# Performance, scalability and reliability issues in web applications

Lakshmi S. Iyer
*Bryan School of Business and Economics, University of North Carolina, Greensboro, North Carolina, USA*

Babita Gupta
*California State University Monterey Bay, Seaside, California, USA, and*

Nakul Johri
*Customer Quality Management Team, I2 Technologies, Dallas, Texas, USA*

## Abstract

**Purpose** – The primary purpose of this paper is to present a comprehensive strategy for performance, reliability and scalability (PSR) testing of multi-tier web applications.

**Design/methodology/approach** – The strategy for PSR testing is presented primarily through examination of the intangible knowledge base in the PSR testing field. The paper also draws on relevant recent work conducted in the area of software performance evaluation.

**Findings** – The study revealed that appropriate testing procedures are critical for the success of web-based multi-tier applications. However, there was little academic work that collectively focused on PSR testing issues. This paper provides step-by-step testing procedures to ensure that web-based applications are functioning well to meet user demands.

**Research limitations/implications** – Given the rapid changes in technology and business environments, more applied research will be needed in the area of PSR testing to ensure the successful functioning of web-based applications. For future studies, structured interviews or case-study methods could be employed to present the views of online companies.

**Originality/value** – This paper provides a comprehensive strategy and the suggested steps for managers and technical personnel to ensure that the multi-tier, web-based applications are effective, scalable and reliable.

**Keywords** Reliability management, Electronic commerce, Computer networks, Performance testing

**Paper type** General review

## Introduction

Testing is always a necessary component in system design to ensure systems operability, performance and availability. In 1980s, the proliferation of client-server environments ensured relative stability in performance terms and the very nature of such architecture meant that such systems were limited to a predefined internal user population, one that could be trained in systems policies and procedures. However, in the early 1990s, the advent of the internet changed all this, exposing mission-critical systems architecture to external networks. Further, the interdependence of such systems meant that a failure in an isolated area could impact the whole network and cause failures anywhere (Aries *et al.*, 2002). In addition, as user load on a web site increases, the computation and communication costs can result in significant delays, leading to poor scalability (Bakalova *et al.*, 2004).

Industrial Management & Data Systems
Vol. 105 No. 5, 2005
pp. 561-576

0263-5577
DOI 10.1108/02635570510599959

While existing client-server architectures gave quantifiable user populations, the web is definitely an unknown terrain in terms of user population. Added to this, the packet-based nature of web traffic puts new demands on system performance and availability. Web technology also brings with it the need to deal with browser-based clients and heavy-duty usage. Where client-server gave relative reliability, the intangible nature of the internet means that what works today may not work tomorrow. This is challenged further by increasingly complex back-end systems, harnessing various technologies and operating environments for deploying mission-critical information. Industry estimates predicts that e-commerce revenues globally are over $1 trillion by 2005 (eMarketer.com). If the full potential of e-business is to be realized, it is essential that appropriate testing procedures be at the forefront of any information systems strategy. Schneider (2002) provides some discussion on the various IT infrastructure elements required for e-commerce. The key to making applications suitable for a high performance level is testing applications' functionality during the development phase (Ameen, 1989; Franz and Shih, 1994; Huq, 2000). In this respect, testing has to be thought of as an integral part of the development process (Nixon, 2000). Hence, appropriate testing procedures for IT infrastructures are important for increased productivity, desirable end-user experience, a positive business image and an optimized technology investment (Datta *et al.*, 2004; Pavur *et al.*, 1999).

Performance and reliability problems can cripple a web site at a moment's notice if the load on a site suddenly increases (Lii *et al.*, 2004; Hooshang *et al.*, 1995; Shaw *et al.*, 2002). Along with the right tools and techniques, a practical performance testing process is needed to be successful in system testing the web site before – and after – it goes live (Empirix, 2004a, b, c; Lars, 2004; Morgan, 2003). The primary purpose of this paper is to discuss various types of testing methods for performance, reliability and scalability of multi-tier web applications.

This paper examines the intangible knowledge base in the performance, scalability and reliability (PSR) testing field and documents the planning and execution methodology that can be adopted for testing multi-tier web applications. A glossary at the beginning includes the definition and discussion of terms related to PSR testing. The background section discusses the need for PSR testing. The PSR planning strategy is then presented followed by the execution of PSR testing and finally, summary and suggestions for future research.

## Background

The growth and adoption of the internet has led to an exponential growth in web-based multi-tier applications used by businesses to provide dynamic content to users (Johnson and Reimer, 2004). As the number of users grows, the load on the systems to perform the communication and computational services also grows. If such systems are not well tested to support the increasing loads, users can see a considerable decrease in response time. For example, Keynote Systems Inc., an internet reporting firm, notes that in December of 2004 FedEX site's average response time during the week of 15th December was about 6 seconds while UPS's site response time about 15 seconds during the same week (Keynote Systems, 2004). Although, the number of visitors at that time is not known, the slow response time might have some impact on the choice of web site from the user's perspective.

There are several academic research studies that focus on the importance of IS performance evaluation (Leem and Kim, 2004), web services (Casati *et al.*, 2003), web site design (Yang and Tang, 2003), management and usability and its impact on user's perception (Frank, 2004; Wen *et al.*, 2003). However, there is little academic research that collectively focuses on web sites reliability, scalability and performance issues. Table I shows the findings on poor web performance, gathered from consultants and engineers, by Empirix (2004c).

They suggest that performance problems in multi-tiered web-based systems are due to very little or the lack of appropriate performance testing. They also discuss the importance of PSR testing for web-based applications. In this paper, we present a PSR testing strategy for multi-tiered web-based applications (Figure 1). The next section discusses the planning phase of the PSR strategy followed by the execution strategy.

To ensure readability and maintain the flow a separate glossary section is added at the beginning of the paper that defines the terms and definition.

## Planning the PSR testing

The objective of any performance testing process is to simulate a production environment. This entails the following issues:

(1) *Production scenarios*. The testing environment ideally should be as close to the actual environment where the software would be used. This allows the generation of important metrics, which estimate the performance limits of the software.

(2) *Production-type loading conditions*. The end-users do not equally utilize all functions of software. The manner in which the software and its various modules would be used determines the overall efficiency and ensures if the product is capable of meeting the requirements. The test plan should take that into consideration while determining the appropriate testing load to be used for various usage levels, etc. (Singleton, 2002; Wen, 2001).

| Where web problems reside | Percentage |
|---|---|
| Network | 20 |
| Web server | 10 |
| Database server | 30 |
| Application server | 40 |

**Source**: Empirix (2004c)

**Table I.**
Where common web performance problems reside

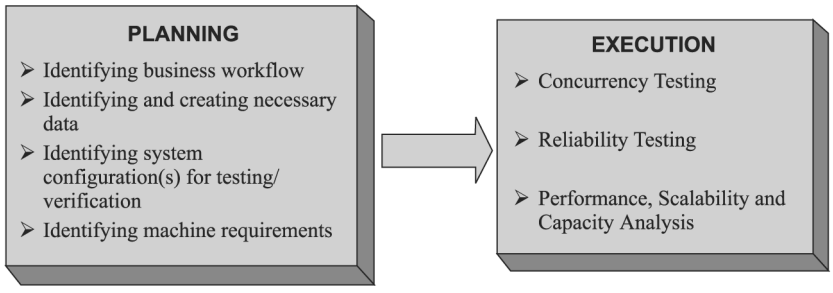


**Figure 1.**
PSR testing strategy

Hence, the steps involved in this planning phase of PSR testing and verification are:

- identifying business workflow;
- identifying and creating necessary data;
- identifying system configuration(s) for testing/verification; and
- identifying machine requirements (type of machine, characteristics).

*Identifying business workflow*

This process depends strictly on the business scope of the product. Key areas or workflow that represent the broader use of the product need to be identified. This should include both end-user and administrative workflow applicable to a production environment. For example, for a marketplace solution, end-user workflow could encompass the following:

- login – search products – add to cart – order products – logoff;
- register – login – search products – logoff; and
- login – approve payments – logoff.

Administrative workflow for the same solution could cover the following:

- login – add new products to catalog – logoff;
- login – edit or maintain user profiles – logoff; and
- login – change product prices – logoff.

The end-user input is very valuable at this point as they help prioritize the scenarios for efficient testing. Once the code-freeze has been implemented, these scenarios can be automated and validated for correctness and parameterized to suit any generic system set up. However, a derivative of this process is to identify data necessary to simulate the short-listed scenarios. Automation effort can begin before appropriate data are finalized, at the risk of modifications and potential rewrite due to data mismatch. Another option is to make the automated scenarios data independent.

*Defining and creating necessary data*

Creating relevant data are a key activity to completing PSR testing. This process should closely follow the business scenarios that have been short-listed for testing. To cover a majority of the customer base for the product, categorizing the data size into small, medium and large is usually helpful. This can be accomplished by creating the basic content to cover the business workflow and building up the volume needed for small, medium and large size data sets. Typically, the basic data content could be used for functional testing while the larger ones could be used for system/integration and PSR testing. Sample data sizes for marketplace solutions could be:

- Small ~1,000 users, 100K items in database, 5 prices for each product, 100 categories of products;
- Medium ~10,000 users, 1M items in database, 5 prices for each product, and 1,000 categories of products; and
- Large ~50,000 users, 5M items in database, 5 prices for each product, and 5,000 categories of products.

There could be other factors that might concern the volume of data that needs to be addressed by product teams based on the nature of business problem and the solution the product is intended for. Ideally, reliability-testing needs to be conducted with a data set that is representative of the majority of the customer base. Unless automated scripts are set-up to be data independent, there is very little value in automating the business workflow to be tested while the appropriate data are unavailable. It is always a good practice to identify and create relevant data sets before starting on procedures or scripts for manual and/or automated testing.

Logically, automation takes the next step in this process. Record and play tools that allow for simulations of the workflow are recommended, for example, mercury interactive's load runner. This reflects directly on timesavings that can be realized by using such tools as the PSR testing often faces critical scheduling and time constraints. After recording the workflow of interest, the scripts can be parameterized to be played back on any set up running the same product version. Often the scripts should be made data independent – this merely refers to the process where the input data for the script can be changed through parameters defined in the script instead of re-recording the entire workflow.

*Identifying system configuration(s)*

Defining one or more system configurations suitable for implementing the product is the first step in determining the product's capabilities while not referring to end-user functionality. Discussion with the product architect(s) and product management would give valuable insight into the logical configuration of the product and hence the different physical configurations possible through the various combinations.

If a logical analysis of the design and physical configuration combinations does not short-list the candidates for a production environment, benchmarking procedures should provide the distinction between the different types. However, this approach would add a huge overhead in analyzing different combinations and will add many cycles of repeated testing in order to gather detailed performance metrics for each type. This, of course, should be a one-time exercise for the product, unless there are fundamental functional and architectural changes between releases.

*Identifying machine requirements*

In multi-tier web-based applications (Figure 2), based on the system configuration(s) chosen for testing and on the logical configuration of the product; different tiers might

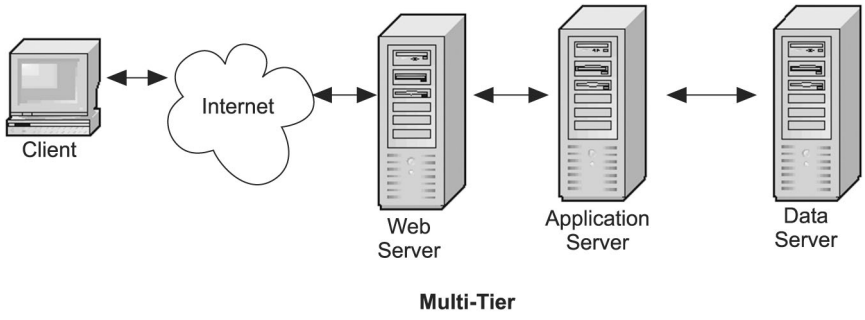


**Figure 2.**
Multi-tier configuration for web-based applications

need different types of machines with different capacities. For example, database machine needs to have more memory, number of processors and hard disk space. While the second tier could be memory intensive and could cache lot of data from the database, third tier could be processor intensive and handle all client requests. Machine requirements have to be planned accordingly for each tier and capacity of machines determined in advance.

Machine characteristics have to be carefully planned for PSR testing, keeping in mind that production systems usually have high-end machines with higher capacities. For example, a desktop computer with a 166 MHz processor and 128 MB ram would not fit well for a web production system and hence should not be a candidate for performance measurements. Ideally, performance metrics should be gathered on different types of machines to establish dependency factors. For concurrency and reliability testing, machines need not be high-end, but should at least have dual processors so that concurrency issues due to system configuration can be isolated early in the testing process.

Selecting between a single processor and a multi-processor machine is dependent on the product's design – whether or not it supports multi-threading. If the application is not designed to run multiple processes or components, multi-processor machines would usually be under-utilized. In either case, the product has to be tested appropriately to determine the number of processors that would be optimum for a given configuration.

## PSR testing execution

Adopting a strategy to encompass all the types of testing is not only a cumbersome activity but is also complex as it is hard to restrict the testing process to short release cycles (Rothman, 2002). Blumenstyk and Decker (2001) discuss some test procedure as well as the downfall of not having appropriate test procedure for web applications. We present the following testing processes that may be adopted to cover the broad area of performance benchmarking:

(1) *Concurrency testing*:
   - under nominal end-user load utilizing a maximum of 50 percent of machine CPU power; and
   - individual workflow tests.

(2) *Reliability testing*:
   - under real-time production-like environment covering key product workflow;
   - test encompassing all end-user workflow; and
   - to cover reliability testing and load testing.

(3) *Performance, scalability and capacity analysis*:
   - measure product performance under different loading conditions;
   - measure product performance on a system by scaling machine resources – processors on a machine, number of machines; and
   - to cover volume and stress testing in addition to performance and scalability testing.

*Concurrency testing*

Concurrency testing pertains to testing the product for multiple users/requests that would trigger simultaneous calls through single or multiple modules. This should be the first step of analysis for production readiness of a release. This yields valuable information about the system's ability to withstand different types of end-user requests under varying loading conditions. Goals for this testing phase should cover the following:

- multi-user simulation on a single processor machine;
- multi-user simulation on multi-processor machine; and
- multi-user simulation on multi-server configuration.

Potential problem areas that may be exposed by the above steps are:

- deadlocks in code;
- sections of code that are processor intensive;
- memory usage under concurrency;
- data I/O synchronization problems with data source (mainly databases);
- issues with product configuration in a multi-server set up; and
- necessary settings that need to be documented for system/sub-system(s).

Strategy to overcome potential problems: a good strategy to accomplish concurrency testing is to work through the following issues systematically:

(1) Define real-time/production system scenarios that need to be validated.

(2) Define the type of data that needs to represent the real-world performance requirements, for example, number of users, number of products, number of product attributes.

(3) Create test data needed to support the above workflow.

(4) Automate the end-user workflow and include parameters in the script for flexibility and to support different types of data.

(5) Verify that the workflow can be consistently used to produce same result using the product installation/configuration.

(6) Execute the automated script on a standalone machine or the standard test environment, not necessarily emulating a production system configuration, with increasing level of severity and complexity as mentioned below. This is typically executed on a multi-processor machine to eliminate issues early when the scope is very narrow and the number of variables that could lend the system unstable are few. This step has to be repeated for every workflow that has been translated into an automated script that would simulate an end-user's actions:

    - one simulated user action running for a fraction of a second for one workflow process cycle/iteration of the selected workflow;
    - one simulated user action running for a fraction of a second for multiple iterations ($>100$) non-stop that could extend over a three to five hours period; and
    - multiple simulated users (initially smaller number, example 10) running for a fraction of a second for one workflow process cycle/iteration of the selected workflow.

This would complete the basic concurrency testing as applied to each standalone workflow, independent of all other end-user scenarios that are possible in a production system. Next phase of testing is a combination of concurrency and reliability analysis that is focused toward attaining a stable system under load that yields consistent and repeatable behavior.

During this phase, concurrency is not restricted to number of processors or users or servers, but also covers concurrent use/execution of different modules of the product that would impose different loads on the system due to different reasons such as design differences and differences in end-user functionality.

*Reliability testing*

Reliability testing is the process of validating, that the product configured to simulate the customer's production system is capable of sustaining continued stress and load applied through a collection of end-user actions that may or may not be predictable for coincidence, with an acceptable level of degradation.

*Potential problem areas* that have to be monitored through this analysis phase are: memory consumption, CPU utilization, performance as measured by response times, throughput (applicable to web servers) and cycle time for each workflow.

*Strategy to avoid potential problem areas* can be implemented in two stages – reliability of each workflow (subset of real-time usage) and reliability of the collective set of workflow. To ensure reliability of the system concerning this subset, automated requests, simulating multiple end-users may be generated for each workflow against the system configuration for an extended period of time to ensure that the product is free of the problems listed earlier. This could be done by establishing a distribution of number of users (percent) that would focus on individual workflow. For example, one of the surveys on online user traffic indicated that among all online shoppers, 65 percent browse for products and never buy them; 25 percent of the shoppers place an order; average surfing time was identified to be 20 minutes.

The following steps help in organizing the reliability testing effort.

(1) Multiple simulated users running for a fraction of a second for multiple iterations ($>100$) non-stop, that could extend over eight hours period.
    - This step could be sub-divided into multiple tasks by gradually increasing the simulated user load until the specifications for the final load requirements for this workflow are met for a single machine configuration. For example, in a typical marketplace environment, if the requirement expects the product to support 100 users for catalog search on a single application server, this step could be accomplished over 10, 25, 50, 100 users.
(2) Repeat the above steps for each end-user workflow that has been identified in the requirements.
(3) Completing the above would ensure reliability of selected end-user workflow under given load conditions, simulated separately on a non-production-like environment/configuration.

An extension of the above testing procedures specific to reliability analysis is the final effort toward determining the reliability of the product suite under real-time operating

conditions. Each independent testing procedure has to be incorporated into a single collective scenario that would simulate the real-time use of the product implementation.

This testing phase, and hence each step involved in it, should meet the following criteria to be declared a success:

(1) no exceptions in log files;

(2) no error messages in application and servlet engine log files;

(3) no deadlocks or memory leaks;

(4) no performance bottlenecks that can be attributed to size of data; and

(5) system performs with stability beyond two hours from startup time (as defined by the following) under the given loading conditions and does not change more than 10 percent with regard to:

- system throughput measured by the total number of requests served per second;
- cycle time for different workflow;
- number of threads of different processes in the configuration; and
- virtual memory utilization of different processes.

*Performance, scalability and capacity analysis*

Performance can be defined as the end-user responsiveness of system under various loading conditions. Most of the product related issues could be resolved if reliability tests are conducted on the workflow as those that need performance benchmarking. Capacity of a unit of production system can be determined by monitoring requests per second (RPS), requests per page and number of active users (van Eijl, 2002; Sia and Ho, 1997).

However, measuring these systems characteristics have to be done in a controlled environment where the following parameters help identify the threshold for acceptable system performance:

*Latency.* The round trip time to service a request at the client. This is typically measured over a 10 MB LAN connection in a B-B scenario.

*RPS per unit of production system.* A request is a single HTTP request served (in case of web server). This number represents the total requests processed in one second and is the best descriptor of throughput per machine. This parameter may be applicable to all types of transaction processing systems (order promising, order management, etc.).

RPS and latency depend on the number of users actively using the system at any point in time. Since the load on a unit can be increased many fold the threshold for acceptable performance is defined through the latency that is acceptable for the business scenario in context. Typical latency on marketplace production systems, which seem to reflect the industry standard, is two seconds.

This could vary drastically for back-end transaction processing systems that may be expected to serve requests in the order of milliseconds. Nonetheless, an acceptable limit has to be set prior to conducting performance/capacity tests. Plan the testing effort to estimate the capacity that meets twice this threshold in order to determine maximum possible capacity under given testing conditions.

Following is the high-level approach to the performance and scalability benchmarking effort:

(1) Identify the configuration that is applicable for implementation:
   - if many system configurations have to be analyzed to identify the production set up, the procedure below has to be repeated for each configuration; and
   - appropriate analysis and comparison can be done after gathering enough data points for the different set ups and workflow.

(2) Based on the reliability testing results, an estimate of number of users that a single unit of a production system can support can be determined.

(3) Set up monitoring tools to gather metrics on parameters of interest.

(4) Establish a repository to store test results so that they can be processed at the end of the test cycle.

(5) Repeatedly add another unit of the system and go through the process to determine the capacity of the scaled set up.

(6) Continue the above steps to determine scalability for at least four or five units of a production system.

(7) If the configuration is multi-tiered, the approach suggested for scalability analysis can be followed.

(8) Performance metrics can be monitored during the above process and tabulated.

In addition to the above, the "workhorse" has to be tested for multi-processor support (useful if the application is multi-threaded) in order to estimate the optimum number of processors that a unit of the production system can utilize. If a system were running multiple processes, multi-processor machine would be the obvious choice.

Dual processor assumption may be a valid approach to start tuning the process, but most high-end systems are capable of four or eight processors (especially Unix). Hence, the question of number of processors in a production system is bound to come up in almost every implementation.

To determine this, the unit has to be tested for performance and scalability. The number of processors on the unit has to be progressively increased – 1, 2, 3, 4. For each machine configuration (with specific number of processors), tests have to be run for different end-user loads.

With the resulting tabulation for these tests, scalability of unit as applied to the number of processors can be determined. If results border near in excess of 80 percent scalability for multi-processors, tests may be executed with 8, 12 and 16 processors in a single unit of system. This process depends on how important and critical it is to develop a thorough capacity planning roadmap for implementations (Ferrari *et al.*, 1983; Sia and Ho, 1997). A key objective of the capacity planning process is that adequate capacity exists for the delivery of information resources required and that the services can be delivered without declines in system service levels (Davis, 1991).

## Scaling machine resources

In a two-tier system configuration (Figure 3), a unit of a production system encompasses all processes that are core to the run-time production system. Two such basic units are needed to support basic fault-tolerance in a production environment. To extend the

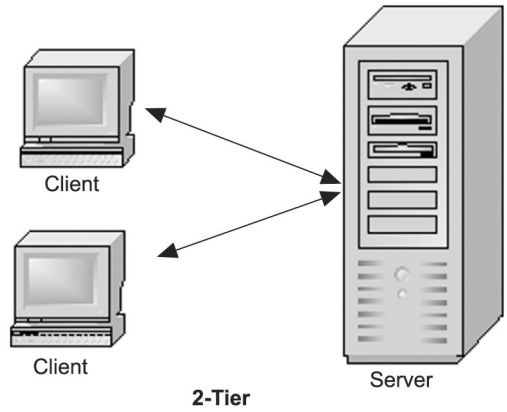


**Figure 3.**
Two-tier configuration

configuration for a scalability analysis, the number of servers in the second tier can be increased to four or five to gather sufficient details to determine the scalability factor for the metrics of interest – response times and number of concurrent users.

Following the same assumptions as a two-tier set-up, in a three-tier configuration (Figure 4), scalability factor has to be estimated at second and third tiers. Starting with a single server on second tier, number of servers has to be increased in the third tier and relevant metrics has to be gathered. Next, another unit of server is added on the second tier and the number of servers in the third tier are increased to estimate scalability.

Scalability of this set up is still measured by the same factors mentioned earlier, viz. response times, and number of concurrent users. The process is little complicated since a unit of production system cannot be isolated due to the split-tier model.

A similar procedure outlined for a three-tier set up can be adopted for multi-tier models (Figure 4). This configuration would involve an additional cycle of testing in two dimensions – vertically and horizontally. Unless there is a compelling reason to

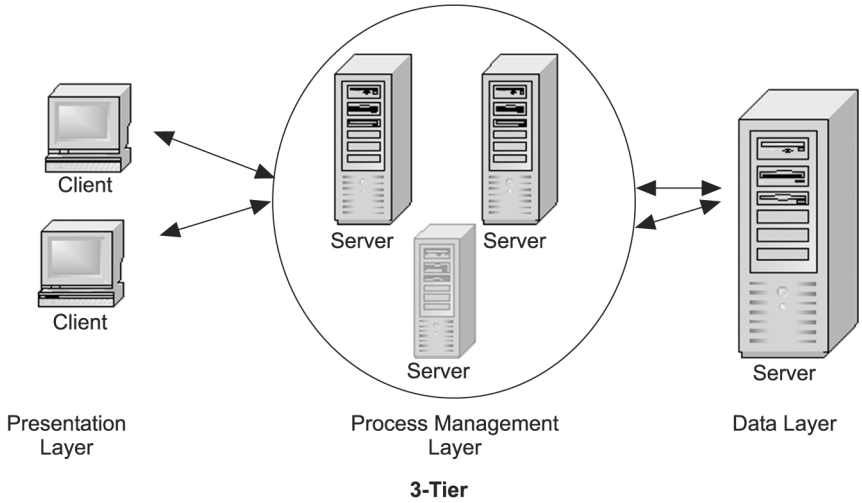


**Figure 4.**
Three-tier configuration

have such a topology it is advisable to design less complicated configurations for implementations. Following notes would support this argument:

(1) lesser maintenance overhead for system administrators;

(2) greater machine resource utilization in a smaller set up;

(3) capacity/scalability analysis/planning is more controlled and accurate;

(4) analysis process is less complicated since capacity or scalability factor need not be determined at additional tiers; and

(5) lesser network complications, delays, expenses and simpler set up.

Though the procedure below is elaborate and involves many cycles of same type of testing, this methodology provides valuable information and helps understand the system's behavior under different types of loading:

(1) Starting with five users measure the end-user latency for the given combination of business workflow and note the details.

(2) Double the number of users and repeat tests for 4 or 5 data points (5, 10, 20, 40, 80, etc.) or until the acceptable latency limit is exceeded, whichever happens later.

(3) Add another unit of the production system and double the number of simulated users (10, 20, 40, 80, 160, etc.). Again, this exercise can stop once the latency limit is reached or continued up to twice the latency to determine max capacity.

(4) Continue the above process by scaling to four units and tabulate the results.

(5) For a multi-tiered set up, capacity of intermediate tier(s) can be estimated by the load on the processors on the machines positioned at the tier(s). Typically, capacity of the intermediate tiers are restricted to an average of 80 percent utilization of machines' processor(s), to allow for peak loading conditions.

(6) During this process, performance bottlenecks can be monitored by evaluating the list of transactions in the business workflow and their respective average latency for a test cycle, viz. ten user tests running for three hours.

(7) Though the average latency of the scenario may fall below the threshold, transactions that indicate exceptions to the condition should be analyzed for potential design issues.

(8) Once the test results are tabulated, following analysis would help judge the scalability of the configuration under study:

- response time scalability per unit;
- latency vs number of users per unit;
- throughput scalability per unit;
- RPS vs number of users per unit;
- capacity scalability – throughput;
- RPS vs load per unit (across different number of units – 1, 2, 3, 4, 5);
- capacity scalability – latency; and
- number of users for capacity threshold (and 2 ×) vs number of units.

All the above details present valuable information that can be analyzed to improve the performance and productivity of the application.

## Study implications and directions for further research

The primary objective of the web-based management of information technology is the design and maintenance of an effective information technology infrastructure capable of delivering required information and services to its various internal and external customers. This requirement has significant bearing in today's highly competitive global marketplace driven by internet technologies that blur geographic boundaries. The timely delivery of information and resources is a critical component of a corporation's online services and has significant effect on its competitiveness in the global digital economy. The ability to deliver online information, and to process online transactions and user requests for information and services in a timely fashion without technical difficulty is directly affected by the information technology infrastructure that supports the e-commerce operations. This mandates appropriate testing procedures to ensure that web-based applications are functioning well to meet user demands. This paper provides a comprehensive strategy and the suggested steps for managers and technical personnel to ensure that the multi-tier web-based applications are effective, scalable and reliable.

In addition to the above-mentioned benefits to practitioners, this paper has relevance to academics as well. As indicated earlier, there is little academic study that collectively focuses on web sites reliability, scalability and performance issues. This paper contributes to the literature by providing a comprehensive discussion of the planning and execution strategies that can be adopted for testing of multi-tier web-based applications. The PSR issues related to multi-tier web-based applications are also presented. An avenue for further research for academics would be to survey firms to investigate in depth the actual practices of firms' PSR testing methods. A study that looks into the factors that limit PSR testing by firms and lessons learnt through PSR testing processes would benefit firms that face challenges in this area.

## References

Ameen, D.A. (1989), "Systems performance evaluation", *Journal of Systems Management*, Vol. 40 No. 3, pp. 33-6.

Aries, J.A., Banerjee, S., Brittan, M.S., Dillon, E., Kowalik, J.S. and Lixvar, J.P. (2002), *Communications of the ACM*, Vol. 45 No. 6, pp. 100-5.

Bakalova, R., Chow, A., Fricano, C., Jain, P., Kodali, N., Poirier, D., Sankaran, S. and Shupp, D. (2004), "WebSphere dynamic cache: improving J2EE application performance", *IBM Systems Journal*, Vol. 43 No. 2, pp. 351-70.

Blumenstyk, M. and Decker, R. (2001), "Performance testing: insurance for web applications", *Consulting to Management*, Vol. 12 No. 4, pp. 57-60.

Casati, F., Shan, E., Dayal, U. and Shan, M.C. (2003), "Business-oriented management of web services", *Communications of the ACM*, Vol. 46 No. 10, pp. 55-60.

Datta, A., Dutta, K., Thomas, H., Vandermeer, D. and Ramamritham, K. (2004), "Proxy-based acceleration of dynamically generated content on the world wide web: an approach and implementation", *ACM Transactions on Database Systems*, Vol. 29 No. 2, pp. 403-43.

Davis, C.K. (1991), "Systems analysis for data communications networks at an electric utility", *Journal of Systems Management*, Vol. 42 No. 7, pp. 9-13.

Empirix (2004a), *Implementing an Effective Web Application Testing Process*, available at: www.empirix.com (accessed 21 October).

Empirix (2004b), *Kelkoo Uses Empirix E-tester to Prevent Customer Problems and Help Strengthen Partnerships*, available at: www.empirix.com (accessed 21 October).

Empirix (2004c), *The Top 25 + Reasons Web Applications Don't Scale*, available at: www.empirix.com (accessed 21 October).

Ferrari, D., Serazzi, G. and Zeigner, A. (1983), *Measurement and Tuning of Computer Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Frank, L. (2004), "Architecture for integration of distributed ERP systems and e-commerce systems", *Industrial Management & Data Systems*, Vol. 104 No. 5, pp. 418-29.

Franz, L.A. and Shih, J.C. (1994), "Estimating the value of inspections for early testing for software projects", *Hewlett-Packard Journal*, Vol. 45 No. 6.

Hooshang, M., Worley, B. and Worley, J. (1995), "Automated systems and reliability", *Industrial Management & Data Systems*, Vol. 95 No. 1, pp. 5-9.

Huq, F. (2000), "Testing in the software development life-cycle: now or later", *International Journal of Project Management*, Vol. 18 No. 4, pp. 243-50.

Johnson, R. and Reimer, D. (2004), "Issues in the development of transactional web applications", *IBM Systems Journal*, Vol. 43 No. 2, pp. 430-40.

Keynote Systems (2004), "Keynote consumer 40 internet performance index", available at: www.keynote.com/solutions/performance_indices/consumer_index/cons40_index-121904.html (accessed 30 December).

Lars, F. (2004), "Architecture for integration of distributed ERP systems and e-commerce systems", *Industrial Management & Data Systems*, Vol. 104 No. 5, pp. 418-29.

Leem, C.S. and Kim, I. (2004), "An integrated evaluation system based on the continuous improvement model of IS performance", *Industrial Management & Data Systems*, Vol. 104 No. 2, pp. 115-28.

Lii, Y., Lim, H.J. and Tseng, L.P. (2004), "The effects of web operational factors on marketing performance", *Journal of American Academy of Business*, Vol. 5 Nos. 1/2, pp. 486-95.

Morgan, B. (2003), "Three keys to a successful architecture", *Wireless Week Magazine*, 15 June, available at: www.keepmedia.com/SaveItem.do?itemID = 271722 (accessed 20 October 2004).

Nixon, B.A. (2000), "Management of performance requirements for information systems", *IEEE Transactions on Software Engineering*, Vol. 26, pp. 1122-46.

Pavur, R., Jayakumar, M. and Clayton, H. (1999), "Software testing metrics: do they have merit?", *Industrial Management & Data Systems*, Vol. 99 No. 1, pp. 5-10.

Rothman, J. (2002), "Release criteria: is this software done? How to know if your software is ready to release", *The Software Testing and Quality Engineering Magazine*, available at: www.stqemagazine.com/featured.asp?id = 21 (accessed 10 October).

Schneider, G.P. (2002), *Electronic Commerce*, 3rd ed., Course Technology, Boston, MA.

Shaw, J., Baisden, C. and Pryke, W. (2002), "Performance testing: a case study of a combined web/telephony system", *BT Technology Journal*, Vol. 20 No. 3, pp. 76-86.

Sia, C. and Ho, Y. (1997), "Predictive capacity planning: a proactive approach", *Information and Software Technology*, Vol. 39 No. 3, pp. 195-204.

Singleton, P. (2002), "Performance modeling: what, why, when and how?", *BT Technology Journal*, Vol. 20 No. 3, pp. 133-43.

van Eijl, C. (2002), "Capacity planning for carrier-scale IP networks", *BT Technology Journal*, Vol. 20 No. 3, pp. 116-23.

Wen, H.J., Lim, B. and Huang, H.L. (2003), "Measuring e-commerce efficiency: a data envelopment analysis (DEA) approach", *Industrial Management & Data Systems*, Vol. 103 Nos. 8/9, pp. 703-10.

Wen, R.B. (2001), "E-business reliability with web performance management", *Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE)*, Hong Kong, 27-30 November.

Yang, H. and Tang, J. (2003), "A three-stage model of requirements elicitation for web-based information systems", *Industrial Management & Data Systems*, Vol. 103 Nos. 5/6, pp. 398-409.

## Further reading

Warren, A. (2003), "Quality of service middleware", *Industrial Management & Data Systems*, Vol. 103 No. 1, pp. 47-51.

## Glossary of terms

This section describes the basics of different terms used to describe different facets of performance benchmarking – concurrency testing, stress testing, volume testing, load testing, reliability testing, performance testing, baseline testing, scalability testing, capacity testing. All these facets are closely linked and would provide rich benefits toward benchmarking a product release if conducted in a single cycle.

*Concurrency testing*. Concurrency testing is about testing the product for multiple users/requests that would trigger simultaneous calls through single or multiple modules. Yields valuable information about the system's ability to withstand different types of end-user requests under varying loading conditions.

*Stress testing a.k.a. capacity testing*. Determines the upper bound of the application's capacity with respect to the server, network, and database. Stress testing will identify which system components are the first to get exhausted. Component stress testing stresses individual component as part of a troubleshooting process to isolate a fault.

*Volume testing*. Subjects the system to heavy volumes of data, similar to the loads expected to be encountered in the course of normal operations in a real-time production environment.

*Load testing*. Provides evidence that the application will sustain the expected number of concurrent users, number of transactions and transaction types per hour as specified.

*Reliability testing a.k.a. stability testing*. Provides evidence that the internet application/infrastructure is stable over extended periods. Used to determine and quantify the following – memory consumption, throughput degradation, response time degradation, and instability of component applications/modules or the infrastructure due to extended use. Also referred to as stability testing.

*Performance testing a.k.a. baseline testing*. Establishes a performance baseline for the application and infrastructure components that will be used for comparison against future upgrades or other implementations. Also referred to as baseline testing. Main goal of this testing process is to identify possible areas of the product that do not meet the standards defined to suit its real-time use.

*Scalability testing*. Follows capacity testing phase that yields information on the capacity of a unit of system or sub-system that represents a real-time production set up. Once the unit capacity is established, scalability testing provides evidence to prove that the application yields scalable results – as applied to end-user load, response time, memory consumption, CPU utilization – on proportionate increase of load and a unit of system/sub-system resource.

(Lakshmi S. Iyer is an assistant professor in the Information Systems and Operations Management Department at The University of North Carolina at Greensboro. She obtained her PhD in Business Administration from the University of Georgia, Athens, GA. Her research interests are in the area of Electronic Commerce, CRM, IT Outsourcing, Global Issues in IS, Intelligent Agents, DSS, and Knowledge Management. Her research work has been published or accepted for publication in *CACM, Annals of OR, DSS, JGITM, Journal of Scientific and Industrial Research, Encyclopedia of ORMS, Journal of Information Technology Management, Journal of Data Warehousing*, and *Industrial Management & Data Systems*.

Babita Gupta is an associate professor of Information Systems at California State University Monterey Bay. She obtained her PhD in Business Administration from the University of Georgia. Her research interests are in the areas of online security and privacy, IT outsourcing, customer relationship management, online consumer behavior, KM and global issues in the internet. Her research work has been published in the *CACM, Journal of Information Technology Cases and Applications, Industrial Management & Data Systems, Journal of Computing and Information Technology* and *Journal of Scientific and Industrial Research*.

Nakul Johri is an applications engineer within the Customer Quality Management Team at I2 Technologies, Dallas, Texas. He obtained his Master of Science in Information Technology and Management from the Information Systems and Operations Management Department at The University of North Carolina at Greensboro.)