!!overridecount0

Performance Evaluation & Measurement of E-Business/ERP Systems

Rakesh Radhakrishnan
I T Architect
Sun Microsystems, Inc.

Packaged applications are being widely adopted at mid- and large-size companies to automate both internal and business-to-business processes. E-Business/ERP implementations are usually part of a broader enterprise computing effort involving the integration of Web, server and host-based applications. In many cases, E-Business/ERP systems constitute the engine upon which new business processes are deployed. These business transactions are essential elements in the drive to deploy new products and services an support new customer relationship management efforts.With so much resting on these systems, it's imperative that they operate at peak performance. IT departments need to adopt measurements for tracking the performance of their E-Business/ERP applications both during implementation and production phases of operation. Companies that deploy these complex, multi-application business applications must continually adjust them to meet performance and capacity demands as the system scales. The rate with which these applications are deployed by organizations, the pace at which data grows and the size of the user population contribute to the type of system and network infrastructure required. These factors also influence the application architecture adopted to support large-scale E-Business/ERP deployments.

When developing an E-Business/ERP deployment plan, factors such as the operating systems, databases, Web servers, and middleware used must be taken into account. Any type of performance evaluation should include extensive stress, system integration, unit and parallel tests to examine every facet of the system and how it performs in relation to the underlying infrastructure.

Testing should be addressed at every stage of the E-Business/ERP project from pilot deployment to full blow roll-out. Tests should also be performed after lifecycle changes and software upgrades are made to the system.

There are 9 basic steps used to evaluate and measure the performance of an E-Business/ERP system:

1. Definition of performance requirements such as minimal acceptable response time for all end-to-end tasks and acceptable network latency of an interface
2. Creation of a test bed with all components in place
3. Configuration of entire infrastructure based on vendor recommendations
4. Execution of unit tests for each application in the package to ensure all required functions work
5. Execution of integration test to ensure compatibility and connectivity

between all components

6. Launch monitoring tools for underlying systems including operating systems, databases and middleware

7.Creation of baseline reference of the response times for all key tasks when the system is not under stress

8.Creation of baseline reference of response times for all key tasks under varying load conditions

9.If requirements are not met, make necessary changes in the hardware, software and networks and repeat the tests

Without successfully completing tests aimed at the functionality offered with each software module in the E-Business/ERP suite in isolation (unit) and in conjunction with each other (integration), a full blown stress test can't be orchestrated. All individual components should be working in isolation and in relation to one another for a successful stress test to be accomplished, since both the functionality and the integration between these components are exercised heavily during a stress test.

When embarking on a detailed test plan, it's important to look at the system architecture and how it influences the E-Business/ERP/ERP deployment and system performance. The following is a list of the architectural components associated with a large-scale web-enabled E-Business/ERP/ERP implementation:

- Back-end network infrastructure such as fiber-based storage area network, clustered storage area network or high speed backbone
- Scaleable servers such as symmetrical multiprocessing and NUMA systems
- Back-end software components such as databases and middleware
- Back-end connection
- Client hardware
- Client software
- Client-side ISP connection

A typical connection flows from the client machine running the OS and browser to the back-end via the virtual private network provided by the ISP, Web server connecting to the application server, and finally connecting to the database server. Performance issues and bottlenecks can stem from any of these connections, or from a component and the interface with which the component interacts.

The enormity and complexity of E-Business/ERP/ERP implementation environments can give rise to a variety of bottlenecks stemming from network connections, database locks, memory usage, router latency, and firewall problems. A subset of such problems could include:

- Too many software modules installed and cached
- Outdated and insufficient hardware Outdated and incompatible software Slow modem access

- Lack of virtual memory or disk space
- Router latency resulting from too many hops
- Clogged and insufficient network infrastructure
- Improper operating system configuration
- Insufficient database locks
- Inefficient use of application partitioning
- Inefficient use of logical memory manager
- Improper RAID configuration
- Slow storage sub-systems
- Components that are not multi-threaded and ones that do not scale across multiple CPU's
- Efficient use of addressable memory space
- Lack of efficient distribution of data between disks or storage sub-systems
- Poor SQL syntax
- No load balancing
- Component interaction latencies Interface latencies
-

It's possible to add several thousand factors like the ones above that contribute to poor performance. However, to identify the critical ones that contribute significantly to slow performance, a detailed analysis of the 9 steps associated with performance measurements is the best place to start.

Step 1

Define performance requirements

Instead of defining performance requirements in terms of the number of transactions per minute, which is relevant when measuring database performance in isolation, or network latency, which is relevant for a network resource planning study, performance requirements should define the end-to-end requirements that are the most relevant to system users. For example, the response time of a human resources or financial system should look at the response time from start-to-finish, and evaluate tasks such as:

Selecting a product and placing it in the shopping cart

Ordering a list of products selected

posting a job opening

submitting a leave application ordering goods

adding a new supplier

submitting a requisition

adding an asset

submitting a purchase order

Users of the system are only concerned about the time it takes for them to accomplish specific tasks using the E-Business/ERP system. They're not concerned about the internals of the system and how many millionoperations per second a server can handle.

By measuring the performance of tasks that exercise all components of the system, an administrator can baseline the application's performance, identify bottlenecks, make changes and baseline the effects once again.

When defining performance requirements it's important to consider the user input and manage expectations. Critical tasks should be identified. If response time for generating a specific report that actually analyzes several gigabits of data is unrealistically set below one second by the user, expectations have to be managed.

It's also important to define other performance parameters along with requirements such as a typical work load or peak workload. A peak workload could include 1,000 WAN users and 200 local users, including 20 power users accessing the financial module. The peak period could include 20 batch processes, 10 application domains, 30 queries, and 30 print jobs running on the system. Measuring the response time of each identified task and sub-task with no workload, typical workload and peak load iscritical to measuring and predicting application performance.

Step 2

Set up a test bed with all the components in place

It's extremely important to establish a test bed that includes all components and is configured as closeto the production environment as possible. All E-Business/ERP application vendors provide certification information about the platforms supported. The back-end hardware used in a test bed should be based on the hardware recommendations for a production environment. If a giga-bit switch is used on the back-end to speed up data transfers between the application server, database and the interfaced system in the production environment, a similar one should be set up as part of the test bed.

To simulate wide area network and Internet traffic, client machines need to be configured outside the router that connects the back-end network to the WAN or the Internet. If access to the application is expected from client machines with 64MB of memory, 300MHz Intel processor and a 56KB modem, this combination of hardware for the client side should be setup as part of the test bed.

Also as part of the test bed setup, sample data, test monitoring tools and test automation tools all need to be in place prior to exercising any efforts with regards to configuration and tuning. These include protocol analyzer, a test automation tool,

operating system and monitoring tool.

Step 3

Configure and tune the entire infrastructure based on vendor recommendations and certifications. Before beginning any testing, it's important to configure and tune the entire infrastructure but only on given parameters and variables. Use the vendor's recommendations as a baseline. It's impossible to get recommendations for all parameters for a platform from specific vendors and even if it's possible, this could introduce unnecessary complexities given the thousands of variables that can be tuned at each layer.

Certain key recommendations could include: total memory configuration for the database, total allowable locks, network packet size, number of I/O demons, number of application domains, buffered I/O size, customized cache configurations, recommended OS patches, recommended client connectivity tools version and patches and network interface card throughput.

Any tests attempted without accomplishing this third step is basically a wasted effort. Baselining the response time without the default product configurations could lead to misleading test results. The configuration and tuning guidelines given by the respective vendors can only act as a starting point. Further changes in specific hardware and software configuration that can improve performance will be dependent on any additional findings based on the remaining steps associated with this process. Keep in mind that steps 3 through 8 are iterative processes that must be repeated until the performance requirements are met.

Step 4

Conduct unit tests to ensure all required functions work. Based on the performance requirements defined in step 1, all identified functions like adding an asset and generating an accounts payable report ought to be tested first in isolation to ensure that these tasks can be accomplished end-to-end using the system. This is a precursor to the remaining tests that need to be accomplished. Unit tests are meant to ensure that the product works

the way it's supposed to. Although the unit tests are conducted in isolation from one another, it's still expected that all the components that are part of the system are in place and function appropriately, but are not running at that point in time. Automated testing tools can be used to accomplish these tasks.

Step 5

Conduct an integration test to ensure compatibility and connectivity between all components. This step is critical since it's the first occasion when the entire system is put to the test. All functions, sub-functions and their interactions are tested concurrently. To accomplish this task the computing environment and all its components must be active. The main purpose of this the test is to ensure that all components can talk to each other. Any compatibility and connectivity issues that exist will be revealed. Before studying and baselining the performance behavior of each of the components in the system, it's important to fix these compatibility and connectivity issues.

An example of a problem could be a connectivity time out by an application server for the 100001st user when the five application servers are configured to provide access to 20000 users each. Another example is bad configuration pointing to different versions of connectivity libraries. By completing this step successfully, you ensure that the system in its entirety works, even if it is not yet at the acceptable level of performance.

Step 6

Kickoff all monitoring. At this phase we kick off all monitoring tools that have been put in place during step 2. Baseline and track the resource consumption of these monitoring tools since they can act as a skew to the test measurements. An example of the monitoring tools that can be used based on the sample platform identified in step 2 includes:

Symon for Solaris 2.6 to measure resource usage of the database, application server, OLAP server memory consumption, CPU usage and I/O activity

Sybase Monitor Server for ASE 11.5.1 to measure database usage, lock contention and cache hit ratio

Performance Monitor for Windows NT 4.0 to measure resource consumption on the file server and secondary application server

LANanalysis tool to identify network bandwidth utilization

When starting these performance measurement tools get output both in the form of real-time visualization to study instant changes to the degradation or boosts in response time/performance and as a file for the purpose of analyzing the data.

Step 7

Baseline response time for all key tasks under no stress. Once all the monitoring tools and the entire infrastructure is up and running, we get a baseline of the response time of all identified critical tasks without simulating the typical or the peak workload. This gives us an idea about how individual tasks perform and how quickly they respond when the back-end is basically idle. The response time for each task is computed by submitting them and measuring the start-time to end-time with a tool like a stopwatch or time scripts. The numbers generated through this step can be used for comparative analysis purposes.

Step 8

Baseline response time for all key tasks under different load conditions

Two key load conditions under which the response times are measured once again include the typicalworkload and the peak load. Assume that a task like submitting a requisition is expected to be completed in less than 40 seconds. When running these tests several times, we come up with averages of 22 seconds under idle conditions, 32 seconds under typical work-load and 52 seconds under peak-load conditions. Now, we know clearly that there would be conditions when the expected levels of performance cannot be achieved with the given infrastructure and its configuration.

Testing Under Various Conditions Task #1 (Submit a Requisition)

	Start-time	End-time	Total-time (Under an idle work load)
#1	10:00:00	10:00:21	21 seconds
#2	10:01:00	10:01:23	23 seconds
#3	10:02:00	10:02:22	22 seconds
#4	10:03:00	10:03:22	22 seconds
#5	10:04:00	10:04:22	22 seconds

The above table illustrates the statistics generated for one such task. A typical test for a large-scale deployment of a web-enabled E-Business/ERP system will include hundreds of such tasks and may even range in the thousands.

Proper analysis of the data generated by the different monitoring tools will reveal the bottleneck or the cause for the slow response to submitting a requisition. By alleviating such a bottleneck the response time can be improved slowly but steadily. By running steps 3 to 8 repeatedly, bottlenecks associated with every task can be identified and fixed. In cases where the performance requirements are met for specific tasks after running through the process the first time, it's important to continue baselining after every change to ensure no performance degradation associated with a specific task has taken place.

Step 9

If requirements are not met, make necessary changes in the form of tuning

Tweaking, tuning, and adding or removing the relevant resource after running this process several times is an optimal way of meeting performance requirements for all tasks. In one situation the bottleneck could be identified as a process such as a credit reporting process. When looking at the process flow, it will become apparent this task can't handle more than a limited number of concurrent requests. Given asituation like this, the configuration of this process could be modified to spawn more of the same to accommodate more requests.

In another situation the firewall and its authentication/authorization process could be identified as a bottleneck. A load-balanced cluster of firewall servers could resolve this issue. A particular server's CPU utilization could cause a bottleneck and something as simple as adding CPU's could be the answer to the problem. Similarly, if the bottleneck is caused by I/O contention on a specific device, implementing striped mirrors for that device could be the answer.

By adopting this methodology, all relevant macro level and micro level issues and bottlenecks can be identified. This enables the overall throughput of the system to be optimized. An added benefit of this process is ensuring functionality requirements and addressing compatibility requirements.

Finally, by embracing this process you can tell if the applications will work under realistic workloads and prove if service level agreements will be met. These tests will also help an IT department understand how the system will affect existing IT infrastructure. These tests should help gauge capacity planning efforts during the system's lifecycle as well.

Rakesh Radhakrishnan

Rakesh Radhakrishnan is an IT architect with Sun Microsystems, Inc., in McLean, Virginia, specializing in e-commerce and enterprise applications.

Rakesh holds an M.B.A. and M.S. (CS) and is an enterprise computing certified systems engineer from Sun Microsystems and a systems architect pro from PeopleSoft.

He can be reached at rakesh.radhakrishnan@east.sun.com.