Capacity and Performance of Distributed Ente

Analytic and simulation models enhance the reengineering and tuning of large client/server distributed systems.

The world of large-scale computing dominated by mainframes was relatively simple. All applications and related data were co-located, and many thorny issues of distributed computing such as remote data access, splitting databases, and network latency were either nonexistent or presented easier problems. Unfortunately, mainframes or even clusters of mainframes did not scale well. The client/server architecture (CSA) has been introduced to overcome this difficulty.

Has CSA delivered as expected? The answer is "Yes, but..." Ideally, CSA solutions scale better than mainframe solutions, but this can be accomplished by expending additional efforts required for architecting and tuning CSA systems. An effective approach to this task is building and exercising system workload and performance models.

Here, we share our experience of reengineering and tuning large CSA systems at Boeing. We consider systems that have been designed for tens of thousands of users and run applications such as Baan, PeopleSoft, and CATIA (a CAD system).

A system of this kind has to satisfy several requirements, including adequate computational performance dictated by the company business processes and scalability, that is, the ability to accept new users and applications without severe performance degradation or expensive system architectural redesigns.

By attempting to solve this problem for several large applications, we have gained valuable experience that has advanced our understanding of the methodologies and tools needed for accomplishing the goal. Our experience may not be universal, but many other large companies face problems similar to Boeing's and could benefit from the lessons we learned.

The key technical challenges we encountered include:

- The workload predictions may be uncertain or inaccurate. Unless the designed system has been at least partly in production, the data needed for modeling may be unavailable.
- In multiple vendor environments, the responsibility for the system integration and performance tends to be ill-defined. By default it becomes the customer's responsibility.
- Very large systems (large number of software, hardware components, huge amounts of data, and numbers of users) are frequently outside the scope of a vendor's experience. They are tempted to advocate linear extrapolations from smaller and less complex systems. The correct determination of system capacity and performance requires accurate modeling that exhibits highly nonlinear behavior.
- In general, hardware solutions advance quickly and are ahead of software that runs on new architectures. An example of this gap is parallel computing, which can offer significant speedups and scaleups on multiprocessor architectures. Often this advantage cannot be realized if software is singlethreaded.

One of the greatest difficulties of any analysis is understanding the customer problem and turning it into a formal problem statement. The problem must be translated from customer terminology into a formal statement amenable to mathematical analysis and programming. The capacity and performance analysis problem can be divided into two broad categories: Design of a new system from requirements, and reengineering and tuning a system already in production.

From a user's perspective, the most common measure of performance is usually stated in a service-level

Analysis rprise Systems

James A. Aries, Subhankar Banerjee, Marc S. Brittan, Eric Dillon, Janusz S. Kowalik, and John P. Lixvar

agreement that describes the level of performance required by the user community. Common user-based metrics include response time, percent uptime, and other metrics.

From the system manager's perspective, there are a large number of performance metrics of overall system health, such as CPU, network, disk utilization, cache hits and misses, and context switches. These metrics serve to measure both current system performance and the maximum system workload that can be supported while still meeting service requirements. The systems manager is interested in both the day-to-day performance of the system, as well as the ability of the system to support growth in the user workload. The manager must also know whether the system can support periodic spikes in workload that can usually be managed by workload scheduling if system capacity is known.

From the mathematical modeler's perspective, the service-level agreements act as hard constraints, with CPU utilization, network utilization, and other metrics serving as measures of performance. The modeler will typically vary a number of system parameters such as network bandwidth, CPU speed, and workload in an effort to improve system performance and assess overall system capacity. For systems currently in use, a typical starting point is to use some of the metrics from production data mining and build a model that mimics production behavior. This gives us a calibration point for future extrapolation. Once we are confident we can reproduce the performance seen in production, we may use the model in its predictive mode by scaling the workload to find bottlenecks, response times, capacities, and other parameters of interest for the system. The overall mathematical model can be broken down into a workload model, a performance model, and a cost model.

In building a workload model, the first step is to determine the system resources (CPU, network traffic, server calls, and so on) required to support it. Each server may in turn spawn requests to other servers before the user's job/query is resolved. To model the system, we need to know the load per query on each machine/resource, including remote calls generated by one node to another node. For large systems, this is typically stated in terms of probability distributions where a user of a particular type generates a probabilistic number of queries per unit time.

The performance model takes into account the physical hardware, connectivity, architecture, software, and workload model. This model is used to analyze the current system performance and predict future performance under varying workload and architecture changes. An important part of modeling a large system is determining whether it will scale well as additional users, servers, and other hardware and software are added to the existing system. The performance model should be flexible enough to perform ad hoc studies, such as moving workload assignments, adding new servers, splitting databases, and changing connectivity.

The cost model may include total system and life cycle costs, including hardware, maintenance costs, and projected costs of future revisions to the system. Cost is frequently modeled as a constraint (find a system with the best performance characteristics subject to a budget constraint), or as part of an objective function (find the lowest cost system that can meet service level agreements).

Designing New Systems, Reengineering Old Ones

We have the greatest opportunity to build performance and scalability into a new system from the outset. Once we have more than 100,000 users online with millions of lines of legacy code and database information, we are limited in our ability to change architecture or software. The cost of retraining the user community and rewriting code can be prohibitive after the system is in place.

For systems not yet built, there is the additional complexity of estimating usage patterns and understanding server loads from programs not yet written. Frequently, we must resort to benchmark studies from vendors and historical studies from the literature to predict resource requirements. We can then make load

estimates in the proposed user environment and make proposals for system hardware and architecture.

It is a relatively low-cost task to perform a large suite of computer simulations of a proposed system compared with the cost of building a system and rebuilding it if it cannot handle the workload. Adding new servers, changing server computing power, or changing connectivity is a straightforward programming task and ensures the right hardware has been ordered.

We are more constrained when redesigning an existing system because there is an existing infrastructure and user base that cannot be shut down during the transition. Unlike a new system, the existing system can be analyzed for usage patterns and other information that characterizes system usage. For a new system we only have projections of production system usage. For an existing system, production data mining provides an excellent starting point for our analysis.

When confronted with the task of reengineering and tuning an existing system, we often have an additional constraint—the reengineered system must be backward compatible with the existing production system and be capable of seamless integration with minimal downtime. This, of course, implies we are usually constrained to operate within an architectural framework close to the existing system. Radically different topologies, workload assignments, and software are usually out of scope.

Workload Characterization, Performance Testing, and Input Parameter Estimation

Capacity planning and performance analysis of a distributed system requires a flexible and representative model of the workload. Such a model must represent the essential workload complexity while remaining simple enough to provide timely analysis results. In most studies it is necessary to collect and analyze performance data to develop a quantitative model of the workload components. Thus, the collection and analysis of performance data should be integrated into the performance study plan. The data must be sufficient to estimate the model parameters and the model must reflect the essential behavior of the system for this process to work. Data collection is a time-consuming activity, and opportunities to collect usable data while the system is under development can be infrequent. It is common that the desired level of detail in a performance model be reduced because the available data will not support it. Thus, identification and collection of the required data early in the lifetime of a study will contribute greatly to the end result. The development of a workload model primarily involves the following four steps:

Workload characterization involves specifying the nature of the tasks that will be performed on the system. This requires a good understanding of the business processes that will be used during the time period of the performance study. User roles can then be defined in terms of the business processes and decomposed into the computational transactions required to complete them. The workload is then defined by specifying the frequency and mixture of the different user roles. These user roles can be scripted to serve as a basis for automated load testing and can also serve as the basis for the workload model when the service demands associated with each user are quantified through measurement.

One of the greatest difficulties of any analysis is understanding the customer problem and turning it into a formal problem statement.

Workload model development. A quantifiable workload model requires that service demands associated with the workload components be estimated. If the model is transaction-based, it is necessary to estimate the transaction arrival rates and the service demands associated with the various transaction types or classes. The service demands will typically include CPU, memory, interprocess communications, and I/O requests.

Observability of the service demands is often an issue due to the extreme complexity of large-scale distributed systems. The service demands will be distributed over the various types of servers and communication links comprising the elements of a multitiered system. The workload is distributed over many software and hardware components that communicate over diverse data links with differing mechanisms. Large distributed database systems will service transaction data requests with a small number of continuously running processes that will spawn many child processes in response to service requests and will defer some of their contribution to the service demand to cache refreshes and other housekeeping tasks. Thus, the direct isolation and measurement of transaction service demands from a production or load test environment is not always possible. It may be necessary to divide the service demand into steady state or periodic components reflecting the overhead and into other compo-

nents proportional to the number of users of a given type. In practical situations it is necessary to combine diverse results from many sources and observations to calibrate a model reflecting the direct and deferred service demands resulting from workload elements.

The number of transaction types can be unmanageably large. This is often addressed by creating a class model of the workload wherein transactions with similar service demands and arrival patterns are clustered together. A major objective of a performance study will be to estimate the distribution of transaction response times as the load increases. Properly classifying transactions into classes is a key step in supporting this process.

Performance testing and data collection. It is necessary to collect data to convert a workload defined in terms of business processes into a model that can be quantified as service demands on a computer system. If data is collected from a production environment, it is not possible to control the environment. The data collection and analysis will then have to be designed to estimate the service demands from a mix of work where their effects cannot be directly measured. This may require more indirect parameter estimation techniques or elevating the level of detail input to the performance model. If load testing is to be performed, this problem is somewhat simplified because the definition of the data collection and analysis should be an integral part of the test design. Detailed planning is necessary to define the test suites, develop scripts for automated load testing, define the monitoring tools to be used, define what data will be collected, and define how the results will be stored and managed. Tools for data collection include:

- Performance monitors installed on each server (and typically supplied by the hardware vendor) are generally specific to the server architecture and operating system. Examples include Hewlett-Packard MeasureWare, Sequent Performance Evaluation Package, and Microsoft Perfmon. This software will supply utilization of system components broken out by process ID and a great many other details abut the status of the system.
- Data logs supported internally by software components of the architecture such as the transaction monitor, the database system, and the application software itself. These logs typically contain information about the number and type of transactions by time interval.
- Code instrumentation specifically inserted into the application software to provide checkpoints. This can be done by the software vendor, the system integrator, or may be automatically supplied as part of the emerging Application Response Measurement (ARM) standard.
- Instrumentation supplied with automated load testing software (for example, LoadRunner, PreVue CS/X) will typically collect transaction response time data and may provide additional information about the distribution of response times over the system components.

Input parameter estimation. Once performance measurement data has been collected, the parameters of the workload model need to be estimated. This consists of fitting the model parameters to match the observed behavior of the system. Regression analysis can be a useful tool for fitting parameters based on multiple test results, while a very simple model with one service class may use simple queueing models to derive service times based on the observed utilization and response times. A more sophisticated model may break out components of the load due to background processes, work rate proportional processes, and deferred processes. Data then must be collected in a manner that allows the model parameters to be independently observed and estimated. A good understanding of the design and behavior of the system is necessary to construct such a model. The usefulness of the model as a predictive tool will depend on how well the parameters reflect the behavioral characteristics of the system as well as the accuracy of the parameter estimation.

Load Testing for Performance Analysis

Load testing (LT) is considered a good approach for both performance and scalability analysis as well as collecting data as input for performance models. There are several competing COTS tools (LoadRunner and Performance Studio, for example) available for LT. Our approach is to combine the two methodologies, namely, load testing with performance modeling to analyze system performance and predict capacity. In this approach, we would use an LT tool with a moderate number of simulated users. We then use the test data collected as input to the performance model to study various what-if scenarios as well as predict knee-of-the-curve or system performance under heavy load conditions.

The client/server systems we encounter at Boeing are very complex, and it is difficult to test the performance and scalability of these systems. Whereas single-user testing focuses primarily on functionality and the user interface of a single application, load testing focuses on the performance, scalability, and reliability of an entire client/server system under various load scenarios. Traditional manual testing methods offer only a partial solution to load testing. A client/server system can be manually tested by constructing an environ-

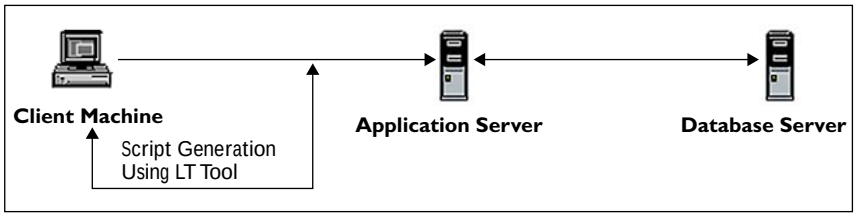

Figure 1. Script generation using load-testing tool.

ment where many users work simultaneously on the system. Each user works at a single machine and submits input to the system. This kind of testing has been used extensively for functional testing of various Boeing enterprise systems. However, the manual testing method is not very reliable for performance analysis and suffers from numerous drawbacks. An LT tool addresses the drawbacks of manual testing with useful solutions. It reduces the personnel requirements by replacing human users with virtual users. These virtual users emulate the behavior of real users operating real applications. Because numerous virtual users can run on a single computer, it reduces the hardware requirements. Because these tests are fully automated, they can be easily repeated.

A real user session can be captured automatically into a script using an LT tool. This script can then be manually modified and parameterized to suit the needs of a test scenario as well as to emulate many users who would be running a similar script. Later, a scenario can be built out of these scripts to represent a complex mix of transactions. Scheduling of script execution within a scenario can be achieved for various arrival rates with different user types and mix of transactions. The commercially available LT tools support various client/server applications, including Database, Web, Baan, Java, Tuxedo, PeopleSoft, and others. The script generation happens on the client end where the LT tool captures the protocols issued from the client software. A three-tier client/server system is illustrated in Figure 1.

The results of the LT data can be used to estimate useful parameters as input to the performance models. One benefit of using data from LT for performance models is these tests can be performed in a controlled environment and repeated with ease, producing reliable input parameters for performance models.

Modeling Objectives

Modeling often becomes the solution of choice in assessing performance and scalability when the system of interest is unavailable for benchmarking. However, building a performance model in the absence of such information is no easy task. Enterprise systems frequently consist of many components arrayed in highly complex configurations: heterogeneous, geographically distributed servers linked through many different network interconnects, COTS applications based on proprietary middleware, databases of enormous size, and so on. Understanding the interactions within such a system is the first, and perhaps most difficult, step in building an effective model (see Figure 2).

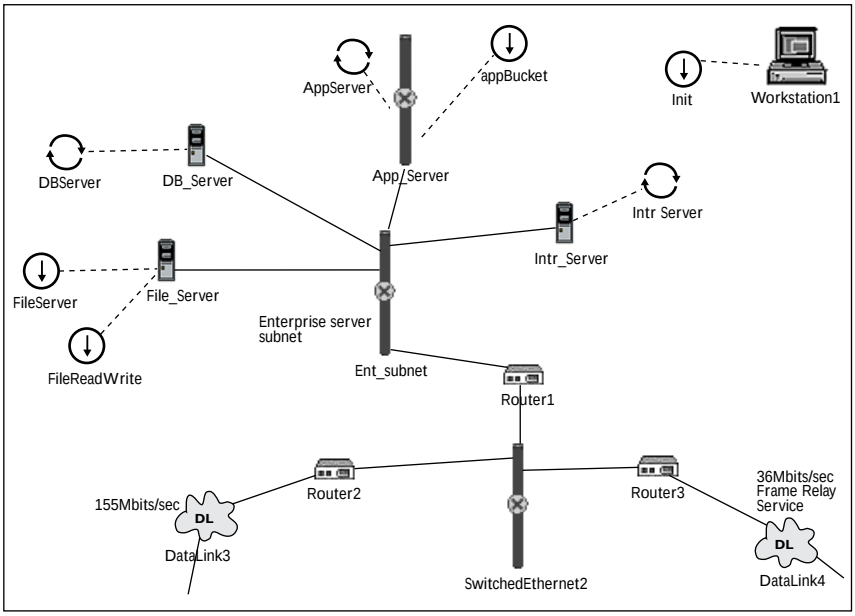

Figure 2. Enterprise system model generated by Strategizer, a product of HyPerformix.

Defining clear and attainable objectives is the key to a successful modeling project: Why is a performance model being built? What specific issues do we hope to address? Typical questions might include the number of CPUs needed for a particular server under a specified workload, or the impact on transaction response time when batch processing is added to the workload.

Objectives may differ depending on the stage of the design process. When used during early design, an enterprise system model can help build performance into the system at a time when the costs of doing so are minimized. The performance model may also be used to study different implementation choices. For example, should data be replicated at different sites to ensure reasonable response time? Are local application servers

a good idea? How much network bandwidth is needed to support the file server connection? During the reengineering or tuning phase of the system, the objectives may focus on identifying bottlenecks, or assessing the impact of implementing new versions of software, or perhaps even consolidating hardware in an overloaded system. Finally, in addition to all these performance considerations, cost objectives must be factored into the model results before a final recommendation is made on system deployment.

Modeling is all about tradeoffs and compromises; not every detail is needed to capture the essentials of real system behavior. Furthermore, not all of the parameters identified in the workload definition may be available to the modeler. Modeling objectives will drive this tradeoff between model abstraction and model accuracy—there is no need to model the entire computing complex if we are only concerned about specific subsystem performance. Likewise, a performance model may be based on vendor specifications and industry standard benchmarks too coarse to support detailed investigations of individual system components or workload elements. In such cases, the modeler may have to aggregate transactions and consolidate hardware components. These choices may affect the accuracy and level of confidence attached to the model, but quite often these compromises are well justified.

Building a valid and effective performance model is a process of refinement where first-generation models are successively replaced by models of increasing fidelity. Only through this validation and calibration process can we develop tools with predictive capability. Once scaling rules have been determined and the model begins to produce effective results, the analyst's work is still not complete. The models need to be maintained to reflect subsequent hardware and software updates, and they must be designed to anticipate the inevitable series of what-if questions that generally accompany the performance modeling exercise.

Modeling Tools Are Not Perfect

The success of a modeling project is a function of the tools and approaches used. Ideally, a tool should combine all essential input parameters into a model that reflects the actual operation of the system. The tool should then be able to produce statistics with metrics of interest such as CPU utilization, or response time.

Two modeling approaches are typically available: analytical models based on mathematical theory, or computer-based simulation models [4].

Analytical models can provide a quick insight into system behavior by applying the simple principles of queueing theory [1]. However, in order to remain mathematically tractable, these models only accommodate a small number of variables—a situation that usually forces the modeler to oversimplify the real system.

As a complement to analytical modeling, computer-based simulations can potentially model the behavior of every system component. Here again, a tradeoff is needed to balance simulation time with the need for precision. Because most of these tools are based on discrete event simulation [2], generating too many events in a highly detailed model can quickly bring any simulation platform to its knees, giving rise to a situation where the performance and scalability of the simulation tool itself must be considered.

Many simulation tools fall into two main classes: general-purpose simulation tools or languages, and domain-specific simulation tools. A general-purpose simulation tool is totally flexible and open. These tools usually offer simulation extensions to programming languages such as C, C++ [5], or Java [3]. Most of these tools perform sequential simulations, which frequently limit the size or number of model runs to be performed. To overcome this restriction, parallel discrete event simulation tools are now the subject of much investigation. However, these techniques must confront all the well-known problems of parallel programming such as synchronization, load balancing, and scalability.

Domain-specific simulation tools offer many attractive advantages. Because these tools focus on specific applications, they can offer predefined modules with plug-and-play capability. Because these tools are easy to use, people with limited expertise in the art of simulation can build performance models. However, be forewarned that relying on these tools without understanding their built-in assumptions and limitations is one of the quickest ways to sink a performance-modeling project. C

References

1. Allen, A.O. *Statistics and Queuing Theory With Computer Science Applications*. Academic Press.
2. Ball, P. *Introduction to Discrete Event Simulation*; www.strath.ac.uk/Departments/DMEM/MSRG/simulate.html.
3. Howell. F. and McNab, R. SIMJAVA: A discrete event simulation package for Java with applications in computer systems modeling. In *Proceedings of the First International Conference on Web-Based Modeling and Simulation*. Society for Computer Simulation, San Diego, CA, Jan. 1998.
4. Law, A.M. and Kelton, W.D. *Simulation Modeling & Analysis*. McGraw-Hill, New York.
5. Little, M.C. and McCue, D.L. Construction and Use of a Simulation Package in C++. Computing Science Technical Report. University of Newcastle Upon Tyne 437, (July 1993).

James A. Aries, Subhankar Banerjee, Marc S. Brittan, Eric Dillon, Janusz S. Kowalik, John P. Lixvar are members of the Distributed Systems Performance and Scalability team within the Math and Computing Technologies division of Boeing Phantom Works, Seattle, WA.