

The Top 25+ Reasons Web Applications Don't Scale

Including the Common Web Site Performance Bottleneck Checklist

Abstract

This checklist presents the most common performance bottlenecks identified and encountered during the testing of thousands of Web applications by Empirix and its consultants. It also presents the mechanisms that Empirix consultants use in finding those bottlenecks. Our experts have categorized these common problems by system area in an attempt to provide an easy-to-follow guide for spotting bottlenecks within any Web application.

Performance bottlenecks exist in all applications. It is up to application development teams to spot, identify, and fix them before application launch by utilizing tools such as the Empirix e-TEST™ suite or the Empirix e-LoadExpert™ testing service. Using the data gathered from these tools, Empirix customers have been able to increase the performance of their Web sites by as much as 1200%.

What is a bottleneck?

Any Web system that is subjected to testing will exhibit performance bottlenecks. But what is a bottleneck? Below is a list of terms used to describe some of the causes and effects of those bottlenecks, as well as solutions to those problems.

Bottleneck—A term used to describe a limiting resource in the system under test. Bottlenecks are typically identified as the cause of slow or unacceptable performance. They directly affect both the performance and scalability of the system under test. Also, keep in mind that at any SINGLE point in time, only one bottleneck is causing the system to slow for a specific transaction.

Performance—In system testing, this generally refers to the ability of a system to meet the requirements defined prior to testing. Systems are usually said to exhibit good or poor performance, depending on what criteria are used to measure that performance. Performance can be expressed in terms of concurrent users, transactions per day, average response time, or any other measure of speed.

Scalability—The ability of a system to perform as the number of concurrent users or requests increases. A system that can handle the specified user count or request rate as load is added is defined as having good scalability.

Symptom or Problem—The symptom or problem is how a bottleneck will show itself in the system under test. Symptoms are seldom the actual cause of the bottleneck. Symptoms are used together with other data to find the causes of the bottlenecks. Examples may include slow page performance, errors, or crashed servers.

Bottleneck Cause or Indicator—The single point in the system under test that is causing the system to exhibit slow performance.

Solution—The fix used to repair the bottleneck cause.

Measurement Point—The specific metric used to determine when and where a bottleneck exists. Examples are CPU utilization, free memory, response time, and error rates.

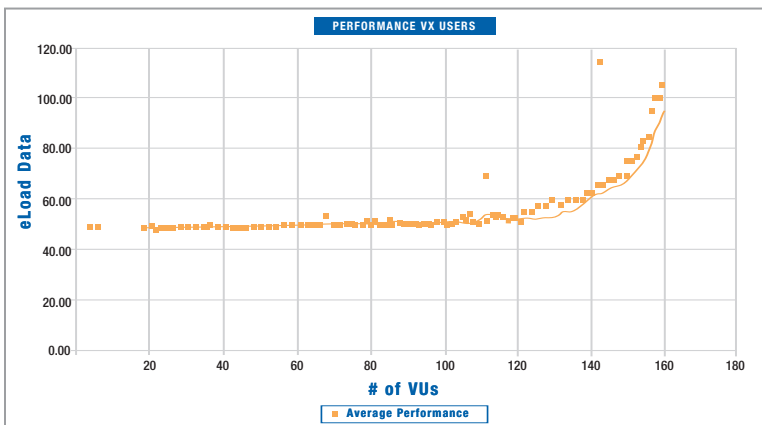


Figure 1. This is an example of a Measurement Point identifying a Symptom.

Empirix customers have been able to increase the performance of their Web sites by as much as 1200%.

Why do bottlenecks exist?

Bottlenecks exist within systems for a number of reasons. These include:

- Inadequate server hardware to support the projected system load
- Inadequate network capacity, both internal and external
- Poor system and architecture design decisions
- Databases that are incorrectly implemented or tuned
- Developers not being mindful of performance considerations during development
- No in-house expertise on system performance
- Tight deadlines and changing requirements

This list may seem like a fairly straightforward group of things to correct; however, during system development, they can be overlooked. Even the best designed, architected, and implemented systems will have some performance problems that make it through unit testing and into system testing. Performance testing is the only way to isolate, identify, and resolve those issues before system launch.

One additional thought to keep in mind is that NO system is infinitely scalable. It is up to the testers, using feedback from the business users and owners of the system, to come up with a series of criteria that will allow them to evaluate and define when the system under test has or has not met performance expectations. The performance team should understand what the immediate performance requirements are and contrast those with growth performance requirements. Systems that meet minimal performance standards can be launched, then improved during subsequent releases, effectively allowing the system to grow as the business grows. But, you must always understand at what level the system, as designed and implemented, will peak, and no longer be able to grow. This level of performance may be defined in terms of number of concurrent users, transactions per second, sales per day, or any other criteria relevant to the business the application supports.

Isolating and Identifying Web Performance Bottlenecks

Introduction

Bottlenecks can reside in any part of the system architecture. Experience has shown that they are more likely to reside in some places than in others. Most of the time, the problems found within Web systems are related to the code developed by the application team or to the database. Server configuration and network setup tend to be less of a factor after the systems have been through some initial tuning because their setup and configuration are fairly straightforward, and there are few 'unknowns' within those components.

The most common problem areas are:

- Database connections and queries
- Application server code
- Web server hardware
- The network itself

Remember, there can be, and often are, more than one bottleneck in systems that have not yet been tuned and that only ONE bottleneck can be found at a time. It's necessary to iteratively test and fix as development continues. This idea is important to remember, as it will assist in identifying bottlenecks. If you keep in mind that for each test the main reason the site is slowing down can be traced back to a single problem, you will spend less time chasing after problems that are not the cause of the bottleneck. Again, the key here is that at any point in time, for a given test, there is only a single bottleneck. Once that bottleneck has been identified and fixed, the next one will be encountered. It is this iterative process that forms performance and scalability testing.

At any single point in time, only one bottleneck is causing the system to slow for a specific transaction.

The chart below presents findings on poor system performance, gathered from thousands of applications by Empirix consultants and engineers.

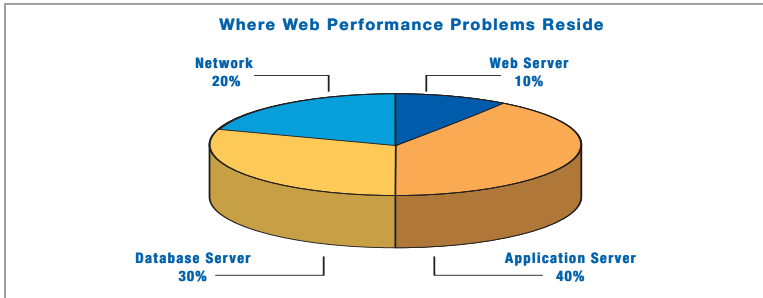


Figure 2. Where Common Problems Reside

The following sections present a methodology that will allow you to quickly and efficiently find, isolate, and resolve performance bottlenecks.

STEP 1—The Testing Approach: Script Setup & Methodology

The Typical Approach

For most Web applications, the application team has an idea of what type of load they expect the application will have to support. This load is usually defined as a number of users. For example, a system might need to support, at peak, 3,000 concurrent users performing typical transactions. The Empirix e-Load® testing tool uses virtual users (VUs), which simulate real users on the site in order to help determine if that goal will be met.

With the target number of VUs in mind, a series of scripts is created. Each script emulates some type of activity that will occur on the Web site. The scripts, using different data and transactions, test various levels of simultaneous users. The results of those tests indicate how the Web site will experience user load.

After testing begins, symptoms of a performance bottleneck typically arise, especially in systems that have been subjected to little or no performance testing. The symptom may be things like very high utilization of the database server or very slow page response times for a critical transaction. The next step—isolating the specific bottleneck that is causing this symptom—can be difficult. Why? Most Web applications involve many transactions occurring at the same time. With many scripts running at the same time, producing many hundreds or thousands of transactions, spotting the problem transaction is very challenging.

Empirix has solved this problem by isolating performance problems and making them easy to identify using the following process.

The Empirix Approach

Contrary to the typical way used to identifying performance bottlenecks, the Empirix approach utilizes short, pointed, and specific scripts that are targeted at key transactions or pieces of key transactions. The shorter the scripts, the easier it is to isolate and identify specific problems. After the bottleneck isolation and identification phase of testing is complete, it is time to run the larger, broader set of tests to validate the performance criteria that have been set for the application.

Level of performance may be defined in terms of concurrent users, transactions per second, sales per day, or any other criteria.

Planning

Before doing any work, it is critical that you plan your load testing. The next few steps will be of use in any test plan.

- ❏ **Strategy**—Lay out the strategy you will use for testing your application.
- ❏ **Objectives**—Document and get buy-in from other relevant parties on the objectives of the load testing.
- ❏ **Scope**—Clearly define the scope by determining what will and will not be tested during the load-testing phase.
- ❏ **Performance Requirements**—Document the system performance requirements, which are used to determine whether the system passes or fails.
- ❏ **Test Cases**—Document your test cases and scripts.
- ❏ **Resources**—Document what resources (system and personnel) will be needed.
- ❏ **Test Data**—Gather and document any data that will be needed for testing. This includes back-end database data and data entered at the UI or front end.
- ❏ **Tools Needed**—Document all tools that will be used during the testing and what role each will play.
- ❏ **Failure Strategy**—In case serious errors or performance problems are found during testing, have a plan in place with buy-in from the all stakeholders on how those problems will be addressed and what type of resources each party is willing to dedicate to resolving them.

Build the Testing Team

Putting together the testing team is probably one of the most overlooked areas of test planning. Performance testing is a much more technical activity than typical functional-regression testing, and therefore requires a broad array of skill sets, which can be addressed by:

- ❏ **Senior Testers**—These are the people who coordinate, drive, and ultimately are held responsible for the performance testing activities. They should have an excellent understanding of performance testing and be able to adeptly use the tool needed to carry out their testing duties.
- ❏ **Database Administrators**—Database experts are needed during the running of the tests to help optimize testing and spot database performance problems.
- ❏ **Network Architects**—Network experts can help spot configuration problems at the network level, including problems with the network itself, firewalls, hubs, switches, routers, load balancers, and SSL accelerators.
- ❏ **Senior Developers/Architects**—These are the individuals who designed the Web site under test. They should be available to answer any questions about functionality and help resolve code-related issues that arise.

Scripting

Scripting should not be taken lightly. Each script should be well thought out and, with the help of the developers, apply to specific functionality and follow the guidelines listed below.

- ❏ Test whichever pages on your site get the most hits (or that you estimate will get the most hits).
- ❏ Even if your home page is not one of your busiest pages, it still has to be fast.
- ❏ Test the most commonly used business transactions on the site.
- ❏ Test the most critical business functions on the site.
- ❏ Test pages and areas that developers suspect may cause bottlenecks.

Putting together the testing team is probably one of the most overlooked areas of test planning.

Most scripts fall into one of three categories:

- End-to-end—Long transactions that cover a wide range of functionality
- Modular—Short, transaction-specific scripts
- Single or low number of pages—Very short, transaction-specific scripts. A transaction in this case is a user action executed on the Web site.

Modular tests or a single/low number of pages work best for isolating bottlenecks. These are the scripts you will want to record for your testing. Use Empirix's e-Tester™ tool to create short tests that focus narrowly on problem pages or functions. (Be sure to use databanks where appropriate.) The end-to-end tests will be used after the bottleneck identification phase to validate final system performance and functionality.

Environment and Load Tool Setup

Next, set up the hardware and software for the tool that will be used. If the testing is being done by a remote testing service, this will include setting up system monitors on the server components and providing access to the application, where necessary, so that it may be tested. In addition, verify that your system under test is ready for performance testing. All servers should be configured and set up, as they would be in production. Databases should be loaded with production data or at least data that is sized appropriately for production.

In cases where replicating the production environment is not feasible or possible, you must take great care in making any assumptions about the scalability of hardware components as they are added. It is inadvisable to assume that a system deployed with two servers will perform two times better than a test system with one server. One could generally assume that the system with two servers will perform better, but benchmarking is required to determine by how much.

One way to build this benchmark relationship is to install identical software on each architecture to determine how the number of servers affects performance and assess each system using identical tests. After the testing, you will have a reasonable comparison factor between the two systems. You could then assume that as long as the hardware remains 100% constant, that the two systems with identical software would perform to that comparison factor. Any changes to the hardware, and possibly the software will affect this factor. You must remember, though, that this is a generalization and an assumption that will not hold true all of the time.

STEP 2—Know Your Architecture

The second step in isolating performance bottlenecks in any application under test is to know your architecture. That is, you must have a firm understanding of the physical components (such as servers, routers, and firewalls) and logical components that make up your system. (For example, know what software is involved and where it is installed.) This will be critical to knowing which counters to watch and where to look for problems.

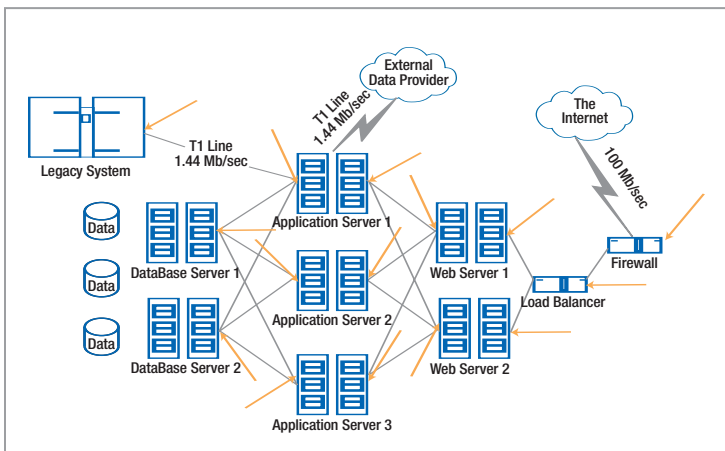


Figure 3. Common Web System Architecture

Modular tests or a single/low number of pages work best for isolating bottlenecks.

During the mapping of your system, make notes of all the systems and monitors you would like to set up. These should cover the critical areas of the system, including the Web server, database server, and application servers. Monitors on network components, if available, should track the performance of the firewalls and load balancers.

The server diagram below shows the complexity that can exist within Web systems. The green arrows represent only some of the places bottlenecks can reside.

STEP 3—Server Management

Server management is the step in which you determine which pieces of the system should be monitored for performance. It provides data that supplements the performance counters collected by the testing tool.

Each piece of the system will be outlined using the data that you gathered in Step 2. From that data, gather a listing of counters that should be watched on each system. Below is the list of counters that we recommend you use (at a minimum). More counters can be added during testing. Empirix ServerStats™ is a tool that makes it easy to configure and add counters at any time. ServerStats details some generic server counter types that you will want to watch on each system component. Although not every system will have this information, most will. This list does not include errors that would be received on the client (through front-end monitoring). The e-Load tool handles errors received on the front-end.

Firewalls

- ❏ Total current connections—This is the number of current established connections. Make sure it does not exceed the limitations set by the license for the firewall.
- ❏ Total current SSL connections—If this number is higher than 100 second per Web server, you might be running into SSL bottlenecks and should consider using SSL accelerators.
- ❏ System utilization (CPU) —This allows you to see if the load is causing the firewall to max its capacity rating.
- ❏ Throughput—Make sure the throughput does not exceed the firewall's rating.

Load Balancers

- ❏ Total current connections—Number of current established connections. Make sure this number does not exceed the license available on the load balancer.
- ❏ Load balancing split—This verifies that the load balancer is properly spreading the load across all available servers.
- ❏ System utilization (CPU) —This is used to see if the load is causing the firewall to meet or surpass its capacity rating.
- ❏ Throughput—Make sure the throughput does not exceed the firewall's rating.

Web Servers

- ❏ System utilization (CPU) —Make sure that the system is not exceeding CPU capacity. Greater than 70% signals a potential problem.
- ❏ Memory utilization—Verify that the available memory on the Web systems is sufficient. Pay close attention to this if the application requires any SSL traffic.
- ❏ Throughput—Verify that the Web servers can keep up with their rated network capacity.
- ❏ Current connections—Make sure that connections and user sessions are in line and are balanced among all Web servers.
- ❏ Disk IO—On a Web site with a lot of images or other static content, make sure disk access is fast.

Application Servers

- ❏ Memory utilization—Check for memory leaks and JVM memory usage.
- ❏ CPU—High CPU can be an indicator of a well-designed system working very hard or a poorly designed system trying to work but being bottlenecked by other problems.

Server management provides data that supplements the performance counters collected by the testing tool.

- Connection pool—Watch the pool of connections to the database and verify they are being used.
- Queue—Watch the queue. When requests start to queue, you have hit a bottleneck.
- Queue wait time—A queue is acceptable, as long as the wait time in the queue is not excessive.
- Disk IO—Verify that disk access is fast.

Database Servers

- Memory utilization—Verify that the database has ample memory for cache. The more memory, the better performance you can expect.
- CPU—High CPU can be an indicator of a well-designed system working very hard or a poorly designed system trying to work, but being bottlenecked by other problems.
- Table scans—Verify low table scans on all critical business transactions.
- Table locks—Make sure that processes are not locking critical tables because that will slow access to them.
- Parse ratio—If parsing is high, convert to stored procedures and update the query plan cache.
- Cache hit ratio—Make sure there is ample cache to store the data.

External Systems

- Current connections—Make sure there are enough connections to handle the requests to the external systems
- Bandwidth consumption—Verify the bandwidth to the external system providers is sufficient
- Queue wait time—A queue is acceptable, as long as the wait time in the queue is not excessive.

STEP 4—Running Tests and Finding the Bottlenecks

During this step, you will execute the tests you have created. This step assumes you have created your scripts, set up e-Load and ServerStats, and have an environment that is ready for test. The following are the quick steps you will use during this phase. Remember, the key here is to run each critical transaction on its own to find any problems. Then fix, retest, fix, retest. When the transaction performs acceptably, move on to the next transaction.

After each transaction is performing acceptably on its own, you can move on to a larger scenario test that will verify all the transactions working together. The idea is to resolve the issues in each piece of the application first, and then to move on to the big picture.

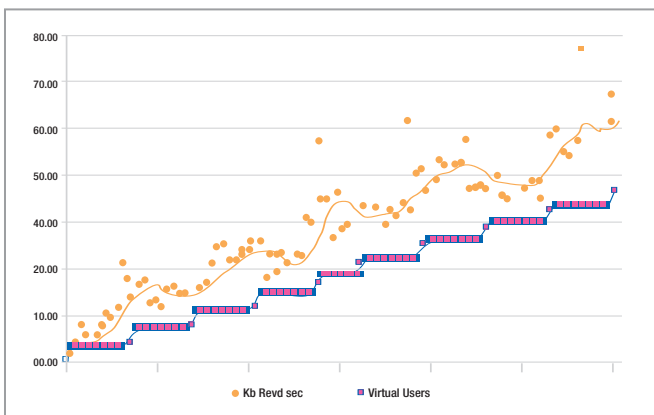


Figure 4. Example of a user ramp with stabilization time

The key is to run each critical transaction on its own to find any problems.

- 1) Start e-Load and ramp at a rate that allows the system to stabilize before adding more users. The time between pages should be set at a fixed amount in order to allow you to understand exactly where the delays are occurring.
- 2) Keep the ramp constant until you notice a slowdown in response times or page/hit rates.
- 3) Check the servers you are monitoring to see if any of them are exceeding CPU or memory resources:
 - a) If so, note which machine is exhibiting a problem, refer to your network diagram, and determine what piece of code is running the problem transaction. Your DBA or programmer can identify potential inefficiencies. Fix and retest.
 - b) If resources are not being exceeded, you may have a problem with network bandwidth or another resource constraint, such as queuing of requests to a specific resource or an outside system. Identify, fix, and retest.

The Checklist

Below is a list of common performance problems that Empirix consultants have found during the thousands of load tests they have conducted. This list is in no particular order, except that each group of common problems is grouped according to the systems they are found in most commonly.

Application Server/Application

Problem #1

You are observing high CPU usage on the application servers.

Indicator: CPU counter, measured from the application servers via ServerStats or through other monitoring tools.

Solution: There is no universal solution. To sort through the many possible causes of this behavior:

- Check the application code of the transaction being executed, and verify that it is optimized.
- Verify that the application servers have been tuned according to manufacture specs.
- Consider, if you have taken the steps outlined above, that additional hardware may be required.

Problem #2

Low memory conditions are observed on the application servers.

Indicator: Available memory, measured from the application servers via ServerStats or through other monitoring tools.

Solution: Verify that there is sufficient memory in the server for the application deployed on it. Add additional memory if needed. Also, verify that the memory isn't INCREASING during the load run, which would indicate a memory leak in the application code.

Problem #3

Low dynamic pages per second, as per manufacturer specifications.

Indicator: Observed through pages-per-second counter within e-Load or on an application server counter.

Solution: Verify that the application server has been tuned according to manufacturer specifications. Check out the application code on the application server to verify it is optimized and not inefficient. Most often, this behavior is caused by inefficient coding on the application server.

Problem #4

Low activity on all systems, but response times increasing; threading issues.

Indicator: Adding load to the system does not increase throughput, but the systems are not exhibiting common stress conditions (CPU or memory usage high).

Solution: Verify that there are no asynchronous transactions causing conditions of queuing. This is fairly common with access to legacy systems through external system connectors.

Problem #5

High CPU usage on the application server and low dynamic pages per second are observed.

Indicator: CPU counters on the application server as observed by Server Stats or other monitoring tools and the page rate counter in e-Load.

Solution: Typically the result of inefficient code including: poor development practices/sloppy coding, inefficient SQL pulling too much data from backend. Check out the code being called for functions being executed, pages that are dynamic that COULD BE static, verify system has been tuned to manufacture specifications. If using a Java platform, make sure all relevant performance patches and manufacturer suggested settings are configured.

Problem #6

Low activity on application server, long response times, and timeouts are observed on the clients at high user loads.

Indicator: Response time indicator in e-Load and CPU counters in ServerStats for the application servers.

Solution: Check the application server configuration: Verify that there are ample threads to execute requests. Many application servers have a setting that allows for the number of concurrent processes to be executed at one time. Verify this is set correctly.

Problem #7

Long response times on the client and low database usage.

Indicator: CPU counters on the database monitored through ServerStats and response time as indicated through e-Load.

Solution: Application Server Connection pooling; verify app server is configured for optimal pooling.

Problem #8

“Spiky” response times. Server performs well, and then response times jump for a bit, and then goes back to normal.

Indicator: Generally observed through CPU usage on the application servers and response times on the client side.

Solution: For Java application servers, configure application servers to cluster, even on single machines. The advantage here is if an application server goes into JAVA garbage collection, it will not affect the ability of users to get dynamically generated pages. Another option is to verify the Java runtime parameters are set up for “frequent” garbage collection (minimized time spent doing garbage collection). There is a switch that can be used on runtime startup to watch the garbage collection intervals. Verify that garbage collection happens when necessary, but not too often.

Problem #9

High levels of disk IO or CPU usage on the application servers.

Indicator: Watch the disk IO or CPU usage on the application server through ServerStats.

Solution: Verify that application server logging levels are set to minimum levels for production. Determine if the level of logging that is currently configured is necessary. The more logs the system must parse and build, the more time will be lost for processing user requests.

Problem #10

Certain pages seem to load slowly, even under low load conditions.

Indicator: High usage of database or application server, as indicated by ServerStats. May also be indicated by high amounts of data being moved from the database server to the application server and high queue wait times on the application server.

Solution: Optimize all slow dynamic pages. Check functions, algorithms, and other code within the .asp, .jsp, java beans of whatever development environment you are using for maximum performance. A common problem is returning large amounts of data, when only pieces are needed. Limit database queries to only return what is needed. For example, only return 20 records from the database at a time instead of returning 500 and letting the application server parse down to only 20.

Problem #11

Certain pages seem to load very slowly, even under lower load conditions.

Indicator: Usually observed by watching CPU usage on the application server during page execution. CPU will “spike” with each page it.

Solution: Limit object includes in the application code to only those that are necessary for the page functions. It is common for developers to include objects that are not needed on .jsp or .asp pages. Minimize those includes so the application server does not have to parse and build those pages.

Problem #12

Poor ASP server performance during initial testing.

Indicator: Application servers are not performing as indicated by performance specifications published by the manufacturer. Usually measured as rate of dynamic pages per second.

Solution: If using ASP, apply all the performance tuning suggestions from Microsoft (MSDN Article: Q253146). This should be standard procedure for all Microsoft ASP deployments prior to testing.

Problem #13

Poor Java server performance during initial testing.

Indicator: Application servers are not performing as indicated by performance specifications published by the manufacturer. Usually measured as rate of dynamic pages per second.

Solution: If using a Java platform, use the fastest JVM you can find. Optimize that JVM by removing functions you do not need for your site. Specify the vendor suggested switches for starting the JVM. These will typically limit memory consumption, help to control garbage collection, and specify server usage or workstation usage. This should be standard procedure for all Java deployments prior to testing.

Web Servers

Problem #14

High CPU usage on the Web server during testing.

Indicator: Watch the CPU counters on the Web servers using ServerStats.

Solution: Verify that the servers are not being overloaded. This is especially common when using high numbers of SSL transactions. Add more Web servers or use an SSL accelerator to handle the SSL traffic.

Problem #15

High amount of disk IO activity on the Web server and lower page throughput than expected given the server specifications.

Indicator: High Disk IO or page throughput as indicated by ServerStats.

Solution: Verify that all source files for the Web and application servers are stored on physical drives other than in the log files. Web servers can perform excessive amounts of logging, and this IO should be able to occur with minimal disruption to the reading of files.

Problem #16

Using NFS shares for Web files, log files, or source files will slow down access significantly.

Indicator: High amounts of network activity to the system sharing the files.

Solution: Store files locally or use high-speed SAN devices for common storage.

Problem #17

Slow page download times over modem connections.

Indicator: In e-Load, when using modem emulation, download times indicate numbers that exceed suggested specifications.

Solution: Check overall page sizes, which should not exceed 75k (for use with modem users). Verify that page size is acceptable and suits users of all networking speeds.

Problem #18

Performance not as expected when running the Netscape iPlanet Web server.

Indicator: Indicated through pages per second or hits per second, as indicated through ServerStats.

Solution: When running Netscape (iPlanet server) running single process to handle connections, add more processes—Netscape suggests four processes, with up to 512 simultaneous connections. Netscape server does not run efficiently as a single process.

Problem #19

High levels of disk IO or CPU usage on the Web servers.

Indicator: Watch the disk IO or CPU usage on the Web server through ServerStats.

Solution: Verify that Web server logging levels are set to minimum levels for production. Determine if the level of logging that is currently configured is necessary. The more logs the system must parse and build, the more time will be lost for processing user requests.

NETWORK

Problem #20

Low CPU usage on all Web systems but poor performance.

Indicator: Response times are increasing as users are added, but the Web systems are not showing any signs of stress.

Solution: Verify that any external system interactions and the network are not exceeding their rated capacities.

Problem #21

Excess packets dropped on network.

Indicator: Watch the SNMP counters on relevant switches or routers to determine the loss rate.

Solution: This problem is generally caused by incorrectly configured network devices or exceeding network capacity. Either add network capacity or verify that the configuration of the network is correct.

Problem #22

Lost connections (12xxx errors in e-Load).

Indicator: e-Load indicates 12xxx series errors during load tests.

Solution: Local LAN exceeding rated capacity (typical when systems do not have segmented external and internal networks). These problems typically are the result of poor configuration of network hardware. Check out the firewalls and load balancers, routers and switches involved in the system.

Problem #23

Network capacity is being exceeded at low numbers of users.

Indicator: SNMP counters through ServerStats on switch performance.

Solution: Use an independent network for all internal traffic. This will minimize the noise on the line that serves the users. The server components will generate a generous amount of traffic communicating back and forth to the various servers. This can be minimized by putting the Web traffic on its own NIC cards and subnet.

Problem #24

When users are being ramped, the firewall starts to send errors about no connections available or the system starts getting timeout errors trying to connect.

Indicator: Errors received by the users in e-Load or through ServerStats configured to watch the firewall.

Solution: Verify firewall licensing and configuration. Verify that the number of concurrent users is not exceeding the rated capacity of the firewall. Remember that each user can instantiate more than one, usually up to four, connections to the server.

Problem #25

Under load, not all Web servers are being utilized when there is a load balancer in use.

Indicator: High amount of connections on a specific Web server as indicated through ServerStats.

Solution: Verify load balancer configuration is set to balance the load properly. Adjust the load balancer rules so that it balances across all Web servers.

Problem #26

Poor network performance.

Indicator: Watch switch performance. Can also be tested utilizing e-Load to verify network throughput.

Solution: Check that all network connections are set up to force 100 MB/sec full duplex or better.

External Systems

Problem #27

All internal Web systems seem to be performing satisfactorily, but overall system performance is still not adequate.

Indicator: Poor performance as indicated by response times, pages per second, or transactions per second. Performance criteria not meeting expected values.

Solution: Verify the performance of external systems and external system interactions. It is common for a call to an external system to be somewhat of a 'black box' and not know what is going on. Many times the developers of those interfaces will make available counters to validate their performance. Some examples of these types of systems might be tax calculation Web sites, shipping tracking Web sites, and credit card validation Web sites. Check out queuing on the external systems, synchronous requests occurring on external calls, and external WAN speeds not being adequate for the traffic needed.

Database

Problem #28

Large number of queued requests on the application server, but the application server usage is low.

Indicator: The queued request indicator from the application server as indicated by ServerStats.

Solution: Verify that there are ample connections to the database from the application server. Many times this is configured as the JDBC pool. Increase the amount of these connections to see if performance increases.

Problem #29

Slow query response.

Indicator: Specific page or transaction is slow.

Solution: Verify that queries and stored procedures are using available indexes in an efficient manner. Check query plan and optimize query. Use a database-profiling tool to spot slow queries. Optimize those queries by minimizing sorting (e.g., sort by index defaults), eliminating unnecessary triggers, minimizing the use of temp tables, using parameter-based stored procedures whenever possible. Optimize SQL statements to properly select on index values, minimize the result-set returned, and minimize the use of joins, order-by, and group-by clauses.

Problem #30

Overall slow performance within Oracle or another database.

Indicator: High database utilization for low numbers of users, especially when testing has just started.

Solution: Verify database has all performance patches and optimizations applied. Verify that the database has been tuned according to manufacturer specifications.

Problem #31

Slow overall database performance and large amounts of data moving between the application server and the database server.

Indicator: High CPU usage on database server and large amount of data moving between application and database servers as indicated through Server Stats.

Solution: Verify using stored procedures or using queries that will not have to be recompiled at each run. Stored procedures have several advantages. First, they are precompiled. Inline SQL must be compiled or parsed each time it is executed. Second, inline SQL code is difficult to optimize and source control. Stored procedures can be more easily optimized by the DBA.

Problem #32

High amount of disk IO on the database.

Indicator: Disk IO number is high as indicated by ServerStats.

Solution: Store logs, data, and indexes on separate disks for the database servers. Use RAID configurations for optimal speed. Verify database is using indexes for all queries. Make sure that database has sufficient memory and the cache is configured.

Problem #33

Poor multi-user database performance.

Indicator: High number of locked tables or blocked transactions, as indicated through ServerStats database monitors.

Solution: Review the transactions being executed and optimize them for multi-user scenarios. Modify the strategy for database locking.

Problem #34

Slow query response.

Indicator: High number of table scans as indicated through ServerStats database monitors.

Solution: Find or add the missing index. Rebuild the index or create it.