

Simplified Performance Estimation

Technical White Paper



© 2002 Sun Microsystems, Inc. All rights reserved.
901 San Antonio Road, Palo Alto, California 94303 U.S.A

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

TRADEMARKS

Sun, Sun Microsystems, the Sun logo, Sun Startup Essentials, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other product names mentioned herein are the trademarks of their respective owners.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



Please
Recycle



Adobe PostScript

Contents

Introduction	1
Motivation	2
Conventional Capacity Planning and Performance Engineering ...	4
Capacity Planning for Existing Systems.	4
Performance Engineering.	5
Performance Verification	6
Simple Estimation Method for Early Capacity Planning	7
Stating Performance Requirements.	8
Mathematical Basics	11
Workload Characterization	14
Capacity Planning	16
Workload Characterization Summary	17
Application	18
Summary	20
References	23

Foreword



Engineering has always been concerned with *predictability*, the expectation for development efforts to result in products which perform desired functions with acceptable reliability and availability on time and at acceptable cost. However, most branches of engineering experience strict physical and technical limitations that impact what can be economically achieved. As a result, strong emphasis is placed on estimation and calculation, such as the amount of materials involved, the strengths of structures, and whether multiple purposes can be served by any single construction.

The use of the word *architecture* in software design and hardware infrastructure is analogous to its use in civil engineering. Architecture is the science, and art, of bringing together all major elements of an intended construction or reconstruction so as to fulfill the primary purposes of the project. The goal of architecture is to simultaneously fulfill both primary and secondary project objectives while satisfying an engineering sense of correctness, appropriateness, and elegance. When designing a new IT hardware and software infrastructure, such as that needed by Web sites that utilize the capabilities of the Internet to promote interest to other businesses as well as customers, similar estimations and calculations must be done to ensure sufficient capacities are available without compromising economy.

There are many disciplines to engineering. Those involved in meeting performance, reliability, availability, security and other requirements — the techniques collected under the term *architecture* — are among the least understood. As a result, IT system architects often learn their craft on the job,



from mentors, and through experience. Consequently, architects often fall prey to two tendencies: they tend to be effective only in developing systems seen before, and every problem is made to fit the architect's preconceptions.

Learning the art of architecture on the job has undesirable consequences. Basing all knowledge on mentoring and experience requires years observing and imitating more seasoned professionals. Eventually, such experience leads to good architectural judgement — but good judgement often comes from learning from mistakes. This slow and error-prone approach is unacceptable in a world where the need for new systems is great. Architecture must be taught to budding architects, and seasoned professionals need better tools with which to evaluate decisions without requiring systems to be built.

To be successful, architecture must be transformed into a rigorous process using quantitative mathematical tools. In an effort to help this process, Sun has developed a series of white papers on *Quantitative Architecture* — the discipline of developing architectures using rigorous techniques that enable architects to predict the non-functional properties of a system to ensure, from the very beginning, that the system meets its goals.

In order to develop a rigorous and quantitative approach to architecture, it is important to understand the disciplines that enable predictions to be made regarding CPU power, network bandwidth, response times, and so on. Toward that end, the white papers address the following topics:

- *Introduction to the Elements of Web Application Architecture*, introduces the sequence of considerations that go into what may be termed modern *dot-com* or *Web-based* architectures, and explains some of the terms and principles that apply.
- *What is Architecture?*, discusses the design processes that lead to an appropriate architecture that meets design and business goals.
- *Simplified Performance Estimation*, presents a simple and approximate performance estimation method for early capacity planning which enables architectural decisions to be made with confidence.
- *Understanding Growth*, discusses ways to make the growth problem easier to understand, and aims to facilitate the finding of good estimates of the effects of growth and alleviate some scalability concerns.



-
- *Understanding Statistics and Queuing — Probability, Distributions and Statistical Models*, delves deeply into the bases of these kinds of computations, making evident the assumptions in identifying the various kinds of scenarios in which it is possible to obtain useful, and interesting, results.
 - *Understanding Capacity and Scalability*, presents both capacity and scalability in a basic mathematical model, and shows how common architectural decisions affect them.
 - *Implementing Enterprise Security Policies*, discusses security policies and the potential for machine verification and other techniques.

This series is based on the Sun Startup EssentialsSM program, a class devoted to teaching entrepreneurs and their colleagues the steps needed to establish and operate business-to-client or business-to-business Web-enabled sites. Discussions with students revealed that quantifying many decisions enabled good architecture — and that the basis of the methodology is rarely fully understood. These papers aim to bridge the gap, making the underlying theory accessible and able to serve as a firm foundation, and demonstrating, by example, how these foundations support resulting hardware and software architectures.



Simplified Performance Estimation



Architects must address capacity planning as part of Web system development, sizing the initial system to meet expected workloads, yet ensure the system scales to meet increasing demands. This technical white paper presents a simple, approximate performance estimation method that enables architects to make early capacity planning decisions quickly and confidently.

Introduction

Web systems are faced with many challenges, including supporting massive numbers of users accessing varying client devices in multiple languages.

Commercial systems development has generally been concerned with sizing systems to ensure production configurations can perform intended tasks. During initial specification and development, system architects often find themselves “flying blind”. What will be the real demands on the system? How can developers plan for demand? Is it possible to take into account variations, including random high-demand bursts without expensive over-engineering?

This problem of *capacity planning* has been the subject of research for decades, including extensive theory and the development of powerful analytic and simulation tools. These tools enable capacity planners to construct accurate models of current and anticipated demand for existing systems, and provide a basis for planning future extensions to those systems. These and other modeling tools suffer from an inherent difficulty — they are only as good as the information on which the models are based. During initial system development,



especially Web system development, little information is available on which to base detailed capacity models. As a result, initial planning is difficult and error prone, no matter how sophisticated the tools.

However, there is hope. It is possible to create an early estimate of needed capacity whether or not a good capacity planning model exists. Some attempt can be made to specify a system to meet that capacity demand, and *anything* that improves initial estimates is worthwhile. Recall an old story of a bear chasing two men. One man stops to put on running shoes, explaining to the other that he does not have to outrun the bear, he only need to outrun him. Early capacity planning is in the same position — it does not need to be extremely accurate, so long as it is more accurate than basic guesswork.

This paper presents a simple methodology for performance estimation in capacity planning. Presented for Web systems, this approach can be applied to a wide range of systems. While limited, this methodology is well founded in theory that can be applied to identify when limits are reached. Equally important, it is simple. It can be applied to early design decisions with some confidence that it will result in appropriate designs. Perhaps most important, the methodology can be applied with minimal mathematical skill, and requires little or no experience with similar systems.

Motivation

Consider a medium-sized bank with an aggressive Web banking program. Originally designed to handle 10,000 users and provide responses in less than 5 seconds at least 95 percent of the time, this successful Web banking system grew to over 200,000 customers, with half using the system on a regular basis (Figure 1).

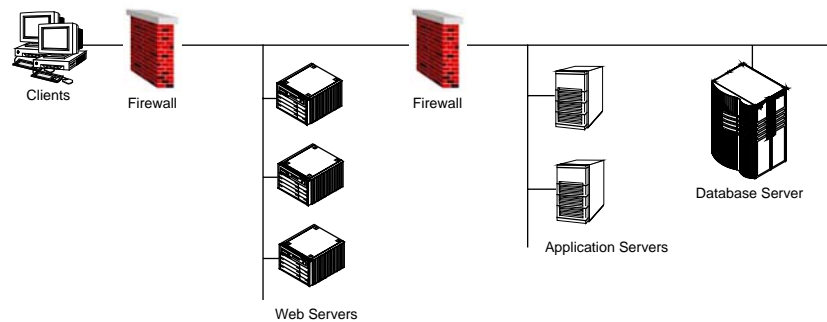


Figure 1 An example Web banking system

During most of the week, response times rarely exceeded the 5 second requirement. At other times, however, CPU utilization soared to nearly 100 percent, and response time increased to minutes. As a result, several attempts were necessary before users could successfully connect to the Web system, and many received screens or messages indicating connections were refused. Most unexpectedly, this behavior came and went randomly. Saturday nights were usually difficult, as were the 15th and 1st of the month. However, even during short periods, system performance ranged from poor to nearly normal and back again.

Performance was clearly the problem. Clever operating system tuning, memory additions, and program profiling during testing were all tried with little success. The bank decided to build a new, extensible and maintainable system which would address their performance problems.

The first step in solving a performance problem involves identifying it. To do so, analysis of the system in operation is essential. The bank observed:

- Using weekly statistics, the average load on the system was approximately 2,500,000 hits per week, or 14881 hits per hour, or 4.13 hits per second.
- Under testing, the system began to “stall” at approximately 15 URL retrievals per second.

These two facts led the bank to conclude their telecommunications provider was the source of the problem.

Was the telecommunications provider really responsible, or was the system hitting the 15 hits per second limit? The answer: the system *was* hitting its inherent limits. The statistics failed to identify the *variance* of the arrival of new URL hits over time. While the system experienced roughly 4 arrivals per second on average, the arrival rate increased to 18 arrivals per second, with bursts of up to 30 arrivals per second or more during peak loads. Once the maximum number of jobs was being processed during peak periods, other jobs were queued by operating system features and the HTTP daemon. The queuing of jobs increased average response time dramatically. When the queue exceeded certain limits, the HTTP daemon socket simply refused further requests, resulting in an HTTP error at the client browser.

The bank naively estimated that adding eight times the capacity would enable the system to catch up to current workload demands, with more capacity needed for future growth. This idea exceeded their hardware budget by a factor of five. It was time to look for more information.



- Was there a more economical way to do the job?
- Was it possible to make a more precise prediction of what would be needed, even with imprecise initial data?
- Was there a simple model that would enable a prediction for what would be needed in the future?
- Could the developer, with confidence, tell the bank that a system within budget could indeed meet their needs?

Conventional Capacity Planning and Performance Engineering

Conventional capacity planning has been in use since the 1960s, and is generally applied to existing installations for the purpose of planning for increases in the scale of an installation to meet future demands. A brief description of conventional techniques is provided below, furnishing background for the simplified techniques presented later in this paper.

Capacity Planning for Existing Systems

In conventional capacity planning, modeling using analytic or simulation methods, separately or in combination, is used to estimate performance and capacity measures of an existing system based on empirical data measured during operation. In general, this approach usually requires two phases.

In the first phase, a model is developed. A wide variety of tools can be used, from spreadsheets to high-powered capacity planning tools such as Best/1. The idea of the model is to represent the behavior of the system as it pertains to capacity, taking into account the fact that a real system always behaves somewhat randomly. This means that one of several stochastic modeling techniques, such as queuing theoretic analysis, generalized Petri net solution, or discrete-event simulation may be used.

The resultant model is evaluated against the current system using existing empirical data or new data collected especially for the effort. If the model's predicted values vary from the empirical data, the model is modified and re-evaluated. This process is repeated until the model's predictions agree sufficiently well with empirical data. Notice that the model does not necessarily exactly reproduce the real behavior of the system, just that the predictions match the empirical data within specification. At this point, the model is said to be

calibrated. On Korzybski's principle that "a difference which makes no difference is no difference", once a model is well calibrated against existing empirical data, it is accepted.

In the second phase, the change in demand is modeled. This step is often a simple linear extrapolation of the current empirical data, and the assumed new demand is used to establish new inputs for the calibrated model. The results from the calibrated model are used to predict changes in the appropriate performance measures.

Over time, as the demand and system change, the model is also modified in a corresponding manner. This enables the system to be modified quickly to meet future demand, rather than forcing a response to a performance crisis. The ability to make these predictions is completely dependent on the ability to estimate *workload*, the behavior of the demand on the system. *Workload characterization* — the modeling of the workload — is key to capacity planning. In many cases, simply characterizing the workload effectively is enough to enable capacity planning to be done. This is particularly true in Web systems.

Performance Engineering

An extension of the discipline of capacity planning is the application of capacity planning techniques to systems during their design phase. Using performance or capacity modeling of a system during design, *performance engineering* presents some new and significant problems. Workload characterization in the form of a stochastic model is more difficult — and more important — than it is for an existing system for which empirical measurements have been performed. More fundamental, if no less important, is the problem of calibrating the model.

Clearly, with no existing system against which to perform experiments and collect empirical data, it is impossible to accept a well calibrated model as is done for conventional capacity planning. The performance engineering model must be capable of being mapped explicitly to the structure of the proposed system, with a convincing argument that the model should predict the performance of the planned system. This argument leads to a *validated* model rather than a calibrated model.

This paper proposes a performance engineering technique intended to be performed on high-level models of a proposed system in order to predict coarse-grained performance and capacity.



Performance Verification

Software engineering distinguishes between *validating* and *verifying* a system. In general, *validation* shows that the requirements are acceptable to users, the system is constructed in an appropriate manner, and all appropriate tests meet their desired conclusion [Pressman]. In contrast, *verification* ensures there is a convincing and rigorous mathematical argument — a proof — that the system meets all requirements. Verification is usually applied only to the functional input/output requirements of highly mission-critical components.

Applied rigorously, performance verification has the potential for demonstrating not only that it is likely a system will meet specified performance bounds, but that it is almost certain to meet those bounds. Formal and rigorous performance verification is beyond the power of current methods, at least for most system development. However, analytic and simulation techniques, applied to system development, can give considerable reassurance that performance bounds will be met.

Simple Estimation Method for Early Capacity Planning

In the days of single-program batch processing using punch cards, capacity planning was an easy task. Processing speed was limited by the speed with which the cards could be read and manipulated. While a form of pipelining was certainly available, it was limited by the facility with which the operators could move card trays from the tabulator to the sorter to the arithmetic unit to the printer.

The Web-enabled world is more complex, and system developers must deal with random events and behavior. Individuals decide to visit Web sites at will, and requests arrive randomly. Different pages have different sizes, resulting in widely varying consumed bandwidth. Arrivals may simply request static text, or they may require significant computation, along with access to secondary storage, a database server, or a legacy system. As a result, the time to process a request varies randomly, often by a factor of 10 or 20.

Consequently, Web system capacity planning is much more difficult. To understand and model the demand on Web systems, statistical and probabilistic methods must be used. The mathematics is elegant, but can be quite complex and difficult. However, by drawing on the results of researchers in stochastic processes, Markov systems, and queuing theory, simple and elegant results can be applied to provide significant insight throughout the architectural process while remaining tractable for non-specialists.

This section presents a methodology for early capacity planning based in queuing theory. This methodology is simple enough that basic estimates can be made in a few lines of calculation, but accurate enough that they can be used to limit risk during early design.

Stating Performance Requirements

The first, and most difficult, problem with any performance engineering or capacity planning effort is stating mutually agreeable, clearly stated performance requirements. In order to plan a Web system, it is important to know the anticipated response time and the number of users expected to visit the system during highly loaded short intervals.

Most projects begin with loosely stated “big rules”, such as:

- The system must respond to the user within 5 seconds
- The system is expected to process 2,500,000 visits per week at the end of one year

At first glance, these rules seem reasonable. Consider how these rules might be tested. What does it mean to respond within five seconds? Must the full page be received and displayed, or does it mean any response at all is adequate? What about latency in the Internet connection between the client and the Web system, or the user’s connection bandwidth? What exactly is a visit? Is a visit one URL hit, a unique visitor, or a single session?

Clearly, it is essential for the capacity planner or system architect to establish firmer requirements, both to guide design efforts and ensure peace of mind. The keys to a good requirement are:

- *Specificity*. The requirement must be clear.
- *Measurement*. The requirement must be stated in terms of a measurable quantity.
- *Testability*. There must exist an effective procedure that can, upon execution, distinguish between systems which do and do not satisfy the requirement.

The SMART Requirement

A mnemonic helps to establish these requirements: SMART, for Specific, Measurable, Agreement, establishing Responsibility and Testability.



- *Specific.* State a particular goal clearly and unambiguously.
- *Measurable.* State the goal in terms of a well-founded measurement.
- *Agreement.* Refine the goal until all stake holders agree that it fairly states the desired outcome.
- *Responsibility.* Establish one capable party responsible for a goal's satisfaction.
- *Testability.* State, or at least sufficiently define, an effective procedure that establishes whether or not the goal is satisfied.

While it is beyond the scope of this paper to give a full treatise on how requirements should be captured and stated, significant insight can be gained by simply considering how big rule requirements fail. For example, the requirement for the system to respond within 5 seconds fails in several areas:

- *Specificity.* There are at least three natural points in a visit to a particular Web page that might be counted as a response: the arrival of the first byte of the first returned HTML, the first appearance of a screen change, the time at which the page finishes downloading.
- *Agreement.* As a consequence of the lack of specificity, it is very unlikely that every stake holder has the same picture in mind of the system's operation.
- *Responsibility.* The failure to specify the end-points from which responses might be tested, for example, leaves open the possibility that a test might include transmission over the open Internet, with potential variation in latency and effective propagation speed. As a result, the requirement as stated could include a very large component over which neither the system nor the designer has any control.

This requirement can be restated more precisely by thinking about the test to be performed. If response times are tested using a client with direct access to the public side of the firewall or Web server, then response times are only affected by controllable components. Furthermore, it is important to establish what it means to respond. Typically, a response is defined as the first appearance of change on the client screen, or the first byte of a response.

These requirements can be stated in probabilistic terms, such as “the response time will be less than 5 seconds more than 95 percent of the time.” Requirements such as these account for the possibility that random events during testing or operation may change the behavior of the system momentarily. With these in mind, the requirement can be restated as follows: The response time of the

system, measured from a client at the public firewall, shall be less than five seconds to the first response byte of the reply, measured from the initiating mouse click, at least 95 percent of the time in 1000 trials.

The testing procedure is then clear. The test system must simulate a mouse click at the appropriate point, and measure the elapsed time between that mouse click and the first response byte. This test must be performed 1000 times, and the result must be that the elapsed time does not exceed 5 seconds more than 20 times.

Mathematical Basics

Performance Estimation at the level of accuracy described in this paper depends on only a few mathematical facts. The interested reader should refer to texts on probability, statistics, and queuing theory for more complete expositions of these facts and the mathematics behind them.

Little's Law

Little's Law states that in a system at equilibrium, the number of jobs being processed is equal to the arrival rate of jobs multiplied by the service rate for an individual job. Most people familiar with grocery store checkout lines understand this intuitively. If the checker is fast enough, the line never goes beyond a certain length. On the other hand, a slow checker in a crowded grocery store can have a very long line.

Little's Law is used later to make simple estimates of the number of concurrent sessions in a Web system.

Pareto's Distribution and Pareto's Law

During the 19th century, the Austrian economist Vilfredo Pareto made the following observation. The distribution of incomes is such that when the incomes of individuals are ranked into small bins, the number of people in each bin varies as x^{-k} . If the number of people in the first bin is x , the number in the second bin is $1/x^2$, the third bin $1/x^3$, and so on. This observation can be treated as the cumulative distribution function of a random variable. These *power laws* are extremely important in many self-organizing systems.



Pareto's observation became part of systems knowledge through the folk theorem known as *Pareto's Law*. Pareto's Law states that 80 percent of the items of interest occur in 20 percent of the items examined. Pareto's Law appears repeatedly in areas in which power laws characterize the distribution of objects. For example:

- In programming, 80 percent of the errors occur in only 20 percent of the modules.
- In FORTRAN programs, 80 percent of the time is consumed by 20 percent of the program statements.
- In Web systems, 80 percent of the total consumed bandwidth is consumed by only 20 percent of the pages.

Random Arrivals and Poisson's Distribution

Consider a common Web system, such as a bank system. If the user population is large, then at any time of the day or night some number of users are probably looking at bank records on that system. Users are independent, and have no knowledge of one another. Each time a user requests a new page, the request is short in comparison to the time spent thinking. Furthermore, requests cannot overlap each other, in part because requests are short, and because the structure of TCP/IP connections enforces it.

Under these conditions, the *instantaneous rate of arrival* of requests is distributed according to *Poisson's Distribution*, which is defined to have the following mass function:

$$f(x) = e^{-\lambda} \frac{\lambda^x}{x!}, \quad x = 0, 1, 2, \dots, \infty$$

In other words, given an arrival rate λ , the probability of x arrivals in a unit time is $f(x)$. As a convenient assumption, and unless otherwise stated, the remainder of this paper will consider arrival rates to have a Poisson distribution.

Random Service Times and the Exponential Distribution

Once a single Web request has been made, it must be processed by the system. The request is scheduled, disks may be accessed, the process may be interrupted and swapped, and so on. As a result, the service time of a single request is also random. Most systems assume the service time of a single job is *exponentially distributed*. That is, the service time has the mass function:

$$f(x) = \lambda e^{-\lambda x}$$

Again, as a convenient assumption, assume that service times are exponentially distributed. Unlike the Poisson distribution of arrival rates, this simplifying assumption has much less justification — the primary justification being that it is much easier to deal with mathematically or numerically. Thankfully, it turns out to be a reasonable assumption for real systems.

Convenient Properties of Memory-Less Distributions

The Poisson and exponential distributions are known as *Markovian*, or *memory-less* distributions, because the value of the distribution is unaffected by the history of previous events. This has obvious advantages from a practical standpoint — it is not necessary to consider anything but the current case. For purposes of discussion, this has one important implication: arrivals of requests to a Web system also tend to occur in groups. As a result, it is important to consider not just what happens with arrivals at the average rate, m , but also during coincidental bursts of arrivals.

A second interesting point about the Poisson and exponential distribution is that the parameter λ that characterizes either the average arrival rate or the average service time is also the mean, or expected value. The variance of the distribution is also λ , and the standard deviation is $\sqrt{\lambda}$. In other words, the shape of the distribution is independent of the rate.

Using a spreadsheet or pocket calculator, it is trivial to compute the exact value of the Poisson or exponential mass function. However, it is convenient to know that the Poisson distribution with parameter λ is approximated by the Gaussian or normal distribution $n(\lambda, \sqrt{\lambda})$. Table 1 illustrates the values for a parameter λ .



Value	σ	Percentile
$E[x] = \lambda$	0	50
$\lambda + \sqrt{\lambda}$	1	~86
$\lambda + 1.68\sqrt{\lambda}$	1.68	~95
$\lambda + 2\sqrt{\lambda}$	2	~98

Table 1 Values for λ in a Poisson distribution

Workload Characterization

With this grounding established, it is now possible to describe how to characterize the workload of a Web system. The goal is to find a reasonable figure for the arrival rate of Web requests per second during peak load conditions. Begin the process with a common big rule — the customer's expectation that the system will experience a number of visits per day. For simplicity, assume each visit is composed of one URL request, with the understanding that in real systems it is important to think about how many pages are seen per visit. However, for estimation, simply use a constant value, making this step a matter of simple multiplication.

It would seem possible to simply divide the number of visits per day by 3600 and arrive at the number of visits per second. The difficulty is that most publicly available Web sites experience loads which vary widely over the course of a day. As a result, there are times of the day in which the mean visits per second will be several or many times the average over the whole day. Figure 2 illustrates this phenomena with the one week load characteristics experienced by the *java.sun.com* Web site during the release of the Java™ Web Service Developer Pack, Early Access Release 1 in January 2002. Clearly, the peak load times are significantly more loaded than the day's average over the entire week.

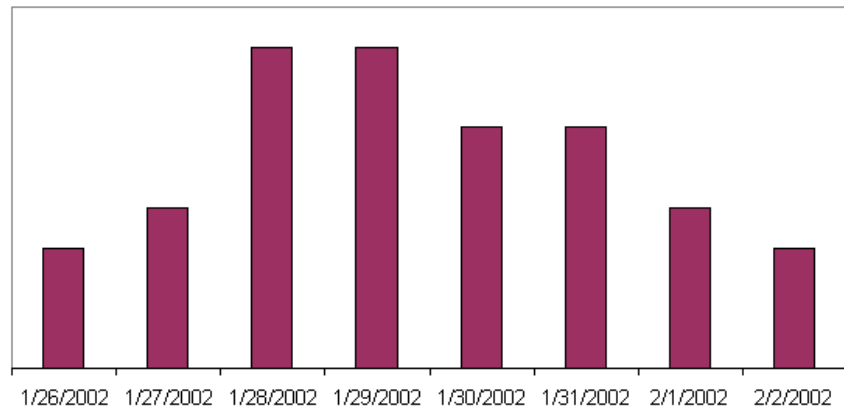


Figure 2 Example workload over one week

Access to actual Web log data could be analyzed to determine the most highly loaded periods. If logs are not available, an estimation method can be used.

The arrival rate of Web site visits obeys a power law distribution — it is safe to assume the visits obey Pareto's Law. That is, 80 percent of the arrivals take place in 20 percent of the time. Therefore, the arrival rate during the heavily loaded part of the day can be estimated by simple arithmetic. Given rate R in arrivals per day, we can compute the number of arrivals r_{\max} in the most heavily loaded period as follows:

$$R \frac{\text{arrivals}}{\text{day}} \times \frac{0.8}{\left(0.2 \times 24 \frac{\text{hours}}{\text{day}}\right)} = \frac{\left(R \frac{\text{arrivals}}{\text{day}} \text{ at maximum load}\right)}{3600} = r_{\max} \frac{\text{arrivals}}{\text{second}}$$

For example, an arrival rate of 2500 arrivals per day results in approximately 28 arrivals per second, on average, during the most heavily loaded periods. This is still an average. What about the most heavily loaded seconds during the heavily loaded period? The answer is found by applying the Poisson distribution. The average rate of arrivals is 29 arrivals per second, and the σ rate is 33.3 arrivals per second. The 95th percentile (1.68σ) rate is 36.8, and the 2σ rate is 38.6. If the specification is to achieve a particular response time 95 percent of the time, the (1.68σ) arrival rate should be used as the basis for further calculations.



Capacity Planning

Once the arrival rate of requests are characterized, the next step is to estimate two essential measures of capacity for Web systems: required bandwidth and throughput.

Bandwidth

It is best to use empirical data from actual estimates. Unfortunately, empirical data is often unavailable during early architectural design. It is known, however, that the size distribution of pages also obeys a power law distribution. The mean size of a page is approximately 10 KB, and the Pareto assumption indicates that 80 percent of the pages sent are less than 15 KB. In order to ensure sufficient capacity at high loads, the 95th percentile rate multiplied by 15 KB yields the result.

$$r \frac{\text{pages}}{\text{second}} \times 15,000 \frac{\text{bytes}}{\text{page}} \times 10 = \text{bits per second}$$

Notice that 10 bits per byte are used to account for framing and overhead.

Response Time and Utilization

Another important task is to determine needed server capacity.

One capacity measure already known is the hits per second rate. Often, hardware can be sized directly from this value [Wong]. However, a closer look reveals some interesting facts.

The *service time* $\frac{1}{\mu}$ of a single job in a queuing system is defined to be the time required for the job to be processed. It does not include time spent in the queue, waiting for I/O, or in other overhead operations. Service time is defined as $\frac{1}{\mu}$ because μ is commonly used for the *service rate*, the inverse of service time. This service rate is also known as *system throughput*.

Using Little's Law, a stable system must have $\lambda < \mu$ — jobs must arrive no more quickly, on average, than they can be serviced. The maximum arrival rate that can be serviced then is $\lambda = \mu$. The maximum throughput occurs when the service time exactly equals the service rate. At this rate, the system is 100 percent utilized. When $\lambda < \mu$, the utilization of the system, u , is simply $u = \frac{\lambda}{\mu}$.

Unfortunately, it is not that simple. Response time requirements must also be met. Figure 3 illustrates the relationship of response time and server utilization. To estimate actual needed capacity, it is essential to plan for higher maximum throughput — approximately a factor of two higher than the expected maximum.

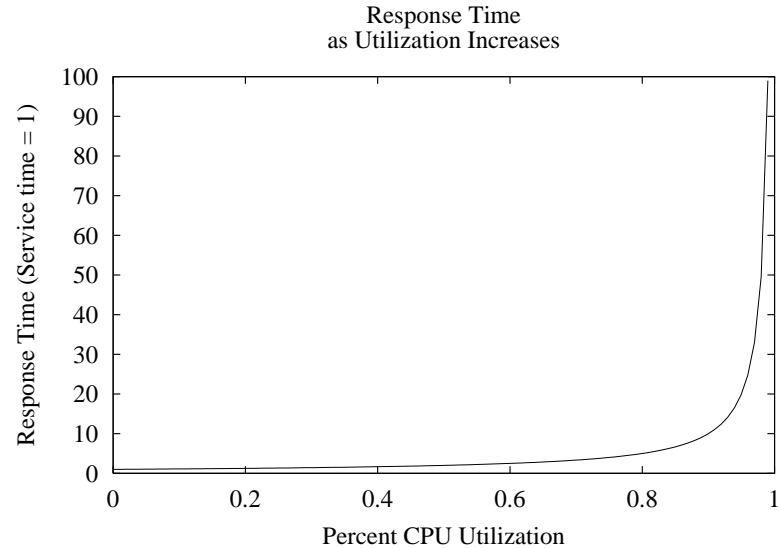


Figure 3 Response time versus server utilization

Workload Characterization Summary

This section outlined the steps needed to estimate workload characterization. From this information, bandwidth and capacity (or throughput) estimates for the proposed system are possible, using only a few facts.

Step	Product
Initial Estimate	Average number of visits per day.
Workload Characterization	Mean visits per second during high load periods. Upper limit (e.g. 95 percent) visits per second during high load periods.
Capacity Estimation	Bandwidth (15 x upper limit rate per second) Capacity ($[1/2 \text{ service time}]^{-1}$)

Table 2 Steps needed to estimate workload characteristics



Application

The tools developed so far can be applied to answer the initial questions posed in a banking example. As the methodology suggests, the first issue involves creating a model with which to estimate actual system workload.

First, consider the actual workload. Consider a bank with 100,000 customers that use the Web banking system on a regular basis. Most customers use the system an average of two or three times per week, resulting in a total of 2,500,000 URL requests per week. When using the system, customers make several URL requests within a half-hour session. As a result, there are many request sources in which the requests are very short (a URL request taking at most a few seconds) and very rare (since each user only makes approximately 10 requests per week.) These facts match the characteristics of a Poisson distribution well, and it is fair to assume the request arrival rate is Poisson, with parameter λ , which remains to be estimated.

The fact that the instantaneous arrival rate is Poisson does not mean the distribution will be uniform over a long time. Indeed, actual Web site statistics indicate this is not true. Instead, arrival rates over a long period take a power-law distribution, and a Pareto assumption is justified. The Pareto estimation rule can be applied, yielding:

$$\begin{aligned}\text{Average Loaded Rate} &= \frac{\left(\frac{2,500,000}{7}\right) \times 0.8}{24 \times 0.2} \\ &= 59,523\end{aligned}$$

The result is 59,523 hits per hour, or 16.5 hits per second. Consequently, the system could expect an average load of 16.5 hits per second during peak times. This explains the observed behavior. While initial system capacity was 9,072,000 hits per week, or 15 hits per second, requests were being received faster than they could be processed during peak load conditions.

Consider Poisson distributions. If the average rate is 16.5 hits per second during high load periods, then 15 percent of the time the load exceeds 20.5 hits per second, 5 percent of the time it exceeds 23.3 hits per second, and 1 percent of the time it exceeds 24.6 hits per second. The current system is under capacity. On the other hand, if the system had available capacity for the appropriate response time at the 95th percentile, the system would almost always have time to catch up if a burst occasionally exceeded that rate. Therefore, the system needs

sufficient capacity to satisfy 23.3 requests per second — rather than increasing the capacity eight-fold, an immediate increase in capacity of 23.3/15 or 1.6 times would suffice.

$$\text{Average Load}(\lambda) \approx \frac{1 \times R(\text{arrivals/day})}{6} \left(\frac{\text{arrivals}}{\text{hour}} \right)$$

$$95\text{th Percentile} \approx \lambda + 1.68\lambda$$

Summary

The methodology described in this paper provides a simple scheme for initial performance estimation in early capacity planning. Such a method is important for several reasons:

- It ensures the architecture is well-matched with the demands it must serve.
- It enables architectural decisions to be made with some assurance that the decisions are reasonable.
- It avoids both under-engineering, which leads to unpleasant crises, and over-engineering, which can lead to expensive systems, or even to the decision to forego a system that would otherwise have been feasible.

Other methods of capacity planning have been proposed and are widely used. These methods are very accurate, and are based on precise measurements and complicated models. The advantage to the method presented here is that it is *not* based on accurate measurements and detailed models. This method can, therefore, be applied very early in the design of a system, while major architectural decisions are still being made.

Even though this method is not intended to be very accurate, it is surprisingly good for many Web systems. This is due, in part, to the inherently well-behaved nature of scalable distributed Web systems. They tend to form series-parallel graphs of processors in which each Web interaction is independent, or nearly independent, both of other concurrent interactions and of the transactions's own histories. The workloads for Web systems are also well-behaved, with tight correspondence to the theoretical Poisson distribution which is itself mathematically convenient. As a result, it has been observed in practice to be predictive to one or two significant figures in many common situations.

This method has its limitations. Consider a system in which the business model called for data to be delivered to the central host system in batch transmissions periodically over the course of a business day, and typically at the end of the day. The data is then accessed for data mining by customers over the Internet. In this



case, this methodology would not be sufficiently predictive, and would need to be supplemented by further analysis. This is because the workloads fail to match some of the underlying Markovian assumptions:

- The batch data transmissions are large and consume a significant length of transmission time.
- Data transmission times tend not to be distributed throughout the day, but instead occur at the same time every day from every customer.
- The interactions with the data mining system include very large computations or queries which consume significant time and resources.

As a result, the real workload does not approximate what might be assumed, and predictions based on the simple workload model will fail. Of course, the solution is to make a detailed analysis of the actual workload, and how the system will respond to that workload.

For all its limitations, this method is effective in practice, both for architectural design and as a way to clearly define user assumptions and expectations. Often, this simple analysis reveals either an area in which user expectations are unrealistic, or one in which expectations are unclear. It is simple and straightforward enough that it literally can be applied quickly and easily during preliminary conversations, fostering confidence in project feasibility, utility, and success.

References



Sun Microsystems Computer Company posts product information in the form of data sheets, specifications, and white papers on its Internet World Wide Web Home page at: <http://www.sun.com>.

Look for abstracts on these and other Sun technology white papers:

Simplified Performance Estimation, Sun Microsystems, 2001.

Implementing Enterprise Security Policies, Sun Microsystems, 2001.

Introduction to the Elements of Web Application Architecture, Sun Microsystems, 2001.

Understanding Capacity and Scalability, Sun Microsystems, 2001.

Understanding Growth, Sun Microsystems, 2001.

Understanding Statistics and Queuing — Probability, Distributions and Statistical Models, Sun Microsystems, 2001.

Other references:

Application and Theory of Petri Nets, 1993: Proceedings of the 14th International Conference, M. Ajmone Marsan, Editor, Chicago, Illinois, USA, June 21-25, 1993.

Capacity Planning for Internet Services, Adrian Cockcroft, Sun Microsystems Press, Prentice Hall, 2001, ISBN 0130894028.



Capacity Planning For Web Services: Metrics, Models, and Methods, Daniel A. Menasce, Virgilio Almeida, ISBN 0130659037

Configuration and Capacity Planning for Solaris Servers, Brian L. Wong, Sun Microsystems Press, Prentice Hall, 1997.

The Design of a Unified Package for the Soution of Stochastic Petri Net Models, J.B. Dugan, A. Bobbio, G. Ciardo, K.S. Trivedi, International Workshop on Timed Petri Nets, Torino, Italy, July 1-3, 1985, pages 6-13, Ieee Computer Society Press, 1985.

Java 2 Platform, Enterprise Edition: Platform and Component Specifications, Bill Shannon, Mark Hapner, Vlada Matena, James Davidson, Eduardo Pelegri-Llopart, Larry Cable, ISBN 0201704560.

Modelling with Generalized Stochastic Petri Nets, D. Kartson, G. Balbo, Giuseppe Conte, S. Donatelli, G. Franceschinis, ISBN 0471930598.

Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package, Robin A. Sahner, Kishor S. Trivedi, Antonio Puliafito, ISBN 0792396502.

Performance Engineering of Software Systems, Connie U. Smith, Addison-Wesley Publishing Company, Reading, MA, 1990.

Probability and Statistics with Reliability, Queueing and Computer Science Applications, Kishor S. Trivedi, ISBN 0471333417.

Software Engineering: A Beginner's Guide, Roger S. Pressman, ISBN 0070507902.

Stochastic Processes Occuring in the Theory of Queues and Their Analysis by the Method of Imbedded Markov Chains, DG Kendall, Annals of Mathematical Statistics, 24, 338-354, 1953.

Sun Performance Tuning: SPARC and Solaris, Adrian Cockcroft, Prentice-Hall PTR, Upper Saddle River, NJ 07458, USA, 1994.

UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers, Curt Schimmel, ISBN 0201633388.



Sales Offices

Africa (North, West and Central): +9714-3366333
Argentina: +5411-4317-5600
Australia: +61-2-9844-5000
Austria: +43-1-60563-0
Belgium: +32-2-704-8000
Brazil: +55-11-5187-2100
Canada: +905-477-6745
Chile: +56-2-3724500
Colombia: +571-629-2323
Commonwealth of Independent States: +7-502-935-8411
Czech Republic: +420-2-3300-9311
Denmark: +45 4556 5000
Egypt +202-570-9442
Estonia: +372-6-308-900
Finland: +358-9-525-561
France: +33-01-30-67-50-00
Germany: +49-89-46008-0
Greece: +30-1-618-8111
Hungary: +36-1-202-4415
Iceland: +354-563-3010
India: +91-80-5599595
Ireland: +353-1-8055-666
Israel: +972-9-9710500
Italy: +39-039-60551
Japan: +81-3-5717-5000
Kazakhstan: +7-3272-466774
Korea: +822-2193-5114
Latvia: +371-750-3700
Lithuania: +370-729-8468
Luxembourg: +352-49 11 33 1
Malaysia: +603-264-9988
Mexico: +52-5-258-6100
The Netherlands: +00-31-33-45-15-000
New Zealand: +64-4-499-2344
Norway: +47 23 36 96 00
People's Republic of China:
 Beijing: +86-10-6803-5588
 Chengdu: +86-28-619-9333
 Guangzhou: +86-20-8755-5900
 Hong Kong: +852-2202-6688
 Shanghai: +86-21-6466-1228
Poland: +48-22-8747800
Portugal: +351-21-4134000
Russia: +7-502-935-8411
Singapore: +65-438-1888
Slovak Republic: +421-7-4342 94 85
South Africa: +2711-805-4305
Spain: +34-91-596-9900
Sweden: +46-8-631-10-00
Switzerland:
 German: 41-1-908-90-00
 French: 41-22-999-0444
Taiwan: +886-2-2514-0567
Thailand: +662-636-1555
Turkey: +90-212-335-22-00
United Arab Emirates: +9714-3366333
United Kingdom: +44-1-276-20444
United States: +1-800-555-9SUN OR +1-650-960-1300
Venezuela: +58-2-905-3800
Worldwide Headquarters:
 Sun Microsystems, Inc.
 901 San Antonio Road
 Palo Alto, CA 94303-4900 USA
 Phone: 650-960-1300 or 800-555-9SUN
 Internet: www.sun.com