IS YOUR DATABASE STRESSED? LOAD TESTING FOR PEAK PERFORMANCE

Noelle A. Stimely, University of California - San Francisco

SPEAKER BACKGROUND

Noelle Stimely is presently a Senior Performance Test Engineer at the University of California San Francisco. Noelle has been an Oracle Database Administrator, holds certifications in Oracle 8/8i/9i/10g and HP LoadRunner. She has held various positions including Lead Database Administrator, Senior Database Administrator and Oracle Instructor during her fifteen year career.

INTRODUCTION

Have you ever experienced an unexpected rush of users on a production system which nearly ground it to a halt?

Have you ever had to approach management with a request for more resources due to poor performance during a period of budgetary constraints within the organization? This presentation will discuss how to potentially avoid these situations in advance though load testing. A load test can be run to gauge end user experience or stress the database to the point of breakage to determine where the limit lies.

This presentation will discuss the purpose of load testing; how, when and why to test using both freeware and commercially available tools. It will also describe how a properly designed load test stresses the system and how to utilize the results for tuning purposes. I will also compare actual load testing versus Oracle's Real Application Testing tool. Finally, I examine the reports produced and how to best use those results to determine current or future resource needs.

WHAT IS LOAD TESTING?

Load testing, stress testing, performance testing can all mean different things to different people. It is extremely important when embarking on a load test scenario to properly define the scope of the test you are planning to perform. It is also equally important to have management/stakeholder buy-in prior to moving forward to ensure the test has credibility.

Load testing can be defined as placing demands on a system to gauge its response under load. The question becomes which type of load to use. Is the purpose of the test to push the system to the point of breakage? Is the test being run with real world loads? Is it being run with anticipated future loads? Is there a certain aspect of the application which needs to be tested? There are many questions which are raised when determining which type of "load test" to perform. Determining the type is one of the most critical points of load testing. It will drive the design and makeup of the test.

TYPES OF LOAD TESTS

- Transaction Response Testing

  A load test may be useful when there is an issue where certain transactions perform poorly on a consistent basis. Do you get calls from end-users at certain points in the month when specific transactions are run? Was the application recently upgraded and transactions which performed in minimal time now perform unacceptability? Wouldn't you want to know and fix poorly performing transactions prior to going live on the new release/system?

  Define acceptable response time requirements. A good method to use is to state an average and a 90th percentile response time for critical transactions. For example, if the maximum response time is 15 seconds and the 90th percentile is 9 seconds it means there are 10% of response times between 9 to 15 seconds…meaning there are some transactions in this area which should be further investigated. If the test was run and found to have a 90th percentile of 14 seconds with the maximum being 15 it would show the largest response times to be consistent.

  Examples of Transaction Response Testing:
  - Order entry transaction must complete within 8 seconds.
  - AP query must return results within 5 seconds.
  - Airline flight reservation must complete within 7 seconds.
  - PDF attachment must upload within 5 seconds.

- End User Experience Testing

  Testing end user experience under load is also an excellent methodology to use as actual human user satisfaction will be measured. During end user experience testing, the load test is executed while a subset of users works on the system. By conducting end user experience testing, some application issues which may not appear in the final numbers can be identified.

  Examples of End User Experience Testing:
  - Run full-scale web load test while a subset of users logs onto system to conduct normal work activity.
  - Have a sub-set of end users log onto the system in the middle of the load test to gauge performance.

- Stress Test

  Under a stress test, a system will be pushed beyond the limits of normal operation to find the "breaking" point where the system will fail. This type of test would be used for an environment designated as "mission critical" that cannot tolerate downtime. The test would be focused on discovering an application's performance under extreme conditions to determine its availability and reliability. Stress testing would include conditions of heavy loads, high concurrency and limited resources to find the system breaking point. Operating system bugs, resource, locking and code bugs can be located and corrected. The results of this test would provide the necessary information as to where additional resources may be needed to avoid downtime.

  Examples of Stress Testing:
  - Multiple users logging into the system at one time (100 users log in at one time)
  - Heavy transaction loads.
  - Users logging into the system very rapidly (e.g. 1 user every second)
  - Extended concurrency times (25 users remain logged into the system running heavy transactions for an extended period)

Before a load test can be conducted a baseline must first be established. The current system performance must be documented and known prior to a load test being performed. Once a baseline has been gathered, system performance improvement or decline can be effectively identified. After running the load test results can be compared against the baseline and performance problems can be easily identified and tuned.

WHY LOAD TEST?

There are many questions which may be answered through load testing. These questions should be asked whenever deploying a new application, upgrading hardware/software and increasing user/data loads.

- Does the application perform acceptably with a minimum amount of users on the system?
- Does the application respond quickly with additional concurrent users?
- Are there hardware bottlenecks in the system?
- Can the application handle a growing number of users/data?
- What is the maximum amount of load the system can handle?

These are only a few of the questions which need to be asked when designing a load test. The purpose of the load test should be to answer these questions so the system will not crash when the unexpected hits. How many of us have gotten the dreaded phone calls right after an upgrade where the end users are complaining about bad performance? Everyone would like to avoid this situation and if a properly defined load test was conducted prior to go-live there is a very high likelihood these calls may be avoided.

PURPOSE OF LOAD TESTING

Before embarking on any load test the purpose and goal(s) must be clearly defined and documented. All load tests are run to eliminate risk and to mitigate potential performance problems. Some of questions asked might be:

- Will the system perform better or the same after the upgrade?

- Can the system handle another 300 users?
- Open enrollment is 6 weeks away…can we handle the additional traffic?
- The system is not currently meeting performance standards. Can we reproduce the issue and correct it on an identical test system?

The bottom line is performance: specifically, end-user satisfaction. We all know end users are never satisfied but if we can reproduce their experience then bottlenecks can be identified and corrected. The ultimate goal is to **best predict the unexpected**.

In working to define the goal(s) of the load test it is very important to work with the application owner(s)/management to gain buy-in. Most of the time these are the people on the front lines who best know the application and know where the problems lie. These are the people who can tell you where to focus so the problem gets solved and the test has validity. If the wrong area of the application is tested, then it is wasted time and the proper test will not be run. When designing the test it is imperative that a proper test plan be developed which requires the input of the application owner. The application owner will assist in designing the test cases necessary for a successful test. They will be point of contact in creating a realistic test which adequately stresses the web application.

WEB APPLICATION COMPONENTS

A web application is made up of three distinct, but interrelated components. These components are:

- Web server – presentation tier
- Application server – business logic tier
- Database server – information tier.

Any one of these components could act as a bottleneck in the application. The key is to discover which component is the culprit. Another way of saying things is that displaying the pretty parts of web pages is one component. The thinking and business decision making another with core information the last. All of these components work together in a joint unit that makes up the web application.

DEVELOPING THE LOAD TEST

LOAD TEST TEAM

After the load test has been defined, then it is time to develop the test itself. First is to identify the load test team. These will be the people who will assist in developing the test, outlining the goals, defining the hardware/software involved and monitoring the system during the test for performance issues. Critical team members are as follows (every environment is different, but these are some of the primary participants):

- Application owner(s)
- Database Administrator(s)
- Web Server Administrator(s)
- System Administrator(s)
- Network Administrator(s)
- Developer(s)
- Project Manager(s)

Application owners(s) will assist with creating test cases, identifying application performance problems, defining/creating test data, creating a load test profile, setup ids and granting access.

Database Administrator(s) know how the database functions on a day to day basis and will need to monitor the database during the load test. They will also be involved with identifying performance issues and tuning the system.

Web Server Administrator(s) understand the web server environment and can assist you with determining the current and projected website hits. They will both monitor and tune the system.

System Administrator(s) understand the server environment be it Windows, UNIX or Linux. They are mostly involved with monitoring and tuning the system during and after the load test. They can also provide you with the current baseline system performance.

Network Administrator(s) know the network and understand how traffic operates in your environment. They can also provide baseline performance and will monitor and tune the network after the test.

Developer(s) know the application code and how it functions. If your application was designed in-house or you have access to the source code of the application developers will be involved with tuning the code after the test has been run. They will also be involved with successive code tuning efforts when/if the test is repeated.

Project Manager(s) may or may not be involved with the load test depending on the scale. If a Project Manager is involved with the load test make sure they understand the definition of the test and the complexities involved. They can either be your best friend (eliminating scope creep/on schedule) or worst nightmare (do not understand the project/technology).

Developing the proper team and team members is critical to the success of the project. They will guide you in developing, executing and final tuning of the load test. Communication is critical. Listening to the viewpoints of the team members will help you to identify performance problems so you can create a robust test which will accomplish all goals set.

IDENTIFY THE TEST ENVIRONMENT

The test environment used for the load/stress test should be as close to an exact replica as the production environment as possible. Exact replicas of production environments are extremely rare, but for a reliable test a close one is preferred. When considering the system to use keep in mind to compare the logical and physical production and test architectures and the available tools on each. Under ideal conditions, the test environment is an exact copy of the production system only with additional load testing tools and monitoring software installed. A significant consideration of the complexity, load and size of test to run is a measure of the similarity of the hardware, software and network configuration of the test system. It is important to keep in mind the results of the load test are only as good as the test environment in which it is run.

IDENTIFY THE PERFORMANCE ACCEPTANCE CRITERIA

Before embarking on the load test design and creation it is important to have the desired performance characteristics of the application/database established. These performance metrics can be defined by a variety of sources such as application owners, end-users, database administrators or a combination of stakeholders. Typical measures of performance satisfaction may include response time, throughput and resource utilization. An example of response time would be a customer's order details must display in less than 3 seconds. Throughput may be defined as being able to support 30 book orders per second. Resource utilization is defining CPU usage to be under 80% during the load test. Other key points which should also be considered in defining performance criteria are as follows:

- Service Level Agreements (SLAs)
- End user expectations
- Business requirements
- Workload – key scenarios representing reality
- Performance optimization objectives
- Scalability
- Range of anticipated workload conditions
- Resource utilization objectives

These are only a few of the elements to keep in mind when developing performance acceptance criteria. It is important to have these objectives nailed down so to design a robust test which will meet everyone's goals. Having the proper criteria defined and agreed upon in advance will make identifying the performance bottlenecks and tuning much easier and make for a successful load test.

IDENTIFY APPLICATION SCENARIOS

The key to a load test are the application scenarios chosen to test. It is imperative to choose the proper scenarios and mix of scenarios to properly stress the system to identify the performance issues and bottlenecks. Many times the load tester does not have intimate knowledge of the application and needs to heavily rely on end user and application owner feedback. There are many questions to be asked in determining the scenario(s) to use. The first question should be is when and where do the performance problems occur? The people most likely to answer this question would be the end users. They are on the frontlines and know when and where the performance declines. Also, the application owners may have knowledge as well since in many cases, communicate with the end users when the system does not perform up to expectation. It is also important the application owner/end user understand the goal(s) of the performance test. If the people assisting/designing the scenarios have a basic understanding of how the load test is going to work, they can better assist with the project.

Consider what will be tested:

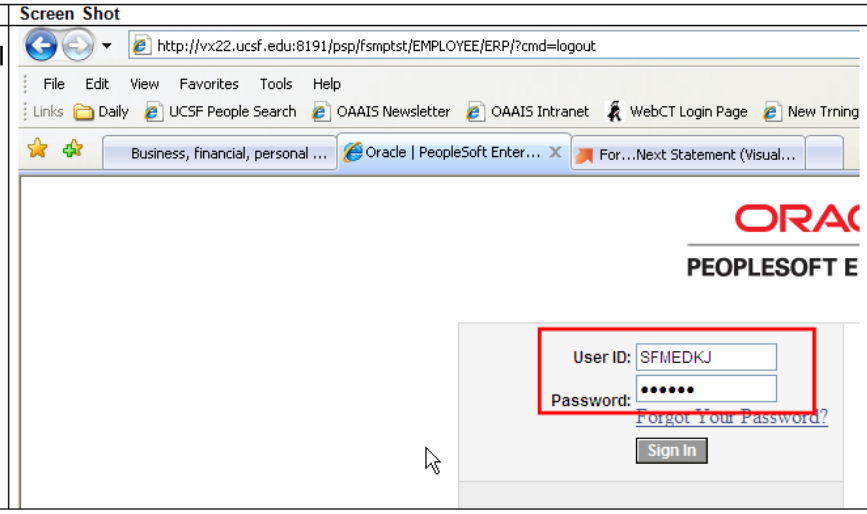
- Which aspect of application going to test?
  - Break down actions in application task into separate tests
  - What actions are performed on the same system concurrently when main task is taking place (i.e. people browsing pages, running reports)

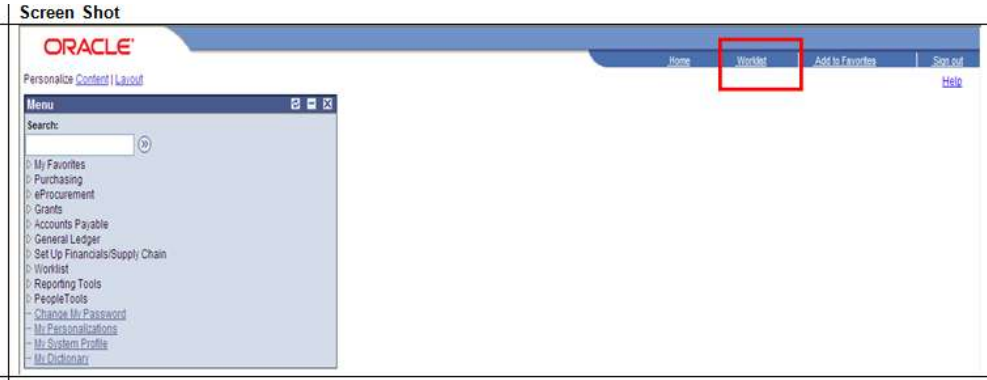
The primary application scenarios are anticipated user actions that generally incorporate multiple application activities. Primary scenarios are those where specific performance goals have been set or have a significant performance impact. These scenarios are chosen because they are commonly executed or because they are extremely resource intensive. An example of an application testing scenario is as follows:

- Log into the application.
- Search airline flights.
- Select specific flight.
- Book flight.
- Pay for flight with credit card.
- Logout of application.

Would the scenario mix include normal day to day actions or specific performance issues? The combination of scenarios to be used is critical to properly load testing the system and identifying bottlenecks.

I typically work with the application owner to provide me with screenshots of each step for the scenario. Each screenshot ties to a page in the application and the steps taken. Below is an example:

Step	Screen Shot
1) Login as SFMEDKJ/PASSWORD	

Step	Screen Shot
2) Click 'Worklist'	
3) Select first link.	

Typically when I run a load test, I am unfamiliar with the intracies of the application. I need a roadmap to follow so I know how it works. After receiving all screenshots for the scenarios you can now record the test in the tool of your choice. Recording the scenario is the easy part. Setting up the load test profile, UIDs and data feeds is much more difficult and is the most important part of the test.

LOAD TEST PROFILE

After the scenarios for the load test have been defined it is now necessary to create a load profile identify the performance characteristics or workload associated with each of the defined scenarios. Each of the scenario(s) must consider the following as part of the load profile:

- How many users have been concurrently logged onto the system at the busiest time?

- How many concurrent users will be used for the test?
- How will users be divided among the script tasks to be tested, i.e. test 4 actions during test and divide users among scripts – 100 for script 1, 200 for script 2, 50 for script 3 and 150 for script 4.

- Number of users: the total number of concurrent and simultaneous users who access the application in a given time frame
- Rate of requests: the requests received from the concurrent load of users per unit time.
- Patterns of requests: A given load of concurrent users may be performing different tasks using the application. Patterns of requests identify the average load of users, and the rate of requests for a given functionality of an application.

- Will user pause before re-logging in again to perform action, i.e. wait 30 seconds before logging in again?
- How long will users spend on each web page of application during test (think time)
- How long will the test run?

APPLICATION USAGE PATTERNS

Now you need to find out the usage patterns of the scenarios being run for the load test. For example, our airline reservation load test will be comprised of 4 scenarios.

- Search and book flight.
- Search for flights without booking.
- Change an existing reservation.
- Ticket agent running a report.

Among these scenarios, you would need to divide your total number of users. The best way to find the division of users is to talk to the application owner. Application owners typically know the application the best and can give you the best guidance on usage numbers. After knowing the numbers then you can tailor the test to meet the performance testing goals. Below is a distribution of scenarios for this load test:

Script	% of users	Total
Search and book flight	20	200
Search for flights without booking	60	600
Change an existing reservation	10	100
Ticket agent running a report	10	100
Total	100	1000

After creating a load profile, begin load testing with a total number of users distributed against your user profile, then start to incrementally increase the load for each test cycle. Continue to increase the load, and record the behavior until you reach the threshold for the resources identified in your performance objectives. You can also continue to increase the number of users until you hit your service level limits, beyond which you would be violating your service level agreements for throughput, response time, and resource utilization.

When designing the load test profile the following needs to be taken into consideration:

- User activity
    - Some transactions occur more often than others and therefore should be a larger part of the test data and scenarios. Most sites often experience large swings in usage depending on the type of users who access it. For example, U.S. retail customers, for example, typically use a web site in the evenings (7:00 p.m. EST to 11:00 p.m. PST). Business customers typically use a web site during regular working hours (9:00 a.m. EST to 4:00 p.m. PST). The functionality of a web site can also have a significant impact on usage patterns. U.S. stock quotes, for example, are typically requested during market trading hours (9:30a.m. EST to 4:00p.m. EST).

      When attempting to model the real world, you should conduct research to determine peak usage ramp-up and ramp-down, peak usage duration, and whether any other load profile parameters vary by time of day, the day of the week, or another time increment. After all the research has been completed you can create a load profile which most resembles real life transactions.
- Ramp-upTime
    - A major component of stress on a site/application is how many and how quickly users log into the system. If a site/application/database is not properly tuned or sized performance can severely degrade or crash if an unexpected rush of users log into the system at once. An example would be a 24 hour airfare sale. If the system was not tested to handle a mass rush of users it could crash. A rigorous load test would have the users logging in at a rate of 1 user per second. Depending on the number of users for the test this rate could easily bring a system down. A more typical ramp-up rate would be 1 user logging in every 10 – 15 seconds. Users would log in according to the ramp-up rate until all users are logged into the system.
- Ramp-down Time
    - Load test ramp down implies gradually stopping the threads during a load run in order to detect memory leaks and check system recovery after the application has reached a threshold. Ramp-down time is the amount of time it take users to log off the system after it has ran with concurrent users for the defined load test time.
- Think Times
    - The time it takes for a user to respond to an application page has a significant impact on the number of users the application can support without performance issues. People have a wide range of different skill levels and abilities. Individual users require varying amounts of time to think about the information that they are viewing. One user may move from one application page to another, barely pausing long enough to comprehend what they've seen; while others need time to contemplate what they've read before moving on to the next page. The length of this pause is called think time. Think time is generally considered to be the length of time from when a user receives the last data packet for the web page currently being viewed to the moment that the user requests a new page.

      Typically a application/database which can support a maximum of 100 "10-second aggressive" users should be able to support 300 "30-second casual" users because both types of users result in roughly the same (600) transactions per minute. Only the most basic of applications/databases would hold true to this thinking as more interactive applications require resources for both active and passive users, meaning that the 300 casual users are more likely to require extra resources than the 100 aggressive users.

      When recording or writing test scripts for performance testing, you should consider how much time each of the various types of users might spend waiting on each page. Having an idea of the mix of application users (aggressive vs. passive), you should be able to create a realistic distribution of timings and, possibly, a randomizing algorithm. A good source of information for estimating think time are web logs. If you find in the web logs users spend about 50 seconds on the site pages I would recommend setting think time for a random time of 20 and 60 seconds.

- Run Length
  - The length of the actual load test can vary based on many factors. You should aim to run the test long enough to get a representative example of true system performance. Typically we run most load tests for a period of 30 minutes and extrapolate performance to the production system. There are some instances where you may want to run the test for a longer duration until the application hits its breaking point. It depends on what you are testing for and what you hope to achieve.
- Workload/users
  - Calculating the number of users for the load test is critical in order to have a valid test which accurately forecasts performance. For example, let's say a travel reservation site reports that the site either has or expects to have an average of 100,000 users per month. The first calculation needed is to find the number of visitors per day. A consideration is site usage. Is the site is used equally seven days a week? Is it used mostly during the week? If it is used on a seven day basis divide the number of users by 30. If the site is used mostly during the week divide the number of users by 21.7 which is the average number of days per month minus weekends.
  - Assuming the site is used equally throughout the week for a per-day user number of 3,333 from the example above. Now it is necessary to figure out how many hours of the day the site is busiest which will differ depending on the site/application. In order to determine peak hour activity the application owner or webserver administrator can be very helpful in finding peak usage. If you are unable to identify peak usage hours then use a 12 hours as a basis.

    Average visits per hour = 3,333 average visitors per day / 12 hours = 278 visits per hour

    In order to find out the average concurrent users for the load test it is necessary to deturmine the average length a user stays on the site. User site visits can be found in the web logs or by your visiting the site yourself, running through the scenarios and timing it.

    If you know the average visit length, then it is possible to calculate the average number of concurrent users. An example of concurrency is if one run through a scenario takes ten minutes then over a sixty minute load test the level of concurrency would be six visits per hour per user.

    Concurrent Users = Visits per hour/60 min/average visit

    If you use the ten minute figure from the scenario above an hourly visit rate of 278 returns an average concurrent user count of 46 which is quite lower than the 100,000 per month figure of user visits.

    After you know how many concurrent users the application should handle the next step is to select the load test amount. Using the example above the application experiences a maximum of 46 concurrent users. If for the purpose of the load test you want to test that the application can handle occasional peaks of 50% more traffic, then you would want to test using 92 concurrent users.

An example of a load configuration might be 4 separate load tests which test different load levels. The first test could start at 25 concurrent users the next ramping up to 50, then 75 and lastly 100 concurrent users. Each test would show you the response times users would experience in low (25 users) and high (50 users) traffic conditions. The load tests of 75 and 100 users would demonstrate what would happen when usage grows or you see an unexpected burst of traffic due to some external event.

Scenario 1 – 25 Users		Load Profile						
Script	**Action**	**# Iterations**	**Think Time sec**	**Users**	**Pacing**	**# Users Ramp Up**	**# Users Ramp Down**	**Duration (mins)**
Script01	Search & Book Flight	1	Use random percentage of recorded think time: Min. 50% - Max 150%. Limit think time to 10 seconds.	10	0 Seconds	1 User every 10 seconds	1 User every 10 seconds	30
Script02	Search for flights without booking	1	Use random percentage of recorded think time: Min. 50% - Max 150%. Limit think time to 10 seconds.	8	0 Seconds	1 User every 10 seconds	1 User every 10 seconds	30
Script03	Change existing reservation	1	Use random percentage of recorded think time: Min. 50% - Max 150%. Limit think time to 10 seconds.	5	0 Seconds	1 User every 10 seconds	1 User every 10 seconds	30
Script04	Ticket agent running report	1	Use random percentage of recorded think time: Min. 50% - Max 150%. Limit think time to 10 seconds.	2	0 Seconds	1 User every 10 seconds	1 User every 10 seconds	30

LOAD TEST DESIGN

After the performance goals of the test have been defined, scenarios established and documented, and a load test tool has been chosen it is now time to start developing the test. The recording of the scenarios is different with each tool chosen, but most likely will involve clicking a button to record and opening a browser to a login page. The following are important items to take into consideration when implementing the test design:

- Application accounts are created for the test with appropriate privileges.
- Test data feeds are properly implemented for testing. Test data feeds are data which is inputed into the system during the load test to create variances in activity thus mimicking real life.
- Validation of scenario transaction on the webserver, database, application server and hosting server.
- Special data used in application which needs to be incremented for proper execution such as product ID.

TEST IDS

When beginning to record the test it is imperative to have at least one login id which has the appropriate privileges to exectute the transactions in the scenario. Without the proper application permissions you may not get far beyond the login page. Use this id to record and execute the initial script. It sometimes takes a while to receive the full list of test ids so this gives you the ability to get a running start on scripting the test. Always after receiving the initial test application id log into the application and go through each scenario step manually to verify the scenario funtions as predicted and the id has the appropriate level of permissions. There is nothing worse than kicking off a load test only to have it fail due to permission issues! The initial scripting of the test is usually the most difficult part of a load test due to things such as permission issues, data issues, correlations and a variety of other problems which can arise.

TEST DATA FEEDS

Another consideration in designing the test are the test data feeds. In order to have a load test which is dynamic and most like a true production enivironment it is imperative to have test data to use. An example of such would be an airline reservation system. When running the load test you would not want to continue rebooking the same flight over and over again since it would not realistic of the application and cached data. For the test you would would want to have parameter files with names of different cities/passengers, etc. to select during the course of the test. An example would be intitially recording the scenario booking a flight from Bangkok to New York. If you did not parameterize the customer and cities when testing the scenario would run over and over again booking the same customer and flight every time. Using the same data for every transaction is not realistic and the data would be cached therefore not exposing potential performance problems. Parameter files need to be created with "dummy" data for testing purposes. In this scenario a list of customer names, departure and arrival cities would need to be generated. When creating the data it is necessary to keep in mind application constraints and work with your application team in generating the files. It is also important to run trial tests using the newly created parameter files to make sure the data works with the application. Another consideration is whether the data is "resuable."

If any of the data entry fields are unique the amount of "dummy" data can be quite extensive. In the airline reservation example above unique fields should not be an issue, but if you were testing an employee new hire system using social security numbers it would be an issue. If the new hire application used a combination of first name, last name and social security number as the unique identifier it would be necessary to have a list of unique "dummy" social security numbers so the test would not fail. Each time the scenario script is run it updates the application with one of the pieces of data. Think of the parameter file as a spreadsheet with a row for each piece of data. The first time the script is run it will select John, Smith, 123-456-7890. The next time the script is ran it will select the next line Jane, Jones, 999-99-999 and so forth. If the script is then ran again it will start at line one selecting John Smith and the test will fail with a unique constraint violation application error. If you are testing against an Oracle database one option you have is to set up Oracle Flashback.

ORACLE FLASHBACK

I typically setup database flashback when I am testing against an Oracle system. Flashback technology was first introduced in Oracle 10g and with this feature in place you can restore to a particular point in time within minutes. I use flashback database because of its ease of use by relying on flashback logs and information mined from the archive logs to access past version of the changed blocks instead of undo data. Using flashback logs and archive data is a much quicker alternative and less disruptive due not having to restore from backup. Flashback is easily setup by enabling archiving and setting up the flash recovery area size. One thing to keep in mind is the amount of undo which may be generated from the load test. The amount

of undo generated needs to be kept in mind when configuring the flash recovery area size. The two initialization parameters used to setup the flash recovery area are:

- DB_RECOVERY_FILE_DEST_SIZE
- DB_RECOVERY_FILE_DEST

The first parameter defines the maximum space available for the flash recovery area files and the second parameter points to the destination on disk. The size parameter needs to be set before the destination can be configured.

The size of the flashback area can be estimated by taking into consideration the size of the datafiles, redo logs, control files and the average frequency and number of block changes during a typical load test run.

Setting up flashback prior to running a load test script which uses unique data can save a good deal of time and work both for the database and application administrators. For example, say you are load testing a new hire application where social security number is the primary key. If you do not have a list of multiple unique numbers for this field it will fail when run for multiple iterations. You need to make sure there is enough data to support multiple runs of the script. Many load tests run for at least 30 minutes or more where the test users login execute the script and logout. Depending on the amount of think-time, pacing and length the scenario can repeat 50 times or more within this period. It is therefore necessary to know the data needed and have enough unqiue test values to cover the testing period. If you use Oracle database flashback a restore point can be set in the database prior to the test execution and data updates. After running the test the database can then be "flashed" back to its original state prior to any updates saving much time and hassle.

If testing against a non-Oracle system you need to keep in mind the role of unique data in the test and how to run trials without error. It may be necessary to have hundreds or thousands pieces of test data to enter into the system or a way to restore the system back to its originial state. When designing and building a load test it is necessary to run multiple trials to validate the scripts, test data, timing and flow of the test to mention a few items. It is extremely important all scenario scripts run smoothly without error prior to the execution of the actual test. It is very embassasing and a waste of many people's time if on the day of the load test a data error or application error is encountered mid-test which causes it to fail due to non-validation.

LOAD TESTING TOOLS – FREEWARE VS. COMMERCIAL

There are many different tools which can be chosen to conduct a load test. The tools range from freeware tools such as JMeter, HammerOra or OpenSTA to commercial tools such as LoadRunner or Silk Performer to name a few.

FREEWARE TOOLS

Freeware tools such as JMeter, HammerOra, Swingbench and OpenSTA offer many features for no cost, but may not offer the complexity needed for a truly robust load test. Many of these tools require additional programming skills which may or may not be available. If the time and skills are available then these tools can offer a truly cost-effective, flexible manner by which to load test a system.

The first freeware tool I will discuss is Apache JMeter. Apache JMeter is a 100% java, open source desktop application designed to test application functional behavior and performance. Apache JMeter may be used to test functional and performance on many different types of resources (files, Servlets, Perl scripts, Java Objects, databases and queries, FTP servers and more). The tool can be used to simulate a substantial load on a server, network or object to test its availability or to analyze system performance under varying loads. JMeter can be used to perform a functional test on websites, databases, LDAPs, webservices or many other areas.

JMeter does not act as a browser. When using web services and testing remote services for all intents and purposes, JMeter does look like a browser, but JMeter does not perform all actions supported by most browsers. Specifically, JMeter does not execute the Javascript found in HTML pages. It does not render the HTML pages as a browser does.

HammerOra is another freeware tool which can be used to load test a system. HammerOra is an open source load test tool which can be used against Oracle, Microsoft SQL Server, MySQL Databases and Web applications. HammerOra is built with the Tcl scripting language and can be used to simulate a real workload using as many client user sessions as your system can handle. HammerOra includes in its install a pre-installed and configured schema with load tests based on the industry standard TPC-C and TPC-H benchmarks. These benchmark tests can be deployed against the Oracle database to test with multiple users. The tool can also converts and replay Oracle trace files and enable Web-tier testing to build robust load tests for an entire Oracle application environment.

Swingbench is a Java based load test tool specifically designed to be used for load testing Oracle databases. The Swingbench install includes four benchmark tests, OrderEntry, SalesHistory, CallingCircle and StressTest. The first two tests are based on the default test schemas included with Oracle's "oe" 11g database. The second two tests are custom written for the application and are designed to simulate a call center and random inserts, updates and deletes to stress the system.

The final open source tool I will discuss is OpenSTA. OpenSTA is written in COBRA and designed to run on Windows. Load tests are architected using the tester's own browser which creates simple scripts that can be edited and controlled with a OpenSTA's scripting language. After the scripts have been finished it can then be played back to simulate many users using a load generation engine. OpenSTA load tests can be designed to scale to hundreds of thousands of virtual users.

COMMERCIAL TOOLS

Commercial tools offer both pluses and minuses. One of the big pluses of using a commercial tool to load test is product support. If you run into an issue scripting or running the load test you can simply pick up the phone and call the software vendor whereas when using freeware one is stuck with either figuring it out themseleves or with "Google" support. A major minus is cost. Many companies do not value the importance of load testing and do not want to spend precious company resources on pricy software. Time is money in most cases and having a website or database crash can be a very costly event which is why it is imperative to load test databases/applications. Using commercial tools to load test can significantly limit the time involved in creating and running the load test depending on tester skill level. If the load tester has a strong programming background using freeware tools might be a good way to go. On the other hand if the tester has weak skills or limited time, commercial might be the best option.

HP LOADRUNNER

The first commercial tool I will discuss is HP Loadrunner which is seen by many as being the industry standard for load testing. Loadrunner was acquired by HP in 2006 as a part of its purchase of Mercury Interactive. It can be used to test a broad range of applictions and up to 51 protocols. Protocols enable communication between a client and server. Examples of this are an airline ticketing system which can use HTTP/HTTPS with Java and Web Services or an application using Citrix as its basis. These scripts can be recorded both in single and multi-protocol modes. Loadrunner is a very versitile tool written primarily in c with protocol extensions enabling it to be used against a variety of applications and databases (at extra cost, of course). The product consists of 4 primary pieces:

- Virtual User Generator (VuGen)
- Controller
- Load Generator
- Analysis

These parts are combined to enable end to end load testing and analysis of an appliation or database.

ORACLE APPLICATION TESTING SUITE

Oracle Application Testing Suite (ATS) is another commercial load test software package. ATS is Oracle's load testing full-lifecycle product built to validate application functionality and test availability under load. Application Testing Suite consists of the following pieces:

- Oracle Load Testing for scalbility, performance and load testing.
- Oracle Functional Testing for automated functional and regression testing.
- Oracle Test Manager for test process management, including test requirements management, test management, test execution and defect tracking.

Oracle Application Testing Suite also contains integrated testing accelerators which are used for testing Oracle packaged and SOA applications. The purpose of the accelerators is to enhanced scripting to create more robust tests.

SILKPERFORMER

Silk Performer by Borland Software is another commonly used load test tool to diagnose application performance. Silk Performer is mostly used to load test Web 2.0 applications which rely on AJAX logic. The latest release, now allows for the use of Web browers (Internet Explorer) to generate load on the system. Using AJAX technology built into Web browsers enables a test which would most likely mimic end-user browsing behavior.

Silk Performer prides itself on its ease of use and a small testing footprint. Ease of use is maintained though the use of simple wizards which allow the tester to input a variety of fields. The load test can also be made more robust during recording by using variying virtual user activities to simulate real world activities to test web connection speeds and network equipment performance. The installation of Silk Performer on the test server only requires a minimal configuration to support the running of thousands of virtual users against an application. The agent software and or controller only require a minium of 512 MB RAM, Pentium 4 processor and 580 MB disk space. After the software has been installed it is now possible to create realistic application scenarios with thousands of users using the latest Web 2.0 technology. Silk Performer also has the following additional components available to use with its load testing stack:

- SilkPerformer Diagnostics – performance analysis package for evaluating Java and .Net applications.
- SilkPerformer Cloudburst – a cloud-based extension which allows a tester to scale a load test to thousands of users without having to setup and manage complex infrastructures.
- SilkCentral Test Manager – integrates multiple aspects of testing activities under a unified umbrella allowing central of the load test management environment.

ORACLE REAL APPLICATION TESTING VS. LOAD TESTING

When load testing tool consideration is discussed among Oracle professionals the tool which most likey might be mentioned is Oracle Real Application Testing (RAT). RAT was first introduced with Oracle 11g. It is a separately licensed option and is only available with Oracle Enterprise Edition. RAT consists of Database Replay and SQL Performance Analyzer. RAT is NOT for load testing, it is more a regression testing tool due to its inability to scale to multiple users. Oracle Real Application Testing is a very powerful tool to use in testing infrastructure changes. It does not have the ability to ramp-up users as needed when conducting a regular load test.

It is possible to use Oracle Real Application Testing along with Oracle's Application Testing Suite to create a robust load test which uses captured data from your production environment. The database test scripts would be created manually or automatically in Application Testing Suite by importing a workload capture from Oracle Real Application Testing replay component. Bottom line is Oracle Real Application Testing cannot be used by itself to create a load test. Both RAT and Oracle Application Testing Suite must be used together for load testing purposes.

MONITORING THE LOAD TEST

Once the load profile has been created it is now necessary to identify the metrics which are derived from the defined performance objectives. The metrics identified will be used to measure the applications performance to compare with your performance objectives. Clearly defined metrics will make identifying any performance bottlenecks easier and better to resolve. Specific areas to be monitored are as follows:

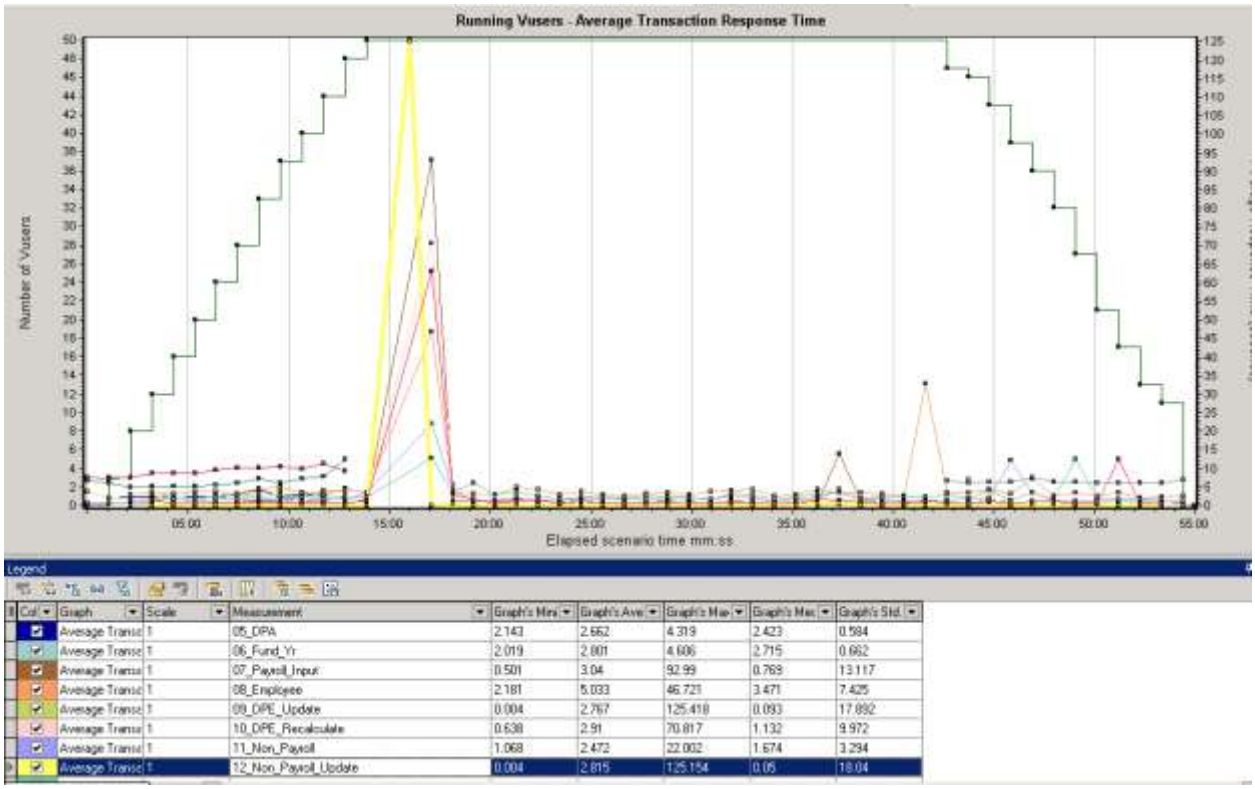
- OS metrics which could include CPU, memory, disk I/O and network I/O statistics.
- Database metrics to watch for are waits, top SQL, Buffer gets, etc.
- Network metrics would include the overall health of your routers, switches and gateways.
- Webserver metrics would include hits per second, number of requests, etc.

After identifying the areas to monitor for the load test it is now necessary to identify the individual team members who will take part in monitoring the test. These people will monitor their respective systems for performance issues while the test is run. These people will monitor during the test and possibly assist in tuning the system after any bottlenecks have been discovered. Typically when we run load tests, there is a bridge call open which includes all team members so they can speak up when issues are encountered as the test is run. Having real-time monitoring occuring while you are running a load test is invaluable as you can see how the addition of more users or faster ramp-up time is directly affecting the system.

ANALYZE AND REPORT LOAD TEST RESULTS

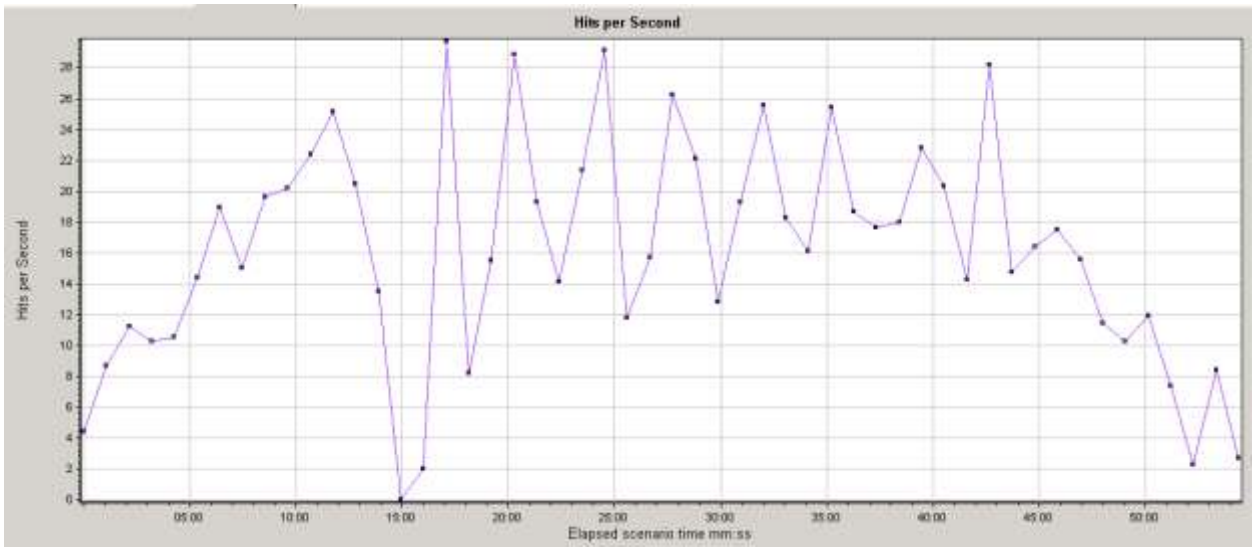
After the load test has been conducted it is time to analyze the results of the test. This presentation is not on tuning and will only show how to use the load test results to pinpoint system bottlenecks. The performance objectives initially set at the beginning of the project should be used to judge the results. For example, checkout should not take longer than 4 seconds under 15 hits per second workload. Other measurements could be CPU utilization for the test should not exceed 80% during any transaction. All of these measurements should have been defined and agreed upon at the beginning of the project in the performance acceptance criteria. Some things to consider when analyzing the results are:

- Transactions in average response time graph breach the performance acceptance criteria.
- Response time that is worse than what the SLA for the customer allows.
- Correlate transactions with other graphs and measurements to find bottlenecks.
- Involve developers, system and network administrators in investigation.
- Overlay response time graph with monitoring graphs and look for performance trends.
- Check standard deviation of transactions if transactions behave inconsistently identify the cause.
- Transactions availability: (count of pass transactions/count of total transactions) x 100. Usually applications have around 99.9% availability. Lower values don't necessarily indicate an issue, but should be verified why lower.



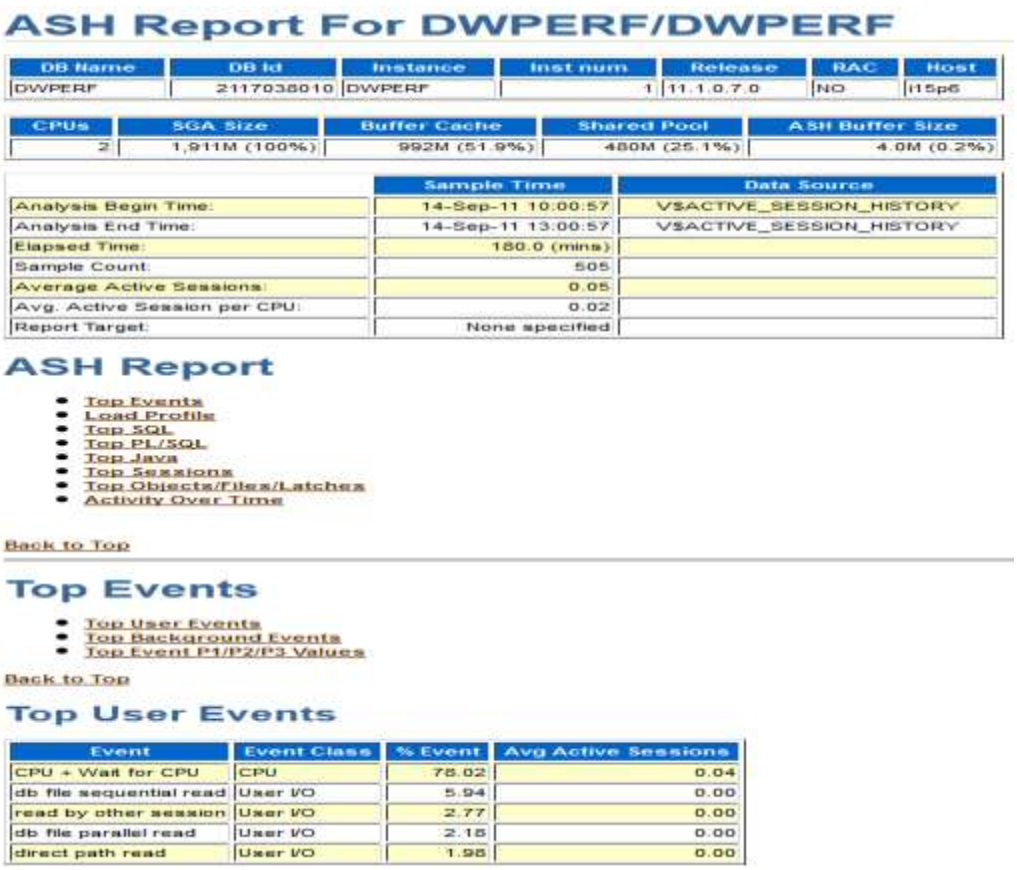
Col	Graph	Scale	Measurement	Graph's Min	Graph's Ave	Graph's Max	Graph's Min	Graph's Std
	Average Transa	1	05_DPA	2.143	2.662	4.319	2.423	0.584
	Average Transa	1	06_Fund_Yr	2.019	2.801	4.606	2.715	0.662
	Average Transa	1	07_Payroll_Input	0.501	3.04	92.99	0.769	13.117
	Average Transa	1	08_Employee	2.181	5.033	46.721	3.471	7.425
	Average Transa	1	09_DPE_Update	0.004	2.767	125.418	0.093	17.892
	Average Transa	1	10_DPE_Recalculate	0.638	2.91	70.817	1.132	9.972
	Average Transa	1	11_Non_Payroll	1.068	2.472	22.002	1.674	3.294
	Average Transa	1	12_Non_Payroll_Update	0.004	2.815	125.154	0.05	18.04

The graph above shows average response time combined with running virtual users. The graph was taken from a load test performed using LoadRunner. Analysis of this graph shows a performance irrgularity occurred between 14 and 18 minutes into the test. The performance hit looks to have taken place when the concurrent level of 50 virtual users was hit. You would then want to take a look at the transactions which were most affected during this time. In this case it looks like the transaction 12_Non_Payroll_Update took the longest. You would also want to investigate the web hits to see if there was a large hit on the webserver during this time. Below is a graph of web hits from the load test:



As you can see 15 minutes into the test there was a huge drop off in web server hits. At this point you would want to get with your webserver administrator to review the webserver logs for this point of the test.

On the database side of things you would want to monitor and pull an ASH report for the load test period. Below is a graph showing database performance for the load test:



ASH Report For DWPERF/DWPERF

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
DWPERF	2117038010	DWPERF	1	11.1.0.7.0	NO	i15p6

CPUs	SGA Size	Buffer Cache	Shared Pool	ASH Buffer Size
2	1,911M (100%)	992M (51.9%)	480M (25.1%)	4.0M (0.2%)

	Sample Time	Data Source
Analysis Begin Time:	14-Sep-11 10:00:57	V$ACTIVE_SESSION_HISTORY
Analysis End Time:	14-Sep-11 13:00:57	V$ACTIVE_SESSION_HISTORY
Elapsed Time:	180.0 (mins)	
Sample Count:	505	
Average Active Sessions:	0.05	
Avg. Active Session per CPU:	0.02	
Report Target:	None specified	

ASH Report

- Top Events
- Load Profile
- Top SQL
- Top PL/SQL
- Top Java
- Top Sessions
- Top Objects/Files/Latches
- Activity Over Time

Back to Top

Top Events

- Top User Events
- Top Background Events
- Top Event P1/P2/P3 Values

Back to Top

Top User Events

Event	Event Class	% Event	Avg Active Sessions
CPU + Wait for CPU	CPU	78.02	0.04
db file sequential read	User I/O	5.94	0.00
read by other session	User I/O	2.77	0.00
db file parallel read	User I/O	2.18	0.00
direct path read	User I/O	1.98	0.00

As you can see during the period the load test was ran database performance was within an acceptible range. If performance is negative you can review the top SQL and Top User Events to see where the waits were found.

REPORTING AUDIENCE

When compiling the analysis reports for the load test it is necessary to keep in mind your audience. Many times management requests the load test report and depending on how technical the audience will dictate what information is included. If your audience is non-technical in nature (loves to see graphs and not numbers), good graphs to include are as follows:

- Running Users – Shows the impact of ramp-up and ramp-down as well as when errors begin to occur during the load test.
- Hits per Second – Shows the amount of load generated by vitual users in terms of hits to web servers.
- Throughput – Shows the amount of load virtual users create on network resources – raw amount of bytes client receives each second as a result of hits on the web server.
- Average Transaction Response Time – Shows the actual load generated by each transaction throughout the entire load test.

The combination of the graphs above will give both a technical and non-technical audience a good overview of the outcome of the load test. The graphs will show where the bottlenecks lie and where to best focus the tuning efforts. These charts can be generated using Excel with data obtained from your system administrators, network administrators and web server

administrators. If you are using a commercial tool such as LoadRunner these graphs are included with the software and are easily generated.

For a more technical audience you may want to include the output of an Oracle Active Session History (ASH), Web server, OS and Network reports. Be sure to work with the respective system owners to boil the reports down to a point where the performance bottleneck can be easily identified for tuning purposes.

LOAD TEST SUMMARY

At the end of the load test summary report should also be included which explains the key results in an easy to read layout. It should list the slowest transactions and explain any correlations with other graphs.

A good example of items to list are as follows:

- Top 5 Slowest Pages
- Top 5 Slowest Transactions
- Top 5 Slowest SQL Statements

The top 5 slowest pages would include the slowest pages encountered in the load test. These pages are ones which do not meet page duration time during the load test. The top 5 slowest transactions are the transactions which do not meet the performance criteria defined at test inception. Identifying these transactions not only show management the areas which need work, but show the technical team where to concentrate their efforts. The top 5 slowest SQL statements will give the DBAs the information needed to tune the application SQL which is typically 80% of performance problems.

FIND BOTTLENECKS, TUNE SYSTEM AND POSSIBLY RETEST

An online application can consist of multiple components which need to be considered in a load test, such as load balancer, web server, application server, database, etc. Performance of the entire application is usually affected by one of these components. Improving or fixing a single component can result in performance improvement, while improving other items may not change anything or will reduce performance. Many times performance improvement can be achieved by changing settings, tuning application code or the purchase of additional hardware.

After any tuning adjustments are completed the decision can be made whether to re-run the test to see if the bottleneck has been resolved. The iterative tests should be run with the same options as the initial test so there is an apples to apples comparison of the results. Load testing is an iterative process and tests may be executed several times before the performance bottleneck is corrected.

CONCLUSIONS

Load testing, like most of IT is both an art and a science. It takes a team effort to design and execute a load test. Load testing enables you to identify application/database issues which would only be visible under extreme conditions: bottlenecks such as SQL execution plans, memory, CPU-bound systems, network bandwidth and disk capacity. And, these are only a few of the potential bottlenecks or issues which could be exposed.

Load testing is focused on the identified performance objectives and key application scenarios with a focus on reliability and stability of the application. It is important that correct methodology is used and all data extracted are properly analyzed. The methodology employed for the load test needs to take into consideration the most realistic reproduction of production workload, volume of data, identification of key scenarios and interpretation of performance metrics.