A Model of Web Server Performance

Louis P. Slothouber, Ph.D.
louis@slothouber.com

***Abstract* -** Despite the increasing number of Web (i.e., HTTP) servers in use each day, little is definitively known about their performance characteristics. We present a simple, high-level, open queueing network model from which we derive several general performance results for Web servers on the Internet. Multiple-server systems are also analyzed. A theoretical upper bound on the serving capacity of Web servers is defined. As Web servers approach this boundary response times increase suddenly toward infinity, disabling the server; but limiting the server's simultaneous connections prevents this problem. The effects of file size, server speed, and network bandwidth on response time are also investigated. In addition, the relative merits of several methods of improving server performance are evaluated.

Keywords - web, server, performance, model, queueing theory

1. Introduction

In the past few years the World Wide Web has experienced phenomenal growth. Not only are millions browsing the Web, but hundreds of new Web sites are added each day [12]. Yet, despite the increasing number of Web (i.e., HTTP) servers in use little is definitively known about their performance characteristics. Web server vendors are quite happy to extol the performance virtues of their products, and industry professionals abound with theories about how to serve data faster; but these virtues and theories are generally based upon anecdotal evidence, gut instinct, or narrow empirical evidence that has little general utility.

Server hardware, Web server software, and a connection to the Internet are required elements of any Web site, and they are all expensive. To generate the best possible performance for any Web site an understanding of the interrelated effects of these three elements on Web server performance is vital.

In this paper we present an analytical performance model of Web servers in which the Web server and the Internet are collectively modeled as an open queueing network. Analysis of this model yields several interesting results. Most importantly, as the load on the Web server increases the time required to serve a file increases

This paper first appeared in June 1995.

very gradually (almost imperceptibly) up to a point; thereafter, it increases suddenly and asymptotically toward infinity. This asymptote defines a clear upper bound on the serving capacity of web servers. This boundary is particularly sensitive to the average size of the files served.

As the load on the server nears the boundary, a minor increase in the load can rapidly plunge the server into a situation resembling deadlock, where it attempts to serve more and more files at slower and slower speeds such that no files (or very few) are successfully served. The majority of today's UNIX based servers allow a large number of simultaneous connections, and are particularly susceptible to this problem. Ironically, it is the servers on Macintosh and Windows platforms, often criticized for their limited number of simultaneous connections, that avoid such server deadlock.

The queueing model was also extended to investigate multiple-server systems. Results indicate that in any multiple-server system, balancing the service load between the servers is crucial to optimal performance. In fact, a two server system in which one server is slower than another appears to perform worse than the faster server alone.

Finally, several common schemes for improving Web server performance were evaluated and compared. As expected, when the network speed is the bottleneck, increasing network speed generates the best performance improvement. But when the bottleneck is the server itself, the best choice depends on several factors.

In the sections that follow a survey of related work is conducted and basic queueing theory is reviewed. The web server model is then presented and analyzed, and the results described above are derived. This analysis is followed by thoughts on future research directions and some concluding remarks. The full text of the model is available as an appendix.

2. Related Work

A Web server together with a browser program (i.e., client) constitutes a client-server system. A Web server is really just a file server connected to its clients via the Internet. Several others have used queueing models to analyze client-server systems [1,2,3,5,6,9,10,11], but not all of these investigations analyze the performance of these systems, focusing instead on fault tolerance [10] or file storage characteristics [1]. But in all cases, the structure of the World Wide Web, the heterogeneous nature of Web clients, and the idiosyncrasies of the HTTP protocol render these models inadequate as models of web servers.

Typically these other investigations predate the emergence of the Web as an entity worthy of study. They are designed with conventional file servers in mind, and assume a homogeneous LAN network architecture where the clients are both limited and well defined. In these situations a closed queueing network model is most appropriate.

Web servers do not conform to the assumptions built into previous models. For any given Web server the number of potential clients is in the tens of millions [12], and consist of a variety of different Web browsers running on various hardware platforms and connected to the Internet at several different speeds [7,8]. Hence, the Web does not represent a closed queueing system.

3. Queueing Theory

As is often the case in computer systems, Web servers typically process many simultaneous jobs (i.e., file requests), each of which contends for various shared resources: processor time, file access, and network bandwidth. Since only one job may use a resource at any time, all other jobs must wait in a queue for their turn at the resource. As jobs receive service at the resource, they are removed from the queue; all the while, new jobs arrive and join the queue. Queueing theory is a tool that helps to compute the size of those queues and the time that jobs spend in them. In this paper, we are concerned with the number of simultaneous HTTP GET file requests handled by a server, and the total time required to service a request.

In this section we will present a very simple review of important concepts from queueing theory. More complete information is presented elsewhere [2,3,4,5,9].

Queueing theory views every service or resource as an abstract system consisting of a single queue feeding one or more servers. Associated with every queue is an *arrival rate* (A) —the average rate at which new jobs arrive at the queue. The average amount of time that it takes a server to process such jobs is the *service time* (T_S) of the server, and the average amount of time a job spends in the queue is the *queueing time* (T_Q). The average *response time* (T) is simply $T_S + T_Q$.

If the arrival rate is less than the *service rate* ($1/T_S$) then the queueing system is said to be *stable;* all jobs will eventually be serviced, and the average queue size is bounded. On the other hand, if $A > (1/T_S)$ then the system is *unstable* and the queue will grow without bound. The product of the arrival rate and service time yields the *utilization* of the server ($U = AT_S$); a dimensionless number between 0 and 1 for all stable systems. A utilization of 0 denotes an idle server, while a utilization of 1 denotes a server being used at maximum capacity.

If amount of time between job arrivals (1/A) is random and unpredictable then the arrivals exhibit an exponential or "memoryless" distribution. This distribution is extremely important to queueing theory. A queue in which the inter-arrival times and the service times are exponentially distributed is known as an M/M/c queue, where the M's represent the Markov or memoryless nature of the arrival and service rates, and the c denotes the number of servers attached to the queue. When service history of a queueing system is irrelevant to its future behavior —only the current state of the system is important— that history can be ignored, greatly simplifying the mathematics. For example, the response time of an M/M/1 queue is simply $T = \frac{T_S}{(1-U)}$.

The response time curve of an M/M/1 queue as a function of utilization is shown in Figure 1. At a utilization of 0 the response time is just the service time; no job has to wait in a queue. As utilization increases, the response time of the queue grows gradually. Only when the utilization approaches 1 does the response time climb sharply toward infinity. As we will demonstrate below, Web servers behave similarly.

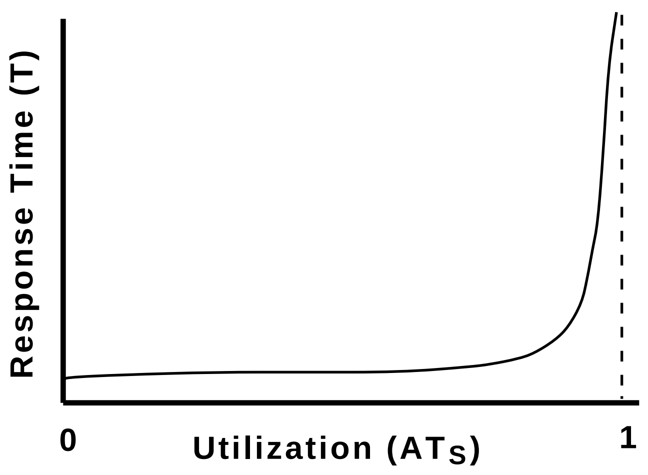

Figure 1. Response Time of a Queueing System

Little's Law (N = AT) states that the average number of jobs waiting in the queue (N) is equal to the product of the average arrival rate and the average response time [4]. Little's Law is surprisingly general, and applies to all queueing systems that are both stable and conservative (i.e., no work is lost when switching between jobs). Little's Law is especially useful when applied to queueing networks.

Typically, a single queue is insufficient for modeling a complex system such as a Web server. In many such cases a system can be modeled as a graph or network in which each queue represents one node. Such queueing networks are called *open* if new jobs arrive from outside the network, and may eventually depart from the network. Under specific conditions that are beyond the scope of this paper, *Jackson's Theorem* implies that the complex interactions between nodes in the network may be ignored [4]. Even if the arrival distribution of a queue is no longer exponential —because it is influenced by the rest of the network— that queue behaves as if it were!

4. A Web Server Model

In this paper we present a very simple, high level view of a Web server, modeled as an open queueing network. Our goal is to produce a generally applicable model that abstracts all hardware and software details, but is specific enough to produce significant performance results regarding the relationship between server and network speeds. In the current model we have ignored the low-level details of the HTTP and TCP/IP protocols, although future versions of the model may benefit from such refinements. Similarly, the current model acts as a simple file server over the Internet, and ignores both common gateway interface applications (CGI) and "if-modified" behavior [12].

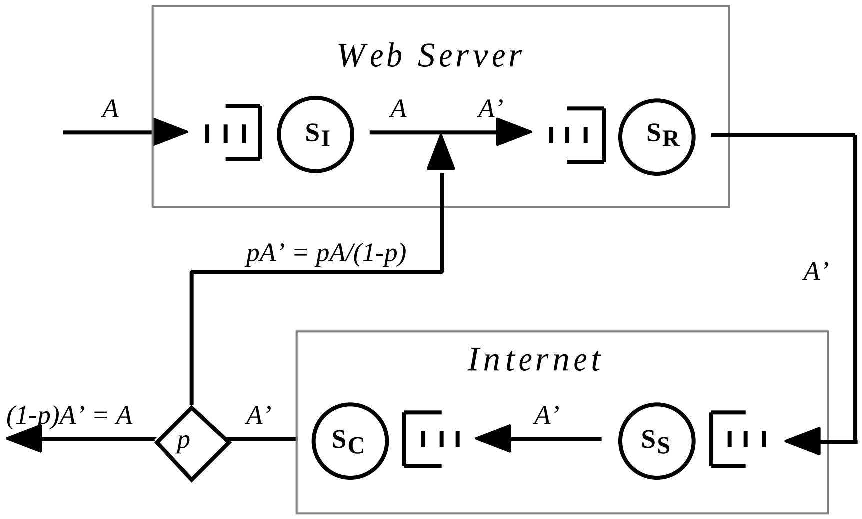

Figure 2. Queueing Network Model of a Web Server

A diagram of our Web server queueing network model is presented in Figure 2. This network consists of four nodes (i.e., single-server queues); two model the Web server itself, and two model the Internet communication network. File requests (i.e., "jobs") arrive at the Web server with frequency A. All one-time "initialization" processing is performed at node S_I. The job then proceeds to node S_R where a single buffer's worth of data is read from the file, processed, and passed on to the network. At node S_S this block of data is transmitted to the Internet at the server's transfer rate (e.g., 1.5 Mbits on a T1 line). This data travels via the Internet and is received by the client's browser, represented by node S_C. If the file has not been fully transmitted, the "job" branches and returns back to node S_R for further processing. Otherwise, the job is complete, and exits the network.

Notice that the branch is a probabilistic one; given an average file size of F and buffer size B, the probability that the file has been fully transmitted is p = B/F. Also, the arrival rate at node S_R (A') is the sum of the network's arrival rate (A), and the rate of the jobs flowing from S_C back to S_R. A' is derived using an operational law of queueing theory: the rate of jobs leaving any stable node must equal its arrival rate.

Several simplifying assumptions are built into the model. The effect of the HTTP GET requests on the network are ignored, since the requests are typically much smaller than the files that are served. Also, it is assumed that the size of requested files (and thus the service times) are distributed exponentially. Although this may not be true for some Web sites, this assumption is conservative; values based on conservative approximations represent an upper bound on the true values. Also, given fixed size buffers the service rates at nodes S_S and S_C are probably not exponential. Again, this is a conservative approximation.

The model has been implemented using Mathematica 2.2 on a Macintosh 8100. A complete text of the implementation is provided in the appendix. Treating the model as a Jackson network, the response time of the queue is given by:

$$T = \frac{F}{C} + \frac{I}{1 - AI} + \frac{F}{S - AF} + \frac{F(B + RY)}{BR - AF(B + RY)}$$

where the seven parameters in the formula are:

- Network Arrival Rate (A)
- Average File Size (F)
- Buffer Size (B)
- Initialization Time (I)
- Static Server Time (Y)
- Dynamic Server Rate (R)
- Server Network Bandwidth (S)
- Client Network Bandwidth (C)

Before analyzing the model, it is important to understand the meaning of the eight model parameters, and how they were applied during the analysis presented below.

Network arrival rate (A) is the average number of HTTP file requests (i.e., "hits") received by the Web server each second. It is important to understand that A denotes an average and not an instantaneous value. Conceptually, it is often easier to mentally translate any reference to A into a corresponding "hits per day" value; just multiply A by 60*60*24 = 86,400. Figure 3 illustrates this correspondence between arrival rate and "hits per day".

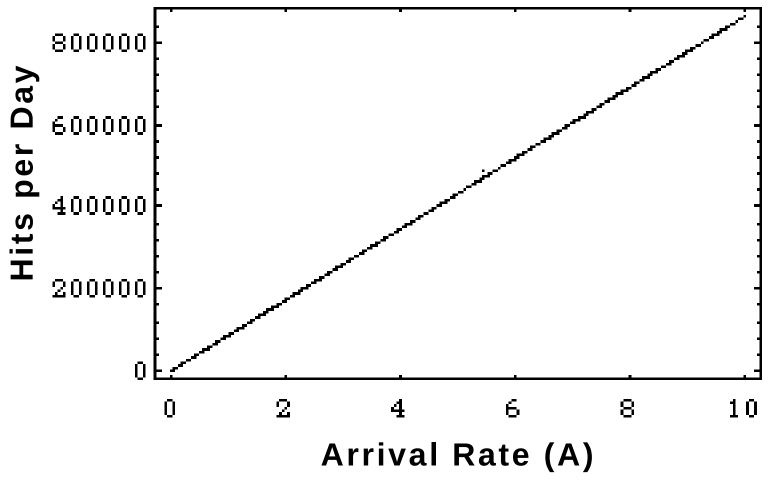

Figure 3. Arrival Rate vs. Hits/Day

Average file size (F) is the average size (in bytes) of the files served. Obviously, this value will vary widely from one Web site to another. However, after visiting 1000 Web pages at random —using the "random link" feature available from several search engines— and noting the size of every file received, including graphics, the average thus obtained was 5,275 bytes. This value has been used to generate some of the graphs below, wherever a fixed value of F was required.

Buffer size (B) is the size of the file chunks that are sent from the server across the Internet to the client's browser. Often, this value corresponds to the disk block size of the server machine. Analysis of our model shows that this value plays an insignificant role on overall server performance. An arbitrary value of 2000 bytes was used to generate all the graphs below.

Initialization time (I), *Static server time* (Y), and *Dynamic server rate* (R) collectively describe the speed at which the Web server handles requests. I represents the average time required to perform various one-time initialization tasks for each job (e.g., suffix mapping). The service rate of the S_I node is 1/I. Y represents the time spent processing a buffer that is independent of the size of that buffer. Finally, R represents the rate (bytes/second) at which the server processes the buffer. The service rate of the S_R node is $1/[Y + (B/R)]$. Web servers running on modern computers can generally serve data much faster than today's networks can transmit it.

Server network bandwidth (S) and *Client network bandwidth* (C) collectively represent the transmission speed of the Internet. S denotes the speed at which the server sends a buffer of data to the Internet. Typical values for S are (128 Kbits/sec - ISDN, 1.5 Mbits/sec - T1, and 6 Mbits/sec - T3). C denotes the average speed at which client software receives a buffer. Averaging the results from a current Internet user's survey [7, 8], a reasonable value for C is 707 Kbits/sec; this value was used to generate the graphs below.

5. Analysis of the Web Server Model

As expected, the response time curves for the Web server model resemble Figure 1. Figure 4 demonstrates the response time for a typical server connected to the Internet via a T1 line. Notice that for values of A less than 35 (that's 3,024,000 hits/day!) the response time (T) is a mere fraction of a second. However, as the server approaches full utilization T grows asymptotically toward infinity. For this Web server 3,024,000 is a theoretical upper bound on the number of hits per day that can be serviced. Henceforth, we will refer to this boundary as the maximum capacity (M) of the Web server —M is also the service rate of the Web server system.

For many people, this result may be counter-intuitive. It is a common misconception that Web servers have no maximum capacity —all jobs will eventually be serviced, albeit slowly— and that response time grows approximately linearly as A increases —the decay in performance is gradual. These misconception could have tragic consequences if it is applied by Web server managers.

Suppose a server is comfortably handling X hits per day, average response times are 50% below unacceptable values, and server utilization is increasing by only 2% of X per week. According to misconceptions above it will take almost a year before server response times double. However, if the server is near maximum capacity then response times may jump well beyond acceptable levels in a single busy day. Worse, the increased response times may be so dramatic that they exceed the patience of people browsing the site. At that point, the Web site is experiencing a situation resembling deadlock, where it attempts to serve more and more files at slower and slower speeds such that no files (or very few) are successfully served.

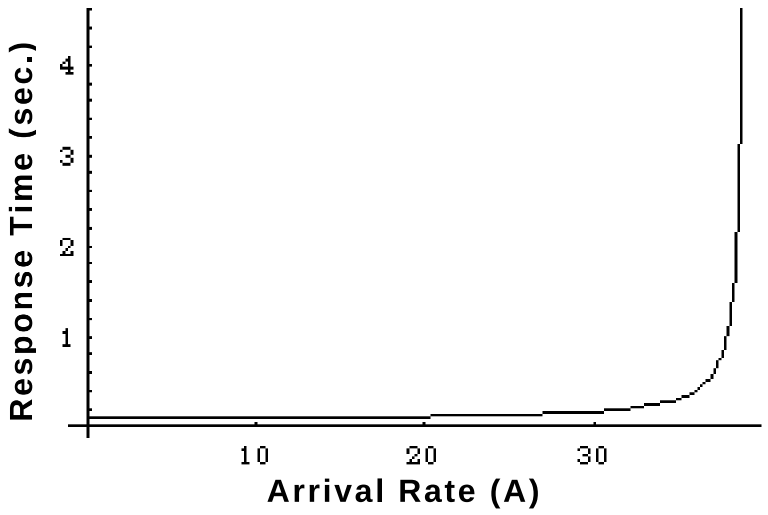

Figure 4. A Typical Response Time Curve

Given this situation, the remainder of this paper is devoted to answering three questions. (1) How do the model parameters above influence response times and, in particular, maximum capacity? (2) How can Web servers operating near maximum capacity avoid a deadlock situation? and (3) What strategies most effectively improve the performance of a Web server?

5.1. What Influences Response Time?

Actually, just about everything influences response time, at least a little. But the influence of B is negligible, and hereafter B is assumed to be 2000 bytes. Similarly, the client speed C has a very small effect on response time and no effect on maximum capacity. Since Web servers have no control over C anyway, it is hereafter assumed to be 707 Kbits/second. The effects of both I and T can be simulated by a slight increase in R. For the purposes of this investigation it is easier to let I and T be 0 and to let R alone represent server speed. The remaining model parameters — F, R, and S— all heavily influence both response time and maximum capacity.

While the effect of F on T and M is always significant, the effects of R and S depend upon whether the system bottleneck is the Web server or the network bandwidth. Because modern computers can serve files at Ethernet speeds and beyond (10+ Mbits), and the typical Internet connections (i.e., 28.8, ISDN, T1, T3) are slower, the network bandwidth is almost always the bottleneck.

Usually only very active multi-server sites (e.g., Yahoo, NCSA, Playboy) have enough network bandwidth that the Web server becomes the bottleneck. In this situation response time and maximum capacity are determined exclusively by network speed (S) and average file size (F); server speed (R) is insignificant. In those rare cases when the server is the bottleneck, it is R and F that are important, and S which can be ignored.

In addition, when the network is the bottleneck, average file size (F) has a significant, effect on response time, as illustrated in Figure 5. This graph was generated assuming the network connection is a T1 (1.5 Mbits). The ridge denotes the maximum capacity asymptote (values behind the ridge are not meaningful). Notice that for any values of F the shape of the (T vs. A) response time curve is essentially the same as in Figure 1. The greatest effect of F is on the maximum capacity asymptote (M), which decreases exponentially with respect to F.

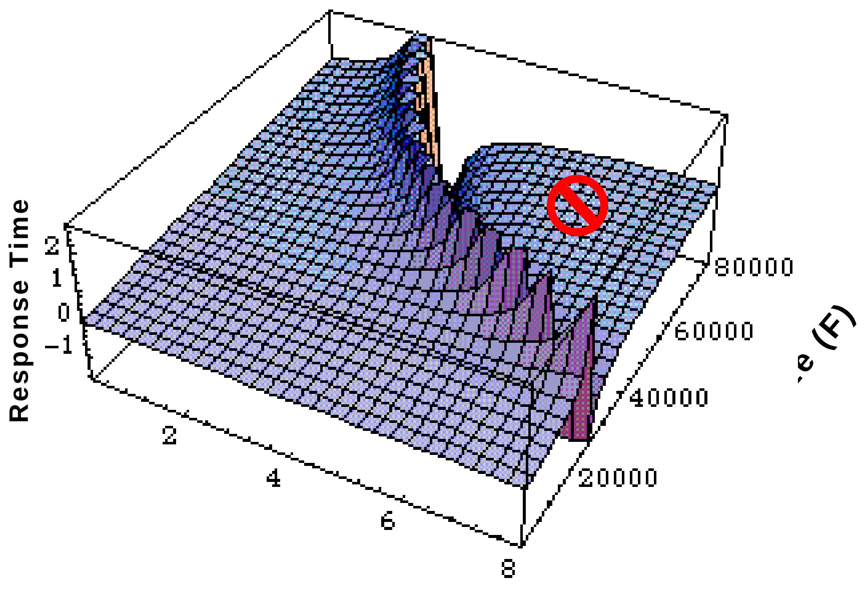

Figure 5. Response Time given A and F

It is understandable that increasing F decreases M. A Web server that serves many large files uses much of the available network bandwidth to do so. However, it is somewhat surprising that this decrease is approximately linear, gradual, and predictable. In fact, when F is relatively small (e.g., 5 Kbytes), a small change in F can have a great effect on M. But when F is already large, maximum capacity is already low, and small changes in F have little effect.

Figure 6. illustrates the relationship between network bandwidth (S) and response time. Notice that the maximum capacity ridge is straight; hence, M grows approximately linearly with respect to S.

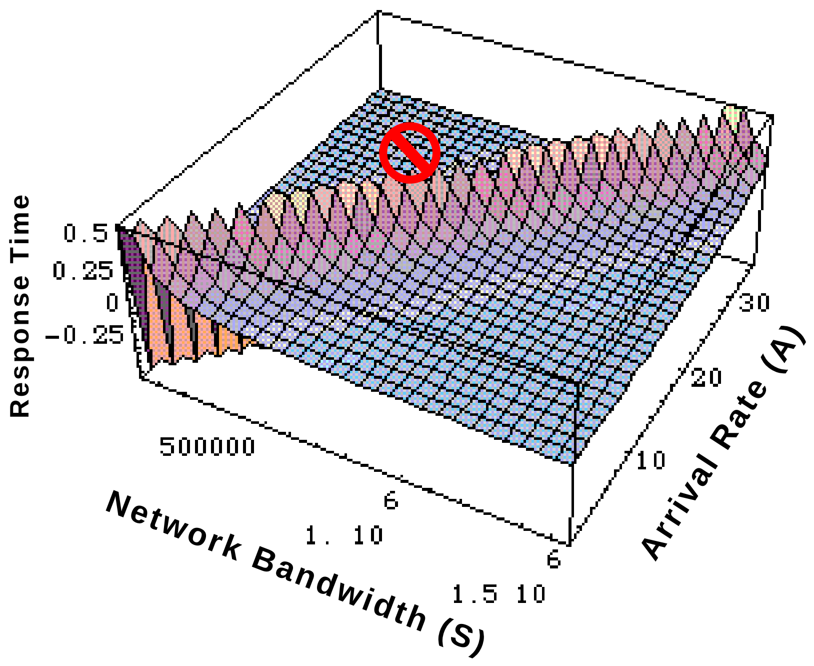

Figure 6. Response Time given A and S.

The combined effects of F and S on M are illustrated in Figure 7. Notice that the effects of F on M are very volatile for average file sizes under 20 Kbytes.

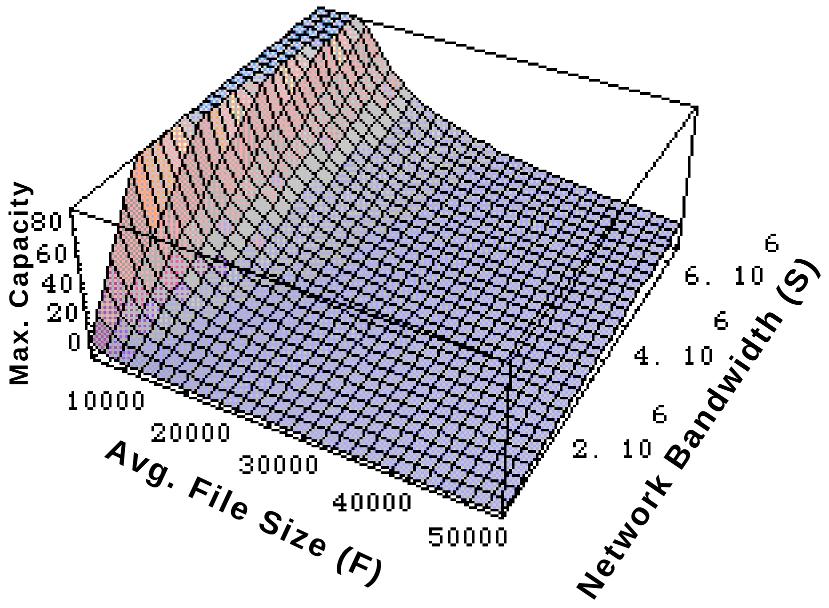

Figure 7. Max. Capacity given F and S

5.2. Avoiding Deadlock Situations

Deadlock situations occurs when new jobs are arriving almost as fast or faster than they are being served ($A \approx 1/T_S$). The only way to avoid this situation is to stop adding jobs to the queue as A nears M. This is difficult for Web servers, because they typically do not monitor arrival rate. But according to Little's Law, the number of jobs in a queueing system (N) is equal to AT. For Web servers, N corresponds to the number of simultaneous open TCP/IP connections: a known quantity. Figure 8 illustrates the relationship between A and N for the same server illustrated in Figure 4. As required by Little's Law, the shape of this curve mimics that of the response time curve, and the asymptote occurs at M. Thus, the magnitude of N can be used to detect imminent deadlock situation.

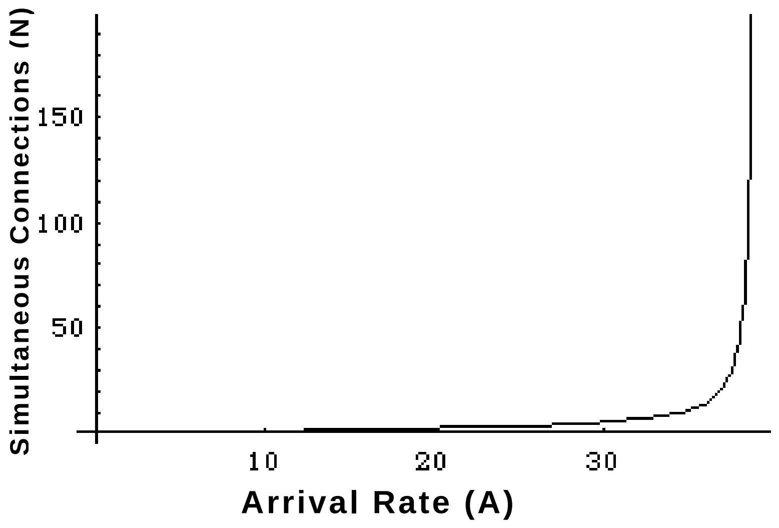

Figure 8. Simultaneous Connections vs. A

For most UNIX and Windows NT based Web servers, the number of simultaneous connections (N) is practically unlimited. It appears that this "feature", often cited as the primary advantage of UNIX based servers, is in fact a curse. Ironically, most Macintosh and Windows based Web servers already have a limit on the number of simultaneous open TCP/IP connections, imposed by either the operating system or the server software. When a Web server is nearing maximum capacity, it should respond to new file requests with the HTTP "come back later" response, and continue to complete the jobs already in its queue. The browser software should then automatically resubmit the request after several seconds when the server is hopefully less busy. Unfortunately, few servers generate this response, and no known browser supports it.

5.3. Improving Web Server Performance

When Web server performance becomes unacceptable, there are three obvious alternatives for improving it:

- Replace the server with a faster one.
- Increase Network Bandwidth
- Add additional servers

We have already demonstrated that using a faster computer (i.e., increasing R) or increasing network bandwidth (S) decreases response times and increases maximum capacity. What we have not done is compare the merits of each. Nor have we described the effect of adding additional servers.

Sometimes it is not cost effective to completely replace a working computer with a faster model. Instead, it is common practice to add additional computers. The Web site content is then either mirrored on all server machines, creating a RAIC (Redundant Array of Inexpensive Computers), or divided between the server machines [12]. This salability is an attractive feature of Web servers that run on relatively inexpensive machines.

In order to evaluate the efficacy of a multi-server system the queueing network model was altered as shown in Figure 9. Jobs are directed to node S_{R1} with probability q, and to S_{R2} with probability (1 – q).

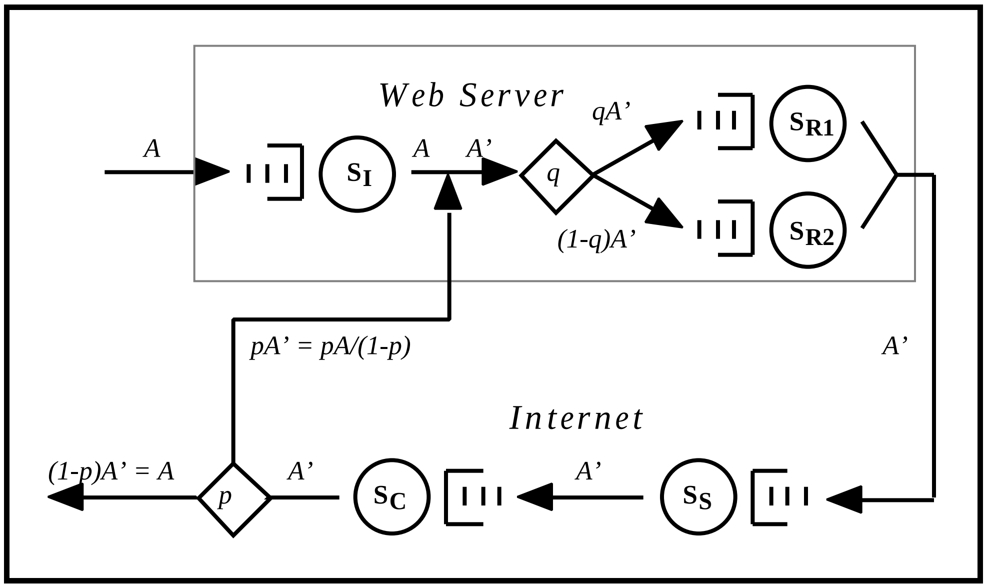

Figure 9. Queueing Network Model of a Multi-Server System

Using this new model we then investigated the relative merits of RAICs. The solid curve in Figure 10 illustrates response times of a Web server in the region well below maximum capacity (R = 10 Mbits (Ethernet), S = 1.5 Mbits (T1), and F = 5000). Four alternatives were investigated.

Obviously, the best alternative in this situation, when the network bandwidth is the bottleneck, is to increase the network bandwidth. Doubling the server speed showed a very slight improvement. Adding a second identical server in a RAIC (not shown) had no effect at all. Finally, adding a second, but slower, server in a RAIC actually increased the response time (i.e., decreased performance).

Like washing machines, multi-server systems are very sensitive to mismatched loads. Mismatched servers in a RAIC (i.e., q ≠ 0.5) overburden the slower server while the faster server may be idle. In non-RAIC, multi-server systems mismatched loads can also be caused by different average file sizes (F). However, this can be exploited to help balance the load between different model server machines.

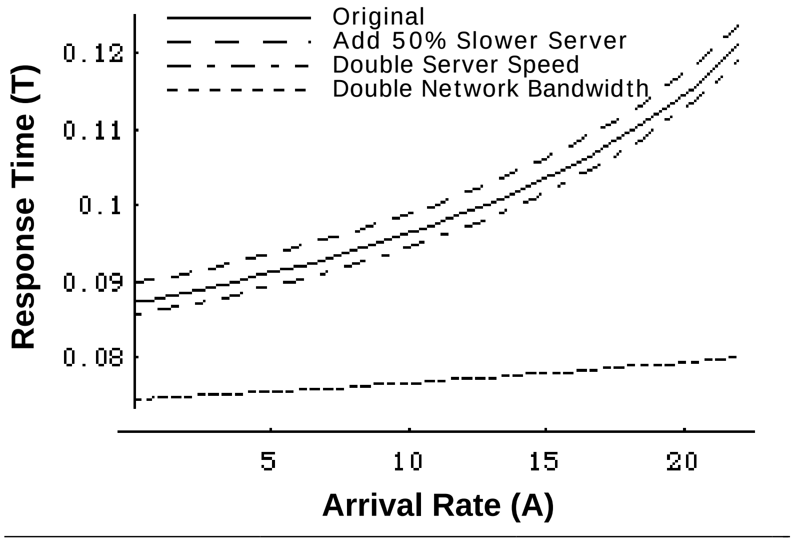

Figure 10. Improving Performance when the Network is the Bottleneck

Improving performance is even more interesting for those very active sites where the server itself is the bottleneck. Figure 11 demonstrates this situation. The best alternative, as expected, is to double the server speed. The next best choice depends upon the arrival rate experienced by the site. For arrival rates under 110 (that's 9,504,000 hits per day!) the second best choice is to double the network bandwidth.

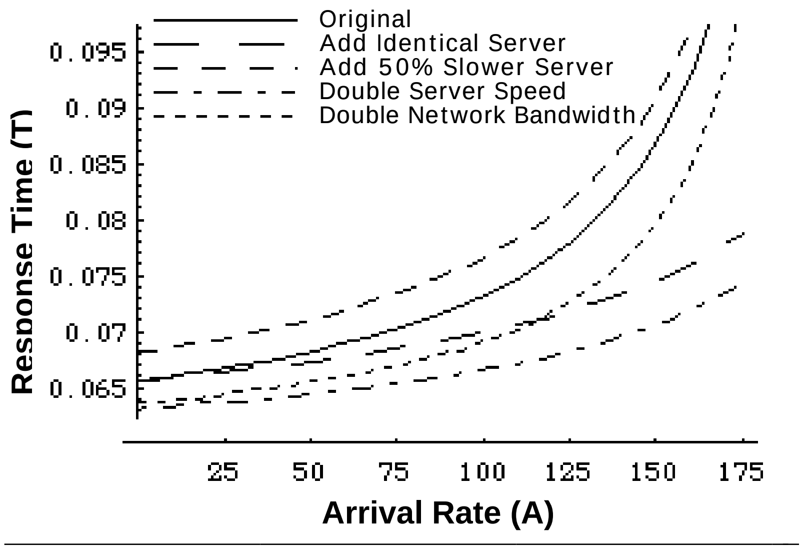

Figure 11. Improving Performance when the Server is the Bottleneck

But for higher arrival rates (whew!) the second best choice is to add another identical server in a RAIC. Finally, the worst choice is to add a slower server in a RAIC, which causes a decrease in performance.

6. Further Work

The queueing network model described in this paper is only a first step. It does not completely model all aspects of the HTTP protocol. Nor has it been empirically validated. Although the model can be used to illustrate several performance characteristics of Web servers, it cannot be employed as a predictive tool until these additional steps are taken.

7. Conclusions

In this paper we have presented an abstract performance model of Web servers in which the Web server and the Internet are collectively modeled as an open queueing network. Analysis of this model yields several interesting results. Most importantly, as the service load on a Web server increases the time required to serve a file increases very gradually (almost imperceptibly) up to a point; thereafter, it increases suddenly and asymptotically toward infinity. This asymptote defines a clear upper bound on the serving capacity of web servers. This maximum capacity boundary is particularly sensitive to the average size of the files served. By limiting the number of simultaneous connections, a Web server can avoid deadlock situations that occur as the server load approaches maximum capacity.

The relative merits of several methods for improving Web server performance were analyzed.

References

[1] Drakopoulos, E. and M. J. Merges, "Performance Analysis of Client-Server Storage Systems", *IEEE Transactions on Computers,* Vol. 41, No. 11, November 1992.

[2] Gelenbe, E. and I. Mitrani, *Analysis and Synthesis of Computer Systems,* Academic Press, New York, 1980.

[3] King, P. J. B., *Computer and Communication Systems Performance Modelling,* Prentice Hall International, UK, 1990.

[4] Kleinrock, L., *Queueing Systems, Volume I: Theory,* John Wiley and Sons, New York, 1976.

[5] Kleinrock, L., *Queueing Systems, Volume II: Computer Applications,* John Wiley and Sons, New York, 1976.

[6] Petriu, D. C. and C. M. Woodside, "Approximate MVA from Markov Model of Software Client/Server Systems", <fill in>, 1991.

[7] Pitkow, James E. and Colleen M. Kehoe, Results from the Third WWW User Survey, *The World Wide Web Journal,* Vol. 1, No. 1, 1995.

[8] Pitkow, James E. and Colleen M. Kehoe, "The Fourth GVU Center WWW User Survey", http://www.cc.gatech.edu/gvu/user_surveys/, 1995.

[9] Pujolle, G. and E. Gelenbe, *Introduction to Queueing Networks*, John Wiley & Sons, New York, 1987.
[10] Puliafito, A., S. Riccobene, and M. Scarpa, "Modelling of Client-Server Systems", <fill in>, 1995.
[11] Trivedi, K. S., O.C., Ibe, and H. Choi, "Performance Evaluation of Client-Server Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, N. 11, November 1993.
[12] Wiederspan, J. and C. Shotton, *Planning and Managing Web Sites on the Macintosh*, Addison-Wesley, 1996.

Appendix

The Web server model below was implemented using *Mathematica* version 2.2 on a Power Macintosh.

World Wide Web Server Response Function

```
wwwServerResponse[      avgFileSize_,
                        dumpBuffSize_,
                        arrivalRate_,
                        startupTime_,
                        serverTime_,
                        serverRate_,
                        senderRate_,
                        receiverRate_
                  ]     :=
Block[
    {probDone = dumpBuffSize / avgFileSize,
    feedbackRate = (- arrivalRate) +
                    (arrivalRate / probDone),
    lambda = arrivalRate + feedbackRate,

    startupMu = 1 / startupTime,
    startupRho = mm1TrafficIntensity[
                arrivalRate, startupMu],
    startupJobs =
        mm1MeanNumberOfJobs[startupRho],

    serverMu = 1 / (serverTime +
                (dumpBuffSize / serverRate)),
    serverRho = mm1TrafficIntensity[
                    lambda, serverMu],
    serverJobs =
            mm1MeanNumberOfJobs[serverRho],

    senderMu = senderRate / dumpBuffSize,
    senderRho = mm1TrafficIntensity[
                    lambda, senderMu],
     senderJobs =
            mm1MeanNumberOfJobs[senderRho],

    receiverMu = receiverRate / dumpBuffSize,
    receiverJobs =
            mmInfMeanNumberOfJobs[
                        lambda, receiverMu],

    totalJobs = startupJobs + serverJobs +
                senderJobs + receiverJobs},

    (* Little's Law: T = N / A *)
    totalJobs / arrivalRate
]
```

M/M/1 Queue

```
(*Traffic intensity*)
mm1TrafficIntensity[lambda_, mu_] := lambda / mu;

(*Probability of n jobs in the system*)
mm1ProbOfNJobs[rho_, n_] := (1 - rho) (rho ^ n)

(*Mean number of jobs in the system*)
mm1MeanNumberOfJobs[rho_] := rho / (1 - rho)

(*Mean number of jobs in the queue*)
mm1MeanNumberOfJobsInQ[rho_] := rho^2 / (1 - rho)

(*Mean waiting time*)
mm1MeanWaitTime[lambda_, mu_] :=
    Block[ {rho},
        rho = mm1TrafficIntensity[lambda, mu];
        rho * (1 / mu) / (1 - rho)
    ]

(*Mean response time*)
mm1MeanResponseTime[lambda_,mu_] :=
    Block[ {rho = lambda / mu},
            (1 / mu) / (1 - rho)
    ]
```

M/M/Infinity Queue

```
(*Traffic Intensity*)
mmInfTrafficIntensity[lambda_, mu_] :=
    lambda / mu

(*Probability of n jobs in the system*)
mmInfProbOfNJobs[lambda_, mu_, n_] :=
    Block[ {rho},
        rho = mmInfTrafficIntensity[lambda,mu];
        (E^-rho/n!) rho^n
    ]

(*Mean number of jobs in system*)
mmInfMeanNumberOfJobs[lambda_, mu_] :=
    mmInfTrafficIntensity[lambda, mu]

(*Mean waiting time == 0*)
mmInfMeanWaitingTime[] := 0

(*Mean response time == service time*)
mmInfMeanResponseTime[mu_] := 1/mu
```