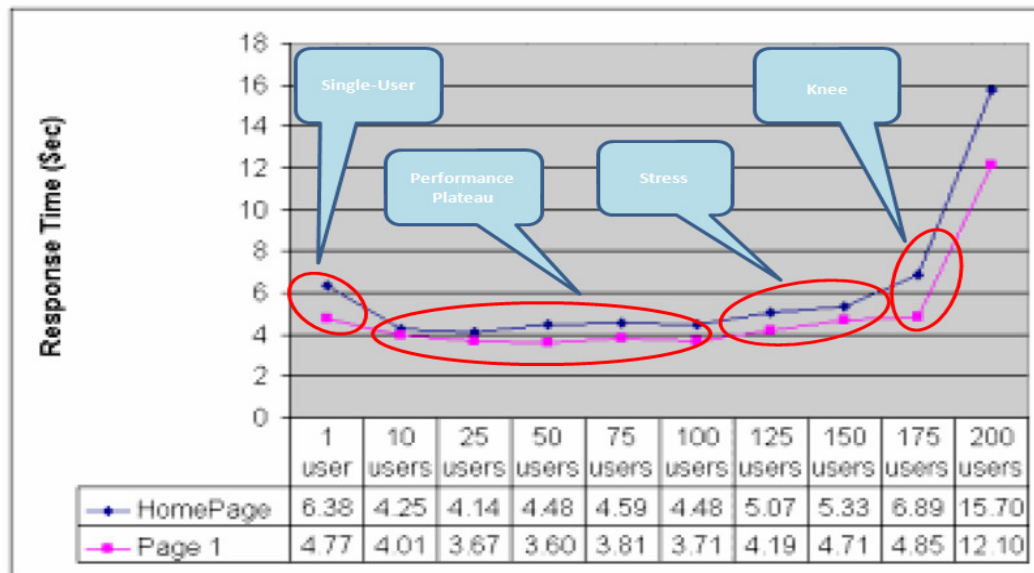


Figure 30: Response Time Degradation Curve

In Single user region, system performance will be little higher than the load condition depending upon system constraints (caching, load test tool dependencies, etc). In Performance Plateau region, the performance becomes steady and consistent where the best performance figures can be seen. This metrics can be used as a benchmark for future comparison. In stress region, the response time starts increasing and ends in knee. Though the server is stressed out, it handles the load. In Knee region, the response time shows a steep increase and system becomes unstable (at 175 users). There will be always a knee in performance testing which is the maximum possible load that a system can encounter.

Chapter 12: Road Ahead – Moving towards advanced Performance Engineering techniques

Moving towards Performance Engineering

As discussed in earlier chapters, mere plans for Performance testing during the end of the SDLC cycle is considered as a reactive approach wherein Performance Engineering approach aims at integrating the performance engineering activities from starting of the SDLC to build a system with high performance. Thus, Performance testing forms a subset of the Performance Engineering activity. Accordingly, the competencies required for a performance engineer is little different from the performance tester.

Software Performance Testing is the act of evaluating the software system for its performance and finding the bottlenecks in the system. Software Performance Engineering is the systematic approach of constructing the software that meets the performance objectives.

Nowadays, IT companies have slowly started adopting the performance engineering activities for critical applications though 70-80% of them have adopted mere performance testing activities.

What is Queuing Theory?

Queuing Theory is the mathematical study of waiting lines (or queues). It is interchangeably spelled as Queueing Theory or Queuing Theory. It is considered as a branch of Probability Theory and Operational Research as it is used in making business decisions. It is used in various fields like telecommunications, networks, traffic management, etc. The notation for describing the queuing model was first introduced by David .G. Kendal in 1953.

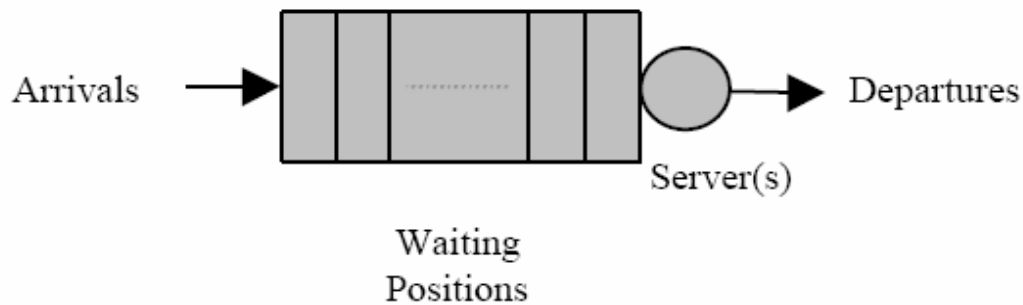


Figure 31: Queuing Theory – Single Queue Model

The figure represents a model of a single queue. Jobs arrive at the queue from the input source, each job bringing a demand for service from the queue. If the queue becomes full, the arriving job is not allowed to join the queue. If the job enters the queue, it may compete with other jobs for service from the queue. When a job completes its service, it departs the queue. We identify models in this class by describing the input source (how jobs arrive), the demand jobs that it brings with them, the rules by which they compete for and receive service, and the capacity of the queue. Thus the elements of the queue includes the input process, service mechanism, queue discipline and output process.

Queuing theory provides answers like mean waiting time in queue, mean response time, service time (service demand), user request arrival pattern and the distribution of users in the system, etc.

Types of Queuing System

The Queuing system can be classified into 2 broad categories as Open or Closed System. A system can be classified as Open or Closed based on certain characteristics it possesses.

Open System

These systems are usually internet facing where the request jobs arrives the system from outside externally and the arrival pattern of the requests are independent of the way the system processes the incoming requests. Thus the load on the system at any point of time is the external decision. The server workload for such systems can be represented as arrivals per unit time.

Closed System

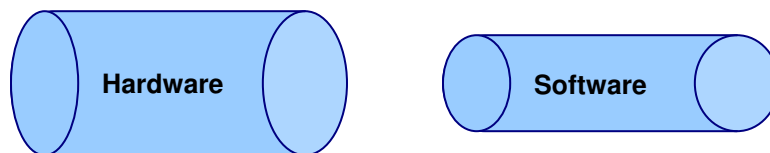
These systems are usually client-server systems where the behavior of incoming request jobs are influenced by the server processing mechanism. The requests circulate continually forming a closed loop. Normally users fire a request, get a response from the server and think for certain amount of time (think time can be zero or non-zero value) and then again fire the request. Thus the load on the system at any point of time is decided by the system. The server workload for such systems can be represented as the number of users and think time.

What can be modeled as Queues?

In the performance context, both server hardware and software can be modeled as queues. But normally modeling the software as a queue is very complex and hence it is very rarely practiced. If you consider the software systems, we can abstract the entire system as two pipes – Hardware and Software.

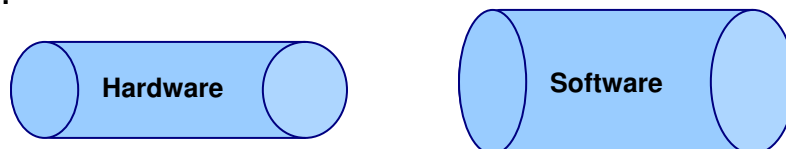
The below two cases explain the situations when we can plan for hardware upgradation and software tuning. In the case 1, the hardware pipe is bigger than the software pipe. If the system doesn't meet the performance targets, then tuning the software (redesigning, code profiling, memory profiling, DB query tuning) might improve performance. Adding hardware doesn't help in this case. In the case 2, hardware capacity is lower than software and hence adding hardware might improve performance in this case.

Case 1:



Tuning Software helps

Case 2:



Adding more Hardware helps

Figure 32: Hardware and Software Bottlenecks

Limitations of Queuing Theory

Queuing theory is mathematically too restrictive to model all the real world conditions accurately as the underlying assumptions of the theory do not always occur in the real world. Thus alternative ways like system simulation have been identified in order to evaluate the system which does not fall under the mathematical scope of queuing theory.

Operational Laws

The operational analysis is a set of basic quantitative relationships between performance quantities. The *operational analysis* is used to establish relationships among quantities based on measured or known data about computer systems.

a. Utilization Law

If we were to observe the abstract system shown in the below figure, we can imagine measuring the following quantities:

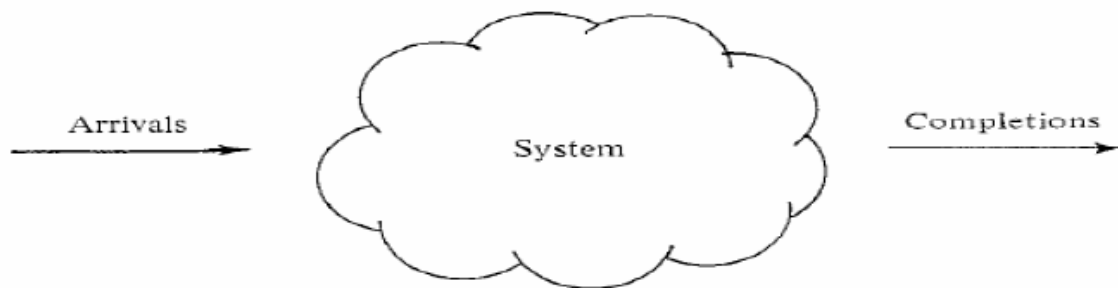


Figure 33: Abstract System

T, the length of time we observed the system

A, the number of request arrivals we observed

C, the number of request completions we observed

From these measurements we can define the following additional quantities:

Arrival Rate λ = A / T , where A is the number of arrivals per unit time.

Throughput X = C / T , where X is the number of completions per unit time.

Utilization $U = B / T$, where B is the time the resource was observed to be busy.

Service Requirement $S = B / C$, where B is the time the resource was observed to be busy and C is the number of completions per unit time.

$$\text{Hence } U = X * S \text{ [as } B / T = C / T * B / C]$$

Utilization, $U = X * S$. This relationship is called Utilization Law.

b. Little's Law:

This law says that "The average number of customers in a stable system (over some time interval), N , is equal to their average arrival rate, λ , multiplied by their average time in the system, T .

$$N = \lambda * T$$

Imagine a small duty paid shop with a single counter and an area where items are placed in the rack, where only one person can be at the counter at a time, and no one leaves without buying something. So the system is roughly:

Entrance → Select items → Counter → Exit

This is a stable system, so the rate at which people enter the store is the rate at which they arrive at the counter and the rate at which they exit as well. We call this the arrival rate.

Little's Law tells us that the average number of customers in the store, N , is the arrival rate, λ , the average time that a customer spends in the store, T . Assume customers arrive at the rate of 20 per hour and stay an average of 0.5 hour. This means we should find the average number of customers in the store at any time to be 10. Now suppose the store is doing more advertising to raise the arrival rate to 40 per hour. The store must either be prepared to host an average of 20 occupants or must reduce the time each customer spends in the store to 0.25 hour. The store might achieve the latter by ringing up the bill faster or by walking up to customers who seem to be taking their time in selecting the items and saying, "Can I help you?" Assume we notice that there are on average 2 customers (N) in the queue and at the counter. We know the arrival rate is 10 per hour (λ), so customers must be spending 0.2 hour (T) on average checking out.

$N = \lambda * T$. This relationship is called as Little's law.

c. Interactive Response Time Law

Consider an interactive client-server system which is composed of N clients interactively accessing the common mainframe server. The clients work independently and alternate between "thinking" (i.e., composing requests for the server) and waiting for a response from the server. The average think time is denoted by Z and the average response time is R . The think time is defined as the time elapsed since a client receives a reply to a request until a subsequent request is submitted. The response time is the time elapsed since the client sends the request until response is provided by the server. Suppose that there are 10 users, average think time is 5 seconds and average response time is 15 seconds. Then Little's law tells us that the system throughput must be $10 / (15+5) = 0.5$ interactions/second.

By applying Little's law, we get the following equation

$$R = N / X - Z. \text{ This relationship is called as Interactive Response Time law.}$$

d. Forced Flow Law

Consider an office where there are multiple reception desks wherein each employee walking into the office needs to sign in the common register placed in the office entrance and then again in the register placed in each of the floor. Hence each employee needs some service from 2 reception desks every day. At any point of time, the arrival rate at the common reception desk must be proportional to the arrival rate at the floor level reception desks. This relationship is expressed by the Forced Flow Law, which states that the flows (throughputs) in all parts of a system must be proportional to one another.

We define the visit count of a resource to be the ratio of the number of completions at that resource to the number of system completions, or, more intuitively, to be the average number of visits that a system-level request makes to that resource. If we let a variable with the subscript k refers to the k -th resource (a variable with no subscript continues to refer to the system as a whole), then we can write this definition as:

$$V_k, \text{ the visit count of resource } k: V_k = C_k / C$$

If we rewrite this definition as $C_k = V_k * C$ and the completion count divided by the length of the

observation interval is defined to be the throughput, then the throughput of resource k is given by:

$$X_k = V_k / X. \text{ This relation is called the Forced Flow Law.}$$

e. Flow Balance Assumption

The flow balance property states that the number of arrivals equals the number of completions, and thus the arrival rate equals the throughput.

$$A = C \text{ hence } \lambda = X. \text{ This relation is called flow balance assumption.}$$

These five operational laws need to be applied to calculate various derived metrics and to validate the correctness of the test metrics collected during performance testing.

f. Bounding Law of Service Demand

In Queuing Theory, service center provides the service for the incoming job request. Each request has service time and queuing time (waiting time). The service time refers to the time required for processing the request and the waiting time refers to the time spend in waiting in the queue to get the service done. If we monitor a resource (say CPU) and it is found to be busy for time T, during which C jobs get completed, then the service demand is calculated as

Service Demand at CPU resource is given by

$$S.D_{cpu} = B_{cpu} / C$$

If the service demand is identified, then the throughput X can be calculated as $X = 1 / S.D$. For example if the service demand is 0.1 seconds, the maximum throughput at that resource is $1 / 0.1$ which is 10 requests per second. The first bottleneck device would be the device with highest service demand. In a system, which consists of multiple service centers (CPU, Disk, Network, etc), the maximum throughput is given by

$$\text{Maximum Throughput, } X = 1 / \text{Max (S.D)}$$

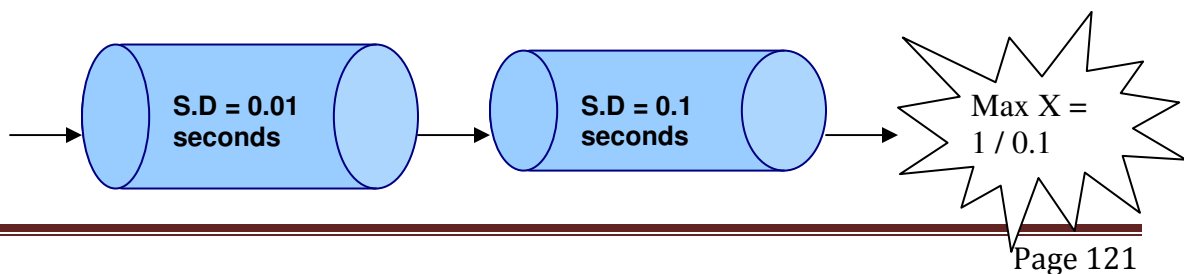


Figure 34: Service Demand Analysis

Capacity Planning

Capacity planning is the process of determining what hardware and software configurations are required to adequately meet the application needs. It helps to define the hardware and network infrastructure needed to handle the expected and forecasted server load. Thus the goal of capacity planning is to provide satisfactory service levels to users in a cost-effective manner.

Capacity planning is carried out by measuring the number of requests the server currently processes and how much demand each request places on the server resources. Using this data, the resources (CPU, RAM, disk space, and network bandwidth) necessary to support current and future usage levels can be calculated.

Why Capacity Planning is required

The first and foremost reason is the user experience factor. Consider a system that can support only a given number of concurrent users while guaranteeing a reasonable response time. As many of us have experienced, when traffic increases on a major Web site that isn't adequately equipped to handle the surge, the response time deteriorates significantly. Am sure that many of us have experienced situations where sites, say payroll, when traffic increases (during last day of the month) on the site, the site isn't adequately equipped to handle the sudden spike and hence resulting in response time deterioration. Studies have shown that if a site's response time is more than 10 seconds, then end users tend to leave. This is generally a bad thing and should be avoided, as there is no secret that a web site's downtime can result in a significant amount of loss to the business.

The second reason is that capacity planning helps you to decide how to allocate resources for a system in terms of the CPUs, RAM, Internet connection bandwidth, and LAN infrastructure needed to support the required performance levels and plan for future growth. Once we understand the limitations of the existing hardware configuration, we can estimate the amount of additional hardware needed to support any increased demands in performance.

Finally, capacity planning is important because it helps to answer the question of what hardware and software infrastructure is required to enable the current system to achieve expected performance objectives.

When is it required?

Capacity Planning activity is usually mandatory for systems which are newly developed. Normally the application capacity planning activity needs lots of inputs from the business analysts regarding the application usage aspects and usually starts from the architecture stage. As more and more details are available, the capacity plan could be refined and baselined. These inputs are provided to the infrastructure team which is very helpful in procurement of the server hardware.

For applications which have not undergone performance testing, it is always advisable to plan for performance testing to identify the software and hardware related bottlenecks. Once the software related bottlenecks are resolved to the maximum extent, then planning for the capacity yields better results.

Normally Capacity Planning involves understanding the business forecasts and planning for the server infrastructure in a cost-effective manner.

References

1. **'Beyond Performance Testing Series'** and **'User Experience, not Metrics series'** by Scott Barber.
2. **'Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning'** by Daniel A. Menasce and Virgilio A.F.Almeida.
3. **Capacity Planning for Web Services: Metrics, Models, and Methods** by Daniel A. Menasce and Virgilio A.F.Almeida.
4. **'The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling'** by R.K.Jain.
5. **'Performance by Design: Computer Capacity Planning By Example'** by Daniel A. Menasce and Virgilio A.F.Almeida, Lawrence W.Dowdy.
6. **'Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software (Addison-Wesley Object Technology Series)'** by Connie U.Smith and Lloyd G.Williams.
7. **'Microsoft patterns & practices Performance Testing Guidance'** by J.D.Meier, Scott Barber, Carlos Farre, Prashant Bansode and Dennis Rea.
8. **Improving .NET Application Performance and Scalability** Forward and select content contributed by Scott Barber.
9. *Load testing, Benchmarking and Application Performance Management for the Web* by Daniel Menasce.
10. **Analyzing Performance Testing Results to correlate Performance Plateaus and Stress areas** by Mike Kelly.
11. **Performance and Bottleneck Analysis** by Sverre Jarpe and Ryszard Jurga.
12. **Web User Behavior Characterization: Techniques, Applications and Research Directions** by Giancarlo Ruffo and Giovanni Ballocca.
13. **An Overview of Load Test Tools** by Buret Julien and Droze Nicolas.
14. **Choosing A Load Testing Strategy** by Borland.
15. Presentations and publications of Scott Barber available at www.perftestplus.com
16. <http://www.loadtester.com>
17. www.Stickyminds.com
18. <http://www.codeplex.com/PerfTesting>
19. <http://www.wilsonmar.com/>
20. http://www.mentora.com/kb_papers.asp