Dominique A. Heger, Fortuitous Technology, (dom@fortuitous.com), Austin, TX, 2006

# Introduction to Modeling Based Performance Engineering


## Abstract

Building efficient applications as hierarchical compositions of cooperating components (modules) depends significantly on having a thorough knowledge of both, component performance, as well as component interaction. Hence, to make effective decisions concerning configuration, deployment, and interconnection, it is important to develop a Performance Engineering (PE) methodology that complements the traditional application development processes. Ideally, performance techniques such as measurement, analysis, and modeling, which extend the programming and execution environment to be both, performance observable and performance aware, should support the methodology. However, the diversity of functionality and the complexity of the implementations (such as different programming languages, hardware platforms, and thread parallelism modes) challenge the area of performance technology to offer more sound solutions.

The discipline of Software Engineering (SE) incorporates the procedures, methods, and tools that control the software development process. SE provides the foundation for building high quality software products in an efficient and effective manner. There are many dimensions to software quality, including availability, maintainability, scalability, functionality, flexibility, security, and performance. Many of the SE methodologies in use today only focus on ensuring that the software product meets functional requirements, while being developed within a certain time-frame and financial budget. In today's market place, performance requirements are increasingly difficult to manage, while at the same time, are getting more and more exposed. The deployment of complex, multi-tier, client-server based software systems causes a pure analytical modeling based performance analyses approach to become much more complex (while still very valuable though), resulting into performance models that require a much more in-depth experience in mathematics and queuing theory. Just as the nature of the software and computer systems has evolved, so has the user community. Most enterprises no longer design, develop, and deploy systems for their own internal use only, as today's systems are primarily intended for direct customer contact. The evolution of the Internet (particularly the area of E-commerce) has placed a much higher emphasis on systems availability, maintainability, and scalability (with a main focus on actual systems performance).

This article outlines a methodology that *enables systems performance engineering*. The underlying focus is on avoiding the case where performance issues are being identified in either the system test phase, in quality assurance, or even in the initial deployment phase. Mitigating any potential performance risks (early on in the systems life cycle), determining and evaluating systems scalability (based on additional physical and logical resources that are made available), as well as evaluating the aspects of systems support, are the key components that directly contribute and impact the methodology. The argument being made is that in order to evaluate *end-to-end performance in a multi-tier environment*, the combination of an analytical and a Petri-Net based (simulation) modeling approach is applicable, as it enables analyzing and evaluating the *dynamics* of an entire infrastructure. This includes any potential outside influences, such as other applications that compete for the shared (hardware and or software) resources. Similar to data modeling, prototyping, or use-case studies conducted by the software design and development teams, Petri-Net based simulation modeling is focused on elaborating, evaluating, and analyzing systems performance under a variety of circumstances. Simulation modeling involves developing, calibrating, verifying, validating, and utilizing a model of an entire system to analyze, evaluate, and estimate key performance metrics such as response time, throughput, and resource utilization. The actual model is derived from either Sequence Diagrams (SD) or Use Case Maps (UCM) that are provided by the design team, and incorporates the hardware and software infrastructure as determined by the systems architects.


## Introduction

In a commercial environment, a *business process* is the driving factor that triggers the actual need for a system. As a user performs a certain task on the system, the application is called upon to basically retrieve data, to perform calculations or comparisons, and to update information. Essential to the performance engineering process is an estimate of *the volume of work* that has to be performed. The *volume*

*of work* is being considered as one of the input parameters into either the analytical or the simulation model. To reiterate, a user application supports the execution of a business process. When being invoked, an application (either through library or system calls) utilizes actual physical and logical resources. Some of the key physical resources are the CPU, memory, I/O, and network related components, whereas some of the logical resources can be identified as the memory tables, the file system block sizes, or the network packet sizes. The called upon resources have to be identified and quantified, as they represent additional input parameters into the model. In practice, these resources are either determined through an educated guessing process, are being extracted from a similar system through an empirical analysis, or are derived from an 'instrumented' prototype that mimics the actual system. As an example, by utilizing tools such as ARM, in conjunction with a prototype, the application can report detailed response time measurements for each *individual business function component*, as well as for the entire transaction. As an example, based on the collected data, it is feasible to separate the think time from the actual system processing time, and to determine which application functionality contributes the most to the overall response time or resource utilization.

**Methodology to Conduct End-to-End Performance Analysis**

Retrofitting performance improvements into an existing product is a daunting and expensive task, and may delay the deploying of the system. The following section outlines a very pragmatic, *divide-and-conquer* based approach, which consists of several cascading steps that can be transformed into a comprehensive systems performance analysis. The methodology utilizes studies conducted by Smith, Jain, Siddiqui et al. as a blueprint (see references). In a nutshell, the overall system is basically divided up into parts that correspond to system responsibilities that are identified as having a certain performance budget. Utilizing the performance budgets (the budgets are either being guessed or determined through an empirical analysis), a system's overall performance can be estimated and potential design changes can be analyzed dynamically. To reiterate, the analytical or simulation models can not only be used to predict overall response time, but also to define and evaluate certain performance goals. In other words, feasible performance budgets can be determined and quantified to meet the overall performance goals, resulting into a process that can be described as supporting an incremental change in performance analysis.

Based on the complexity of the system and the question to be answered, either the analytical or the simulation model is the key to understanding the individual performance budgets, and to meet the overall performance goal. Once the model is developed, the analysis can be conducted either by utilizing a *forward* or a *reverse direction* based analysis approach. In a *forward direction* based analysis, the logical flow starts with the individual performance budgets and ends up with an overall quantification of the entire system, ergo can be used to conduct a feasibility analysis in regards to the system's requirements. As already discussed, the initial performance budgets might have to be determined either based on experience (normally incorporating a *reserve* for contingencies), or by utilizing a prototype-based approach. *A reverse direction* based study incorporates an analysis that starts with the overall performance goal that is being decomposed into individual performance budgets (on a per operation level). The reverse path based approach basically generates (again, on a per-operation based level) the *has-to-be-achieved* individual performance budgets, and might serve as a tool to identify infeasible performance goals.

***Step 1: Design Specification***

The design specification captures the system behavior as a set of scenarios (at a certain level of abstraction), outlining the invoked software subsystems, operations and responsibilities on a per business function level. Sequence Diagrams (SD) or Use Case Maps (UCM) provide the flexibility and detail necessary to develop the model, and to conduct sensitivity and capacity studies.

***Step 2: Performance (Demand) Budgets***

It is imperative to not only identify the performance goals prior to any modeling activities, but also to have a very clear picture of what performance questions are supposed to be answered in the study. This is crucial as the performance goals *drive the level of (modeling) detail necessary* to answer the performance questions. The individual performance budgets are actual values that describe the resource demand of the

operations and responsibilities as identified in the SD or the UCM. As an example, the units to be determined could be in CPU-seconds, operation counts, packet sizes, or Mbytes moved through the networking and/or the I/O subsystem. To reiterate, the initial budgets have to be either assumed or guessed based on experience, derived from a similar system, measured utilizing existing components, or determined based on a empirical analysis conducted by using a prototype. Further, it is imperative to identify certain *workload profiles* that identify the set of *operation mixes* (as being anticipated in real world scenarios).

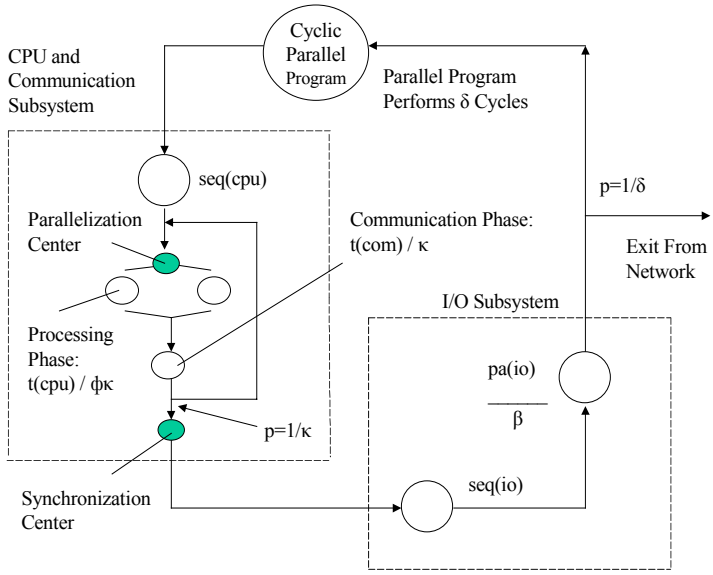
***Step 3: Identifying the Hardware, Software, and Communication Infrastructure***

The application specifications do not completely define an entire system per se. In most performance studies that operate in a multi-tier environment, it is necessary to obtain the hardware layout, the operating system specifications, as well as the incorporated network and middle-ware infrastructure from the architecture team. All these elements have to be determined prior to developing the model (this process might be assumption based as well).

***Step 4: Developing the (Stochastic Simulation) Model***

While conducting a performance analysis, analysts normally utilize a combination of analytical model, simulation model, and empirical analysis based techniques. An analytical model simulates the behavior of a system, is build through equations, and the actual performance is derived mathematically. With an analytical model, it is normally necessary to construct a number of parameterized functions that approximate the workload characteristics of the system components (see Figure 1). In other words, an analytical model provides *average* and *standard deviation* types of statistics, as the actual workload distribution is *normally* ignored. A number of simplifying assumptions has to be made to trace the model. How these assumptions reflect the real world behavior and the extent to which they can be formulated as mathematical equations basically determines the accuracy and usefulness of this approach. Analytical models are generally cheaper and faster to build than simulation models, and can easily be tailored to analyze and predict (after calibrating the model) a wide variety of performance-related aspects of a system.

Figure 1: Analytical Queuing Model representing a Single Node in a Cluster with Parallelization and Synchronization Centers

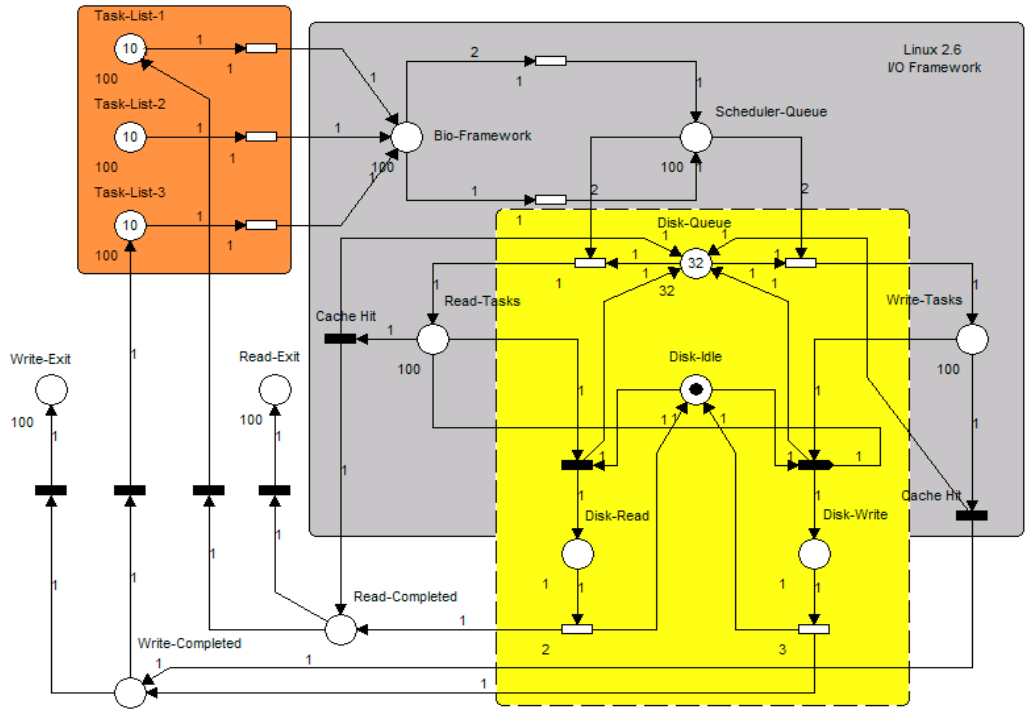



Simulation models on the other hand consist of a suite of modules or programs that are designed to capture the characteristics of a system under real world conditions. The basic idea is to model the entire system and to simulate its dynamic behavior by taking the actual workload distribution into account. More specifically, each event (or state transition) in a computing environment is being considered, analyzed, and

modeled. Simulation models are normally rather complex, and hence more expensive to build than an analytical model, but may provide a higher degree of accuracy (and are extensively being used to conduct sensitivity studies).

An empirical analysis on the other hand involves executing an application or benchmark program on an *existing system* and to measure actual performance. This approach provides the most accurate performance data of the three discussed techniques, but it only analyzes one specific workload (the one that is being executed with a specific set of parameter values) and normally can not be used to conduct a sensitivity analysis per se. In some circumstances, this approach can be very time intensive (and expensive), as the test environment either has to be made available, or has to be built from scratch. To re-emphasize, most performance studies consist of a mixture of the three approaches discussed in this article. The proper approach (that is applicable to conduct the analysis) is chosen based on *the performance goals and the to-be-answered performance questions*.

Figure 2: DSPN Model Representing the Linux 2.6 bio and IO scheduler Abstractions




Commercial simulation modeling tools, such as the HyPerformix Infrastructure Optimizer (IO), represent actual state-event systems where a task can be analyzed as it flows through an entire system, while *visiting* different software and hardware components. On the other hand, deterministic stochastic Petri-net (DSPN) solutions (some DSPN packages are available as freeware) are utilized to facilitate and formulate the high-level modeling abstractions of discrete-event systems that consist of exponential and deterministic subtasks (see Figure 2). The data collected in steps 1, 2 and 3 of the outlined methodology serves as the blueprint to develop and calibrate the model. If the individual performance budgets were obtained by conducting an empirical analysis, the final model can be verified and validated against the actually measured systems performance (to ensure that the model reflects reality, and can be used to conduct a comprehensive sensitivity or capacity study). If the model cannot be validated at this stage of the system life cycle, the model can be used (as an example) in the design phase to evaluate different design alternatives. Further, it is feasible to analyze and quantify the impact that an infrastructure change has on systems performance, an approach that might require normalizing the performance data, as the analysis is now focused on relative and not on actual performance.

### *Step 5: Analysis and Recommendations*

Running an actual simulation (either in a homogeneous or heterogeneous environment) allows comparing the obtained (simulation based) results to the actual performance requirements. An artifact of running a simulation is to utilize the performance results to verify that *if the performance budgets are being met*, overall systems performance is adequate. Another benefit is the potential of adjusting the capacity of certain physical (such as the CPU speed) and logical (such as the number of concurrent worker threads) resources, so that given the overall performance budget, the performance goals might either be met or more closely be approached.

In general, actual recommendations on how to improve performance (by for example either altering the design or adjusting hardware or software components) are *result dependent*. In most circumstances, changes are required if overall performance does not meet the goals. A best practice approach is to change one component (in the model) at a time, and to rerun the simulation after the change has been implemented to clearly understand, quantify, and document the progress.

## Summary

A modeling based approach (normally in conjunction with other tools and techniques) is applicable throughout the system life cycle. In the design phase, evaluating overall performance, analyzing and quantifying different design alternatives and/or different hardware configurations is imperative to mitigate the risk of encountering (expensive) performance deficiencies later on in the project. In most cases, in the development phase (where more performance related data becomes available), the model can be re-calibrated and additional sensitivity and capacity studies can be conducted. This evolutionary process generally leads to comprehensive test scenarios that can be simulated in the (stress) test phase. The outlined methodology enables a comprehensive performance analysis and documentation process that can further be used as a communication tool among the involved design, development, and test teams. The process represents the common denominator for all the involved parties, and can be considered as the tool that drives most of the technical discussions. A sequel to this article will discuss an actual 3-tier architecture and elaborate on the combination of analytical and simulation models that were being used to conduct the PE project.

## References


1. Al-Fatha, I., Majumdar, S., "*Performance Comparison for Client-Server Interactions in CORBA*", ICDCS 98, May 1998
2. Buhr, R., "*Use Case Maps for Object Oriented Systems*", Prentice-Hall, 1995
3. Franks, G., Majumdar, S., "*Performance Analysis of Distributed Server Systems*", 6th International Conference on Software Quality, Ottawa, Canada, 1996
4. Hennessy, J., Patterson, D., "*Computer Architecture, a Quantitative Approach*", Third Edition, Morgan Kaufmann, 2002.
5. Jain, R., "*The Art of Computer System Performance Analysis*", John Wiley, 1991
6. Koenigs, A. "*Industrial Strength Parallel Computing*", Morgan Kaufmann, San Francisco, 2000
7. Siddiqui, K., "*Time Performance Budgeting of Software Design*", Masters Thesis, Carleton University, Ottawa, Canada, 2001
8. Smith, C., "Performance Engineering of Software Systems", Addison-Wesley, 1990
9. Stone, H., "*High Performance Computers*", Addison-Wesley, First Edition, New York, NY, 1991
10. Pfister, G. F., "*In Search of Clusters*", Second Edition, Prentice Hall PTR, NY, 1998.
11. ARM 3.0 for JAVA http://regions.cmg.org/regions/cmgarmw/CMG00_paper_final.pdf
12. Infrastructure Optimizer (HyPerformix) http://www.hyperformix.com