Chapter 9

Test Environment Construction and Tuning

A good stress test uncovers any problem areas before deploying the web site to production. The time and expense of performance tests pays off in the gains realized in customer satisfaction and overall site reliability. So far, we've discussed building good performance tests for your web site. Now let's cover building a realistic environment in which to run these tests.

In short, you cannot go cheap in building your test environment. Poor infrastructure impacts performance and stress tests more than any other type of tests you'll run against your web site. Don't expect to meet your web site performance goals if you build the test environment with cheap cable, underpowered client machines, and low-bandwidth networks. To get the most out of your performance test efforts, the test environment must mimic the production environment as closely as possible, given the ever-present constraints of time and expense. Obviously, if the web site contains hundreds of servers, you cannot recreate a huge server farm for a performance and stress test. In these cases, scale down the test to a few machines, keeping the scale proportional to the production system. Figure 9.1 shows a typical small test cluster with peripheral systems.

Proportion remains important in "scale environments" in other ways as well. Again, if the 20 machines in the production cluster use 200 database connections, assume that the small cluster of two machines needs 20. The same goes for network bandwidth, test clients, and other resources you might need for the test. Also, when using a smaller environment, keep in mind the behavior of the system at two servers might be drastically different than with twenty. The web site team must test scalability in a small-scale environment.

Of course, the reduced costs of small-scale environments lead to reduced coverage of all the situations your large-scale web site faces in production. If you test with a small-scale environment, you may encounter undiscovered problems when you move to production. In fact, you may encounter problems you can *only* recreate on

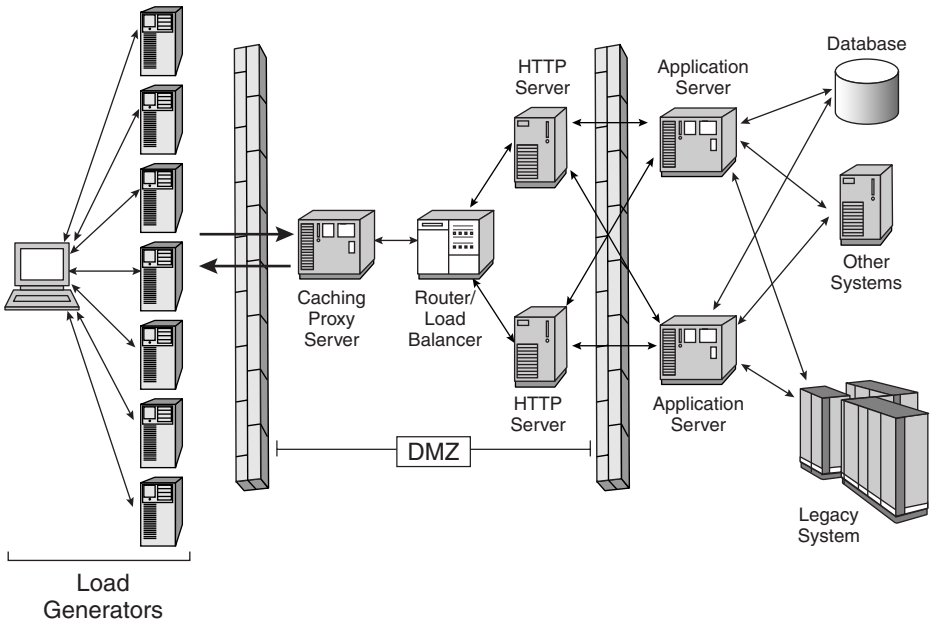

Figure 9.1 An example test environment

the production web site. The best test environment remains a full-scale reproduction of the production web site, whenever possible.

The Network

The network plays an enormous role in performance and stress testing, yet it rarely receives the attention it requires. The network often becomes a hidden source of problems and limitations during the test. Only after days or weeks of fruitless testing and problem resolution of higher-level components does the network come under scrutiny. Consider the network before testing begins. Estimate the amount of data the network must carry and plan sufficient network capacity for testing.

Network Isolation

As always, we want the test network as close as possible to the production setup. However, even if the production environment shares its network with other systems,

build an isolated network for the stress and performance testing. Some companies balk at the expense and time that building an isolated network requires. However, without an isolated network, you cannot control the traffic volumes on the network during testing. We're often amazed at some of the traffic moving across an internal network, even during normal business hours. A few of the things we've seen in the field include

- Heavy network traffic from employees connecting to a company-sponsored gaming server
- Network "storms" created by a faulty network card somewhere on the network
- Massive system backups moving across the network for hours at a time.

In short, if you don't control the network, you don't control its traffic, either. Figure 9.2 shows some of the daily network traffic fluctuation factors. Some test teams try to work around this problem by running their tests at night or early in the morning. Sometimes this works, but often they discover their company uses the network 24 hours a day. As mentioned earlier, the networks might be in use at night for large data transfers and backups.

If you must use a non-isolated network, try to set up a network protocol analyzer (more on how these work below) to monitor network traffic volumes. *Before you do this*, check with your corporate network team. Many companies restrict or forbid the use of network protocol analyzers on their networks. And, yes, they can find out if you install one anyway. The best solution remains building an isolated network.

Remember, you need *repeatability* to perform an effective performance and stress test. Some test teams, however, cannot get the same results twice from the same performance test, even if they don't make any adjustments to the system under test. In such an environment, you cannot effectively tune the system. If you make a change, and see better performance, you cannot know if you've found a legitimate improvement or if the environment actually factors in the solution. Testing on an open network introduces more environment variability than you can ever hope to control.

Network isolation is probably one of the most overlooked issues in performance testing, and yet it is one of the most important. If you can't know from one run to the next whether a change made improved things or not, your test dissolves into an exercise in dart-throwing as you struggle to understand your web site bottlenecks and optimal tuning.

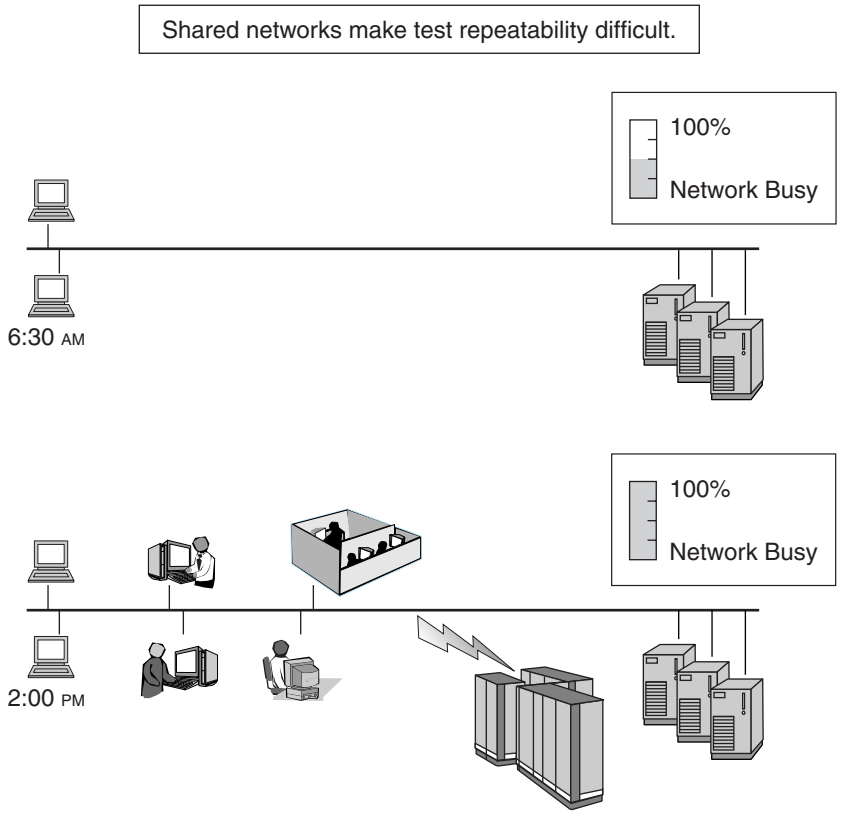

Figure 9.2 Shared network impact

Network Capacity

The network carries all the data for the test (and subsequently for the production environment). You need to do some network capacity planning prior to building the network, and certainly before beginning any tests. The network needs enough capacity to carry the data generated by the test. This begs the question: What data does the test generate, exactly? Here's a list of some common data packages the network handles during a performance test:

- User/server communications
  - User HTTP requests
  - Server HTML responses
  - Embedded HTML page elements, such as *gifs*, *jpegs*, and JavaScript
  - Embedded frame elements (usually resulting in additional page requests)

- Mid-tier (server-to-server) communications
  - HTTP session data sharing within a cluster
  - Application database transfers
  - Traffic to services servers (for example, a stock quote server)
  - Traffic to mail/messaging services
  - LDAP requests/responses
  - DNS requests/responses
- Back-end (server-to-host) communications
  - Host databases transfers
  - Host application communications

Usually the largest network impact comes from a few major sources:

- HTML responses
- Embedded elements such as *gifs* and *jpegs*
- HTTP session data sharing within a cluster
- Application database transfers

However, as we've discussed before, every web site and web application differ. A good understanding of your particular web application, the kinds of pages it returns, and its interactions with other systems helps you put together a reasonable network traffic estimate.

e-Commerce Network Capacity Planning Example

e-Commerce web sites require lots of network bandwidth. As we discussed in Chapter 5, the pages returned by these sites usually contain lots of embedded pictures in the form of *gifs* and *jpegs*. Users browse the pages of the e-Commerce web site and perhaps search for groups of items ("Show me a selection of coffee pots," for example). Because these pages return items the customer probably hasn't seen before, the user's browser does not contain cached copies of the images. To build a network estimate, figure out the average and maximum page size returned to the users. Decide how frequently the maximum page size might go out to the users, and whether it merits special calculation as a worse case scenario. Again, this exercise results in estimates. Use the performance test to validate these estimates.

Let's make the following assumptions about our e-Commerce site:

- Average page size: 45KB
- Maximum page size: 70KB

Let's also assume you want to use a 100Mbps Ethernet network to support the web site. If the web site team hopes to move 100 pages per second through the site at peak

(remember, we *always* plan for peak loading), the network receives *sustained* traffic of 4.5MBps.

```
45,000 bytes per page * 100 pages per second = 4.5MBps
```

Note that 4.5MB approaches the upper bound of sustained traffic we consider acceptable for a 100Mbps Ethernet web site. Under ideal conditions, a 100Mbps network could potentially handle 12.5MB of sustained traffic:

```
100Mbps / 8 bits per byte = 12.5MBps
```

However, Ethernet networks lose efficiency because of traffic collisions and retransmissions. Academics tell us to expect an Ethernet to support 66% of its potential capacity, or about 8.3MBps of sustained traffic. For planning purposes, we prefer to use a more conservative estimate of 5MBps of sustained traffic.[1] This allows the network to absorb unexpected traffic spikes. Web sites operating through a switched network are the exception to this rule. A switched network acts as a point-to-point network, which makes it more efficient. When dealing with switched networks, we raise the planning estimate to 8MBps.

The 4.5MBps we estimated above tells us the peak *outbound* traffic on the web site. How much *inbound* traffic does the web site carry? Inbound traffic consists of HTTP requests, which we usually measure in terms of a few bytes. For example, if the average HTTP request requires 100 bytes, the network load generated at peak is

```
100 bytes/request * 100 requests/second = 10KB/second
```

This is less than 1% of the outbound HTML content traffic. Keep in mind that TCP/IP generates lots of overhead packets to support "guaranteed delivery" (a hallmark of the TCP/IP protocol). So, as a rule of thumb, we estimate inbound traffic at about 20% of outbound traffic.[2] In this case, we'll use a planning estimate of

```
4.5MBps * 20% = 900KBps
```

Now, let's consider the transfer of data between the application servers and the application database on the network. Let's assume each user request requires the transfer of 10KB of data from the database. At peak, this gives us the following traffic:

```
10,000 bytes per request * 100 requests per second = 1MBps
```

1. Thanks to Carolyn Norton for sharing her expertise in this area.
2. Thanks to Susan Hanis for sharing her TCP/IP expertise here.

If the web site uses HTTP session persistence, we need to account for this traffic as well. If each request generates 1KB of HTTP session traffic, we estimate the following HTTP session burden:

```
1,000 bytes per request * 100 requests per second = 100KBps
```

Lots of installations grossly underestimate the size of the HTTP session data they maintain for each user. Check the HTTP session database, and check the size of the data stored for your users to properly size the average HTTP session. See Chapter 2 for more details on HTTP session management.

Other factors also influence the amount of HTTP session data on your network. HTTP session caching combined with affinity routing reduces the data read from a persistent HTTP session store. On the other hand, if your application server vendor supports a distributed HTTP session scheme, this sometimes generates more network traffic, depending on the implementation.

The network burden estimates so far add up to the following, as shown in Table 9.1.

Table 9.1 Estimated Network Traffic Burden

Data	Network Burden
Outbound HTML/static elements	4.5MBps
Inbound HTTP requests	900KBps
Application data transfer	1MBps
HTTP session data transfer	100KBps
Total	6.5MBps

At 6.5MBps of sustained traffic, this web site exceeds our planning limit of 5MBps for the 100Mbps Ethernet. The web site needs a more sophisticated network plan. We might consider a switched 100Mbps network rated at 8MBps, but this doesn't give the web site a lot of room for growth or for unexpected load peaks. A gigabit network might be a better fit for this web site.

Network Components

Networks consist of more than cable. Any number of switches, routers, load balancers, and other equipment make up the test environment network. The brands and types used largely depend on what's available for the test, as well as company standards. Frequently the test team reuses network equipment from other test projects or receives the equipment as a loan from a production group. Often, the equipment becomes a part of the test network without anyone really understanding how it works or how to configure it properly. The end result is a piece of equipment that may

impact web site performance in ways difficult to detect without specialized monitoring equipment and skills.

Keep in mind the following questions when dealing with network components:

- Is the component rated for this network? We regularly find customers trying to use network equipment rated for a 100Mbps network on a gigabit network. Also, your diagnostic equipment, such as network protocol analyzers, may not work with ultra-high-speed networks.
- How was the equipment used previously? Borrowed equipment often contains filters, limits, and other settings still in place from a previous assignment. For example, if you borrow a router previously used in a production web site, its current settings may intentionally limit HTTP connections to a defined maximum. While this protects against denial of service attacks in production, it limits the load you're able to generate against the web site during the test. Review the settings on all equipment to avoid unintentional limitations on performance.
- Does the component support all the features the test environment requires? Routers, load balancers, and other components may or may not have features required by the web site. Know your requirements for key features such as affinity routing and SSL support, and understand how well the equipment supports your desired configuration.

Network Protocol Analyzers and Network Monitoring

A network protocol analyzer monitors traffic flows across a network and allows you to find out exactly how much traffic passes over the network during a test. If permitted, we highly recommend you use a network protocol analyzer to validate the network load during testing. Don't be surprised if your test generates a very different network load than you originally estimated. Often your estimates fail to consider all the factors at play during execution. Also, you may find the page sizes you used for your estimates incorrect.

Network protocol analyzers vary greatly in sophistication and expense. The most expensive network protocol analyzers cost tens of thousands of dollars and work with high-speed networks. These high-end network protocol analyzers usually come with tools for analyzing the network traffic at various levels in the protocol stack. For test teams on a more limited budget, many free or inexpensive network protocol analyzers exist, and they usually run on a machine already connected to the network. They provide limited functionality and don't always work with high-speed networks. However, for providing a gauge of network activity on many classes of networks, these

tools work just fine. Just be sure whatever tool you pick works with the network it will monitor. This requires checking the tool's tolerance for network type and network speed. Chapter 12 discusses these tools in more detail, and Appendix C contains a list of some vendors of these products.

Warning: Many companies actively monitor their networks for network protocol analyzer activity. Some companies consider a network protocol analyzer a breech in their security and do not allow them. We've visited companies where using a network protocol analyzer on a company network results in instant job termination. Please take this warning seriously; obtain permission before inserting a network protocol analyzer into a network.

The Servers

The network connects all the components of your web site. Let's next go up a level and consider the server components you need in the test environment.

Application Server Machines

In terms of configuring the web application servers, strive to make the test configuration as close to the deployment configuration as possible. If you're deploying to a farm of four-way Sun machines with 8GB of RAM, you should use the same equipment during your testing, if at all possible.

Even though the Java web application server might port to different platforms, web applications do not perform and scale the same on a one-way NT box as on a four-way Sun box. If you must choose between fewer servers for your test or smaller servers, pick fewer servers. When you use fewer servers, scale the other components of the test (databases, HTTP servers, and so on) proportionally as well.

If you deploy the web application on multiple machines in a cluster, then test on multiple machines in a cluster as well. Get a representative number: If you plan to deploy on more than four machines, test on more than two. If your target environment contains eight application server machines or more, then four application servers in your test environment should suffice, as long as you scale the rest of the test web site proportionally to match.

We often encounter test teams trying to test and tune all the software used in their web site (application servers, databases, HTTP servers, etc.) on a single machine. However, this technique doesn't work if the production web site actually uses multiple machines. For example, many production web sites separate their HTTP servers

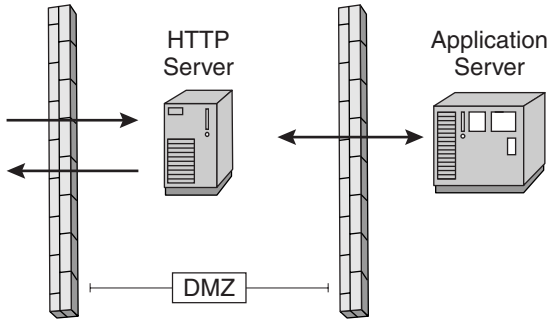

Figure 9.3 Typical DMZ configuration for application servers

from their application servers. Often the production team places the HTTP servers in a DMZ with firewalls in front and behind, as shown in Figure 9.3. It is impossible to successfully performance test for this configuration with a single server machine. If the web site uses SSL, for example, the HTTP server needs so much CPU for encryption/decryption that it chokes the performance of the application server.

Database Servers

Most production web sites use mid-tier database servers to hold application data or data specific to the web site (such as an HTTP session database). Often the web applications access these databases on every user request. Despite their central role in the operation of the web site, the mid-tier databases sometimes receive little or no tuning before the site enters production deployment. Often the test team lacks the database administrator (DBA) skills required to tune the database properly. Thus the team ignores the database unless they stumble across a specific problem in production. The problems we see frequently with databases usually fall into one of two broad categories: Poor software configuration or poor hardware configuration.

Poor Database Software Configuration: Indexing

By far the biggest database tuning problem we encounter is poor table indexing. Usually the problem starts this way: The test team receives a database backup from the production database staff. They dutifully load the database definition and the data itself, but never bother to build indexes for the tables they've just loaded. In fact, they may not even know what an index is, much less how to build one. Chapter 6 discusses how to put together the performance test team, including DBA skills needed to tune environments using databases.

The index allows the database software to find elements in a table without scanning the table repeatedly. This saves tremendous amounts of resource, particularly CPU. Also worth noting, the DBA may build new indexes and remove old ones as usage patterns change over time. New indexes apply even when web applications use existing databases. Often these databases contain indexes tailored for existing applications, but they might require new ones to better support the web application. By monitoring database reports, the DBA determines the web application's usage patterns and makes appropriate adjustments. The DBA may also review the SQL used in the web application to find out where indexes might be most beneficial.

Poor Database Software Configuration: Internal Resources

Web applications receive many simultaneous requests. In turn, they make a proportionally large number of simultaneous requests to the database servers supporting them. These database servers need enough resources to support large volumes of simultaneous requests. These resources include things such as buffer pools, cursors, and sockets to support high-bandwidth operation.

Regrettably, we often see high-concurrency web sites struggling to pull data from databases tuned for small, fat client applications. New applications and usage patterns require a fresh look at the tuning parameters for the database. Do not assume the database is tuned properly because the DBA made a few tweaks some years ago for the usage patterns of a fat client application.

Poor Database Software Configuration: Caching

Particularly for sites with enormous application databases, the database cache becomes very important for optimal site performance. The cache holds the results of the most common queries and makes an impact on sites with large catalogs of items but a few frequently accessed "best sellers." These items return from the cache quickly without an expensive retrieval from the hard disk.

Caching helps some web sites, but a few cannot take advantage of this feature. For example, if every query submitted by the web site is unique (if, say, every query contained the user's account number), the cache may not return a hit even though the query may return items retrieved many times before. The web site team might want to work with the DBA to build queries better able to use the caching mechanisms available.

Poor Database Hardware Configuration: I/O Management

Eventually, all databases interact with the hard disks to read or write data. Database tuning often focuses on the CPU required by the database server without focusing

on how to manage the storage required by the same database. We sometimes see very large multiple-processor database machines spending most of their time waiting to access one tiny hard disk. Adding CPU does not solve disk I/O problems.

If the database server spends most of its time waiting to read or write from the disk, tune the I/O resources. If the database supports it, try adding a multiplatter disk array to the database server. By using multiple disks, the database spreads out the read and write operations for better simultaneous access. The database reads and writes spread out across multiple points rather than queuing up on a single disk. Figure 9.4 shows an example using multiple disk platters.

Likewise, the hard disks themselves often contain tunable features. Some disks allow the system administrator to specify buffering to the disk, which may also improve

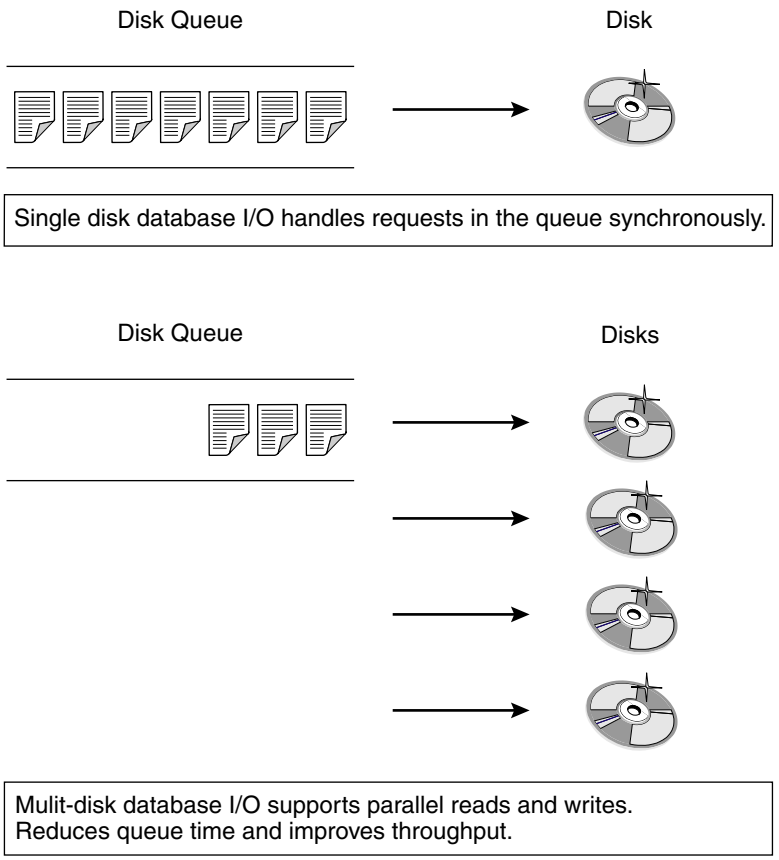

Figure 9.4 Single-disk versus multidisk database I/O

performance. Some databases write more efficiently to disk than even the native file system itself. For example, IBM's DB2® database product provides mechanisms for "raw" I/O management (DB2 bypasses the file system to write directly to the disk) and for multidisk data writing.

Finally, when spreading the data to multiple hard disks, don't forget about the database logs. Databases keep detailed logs for rollback and recovery purposes. Every action taken by the database must be logged, and this requires a write to the hard disk. "Striping" these logs across multiple hard disks, if supported by the database, often improves disk I/O wait times.

Legacy Servers

Legacy servers and their applications present daunting challenges to web site test teams. First, the legacy system usually resides beyond the control of your test team, on a host machine somewhere. (The system may not even reside in the same state as the test team!) Secondly, customers and internal user applications run against the legacy systems during the day and sometimes even during nighttime hours. Finally, these systems require tuning, just like the web application and web site, particularly if the system doesn't currently support a high-volume, multi-threaded application.

The first issue, *access*, requires teamwork to resolve. You need help from the teams supporting the legacy systems to collect performance statistics during the test runs. Also, since the test team usually does not have the skills necessary to tune the legacy systems, you need the legacy support team available to make any tuning adjustments on these systems. We frequently encounter teams attempting a test using legacy systems without assistance from the legacy support team. Invariably, these tests fail because the test team cannot on their own find and resolve bottlenecks on these systems.

The support teams also come in handy to resolve the next issue: *scheduling*. Do not run performance tests against a legacy system in use by customers or in-house applications. Performance tests, by their nature, attempt to drive resources involved in the test to their full utilization. For example, if the web site makes use of a host database shared by multiple applications, the performance test might drive the database to 100% utilization, effectively locking the other applications out of the database. Figure 9.5 illustrates this point.

The legacy system, however, might be available in the evenings or late at night. Many test teams use these off-shift hours to run performance tests against web sites using these systems. This solution works well if you keep a couple of points in mind.

- The tests cannot run unattended. Despite the impact on your social life, performance testing requires many, small iterations to obtain measurements and make tuning adjustments. You also need assistance during these runs from the legacy support team to monitor their systems and make necessary performance adjustments. Later, after finalizing the key performance adjustments, the subsequent long run and stability tests, happily, run overnight with little human intervention.
- Know your legacy system backup and maintenance schedules. Again, the support teams for the legacy system play a big role in obtaining this information. The legacy systems frequently use the nighttime hours for system maintenance and backups. The performance test cannot run during these operations: Backups and maintenance take lots of system resource and result in abnormal readings for the performance test.

  Before scheduling an off-shift test, discuss the maintenance schedule with the legacy support team to either fit the test into an existing schedule or to modify the schedule to fit the test. Also, be aware of maintenance on other systems with the potential to impact the test environment. For example, a system backup crossing the network might disrupt your testing.
- Sometimes you cannot run tests against the production legacy system at any time. If this is the case, consider rebuilding the legacy system inside a test system.

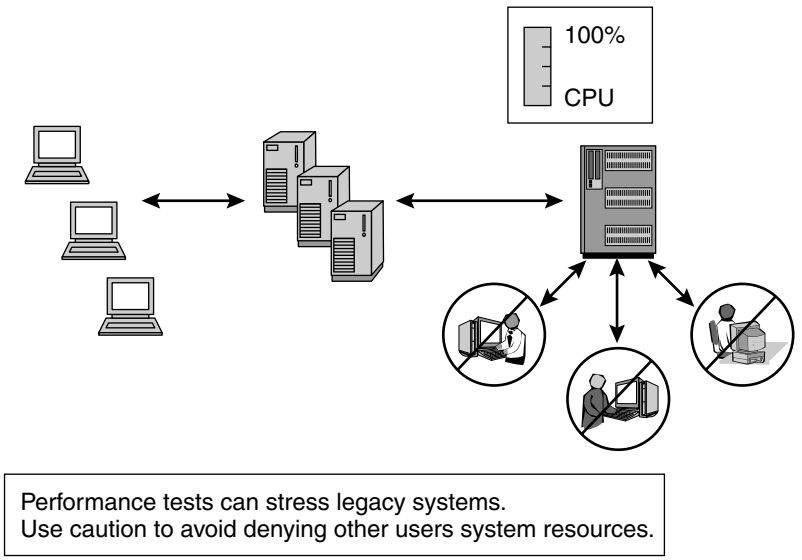

Figure 9.5 Testing live legacy systems

Finally, tuning the web application requires tuning the legacy system to which it connects. When visiting web site teams testing web applications using legacy systems, we commonly encounter the following problems:

- The web site generates more traffic than ever seen before at the legacy system. Remember, a lot of legacy systems exist to support fat client applications. These applications generate less traffic than a popular public web site. The legacy system needs tuning to support such a site, and a good performance test normally brings this issue to light. Legacy application tuning tasks fall into the following general areas:
  - Adding connections to handle the large number of simultaneous requests from the web applications.
  - Providing a larger region for the legacy application to execute.
  - Increasing memory buffers for the internals of the applications (particularly important for database applications).
  - Adding processing capacity. Sometimes the increased burden on the host system warrants additional processing capacity.
- The connection software used by the web application also frequently requires tuning. Here's a list of some of the problems we run into regularly with connection software:
  - The connection software needs more bandwidth. The software often requires adjustment to support the large number of simultaneous connections the web application requires.
  - Home-grown connection software designed for a fat client application does not usually work for a web application. Most web applications run as multi-threaded applications. They need the ability to send multiple, simultaneous requests to the back-end legacy host system, something most fat client applications don't require. Often the web site team discovers too late that their custom connection software isn't thread safe for multi-threaded access. Whenever possible, we recommend using commercially available connection software. Otherwise, check the connection software early on for thread safeness.

Production web sites often use back-end legacy systems. Most of the problems we encounter with these systems during performance tests find their roots in a lack of communication between the web site team and the legacy support teams. The legacy support team belongs in the planning and tuning effort for every web site using the resources under their care.

Host Database

Host databases share all the problems of other common legacy systems, as well as a few unique issues. Many host databases contain huge amounts of data and support multiple applications. The company cannot afford to use these production databases

to support a performance test. This leaves you with the problem of replicating a massive database for performance testing. Of course, the performance test works best when it runs against the full database. Some databases, however, contain terabytes of data and use many, massive disk farms to contain everything. You cannot afford to move the full contents of these databases to the test environment.

For extremely large databases, you require a working subset of the data in the test environment. Taking a meaningful subset of a monster database requires skill, as the data tends to be intertwined with cross-references. Taking a sample indiscriminately often results in data the web application cannot use: The cross-references the web application seeks in the data do not exist in the sample. The sample must also be large enough to generate the proper behavior from the database. An undersized sample, for instance, might be largely held in the database's cache, resulting in overly optimistic database response times.

Also, coordinate your test scripts with the sample data. If you create the scripts using the full database, make sure the data referenced by these scripts exists in the sample as well. For example, if the script tries to purchase a toaster not available in the test database, the script fails.

Finally, apply the key tuning parameters of the production database to your test database, as appropriate. These include things like buffer sizes, indexes, and connection settings. Since your sample database represents a scaled-down model of the production database, scale the tuning parameters as appropriate for the sample.

The Load Generators

Load generators (part of a performance test tool, as discussed in Chapter 8) generate the test environment "traffic." A performance test requires hundreds or thousands of users to simulate production conditions. The load generator uses prewritten test scripts to simulate the users and their activities at the web site. Generating large numbers of "virtual users" requires the right supporting equipment. In the field, we often find test teams running expensive, state-of-the-art performance test tools on old, underpowered PCs retrieved from storage. Without sufficient supporting equipment, the best tools on the market cannot generate sufficient load to properly test the web site. If the load generator cannot provide sufficient load to the web site, the web site never achieves the target throughput. Often the test team misreads the symptoms of this condition and spends weeks tuning the web site when, in fact, they need to add capacity to the load generator machines.

Likewise, the test team needs a network analysis of the traffic generated by each client machine. This includes inbound and outbound requests, just as we discussed earlier in this chapter. The traffic generated by the test tools sometimes overloads the network subnet supporting the client machines. Also, the traffic burden sometimes overloads the network cards in individual client machines.

In short, take the load generator environment seriously. The load generator requires more capacity than most teams originally estimate. If the client machines reach 75% CPU utilization or the network traffic passes the safety threshold, increase capacity on these devices. You cannot drive load against your web site if the load cannot make it to the intended servers.

Master/Slave Configurations

Many industrial performance tools use the master-slave test configuration, shown in Figure 9.6. The "master" machine manages each test and collects data. The "slaves" actually manage the threads and sockets representing the virtual users, and run the corresponding test scripts. For extremely small configurations (10 to 20 virtual users), the master and slave both run on the same machine. However, for larger tests, the master runs on a different machine, with one or more slaves also running on different boxes.

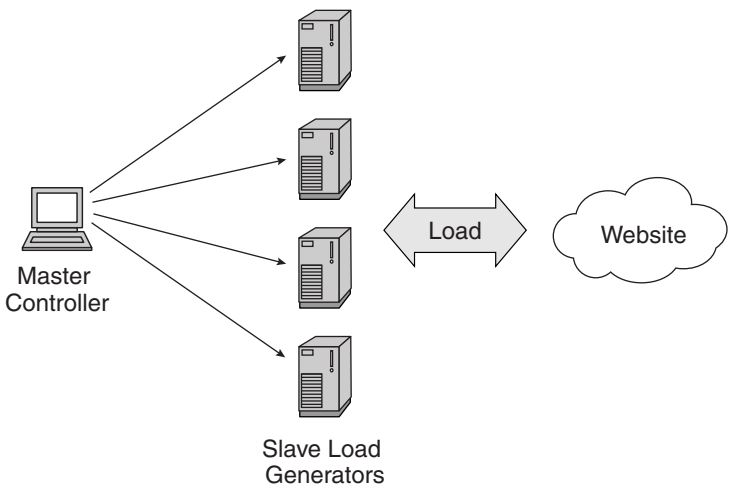

Figure 9.6 Master/slave load generator configuration

The best practices to remember with a master/slave configuration include the following:

- Keep traffic between the master and slaves to a minimum. Many performance test tools allow the test manager to define the frequency with which updates and status travel between the slave and the master. If you're using a new performance test tool for this first time, use a protocol analyzer to determine the actual network burden on the client subnet.
- The master (also known as the "controller" or "coordinator" machine) often requires less CPU capacity than the slave machines. Conversely, if it stores reports for the test cluster, it may require a larger hard drive than a slave machine.
- Watch the CPU on *all* test machines during the performance test. If CPU utilization exceeds 75%, the machine needs more capacity.
- Watch the logs and hard disk on the client machines. Often the clients accumulate large log files and pass these logs back to the master after the test completes. Frequent logging, of course, increases the disk I/O burden for the client machine and impacts testing. Likewise, if these logs accumulate over time, the client machines may not have enough room for subsequent runs. Before starting a run, make sure the client machines contain sufficient free disk space for any logging they may perform.
- Recycle the machines frequently. The slaves and the controller sometimes throw odd errors or stop responding after several testing cycles. In general, we find it best to recycle the test machines once or twice a day during periods of extended testing.
- Try to keep the test slave hardware homogenous. Often, because of load balancing techniques, one or two client machines may drive all the load a server in a clustered environment receives. If the test cluster contains one machine significantly more powerful than the others, some servers in the web site cluster may not achieve full loading.
- It's often useful to simulate traffic from a number of different client IP addresses, especially when performing a test that utilizes an IP sprayer in front of several HTTP servers. Web sites often configure IP sprayers for affinity routing between the incoming user and a particular sever in the cluster.

  You need sufficient test client machines, NICs, and supporting performance test software to make IP affinity work during your testing. See Chapter 8 for more details.

After the Performance Test

Many companies treat the performance test environment as a transient entity. The environment exists for a few weeks to test the performance and scalability of the web

site. Afterward, the team pulls the test environment apart and uses the components to build other test environments or sends the equipment to production. This works well for most test scenarios, but major production web sites often require a permanent, separate test environment. A permanent test environment allows you to

- Safely test new features and bug fixes prior to their introduction into the production web site.
- Recreate problems seen on the production site without using production resources. Because of the multi-threaded nature of web applications, some problems only appear under load conditions. You need a reliable test environment to find these problems.

At a minimum, keep enough test client capacity to stress at least one production server. These clients may reside either with the test environment, if one exists, or may be a part of the production environment. Ideally, configure these clients to drive load in either the production or test environment, if needed.

Few companies dedicate test machines to a particular web site or web application, but this increases another risk: If you run into a problem in production, you don't have machines set aside where you can immediately try to reproduce the situation. The problem determination cycle often takes much longer when you can only work with the problem in production. Having available machines dedicated to testing and problem determination often makes it simpler and cheaper to recreate and debug production problems.

Hardware and Test Planning

Many teams bring all of their hardware online and start testing. Invariably, they experience performance problems and spend lots of time trying to isolate the components or applications contributing to the bottleneck. A better approach calls for bringing hardware components into the test systematically to gauge the performance impact of each piece of equipment.

We recommend limiting the hardware involved in early phases of testing. The first step in the test might be a network performance test, as shown in Figure 9.7. If the tester sends a file from the HTTP server to the application server via FTP, does it take more or less time than sending it from the application server to a legacy system? This simple test validates the network across multiple points, and it can be very important if multiple network segments are engaged in your environment.[3]

3. Thanks again to Susan Hanis for her networking expertise.

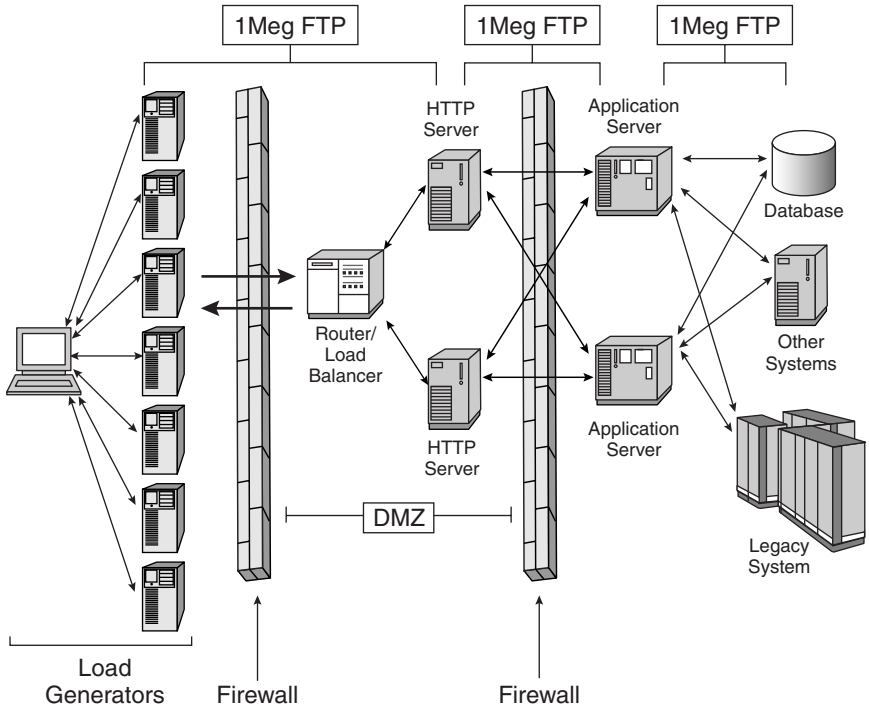

Figure 9.7 Testing the networks with FTP

Next, bring an HTTP server and an application server on-line for testing. Initially, consider stubbing out calls to mid-tier and legacy databases, or use test script primitives (as discussed in Chapter 7) to test only HTTP and servlet function without exercising the back-end systems. (Figure 9.8 outlines a progressive performance test strategy using a typical test environment.)

Add these systems into the mix only when you're certain of the soundness of the HTTP server and application server configuration. After successfully testing and tuning a single HTTP server and application server with the back-end systems, add multiple HTTP servers and application servers to the cluster to test scalability. Next, increase the complexity of the test by turning on the firewalls and, later, the reverse proxy server. At this point, you might decide to exercise the SSL portions of the web application using test scripts.

This approach allows you to bring hardware into the test in a layered fashion. Key components come into the test, receive intense test scrutiny and tuning, and, as a result, stabilize. Of course, this technique requires planning. Plan the network

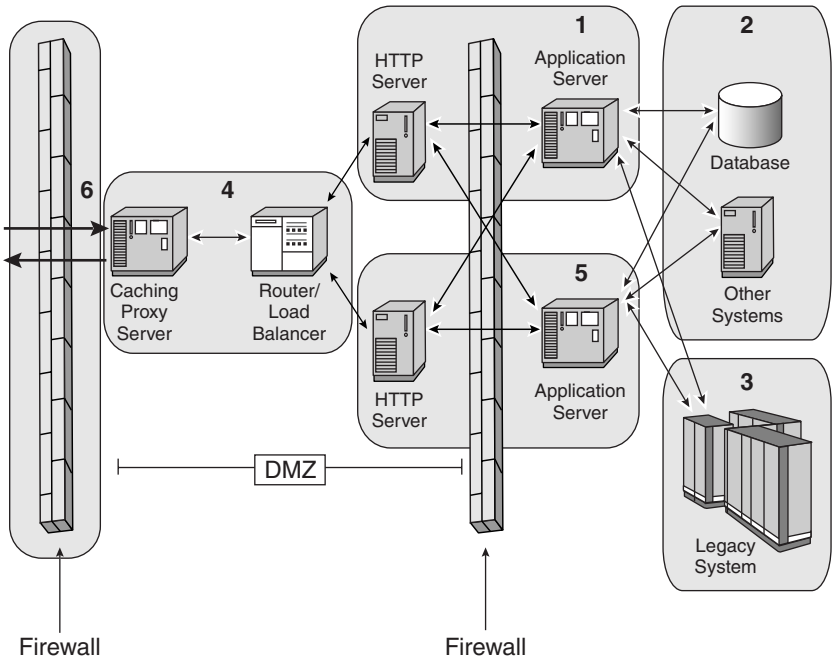

Figure 9.8 One possible test ordering for a large test environment

topology carefully to support easy removal and addition of key components. In the long run, you benefit by understanding the performance impact of each piece of hardware used in the web site.

Summary

This chapter covered some test environment planning basics like using an isolated network to ensure repeatability. We walked through an example of how to develop network capacity estimates for your web site and test environment, and how to scale a test and still accurately predict how your production web site will behave. We also discussed hardware and software tuning issues common to database servers, test clients, application servers and other key web site components. This chapter also recommended a strategy for progressively performance testing these components.

All the setup in the world won't help you if you schedule your test while some of your legacy or database systems are under load or offline for maintenance. Be alert to the

production and maintenance schedules of the components of your test environment. In addition to systems, you need to make sure that the right people are available to assist with monitoring and tuning during the test.

The next chapter marks the beginning of a three-part case study. The example in this study combines many of the topics covered so far and provides concrete examples for their application.