Performance Prediction, Sizing and Capacity Planning for Distributed E-Commerce Applications

by Samuel D. Kounev (skounev@ito.tu-darmstadt.de)
Information Technology Transfer Office

Abstract

Modern e-commerce systems are becoming increasingly complex and dependent on middleware technologies to provide the scalability and performance levels needed to survive in the competitive e-business landscape. Developing scalable middleware applications is a challenging task both in assuring that the required functionality will be present and in guaranteeing that the functionality will be delivered with an acceptable performance. This document starts by discussing the issues of performance prediction, sizing and capacity planning for distributed e-commerce applications. Then it goes on by describing the approaches to dealing with these issues and examines the difficulties caused by the complexity of modern e-commerce application architectures.

Contents:

1. Sizing and Capacity Planning Issues

Most of today's e-commerce environments are based on distributed, multi-tiered, component-based architectures. The inherent complexity of such architectures makes it extremely difficult for system deployers to estimate and plan the size and

capacity needed to provide the desired service levels. System deployers are often faced with the following questions:

- What are the maximum load levels that the system can handle in its current state? For example - how many concurrent users / requests / transactions per second can be served?

- What is the average response time, throughput and processor utilization?

- How does performance change as workload increases?

- How much hardware (e.g. servers, CPUs) would be required to make sure that the system will be able to handle the expected workload while keeping certain performance levels?

- How much costs would be incurred in providing the needed resources?

- Which components of the system affect performance the most? Are they potential bottlenecks?

Since the issues of sizing and performance prediction are often considered "way too complex" to address formally, system deployers usually rely on their intuition, ad hoc procedures and general rules of thumb. In most cases the overall capacity of systems is unknown and capacity planning and procurement is done without a clearly defined methodology. This lack of proactive and continuous capacity planning procedure may lead to unexpected unavailability and performance problems, caused by the inability of the system to handle increasing workloads.

2. Approaches to Performance Prediction and Capacity Planning

In order to avoid the above mentioned problems system designers need to address the issues of sizing and capacity planning in a more formal and systematic way. In general there are two most popular approaches to the problem of performance prediction and capacity planning:

A. System Load Testing
B. System Performance Modeling

Both of these approaches have their strong and weak sides. The former is more practical, the latter more mathematical. Both of them provide information that can be used as a basis for sizing and capacity planning. In fact they should be considered as complementing each other, rather than, as two alternative

approaches. Experience shows that best results are obtained only when both approaches are used together in the sizing and capacity planning process.

(A). System Load Testing

System load testing is a method used to measure the end-to-end performance and scalability characteristics of an application. System load testing uses **load-testing tools** that generate artificial workloads on the system and measure its performance. Sophisticated load testing tools can emulate hundreds of thousands of so-called 'virtual users' that mimic real users interacting with the entire system infrastructure. While tests are run, system components are monitored and performance metrics (e.g. response time, latency, utilization and throughput) are measured. Results obtained in this way can be used to identify and isolate system bottlenecks, fine-tune the application components and predict the end-to-end system scalability.

The main advantage of this approach is that it is quite practical and can provide some accurate performance data that can be used as a basis for performance tuning and optimization. The results from load testing can also be used to make some rough estimates of the adequate sizing and capacity of the production environment. On the downside, we can mention that load testing is usually quite expensive and time-consuming. Costs come from the purchase of load testing tools, the setting up of the testing environment and the organization of the actual tests.

Another disadvantage stems from the fact that, as already alluded to, testing is usually carried out in a separate testing environment, which is much smaller and much less scalable than the production environment where the system is to be deployed. This is due to the fact that it would be too expensive to set up a production-like environment for the testing process. Load testing therefore measures the scalability and performance of the system in the testing environment and does not directly address the issue of predicting performance in the production environment.

While load testing can help system designers to identify bottlenecks and fine-tune systems prior to production, it does not in itself provide a comprehensive solution to the sizing and capacity planning issues discussed earlier. Nonetheless, as will be shown later, load testing results can be used as a basis for performance prediction and capacity planning. Load testing could therefore be considered the first step in the capacity planning process.

(B). System Performance Modeling

In the second approach performance models are built which capture the performance and scalability characteristics of the system under study. Models represent the way system resources are used by the workload and capture the main factors determining system performance. Basically performance models can be grouped into two most common categories: *analytic* and *simulation models*.

Simulation models are computer programs that mimic the behavior of a system as transactions flow through the various simulated resources. The structure of a simulation program is based on the states of the simulated system and events that change the system state. Simulation programs measure performance by counting events and the duration of relevant conditions of the system. The main advantage of simulation models is their great generality and the fact that they can be made as accurate as desired. On the other hand simulation programs may be expensive to develop and to run. More detailed simulation models tend to require more detailed data and more time for execution, thus increasing the cost of using simulation.

Analytic models are based on a set of formulas and computational algorithms used to generate performance metrics from model parameters. Such models are normally based on the *theory of queueing networks* and are used to predict the performance of a system as a function of the system's description and workload parameters. Basically, analytic performance models require as input information such as workload intensity (e.g. arrival rate, number of clients) and the service demands placed by the load on each resource of the system. By applying some mathematical formulas and laws one can estimate performance measures of the system such as average response time, transaction throughput and resource utilization. The results could be used to plan the sizing and capacity of the production environment needed to guarantee that applications would be able to meet future workloads while keeping performance at an acceptable level. Of course any results obtained in this way would always be approximate and there would be no absolute guarantee that systems will behave exactly as indicated by the preliminary analysis. The more detailed and representative a model is, the more accurate would the results be. Analytical models are, in general, the technique of choice for capacity planning and scalability analysis of distributed e-commerce systems.

As already mentioned, in order to solve an analytical model, one needs to first measure the service demands placed on the various system resources by the workload components. The service demand parameters specify the total amount of service time required by each basic workload component at each system resource. Examples include CPU time of transactions at the application server, the total transmission time of replies from the Web server back to the user, and the total I/O

time per transaction at the database server. In order to obtain these parameters one needs to run some tests on the system and make some measurements. The most natural approach to do this is to use the load testing tools described earlier in order to simulate load on the system so that measurements can be made. Once load is generated one can use available system monitoring tools, accounting systems and program analyzers to get some performance data. This data can then be used to provide some estimate values for the service demand parameters. So, as we can see, load testing and performance modeling can be used as two complementary methods in the process of capacity planning.

3. Technological Architectures of Modern E-Commerce Applications

Unfortunately, as already alluded to, the complexity of modern e-commerce applications makes the task of performance modeling and analysis extremely difficult and challenging. The main difficulty comes from the distribution of application logic in multi-tiered component-based systems. Scaling such systems from end-to-end means managing the performance and capacities of each individual component within each tier.

A typical distributed web-based e-commerce application would have the following logical application logic partitioning:

- **Presentation Layer**
  Manages user interfaces and user interaction.

- **Business Logic Layer**
  Performs business operations and models business processes.

- **Data Layer**
  Used by the business logic layer to persist business data.

These layers are mapped to corresponding physical tiers where the actual application code resides and is executed. Usually we can expect to have the following tiers:

- **Presentation Tier**
  Runs within a *Web Server*, which hosts a number of so-called *web components*, based on presentation-related technologies such as: HTML, ASP, Servlets, JSP, etc. In a clustered environment we can have several Web servers hosting the same web components in order to provide maximum scalability of the presentation tier. Web Routers can be used to distribute traffic among the available web servers.

- **Business Logic Tier**
  Usually runs within an *Application Server*, which provides a containment environment for *business logic components* and a number of middleware services such as transaction management, persistence, state management, security, resource pooling, cashing, etc. In a multi-node configuration this tier consists of several machines each running an instance of the Application Server. Modern application servers provide *component clustering functionality*, which handles the complexities of replication, load balancing and failover for business logic components (services typical to TP Monitors). More concretely, an intelligent active load balancer tracks information about the load of each application server and distributes client requests so that none of the servers is overloaded. Furthermore, the state of business components is replicated across servers. In this way, in case one of the servers fails, requests can be rerouted to another server, which can automatically restore the needed components and provide transparent failover.

- **Data Tier**
  Consists of one or more databases and a *Database Server*, which manages data persistence. This tier may contain data-related logic in the form of stored procedures. As with web servers and application servers, database servers can also be clustered in order to provide maximum scalability, load balancing and fault tolerance in the database tier.

The above-described N-tiered component-based architecture is depicted in Fig. 1. As we can see the situation is further complicated by the addition of *firewalls* and the fact that often *legacy systems* must be integrated into the overall application architecture.

4. Conclusion

Performance of applications based on the described architecture seems to be a function of too many variables whose behavior is really hard to predict. Little work has been done to address the issues of performance prediction and sizing. There is some work under way, but as of this writing, there exists no standard procedure or methodology, which can be followed. Solutions are usually proprietary and address particular situations. Moreover they usually target specific components of the system and rarely address the overall system scalability and end-to-end performance. There remains the need to bridge the gap between the theoretical and practical work and combine different techniques in a way that would give cost-effective and practical results.

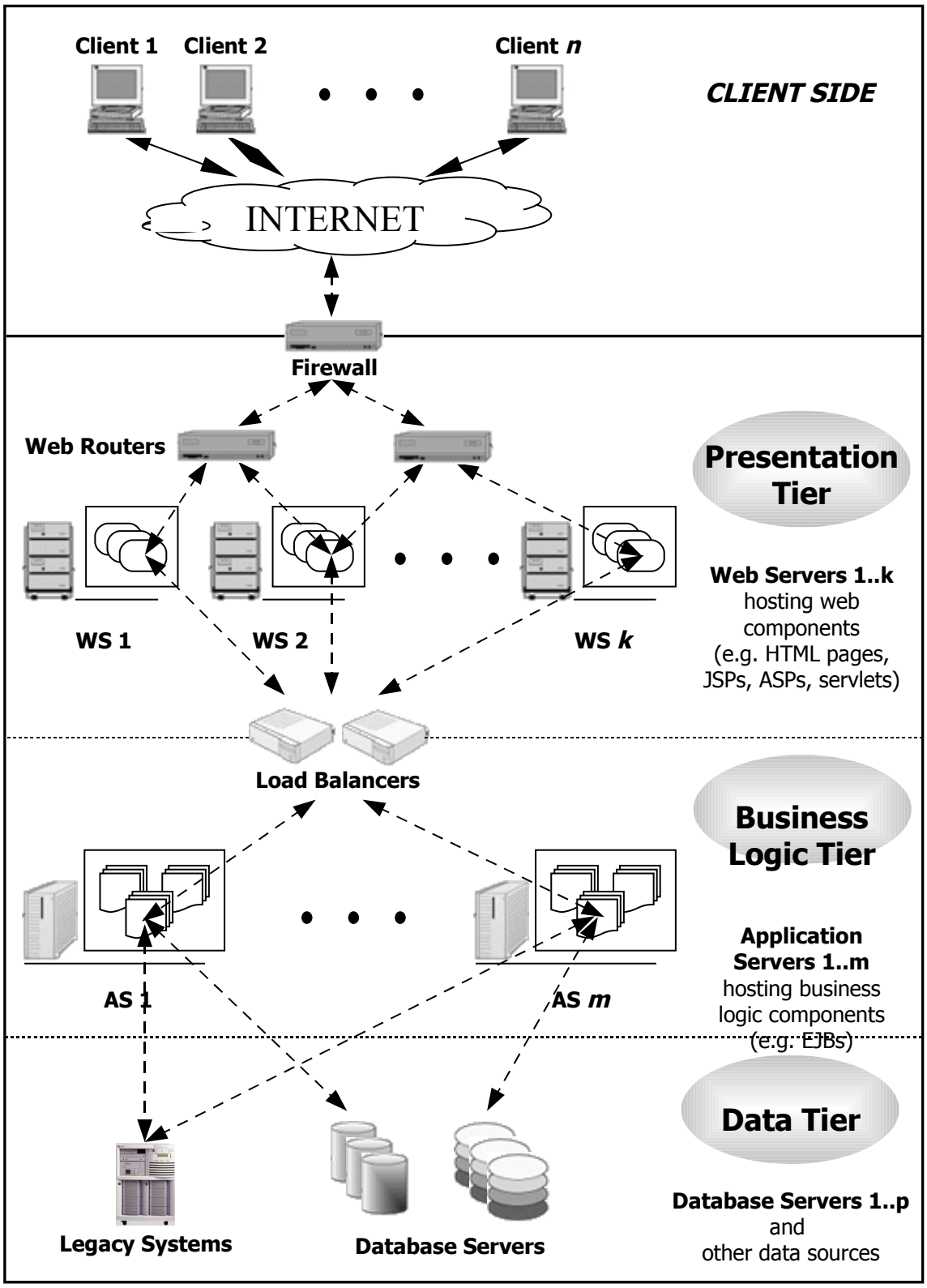

Figure 1: Architecture of Distributed N-Tiered Component-Based Application