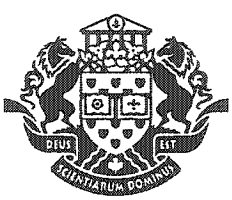


Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Zhengdao XU

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Master of Computer Science

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Study of Oscillation in Admission Control Algorithm for Web Servers

G. V. Bochmann

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

C.-H. Lung

J. Zhao

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Study of oscillations in admission control algorithm for web servers

Zhengdao Xu

A Thesis submitted to the Faculty of Graduate Studies and Postdoctoral Studies in partial fulfillment of the requirement for the degree of Master of Computer Science

May 2003

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 0-612-90358-3
Our file Notre référence
ISBN: 0-612-90358-3

Abstract

In recent years, the advance of the Internet technology and e-commence applications becomes the motivation for the development of scalable server brokerage architectures for the purpose of load sharing.

There are many on-going researches and proposed solutions in solving this problem, but our project focus on providing the satisfactory QoS with the brokerage architecture. Besides the load sharing among all the servers in the server pool the brokerage architecture has to guarantee the response time provided by the web server since it is an important factor for user satisfaction. But in the earlier server selection algorithm of the brokerage architecture, a threshold is used to decide whether to accept/reject users depending on the current response time. A problem with this approach is that oscillations occur. Due to the abrupt manner of accepting/rejecting user, the system experiences unavoidable oscillations in terms of the response time, the number of users in the system and even the utilization of the servers.

The work of this thesis is oriented towards solving the problem of system performance oscillations. We first establish the relationship between the number of users in the system and the average response time. Then we study a theoretical model of the oscillation of the number of users in the system. Then we propose a probabilistic approach to admission control where the probability of rejecting a new user increases as the load increases. Using the theoretical model, we prove that with a probabilistic approach, the oscillations will normally be suppressed, and the number of users in the system reaches a stable point.

We also test the effect of different probability functions and the impact of different inter-observation time intervals on the oscillation by careful simulation experiment. Finally, the probabilistic approach is used to provide differentiated classes of service to different user groups. We show that the probabilistic approach provides better performance than the on-off decision approach.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Gregor v. Bochmann. His insightful conversations about the ideas in this project and helpful comments on my work inspired me greatly during the whole process of my project.

I thank PhD student Mohamed-Val M. Salem for giving me a lot of help on the CSIM18 simulation library tool and building up the prototype of the broker, which made it possible to start my project.

I also thank everybody in the DSR group. They constantly encouraged me and gave me good suggestions.

Finally, I'd like to thank my wife, Zhao Min, who always supports me through difficulties and shares the same dream with me.

Table of contents

Table of figures

CHAPTER 5

CHAPTER 6

1. Introduction

With the exponential expansion of the Internet infrastructure, many Internet-based businesses are experiencing a fast growth, especially in the e-commerce context, and multimedia applications like video-on-demand. Those applications either have a large number of clients, or need to transmit a vast amount of data. They put heavy stress onto the web servers and pose a great challenge to the QoS that is promised to the clients.

The capacity of one server is very limited. For a large electronic merchant, millions of requests per second are expected. To handle those requests, hundreds and thousand machines are needed. And also a complex electronic commerce application usually consists of cooperating pieces of software located on different machines, and they are not even geographically together. Those separations of software are usually functional. Different machine can handle different details to support an electronic shopping model, some for interaction with customers (web page server), some for database (like users' personal profiles), some for security (like in registration and banking). To a large extent, they need the system to scale to a large number of users. Another issue is quality of service. Nowadays, lots of customers do electronic shopping. They will submit the purchase request of some advertised goods to web servers. The server has to make responses to those requests. In the user's point of view, the response time is defined as the time the user spends waiting for the request to be completed. It depends on a lot of factors like the server speed, the network bandwidth, size of files and processing time for ciphering/deciphering etc. The response time has a direct impact on user satisfaction and the reputation of the merchant sites. So the maximum of the response time becomes a basic measure of QoS.

In this context, some questions are posed here: on the one hand, how can we distribute

the requests from the clients evenly among all the web servers; and on the other hand, how can we still provide satisfactory QoS even during the time of heavy load. There have already been quite a few research efforts coping with these problems for optimising the system performance and decreasing the response time. We will examine some of them in detail shortly. Unfortunately, those approaches have not taken the oscillation of the system performance into consideration, so when the workload is high, the response time will sometimes become very long (a long waiting time), and sometimes very short (higher percentage of user rejection). That is exactly why we use a probabilistic approach in our project. Although such a gradual approach (also called proportional control) has long been used as a way of system control in industry, it is the first time to be used for admission control for web servers, so far as we know. In this new approach, each time a user comes to ask for admission, the broker will grant it with some probability based on the current measured response time. By doing this, we no longer admit or reject all users; the users are always accepted by some percentage, which depends on the workload of the system. In this thesis, we start with studying the nature of the performance oscillations by establishing its theoretical model; with some mathematical proof, we show that the probabilistic approach suppresses the oscillation and the performance reaches a so-called "stable point". Then we study the probabilistic approach further in a more realistic setting by simulation studies.

Since many distributed applications are expected to provide different levels of service to different classes of users (for example, the e-commerce system should distinguish between a casual user and a registered user, those registered user should receive the best service in terms of the priority of acceptance and reasonable response time), we use the probabilistic approach to provide differentiated services to different user groups. Through simulation studies, we show that its performance is better than the on-off approach. In the following, we briefly list the contributions and give an outline of the thesis.

1.1 Contribution of this thesis

1. We use a probabilistic approach for admission control to the web servers rather than the simple on-off algorithm in order to avoid performance oscillations and provide the desired quality of service to the client.

2. We established the relationship between the average response time/server utilization and the number of users in the system by means of the simulation experiments. Based on these results, a reasonable cut-off point is chosen.

3. A theoretical model of the oscillation of the number of users in the system with the on-off decision-making approach is established. By studying the theoretical model, the amplitude of the oscillation can be computed and the nature of the oscillation is well understood.

4. A theoretical model of the oscillation of the number of users in the system with the probabilistic approach is established and studies. We prove that with the probabilistic approach, the oscillation will normally be suppressed and the performance reaches a so-called stable point.

5. The effect of the probability functions with different slopes on the avoidance of oscillation is tested, and we find that a more gradual probability function eliminates the oscillation better than a less gradual probability function.

6. Through simulation experiments, we show that with a given probability function, a smaller inter-observation time period can also eliminate the oscillation.

7. Since the number of users in the system is a direct indication of the system response time, we test the efficiency of using an upper limit of the number of users and conclude through the simulation result that the upper number of users in the system has an effect of avoiding oscillations only when the workload of the client reaches/exceeds that limit.

8. We show that the probabilistic approach of admission control can be used for two differentiated classes of users: one group with higher priority and another one with lower priority. We show that with the probabilistic approach, the QoS provided to the higher priority user group is much better than when we use the traditional on-off threshold approach.

9. We propose a combined probability function approach to provide the different QoS to differentiated classes of users. This approach has the advantage over the two probability function approach, in that with the single probability function approach, for a given combined user incoming rate, the resulting response time is predictable regardless of the composition change of the different groups.

1.2 Organization

Following is the organization of this thesis paper:

CHAPTER 2: Load sharing control for web servers: In this chapter, we review related work on providing scalable architecture for web servers. The basic brokerage architecture, its dynamic behaviour are also introduced here.

CHAPTER 3: Simulation principles and tools: In this chapter, some of the very basic principles of simulation modelling are reviewed, and the simulation tool CSIM18 is introduced. We also give a simple code example of a queuing system. Finally, the simulation model used in our project is studied.

CHAPTER 4: The problem of performance oscillation: The problem of the performance oscillation (to be more specific, the oscillation of the number of users in the system) is studied. The relationship between the response time and the number of users in the system is established and we also study the theoretical model of the oscillation of the number of users in the system in the cut-off threshold case. Some basic principles of the system control are also introduced in this chapter, which give us the hint of solving the problem of oscillation using the system control method.

CHAPTER 5: Probabilistic approach to admission control: The detailed design of the probabilistic algorithm for gradual user rejection is given. A careful mathematical study of the oscillation in the probabilistic approach is also provided. The performance of this probabilistic approach is evaluated by simulation studies. All the results are collected and presented.

CHAPTER 6: Probabilistic approach used for differentiated classes of users: The probabilistic approach is used on two groups of users, by either using two probability functions on each of them or using a single probability function but treating users differently according to the class to which they belong. The difference of these two approaches is also studied.

CHAPTER 7: Conclusions and future works: Finally, a conclusion for our project is drawn, and some possible future works are mentioned.

2 Load sharing for web servers

In this chapter, we review the approaches to the load sharing of web servers. We first start with the overview of related work in providing load-sharing function to the web servers and how they cope with performance oscillations. And after a brief introduction of the brokerage architecture of [4], we introduce its load sharing and dynamic properties. This is important for us, since in this project, we base our study on the brokerage architecture. By dealing with the problem of oscillations in the brokerage system, we hope to provide a satisfactory QoS to the users without sacrificing its nice properties of load sharing and scalability.

2.1 Overview of related work

The research of how to distribute the large number of clients among a group of replicated servers in order to provide satisfactory quality of service has be around for over a decade. The goal is to allocate servers to the clients in such a way that the response time experienced by the clients is minimized. Quite a lot of efforts have been made according to specific optimization criteria, and lots of different approaches have been published, some of which will be discussed in the following.

2.1.1 Servers are picked up by clients

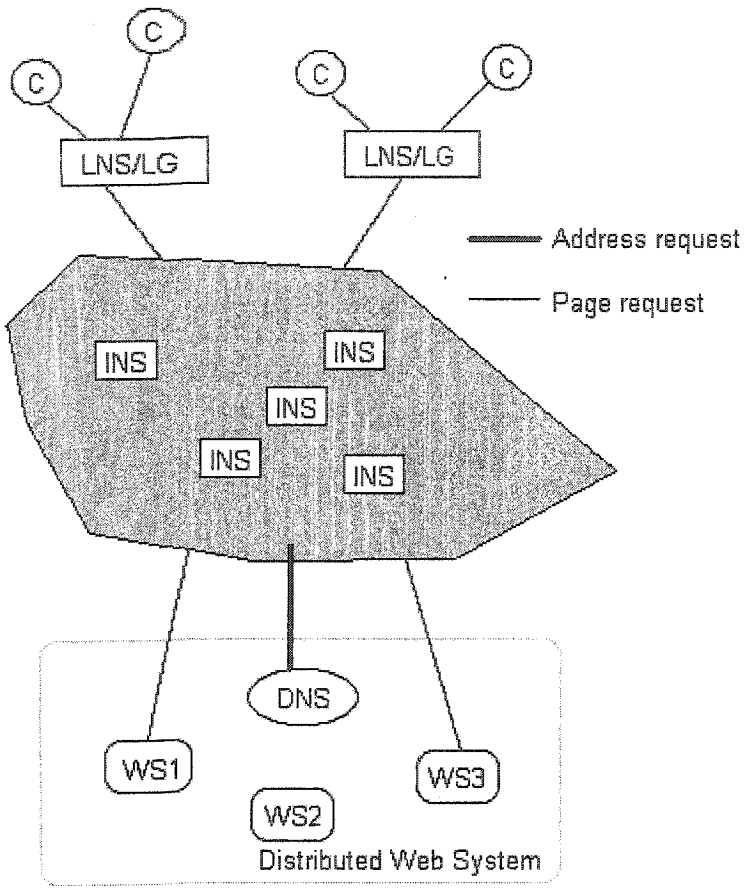
Some of the load sharing approaches are very rudimental. One of them broadcasts a list of servers, from which the clients has to pick the best one. The difficulty of this approach is how the client can know which server is the "best". A server geographically closer does not necessarily guarantee a shorter response time. Also, whenever the configuration of the servers is changed (server down, or a new server added), some site has to be aware of this,

and a new server list has to be broadcast. Although not impossible, this method is not very scalable.

2.1.2 Distribute the workload by the DNS

Another approach uses the DNS (Domain Name System) to return the IP address of one server among the server pool when it is queried about the IP address of the website. But the DNS does not usually keep performance information about servers. Usually, it can only distribute the servers to each client in a round robin manner. It is really a bad approach if the computing capacity of the servers varies, in which case a slower server get the same workload as a faster one. Also most DNS control only a very small portion of the server requests (actually only the initial name to IP address resolution requests), as show in the

1. The local name servers (LNS), the intermediate name servers (INS) and even the clients (C) themselves can usually cache the result of the previous address resolution and the same requests will never go back to the DNS that controls the multi-server (WS) domain. Address caching bypasses the remote DNS, and therefore limits the control of the DNS, and makes the server performance independent of the DNS decisions [12]. This is exactly where the difficulty lies.



C: Client, LNS: Local Name Server, LG: Local Gateway, INS: Intermediate Name Server, DNS: Domain Name Server, WSn: Web Server number n

Figure 1. Approach using DNS to distribute client requests

According to the work accomplished by Michele Colajanni et al [12], this problem can be solved by providing a TTL (time to live) to every name server (from INS to LNS) along the path from the DNS to the client when the DNS returns the IP address of the chosen web server to the client. The name to address mapping is kept in the database at the intermediate name server and the local name server only for a time period specified by TTL. After the period TTL, this mapping entry is simply deleted, and the next request should again be forwarded to the remote DNS. By doing this, the DNS gets more control over the flow on the network. Based on this idea, several possible server-scheduling

algorithms that are extended on the DNS as in the following.

Based on the source of information used by the DNS, the DNS scheduling algorithms can be classified into three categories; namely (1) the algorithms using domain information, (2) the algorithms using load information from the web servers, and (3) the algorithms combining both domain and server information. We briefly describe these algorithms as follows.

(1) Algorithms using domain information

Two-tier-Round-Robin (RR2)

Based on the fact that the hidden load weight (average number of web requests from the domain per name-to-address mapping) of the clients under each LG (Local Gateway) can be very different, this algorithm partitions the domain under different LGs roughly into two classes, i.e. hot (with higher hidden load weight), and normal (with moderate hidden load weight). The round robin scheduling is used on the LGs under each class. RR2 algorithm avoids assigning too many requests from hot class domains to the same server, and therefore it tends to average the load to every server in the pool.

Dynamically Accumulated Load (DAL)

In every measurement period, DNS accumulates the hidden load weight of every assignment for each web server in a variable *bin.* And the web server with the lowest *bin* value is chosen when the address resolution is requested. The *bin* value is increased by the hidden load weight of that LG afterwards to represent the increased load that will arrive.

(2) Algorithms using load information on web servers

Lowest Utilization (LU)

The web server utilization (in the most recent measured interval) is used for the server selection purpose, the server with the lowest utilization is chosen during address resolution.

Lowest among Past and Present Utilizations (LPPU)

Like the LU algorithm, only several recent measures are used, and each one is weighed with different weight value (with the most recent measure being the highest).

(3) Algorithms combining domain and server information

Single threshold (Thr1)

Basically, this approach use the RR2 or DAL algorithm, but it also keeps track of the alarm message coming from any of the web servers which announces that its utilization has exceeded a certain threshold, and excludes them from the candidate list. These overloaded web servers will not be assigned to any domain until their workload drops back below the threshold at which moment another message will be sent to the DNS to notify this.

Double threshold (Thr2)

Similar to the Thr1 algorithm, but to avoid the performance thrashing, this algorithm uses a second threshold (lower than the upper one) to tell when the excluded server should be "re-activated". In this way, the newly included server will not be excluded again too soon during the time of heavy load.

Temporal threshold (ThrT)

Same as the Thr1 algorithm, except that the re-activation of the server is triggered by a timer, which specifies a period of time long enough for the excluded server to finish its currently assigned load.

These algorithms are quite simple to implement, and the expiration of the name-to-address mapping enables the DNS to get back more control over the network flow, thus realize the load balance.

2.1.3 Using an anycast resolver to distribute the workload

The most recent technique is to make use of the anycast domain names (ADNs) ([14], [38], [41], [42]). Such a name identifies a group of IP addresses of the replicated servers. It is assumed that an anycast resolver stands between the clients and the servers, and maps the ADN into the IP address of one of the servers. The web service request is started with the anycast query, and the resolver responds with a server IP address. Then the client talks to this assigned server until he finishes. To guarantee the quality of service, the performance information associated with each server has to be maintained in a performance database. This information is used for the purpose of server selection. Upon this basic anycast architecture, some extensions have been proposed. Among them, one study [9] worth noticing is done by Z. Fei, S. Bhattacharjee et al from Georgia Institute of Technology. Their approach deals not only with the problem of providing reasonable response time to the client, but also with the problem of performance oscillation.

In their anycasting system architecture, they use a hybrid of Push/Probe techniques to keep the performance information of the servers updated. They use the concept of a so-called "set of equivalent servers (ES)" to pick up a good quality server from the server

pool. For the push algorithm on the server side, the server pushes the performance information to the resolver whenever the change of the measured performance exceeds some predefined threshold. The probing mechanism is realized by a *probing agent*, which is co-located with the resolver. This agent periodically queries a well-known file on the server to measure the real performance of the server. The reason to separate the functions of probing and resolving and put them on two different sites is that they want the resolver to be server-protocol independent. But the probing agent still has to be aware of the protocol on the server side.

To insure a reasonable response time for the clients and to prevent the clients from oscillating among different servers (some servers may be favored at one time and over-loaded at another time), they use the idea of a set of equivalent servers (ES) to define a set of servers in the server pool, which can still provide good quality of service. And when queried by the client, the resolver randomly picks up one server in the ES, and sends back its IP address. The ES group is re-calculated each time some server pushes performance data to the resolver. Their ES computing algorithm [9] keeps two thresholds τ_j and τ_l for response time to control when a server can be included into the ES and when it should be kicked out owing to its poor service. τ_j and τ_l are called joining threshold and leaving threshold, respectively.

2.1.4 Weaknesses of the above approaches

The approaches described above can distribute the workload to different servers, thus realizing load-sharing control. But they each have some weakness as described below.

(a) Servers are picked up by the clients

Despite the difficulty of broadcasting the new server list, and the poor scalability, this approach defines the "best" server by some threshold of the response time. Servers whose

response time is below that threshold are all "best". But this inevitably incurs the problem of performance oscillation. At one point in time, a server may be considered to be the best, and all the clients will choose that server since it is believed to provide best service at that time. Soon after, it is very heavily loaded and excluded from the "best" choices. As the server switches back and forth between the "best" and "not best", the workload of the server varies periodically between heavy and low.

(b) Distribute the workload by the DNS

The weakness of this algorithms is that the classification into the hot and normal domains is still too rough a measure, the actual requests/mapping of individuals under each domain can be very different. Also they use the utilization of the server as a measure of the threshold rather than the response time of the client's perspective. The quality of service provided to the clients is surely not a very serious issue under consideration. And most importantly, the abrupt switch at the threshold to re-activate or deactivate a server will cause workload oscillations on the server side similar to the previous approach.

(c) Using an anycast resolver to distribute the workload

Complicated as it is, there are still some disadvantages in this approach. First of all, the server protocol must be modified to add the push function, this affects its scalability. Secondly, the server push mechanism only measures the server performance, and probing checks the performance of the network links as well. However, it is not clear how this information can be combined to make it useful. Finally, the equivalent server set is maintained by joining and leaving thresholds, which dumps out, and pulls in servers abruptly, therefore, the workload oscillation is unavoidable. Even though the authors claimed that by choosing larger thresholds τ_j and τ_i (make $|\tau_j - \tau_i|$ larger), the oscillation could be minimized. But in our perspective, a larger threshold τ_j and τ_i means keeping

larger number of servers in the ES set, which implies that we sometimes have to keep servers with less good service quality in the ES set and thus degrade the quality we provide. This is actually an approach, which trades the stable performance at the server side with bad quality of service on the client side. This approach works well with low to moderate server loads, but uncontrolled oscillations are unavoidable when the load becomes high.

To conclude, all the above approaches fail to take the stability of system performance under consideration. They all use the threshold to dump out the clients. Due to the abrupt nature of the threshold, the workload oscillation among the servers is very hard to avoid. The effect of this kind of on-off decision depending on a threshold can be so bad that in one time interval too many users may rush into the system and the response time soon becomes intolerable, and in the next interval not a single user can be admitted, resulting in very low server utilization. So in our project, we will focus on how to avoid the performance oscillations, yet without impairing the system scalability. In particular, we will use the brokerage architecture, which is introduced below as our basic system architecture; by improving its server selection algorithm, we can eliminate the oscillations, and provide satisfactory QoS to the clients.

2.2 Brokerage architecture

The brokerage architecture was first introduced in Mohamed-Vall M. Salem's paper "A Scalable Load-Sharing Architecture for Distributed Applications" [4]. In this proposed architecture, a delegate server, so-called broker, is used to distribute the client requests among different servers in its server pool. By implementing some load-sharing algorithms that assign the servers to different clients depending on some rules, no server

will be overwhelmed by too heavy load while others have few requests. Several server selection algorithms will be briefly discussed in section 2.2.2. In section 2.2.3, we introduce the dynamic property that makes the brokerage architecture scale.

2.2.1 Introduction to the brokerage architecture

In this section, we make a brief introduction to the brokerage architecture, which is shown in the figure below [4].

In the following Figure 2, the broker is dedicated to assign the client the IP address of the server (from the server pool) that is granted to this client for the duration of a session. It has the responsibility to control the admission of the clients depending on some criteria that is implemented in the server selection algorithm. Since the response time is one of the major factors to customer satisfaction, we use it as a criteria for the admission decision. The main function of the broker is load balancing among the servers in its pool and monitoring the response time of these servers. The load balancing is well studied in Salem's work [4], so in this project, we focus on the impact of performance monitoring, and in particular the impact of the observation time period. We notice that the broker considers the server pool as a single working unit, and within this unit all servers are equally treated in the case of homogeneous servers. In our simulation study, since the performance monitoring aspect and the observation time period are independent of the number of servers in the pool, we consider in the following, for simplicity, that the server pool contains a single server.

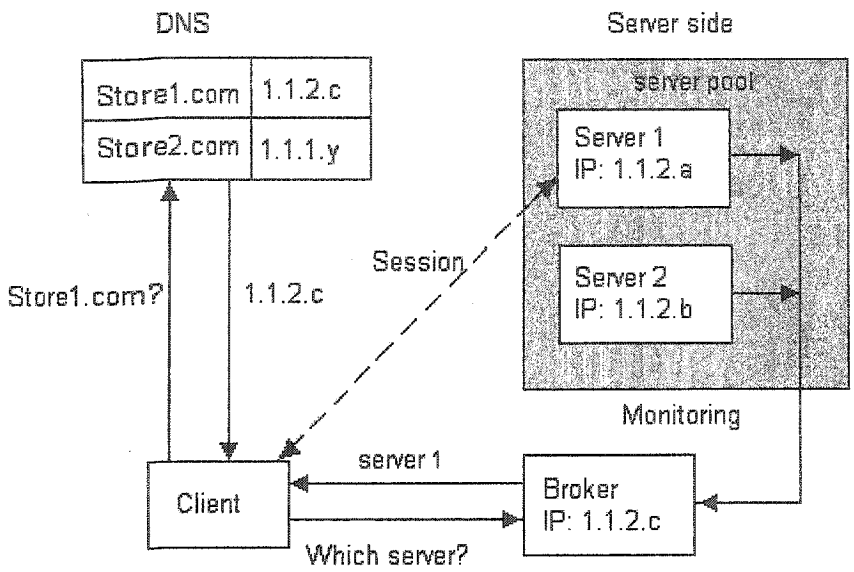


Figure 2. Basic architecture (from [4])

The brokerage architecture works as follows. After the client gets the IP address of the broker from the DNS server, he uses this IP address to ask the broker for the server's address. The broker will either accept this user or turn his request down, depending on the current load of the web servers. If the admission is granted, the IP address of the server will be sent to the client. And then the client is allowed to send any number of HTTP requests to the assigned server. In Salem's work [4], he uses the idea of quantum, which defines the time period this server assignment is valid (a similar approach can be found in [31]). But later we realized that in the quantum-based approach, we dump out users already in the system when the load is high without letting them finish their session. Imagine how unreasonable it is to dump out a client who has already been browsing on this web site for half an hour, and is just waiting for the last web page to finish. So in this project, we try to improve our brokerage model by removing the quantum control and to reject only new users that are not yet in the system. Once admitted, clients will be allowed to stay until they finish. So in the following discussion, we do not use the quantum time, no user who is already in the system will be dumped out due to the long

response time. Every user admitted will successfully finish all his requests no matter how bad the current response time is.

On the server side, every server keeps track of the average response time it is experiencing, and pushes this data periodically to the broker (in practice, these data can also be collected by the broker by probing the servers). These data help the broker to make server selection decision. The time period between the collections of these response time data is called inter-observation time. It has a pre-defined value, and indicates how closely the broker keeps watching the performance of the servers. The determination of the inter-observation time is up to the administrative management; it should be chosen properly. If it is too long, the admission control over the system will be too loose; and if it is too short, the workload of data collection can be too heavy for both broker and servers. An improper length of the inter-observation time and the on-off decision making server selection algorithm are two major factors that contribute to the instability of the system performance, as we will see later. In Section 5.3.2 we are trying to find out an appropriate inter-observation time by simulation experiment.

2.2.2 Load sharing in the brokerage architecture

The goal of load sharing control is to balance the workload of each server in the multi-server pool so that no server will be overloaded or under-utilized. In terms of load sharing control, the brokerage architecture performs exceptionally well. In Mohamed-Vall M. Salem's work, the load sharing function of the broker has been very well studied. Two kinds of algorithms, namely static algorithms and dynamic algorithms (depending on whether the run-time performance measurements are used or not), are discussed and compared. We briefly list these algorithms and shortly describe their operations here.

Static Algorithms (run-time performance measurements are not used in the server selection decision)

(A) Round Robin (RR) – Servers are selected in a cyclic order.

(B) Weighted Round Robin (WRR) – Cyclic order is used, but the faster servers are more frequently selected than the slower ones.

Dynamic Algorithms (run-time performance measurements are used in the server selection decision)

(C) Least Active Session (LAS): The number of sessions assigned to each server can be estimated and recorded by the broker, and the server with least active sessions is selected.

(D) Least Utilization (LU): The utilization of the server is recorded by each server and periodically transferred to the broker and the server with least utilization is selected.

The experiment shows that RR is faster and easier to implement, but it only works well where all servers have the same capacity. WRR has the advantage of simplicity, and is very effective in balancing the load among the servers if the available capacities at various servers do not change very frequently. In cases where the available capacities at various servers change frequently, an adaptive mechanism like LAS or LU will be a good alternative. Better than RR and WRR algorithms, LAS and LU do not have to know the speed of each server, which can also change from time to time and might be very difficult to measure. What is more, these algorithms do not care how many servers there are in the domain; the scheduling is based on the current status of the server instead of the specific server configuration.

With these algorithms, the load-sharing problem among servers can be solved pretty well. Every server gets a compatible workload, and no one will be extremely heavily loaded.

2.2.3 Dynamic properties of the brokerage architecture

Compared with other methods of load balancing, the great strength of the brokerage system lies in its dynamic nature of configuration. We deem this an important asset that makes our system truly distinguishable and outwits the other approaches we discussed in Section 2.1.

Concerning its dynamic properties, first of all, the configuration of the broker is very flexible; the broker can be co-located on the server side, and under the same management as the replicated servers. But this is not absolutely necessary, it could be placed anywhere on the global scale. Several different web sites can even share the same broker. Usually, we suggest that the web site that has very heavy workload manages the broker of his own, only those web sites with comparatively low workload justify the sharing of a broker.

Secondly, the brokerage system could scale up to a system of replicated brokers when the web requests from the clients exceed the number where a single broker can no longer support them. This multi-broker architecture is also studied in Mohamed Salem's work [21]. In the multi-broker system, besides the normal function of distributing requests under its own domain (a domain is usually geographically based in order to minimize the delay on the network), the brokers also need to communicate and cooperate with each

other so as to balance the load globally. In the Figure 3, we show two independent clusters of servers in the multiple broker architecture.

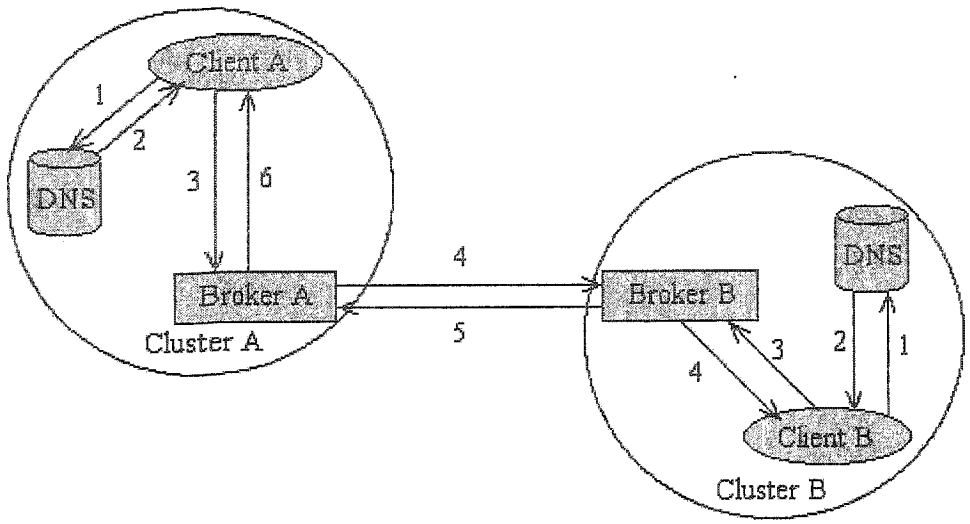


Figure 3. Two brokers communicate to balance the load between two clusters [21]

To allow the exchange of the load status between the brokers, they are managed in a group. A special protocol is needed for a broker to join or leave the group. A simple protocol is to broadcast a join/leave message when a broker joins/leaves the group, so that every member knows who is the newcomer/leaver. And every member in the group exchanges its status information periodically. Each of them will ask other broker for help when its load exceeds some predefined threshold. The broker that receives a request for help can either return the address of a server under its domain, or simply send back a rejection message. It is also possible that the broker forwards the request to other brokers, but in that case a more complicated protocol is needed to prevent endless loops.

Depending on how the threshold is set, the server selection method can be classified as "static global least utilized algorithm" (using a predefined threshold) and "dynamic global least utilized algorithm" (using the overall average of the utilization of all the clusters as the threshold). Simulation experiments show that the multi-broker system significantly improves the load balance of all the clusters. And the dynamic algorithm has

better performance than the static one.

The multi-broker architecture makes the system scale very well. A single broker can balance the load under its domain, and several brokers can cooperate and share the load among several domains. In this way, the geographic barrier to the resource allocation is broken and all the sites work as a whole.

2.2.4 Something to be improved

In the brokerage approach, the broker can guarantee the QoS provided to the client by putting an upper limit to the response time [1]. Whenever the current response time exceeds some pre-defined upper limit, no user can be admitted to the system. In this way, we can control the maximum response time the server will provide. And also if we set different upper limits to different user groups, we can provide a differentiated QoS to different classes of users. But there is still a weakness in this approach: because the server can only report its current response time to the broker periodically (say every minute), for each time interval the broker will either admit all the users (if the threshold is not reached) or reject all the requests (if the threshold is exceeded), thus introducing an oscillation of the number of users and response time in the system. We will discuss such oscillation in detail in Chapter 4.

Despite of the disadvantage of the performance oscillation, the brokerage system is still very flexible, since it does not need support from the DNS. It is more like, yet simpler than, the anycasting approaches. But in the brokerage system, we only need one delegate server to act as a broker; there is no need of the anycast resolver and probing agent. And it is worth noticing that the brokerage architecture can be used not only on web servers, but also in any distributed application, any service provided on the web. So in our project,

we decided to use the brokerage system as the basic architecture to inherit all these advantages and improve its scheduling algorithm to avoid oscillations.

3. Simulation principles and tools

In our project, the simulation is the basic tool we use to study the performance of the server selection algorithm. So we feel obliged to make some introduction to the basic simulation principles and simulation tools we used in the project in this chapter, which may help the readers to understand our work much better.

3.1 Simulation principles

The study of simulation is to build a (simulation) model, which is executed to imitate the operation of a real-world process or system over time in order to solve the real-world problems [7]. It has long been considered as an important methodology in the field of industrial, management, and research.

Simulation helps us to solve the "what if" question in an efficient and economical manner, allowing us to speed up or slow down the process for a thorough checkup and diagnosis. It also makes it easy to make changes or corrections to explore all the different possibilities, which would be extremely expensive to realize in the real system. There are quite a few application areas of the simulation, including simulation of manufacturing and material handling systems, simulation of automobile industry and transportation systems, simulation of healthcare and service systems, and even the simulation in the military field. In all these fields, simulation is an indispensable tool to find all kinds of answers for the real world.

Simulation generally consists of three phases, namely the design of a model, model execution, and the analysis of the data obtained from the execution. In the phase of model design, we have to define a concept model according to the knowledge of the real system.

Then we should consider the model execution where some mathematic languages are chosen to express the concept model. Also in this phase we should consider a proper simulation toolkits. There are many simulation toolkits, and we should decide the one most suitable for our purpose. After the model execution, some results may come up, and then we should start the final phase - the analysis. During the analysis phase, the data of simulation results are put together, maybe in some visualized way; and statistical analysis are made, which allows us to better understand the nature of the system and make further inferences. At this stage, the verification (process to make sure that the concept/mathematic model corresponds precisely to the real system) and validation (the process to check whether the output of our concept model is exactly what we have expected in the real system) can also be made. The results are taken as feedback to further improve or correct the concept model. In this chapter, we will introduce some basic notions of the simulation principles and the simulation tools we used in our brokerage service project.

3.1.1 Modeling principles

In the simulation world, one of the most important concepts is the modeling. A model is actually a representation of the real-world system. Designing a model is more like an art than a technology because there can be many ways we can abstract the conceptual model from a real system. There is simply no best model, therefore any model complex enough to represent all the details of the system that are necessary for the problem under investigation is a good model. Models showing too many details, or not including factors that will alter the simulation results are not good ones.

Depending on the nature of the occurrence of the simulation event, the simulation models can be classified into three categories: namely discrete model, continuous model, and the

combined model [7]. A discrete model is a model with dependent variables that change only at distinct points in simulated time (so called event times). A continuous simulation model has dependent variables that change continuously over time (usually they can be represented by some forms of differential equations). A combined model simply consists of dependent variables that may change discretely or continuously. The discrete model is good at modeling the system where the state of the system changes at discrete point of time, like the problem of resource management, queuing, and any problems that can be modeled by a finite state automata (FSA). The continuous model is usually used in modeling the problems, having variables that can be defined by some mathematic equation and changes continuously over time, like physical experiment, laws of nature etc.

According to the nature of the problem to be modeled, and from the real life experience, we can roughly conclude some of the most often used typical models as follows: conceptual models, discrete event models, functional models, constraint models, spatial models, and multimodels etc. [8] We introduce them briefly here:

Conceptual models: models containing components that cannot be clearly identified in terms of system-theoretic categories such as states, events, and functions are called conceptual models. A conceptual model is very abstract and vague and considered to be a very high-level system model; it will normally progress to some more detailed system-theoretic models.

Discrete event models: a declarative model contains two primary components: states and events. It is especially suitable for mimicking the behavior of the real system whose action is considered to be the transition from one state to another.

Functional models: a functional model contains two primary components: functions and variables. The function works on some input variables and produces some output, which may be used as an input for another function. It can be used in the situation where the problem can be defined as a series of functions, like the law of physics.

Constraint models: a constraint model is similar to the functional model, but it focuses more on the balance and causality of the variables in the system. They are usually defined in terms of some equations and are very powerful to represent laws of nature.

Spatial models: a spatial model deals with the decomposition of space, with clear boundaries, and is useful to fragment the whole system into small pieces, and model each of them in the divide-and-conquer way.

Multimodels: multimodels are composed of several models listed above. Real-world systems are usually too complex to be portrayed as a single simple model.

To design the simulation model, we start with analyzing the concept model of the real system, and break the whole system into a number of smaller abstract modules depending on its functionality. Finally, we choose a proper model to represent each of these modules. Since no rules can be followed to choose a model, sometimes we have to use some heuristic approaches to make decisions.

3.1.2 Procedure of simulation

Generally speaking, there are some commonly followed steps in the simulation study (as stated in [7]). The flow of these steps is shown in the following figures and the brief

explanation of each step is listed as below.

Problem formulation: the very first step in the simulation study, which provides a precise statement of the real problem.

Setting of objectives and overall project plan: the preparation of the proposal, which states the goal, schedule, cost etc. of the simulation work.

Model conceptualization: defines an abstracts conceptual model and mathematical relationship of the components of the real system.

Data collection: the real system data that is required by the simulation model is collected in this step.

Model translation: to translate from the conceptual model to the operational model simulated on the computer.

Verified?: to determine whether the operational model built in the previous step performs properly.

Validated?: the comparison of the conceptual model and the real system is made to see if the conceptual model is the accurate representation of the real system.

Experimental design: to design for each scenario the number of runs, the run length, the initial parameters of the run etc.

Production runs and analysis: to estimate the performance of the scenarios

More runs?: analysis from the previous production runs, to see if more runs are needed.

Documentation and reporting: adequate documentation and reporting is clearly necessary for the simulation model reuse, and modification.

Implementation: the documentation produced in the previous step help people to make implementation decisions for the real system.

These steps are usually followed in every simulation project. Sometimes some steps might not seem very necessary in a specific project, but following these procedures is definitely helpful and makes your simulation model less error prone, especially in a large project.

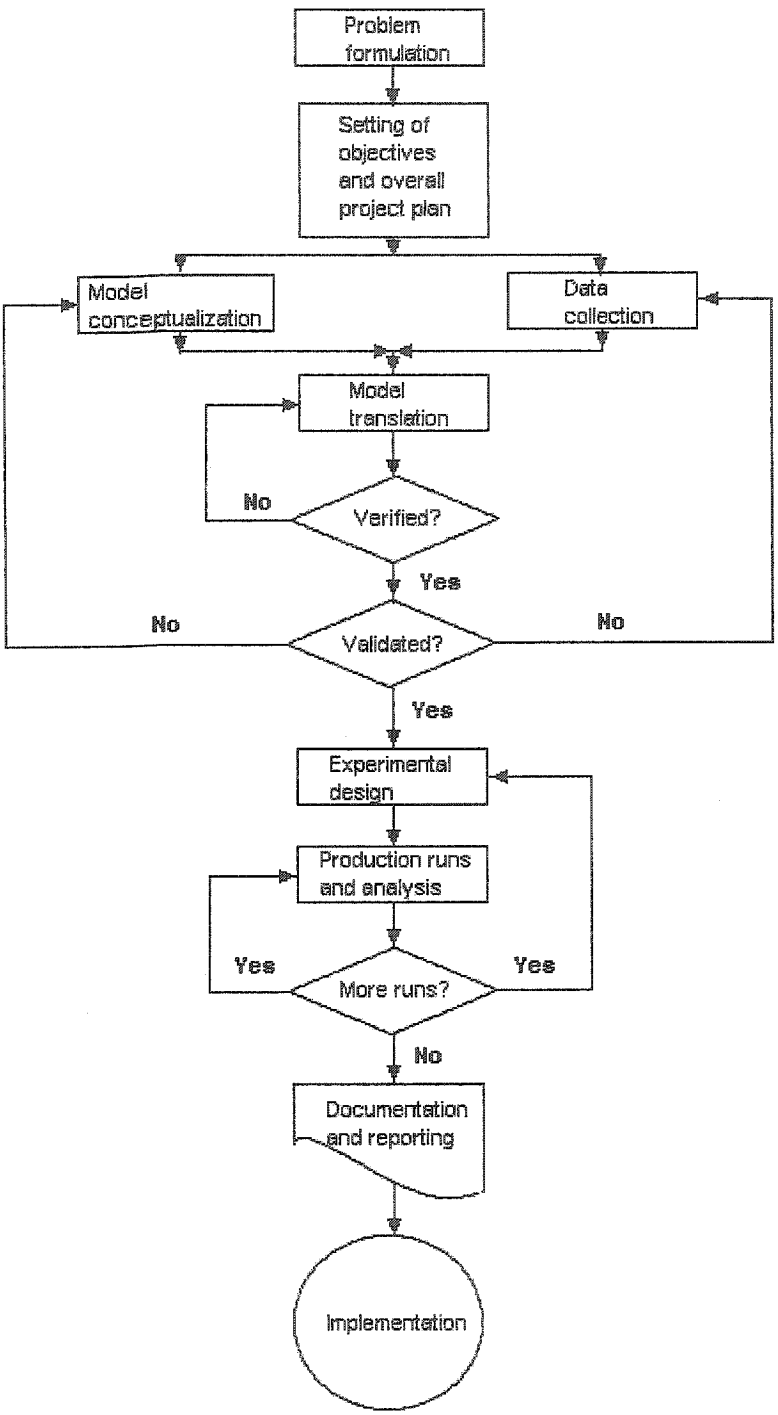


Figure 4. Procedure in simulation study (from [7])

3.2 Simulation tools

There are too many simulation tools available, we should choose the one that is most appropriate to serve our purpose. In our project, we used the CSIM18 simulation package to do the job because of its convenience to use (written in C++) and fast execution speed. Here we present some basic knowledge needed to understand this simulation tool.

3.2.1 Introduction to CSIM18

In our project, we used the simulation engine CSIM18 to do all the experimental tests. The CSIM18 simulation engine is developed by Mesquite Software, Inc. Austin, Texas. And it is a kind of general-purpose model-building simulation toolkit, which enables developers to build up process-oriented, discrete-event simulation models. The model can be any real life model from a simple queuing system to an atomic bombing experiment. All kinds of details like the interrelationships of components, scheduling rules and message exchanges can be represented in the model. After we create a computer simulation program, which accurately realizes the simulation system model, the CSIM18 engine can easily collect all the statistical data that is necessary for the analysis.

The CSIM18 simulation engine is very compact and efficient, and can be embedded into any code written in C/C++, so the users do not have to learn a particular programming language for CSIM18. Like C++, the simulation engine itself is object oriented, thus it provides a convenient and easy-to-use interfaces. It provides a library of classes, methods, and functions, which enable us to implement general simulation models. By inheriting the base class of the simulation engine, the user can easily modify and extend the behavior of the basic models to simplify the realization of more complicated systems.

Furthermore, the CSIM18 simulation engine is a multi-platform library. According to what is claimed by Mesquite [6], this simulation package has versions that are compatible with operating systems such as Windows 3.1, Windows 95, Windows NT, OS/2 Warp and Linux. It also has versions on almost all UNIX workstations, including Sun SPARC (SunOS and Solaris), DEC Alpha (with OSF/1), HP PA (with HP/UX), IBM RS/6000 (with AIX), SGI workstations and Power Mac (with the Metroworks C++ compiler). It is really convenient to transfer the simulation system from one platform to another. We do not have to do any change in the code, recompiling the original code is enough.

3.2.2 Simulation components (classes) in CSIM18

There are a number of simulation components (classes) provided by CSIM18. We now briefly introduce the most important ones:

Processes: A CSIM process is an independent thread (lightweight process), which can mimic certain activities of an entity; several processes can appear to be executing simultaneously, although they are actually executing sequentially on the processor. Just like a real process, a CSIM process can be in the states of ready, active, holding (allowing simulation time to pass), and waiting (for some event to happen). Their transitions are controlled by certain methods provided by the process class. Process has a priority for execution; different processes may have different priorities.

Facilities: A CSIM facility is a resource that is typically "used" by processes in the model; Usually a facility consists of a server and a queue used for the processes waiting to be served by the server. A multi-server facility has a single queue for several servers. During the time of heavy load, the processes are queued up for access to a server. Processes with

higher priorities are queued ahead of the process with lower priority.

Storages: A storage is a resource that can be allocated to the processes. It consists of a counter (amount of storage) and a queue used for queuing the processes waiting for storages. Storages can be set to be synchronous, which means several of them can be allocated in the same clock cycle. When the storage unit is insufficient to allocate to any process, the process will simply wait in line until other processes release the storage unit that is previously allocated.

Events: An event is used to synchronize the behavior of different processes, and it has two states: occurred or not occurred. A process can be suspended when waiting for a not-occurred event and it also can be resumed when that event occurs. The state of an event can be and usually is set by some other processes.

Mailboxes: A mailbox is used to exchange information between processes. Any process can send a message to or receive a message from a mailbox. A mailbox maintains two FIFO queue, one for incoming messages, and the other for waiting processes. When a message arrives and there is no process waiting for it, the message will go to the message queue waiting to be picked up. On the other hand, if a process execute a receive action while there is no message in the message queue or the mailbox is empty, the process will wait until there is some message coming in.

Tables: A table is an object that is used to collect individual data values and to report its statistical properties generated from that table. The properties of the report include mean, variance (and standard deviation), standard deviation, coefficient of variation, minimum, maximum, and the number of observations, etc. The report also support features like histogram, which reports the relative frequency of specified ranges of values, confidence

intervals with which we can estimate the accuracy of some values collected, and moving window (which determine the sample size) etc.

Qtables: A Qtable is pretty much the same as a table described above, except that it is used solely to collect integer values (e.g. number of clients, queue lengths) and to report their statistical properties.

Meters: A Meter is used to measure the flow rate of entities passing a certain point in the system module and to keep track of the times between successive passages.

Boxes: A Box is used to collect data of time spent in a specified entity, and the number of processes inside the box.

With these basic classes, the simulation modeling and the result data collection become an easy job. Users only need to focus their attention on the model itself rather than many tedious details.

3.2.3 The accuracy of the simulation in CSIM18

The CSIM18 simulation engine has the facilities to reach a pre-defined accuracy of some estimation. No one can run a simulation model for a indefinitely long period of time. Sometimes the expected accuracy of some value can never be reached. In other words, the "true value" of some estimation will never be known in a given period of time. That is why we need a way in our simulation engine to tell us whether a given accuracy of the simulation result can be achieved. If yes, how long will it take before such accuracy can be achieved? Fortunately, CSIM18 provides such techniques as confidence intervals and run-length control, which allow us to cope with these difficulties.

In short, a confidence interval is a range of values in which an estimated value is believed to fall with a high probability. That range is usually considered to be the "best guess" of true value. Here we show a typical report for confidence interval.

results of run length control using confidence intervals

cpu time limit	**606.0**	**accuracy requested**	**5.000000**
cpu time used	**606.6**	**accuracy achieved**	**5.000000**

95.0% confidence interval: 642.699255 +/- 4.404560 = [638.294695, 647.103815]

The above report shows that we have 95% confidence that the collected data values fall into the range of [638.294695, 647.103815]. It is worthwhile to mention that the method of batch is used to compute confidence intervals. And by default, CSIM18 simulation engine provides us confidence levels of 90%, 95% or 98% respectively.

Next, we need to determine how long our simulation model should run. CSIM18 provides run-length control, which can determine when the level of confidence has been reached. With run-length control, the simulation program will keep running until a specified accuracy is achieved, or until a predefined simulation time limit has elapsed. That is to say, in some circumstances, the execution can be ceased, but the confidence level is not yet achieved. The final report will show whether the termination of the execution is due to the simulation model being converged to some level of accuracy or simply the maximum CPU time is exceeded. This function enables us to execute our simulation program in an efficient way, and within a reasonable amount of time and computational cost.

3.2.4 Random number generation

In any simulation program, random number generation is an important part. A random number generator should produce random number series (called stream in CSIM18) without any recognizable pattern. Unfortunately, there is no true random number generator up to now. Most generators today only provide pseudo-random number because they produce a series of random numbers in which the number values are calculated from the previous numbers. The very first random number depends on a so-called seed. Different seeds can produce different series of random numbers.

In our simulation model, we use the random number generator to produces values such as inter-arrival time of customers, the number of files each customer hopes to download, the number of objects in each file, and the size of the object. They are all random numbers following some kinds of distributions. We will describe them shortly. These distributions describing the user behavior are carefully studied and explained by Paul Barford [5]. Here we simply use his research results to build up the model of our own. The CSIM18 simulation library provides both continuous (real) and discrete (integer) random numbers series from up to 18 distributions, including uniform, beta, exponential, gamma, erlang, weibull, normal, cauchy, poisson, geometric, and binomial etc. These functions make the simulation tool really handy in designing random aspect of the simulation models. The change of seeds can be realized by the "reseed" function, which gives us a different series of random number still following the same distribution. Reseed enables us to find more stable and accurate result independent of any particular sequence of random numbers, which makes our simulation results more convincing.

We show a piece of code of random number generation in the following. It generates two random numbers 10000 (NUM_SAMPLES) times; one follows an exponential

distribution with the mean of 1.0 (MEAN), while the other follows a uniform distribution within the range [0.1, 10000.0] ([UNIF_LOW, UNIF_HIGH]). It also records them in the tables exp_distribution and unif_distribution respectively.

```
#define NUM_SAMPLES 10000
#define MEAN 1.0
#define UNIF_LOW 0.1
#define UNIF_HIGH 10000.0
...
table *exp_distribution;
table *unif_distribution;
i = 0;
while(++i < NUM_SAMPLES)  {
      exp_distribution->record(exponential(MEAN));
      unif_distribution->record(uniform(UNIF_LOW, UNIF_HIGH));
}
```

3.2.5 A simple example of using the simulation engine CSIM18

To see how this tool works, we show in the following an example of CSIM18 simulation engine used in a queuing system.

This program simulates a queuing system with only one server (facility). There will be 5000 customers coming in and waiting to be served. The inter-arrival time of the customer follows an exponential distribution with the mean of 2 (IAR_TM) seconds, and the length of service time also follows an exponential distribution with the mean of 1 (SRV_TM) second. The function customer() mimics the behavior of a customer, coming in, being served and leaving. At the same time, the variable tbl (of type table) records how long this customer stays in the system or the customer's response time (the time from when he enters the system to when he leaves), the variable f (of type facility) records how much time it takes for the server to serve the customers, and qtbl (of type

qhistogram, almost the same as a qtable) counts the number of the customers in the system.

```
// C++/CSIM Model of M/M/1 queue
#include "cpp.h"                // class definitions

#define NARS 5000
#define IAR_TM 2.0
#define SRV_TM 1.0

event done("done");                     // the event named done
facility f("facility");                 // the facility named f
table tbl("response time");                     // table of response time
qhistogram qtbl("number in system", 10l);       // qhistogram of number in system
int cnt;                        // count of remaining processes

void customer();

extern "C" void sim(int, char **);

void sim(int argc, char *argv[])
{
        set_model_name("M/M/1 Queue");
        create("sim");
        cnt = NARS;
        for(int i = 1; i <= NARS; i++) {
                hold(expntl(IAR_TM));       // interarrival interval
                customer();         // generate next customer
                }
        done.wait();                    // wait for last customer to depart
        report();               // model report
        mdlstat();              // model statistics
}

void customer()                         // arriving customer
{
        double t1;
```

```
    create("cust");
    t1 = clock;              // record start time
    qtbl.note_entry();       // note arrival
    f.reserve();             // reserve facility
        hold(expntl(SRV_TM));    // service interval
    f.release();             // release facility
    tbl.record(clock - t1);  // record response time
    qtbl.note_exit();        // note departure
    if(--cnt == 0)
        done.set();          // if last customer, set done
}
```

After we run the simulation program, we can get the following output. It shows statistics of the facility summary of the server (like utilizations, response time etc), a table of response time, and a table of the customer number in the system (or the queue length because the customers are served in the FCFS order) in the form of a histogram.

FACILITY SUMMARY

facility name	service disc	service time	util.	through-put	queue length	response time	compl count
facility	fcfs	0.99206	0.494	0.49793	0.99059	1.98943	5000

TABLE 1: response time

minimum	0.000145	mean	1.989433
maximum	14.273079	variance	3.813342
range	14.272934	standard deviation	1.952778
observations	5000	coefficient of var	0.981575

QTABLE 1: number in system

initial	0	minimum	0	mean	0.990590
final	0	maximum	13	variance	1.937727
entries	5000	range	13	standard deviation	1.392022
exits	5000			coeff of variation	1.405246

	number	total time	proportion	cumulative proportion	
	0	5081.38161	0.506030	0.506030	**********
	1	2426.95194	0.241688	0.747718	**********
	2	1238.22169	0.123308	0.871027	*****
	3	667.95025	0.066518	0.937545	***
	4	350.00001	0.034855	0.972399	*
	5	152.62571	0.015199	0.987599	*
	6	69.33696	0.006905	0.994504	.
	7	25.09331	0.002499	0.997003	.
	8	9.84005	0.000980	0.997982	.
	9	10.69388	0.001065	0.999047	.
>=	10	9.56521	0.000953	1.000000	.

3.3 Simulation model for the brokerage architecture

As part of our simulation model, a realistic web workload needs to be created (for example, a stream of HTTP requests that the real web server users generate), and it is used to evaluate the performance of our brokerage system. Web workload simulation became a topic under research years ago. Basically, there are two ways of generating a typical web workload, namely the trace-based approach and the analytic approach [5]. The trace-based approach takes the workload as a black box. It simply mimics the workload by replaying the recorded past workload. Although it is very easy to be realized by simulation tools, it hardly reveals any insight into the system behavior. The analytic approach uses the mathematical models to simulate different characteristics of the workload. But the challenge of this approach lies in the difficulty of combining a large number of mathematical characteristics into a single stream of HTTP request. Paul Barford and Mark Crovella from Boston University have done a lot of work in this field, and they even built up a simulation tool SURGE (Scalable URL Reference Generator) for workload generation [5], which has both of the following two major characteristics, user

equivalents and certain model distribution, as explained in the following.

User Equivalents: The workload generated by the generator should roughly correspond to the workload of some known number of users. SURGE realizes this by creating a set of processes; each mimics one user by endlessly alternating between web page requests and user think time. Each web page request consists of the transmission of multiple file requests (web objects), as shown in the following chart. OFF stands for the idle time when there is no message transmitted on line. Active OFF is the time between the transmissions of two objects, while the inactive OFF is the duration between two web page requests (called "think time" in our model). The web page requests, the length of idle time and object size must follow certain distributions and exhibit properties of the real web users. [5]

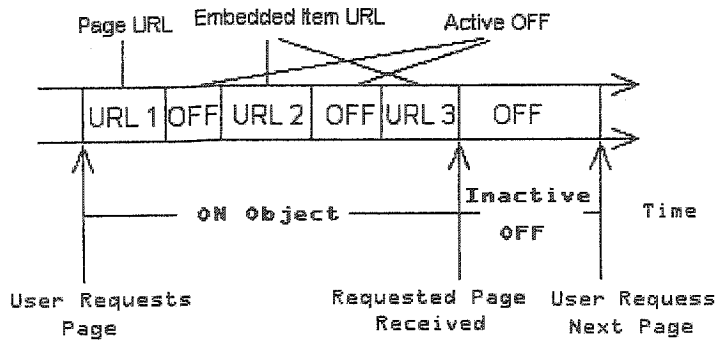


Figure 5. ON/OFF model used in SURGE (from [5])

Distribution Models: In the study of the workload distribution model, they mainly focused on the discussion of the distribution of several major workload characteristics: file sizes, request sizes, popularity, embedded references, temporal locality, and OFF times. These properties have been proved to be ubiquitous and comply with empirical measurements. More and more researches on network traffic nowadays are based on these models, and they are surely becoming more and more popular. The mathematical

rationale behind these distributions is discussed in several papers [15-20], and is beyond the scope of this thesis, but we do use these results to build our simulation models.

As explained before, to make the matter simpler, our simulation is done on just one server, and one broker, which controls the admission of new users. We keep generating new clients with certain time interval (called inter-arrival time). To be more precise, the inter-arrival time follows an exponential distribution with a certain mean. So the arrival of the incoming user is a Poisson process. Each client will launch on average a certain number web page requests. However, before they start the first web page request, they ask the broker for admission permission, and the server id. After it is accepted, the client goes fetching the web pages he wants just as required by HTTP 1.0 (the result can be easily extended to pipelining model of HTTP 1.1 according to [5]). Within each web page, there are a certain number of embedded objects (such as images, wave files, text file etc), which also follows some distribution as illustrated in the following table. After the client gets the web page and processes them, he starts the think time before he fetches the next one. The response time is calculated based on each fetched object, and the mean of the response time during the whole inter-observation time is sent periodically to the broker for the purpose of admission control. Needless to say, the network traffic based on such simulation model will exhibit a self-similar behavior [19]; actually all the network traffic characteristics can be traced back to the particular distribution of the files on the server and the particular behavior of the clients etc. But in our project, the self-similarity of the network traffic is beyond our research scope. We focus the study on how to avoid the oscillations rather than the nature of the oscillation. So in the following discussion, the self-similarity nature will not be discussed.

Notice that the way we build our model and the parameters we use are largely based on Paul Barford's work [5]. Here we list some of the important parameters in the following:

Parameter	**Description**

Server speed	10^{-6} second/byte
Inter-arrival time of the client	exponential distribution with a certain mean
Total number of pages each client requests	exponential distribution (mean = 36)
Number of embedded objects per page	Pareto distribution (α = 2.43, k=2.3)
Object size (in octets)	Bounded Pareto distribution (α = 1.25, k = 1800, p =10^8)
Object processing time (in seconds)	Weibull distribution (α = 0.146, β=0.382)
User think time (in seconds)	Pareto distribution (α = 1.5, k=3)

4. The problem of performance oscillations

4.1 Introduction to the oscillation problem

As we have mentioned in the previous chapter, the abrupt user rejection at the upper threshold, and the manner of periodical data collection by the broker contribute to the oscillation of the server performance.

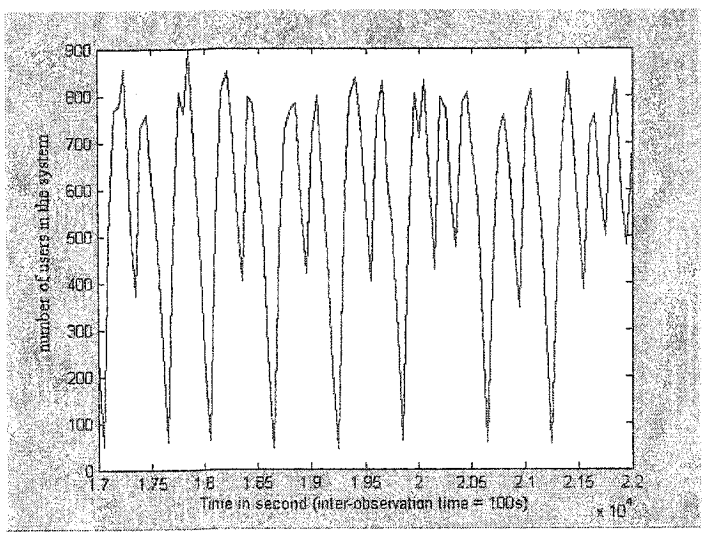
At one point of time, there are very few users in the system, and the average response time of the server is low (below the threshold), so the server accept all the new users; in the next inter-observation time period, many users will enter the system; if the user incoming rate is high enough, soon the server will be overloaded in the next inter-observation time interval, making the average response time exceed the threshold, and the system stop accepting new users; the number of users in the system keeps dropping; sooner or later, the average response time will drop below the threshold again and a new round of oscillation starts again. Accompanied with the oscillation of the number of users and the response time is the oscillation of server utilization. As we will see in the next section, the utilization of the server oscillates as time goes by.

On the server side, the broker can observe and control the response time, the number of users, and the utilization, etc. To the broker, the fluctuation of these performance data is surely an undesirable situation. Fluctuation of the utilization signals a waste of CPU time (especially for the one with longer inter-observation time), while fluctuation of response time implies the suffering of end users. In one way or the other, the broker has to rule out such oscillations so as to provide a stable quality of service.

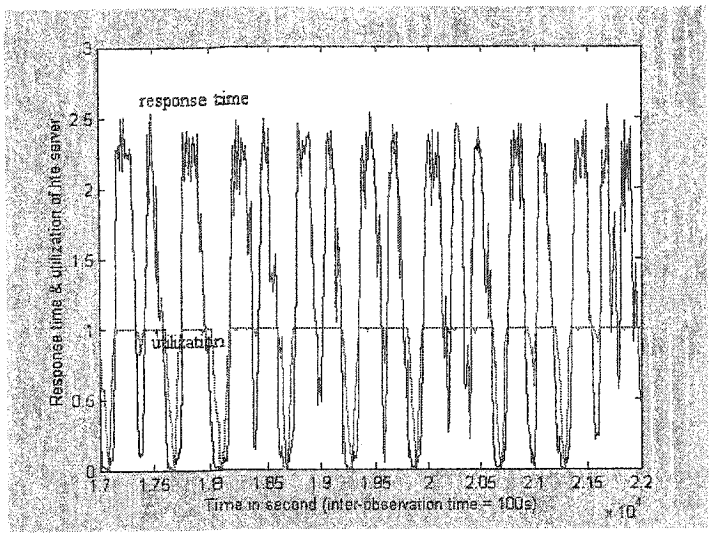
In the next few sections, we will present the simulation result of the oscillations. And then, the theoretical model of the oscillation is studied. With this theoretical model, people can better understand this oscillation problem, thus can avoid it.

4.2 Simulation result of the oscillation

Here we show two groups of results from our simulation experiment to illustrate the problem of oscillation. They present not only the oscillation of the number of users in the system but also the response time and utilization of the server as a function of time. In this experiment, our simulation model keeps the average customer inter-arrival time at 0.1 second (actually it follows an exponential distribution with the mean of 0.1 second). We use only one server with the speed 10^{-6} Bytes/second, and with other settings being the same as shown in the Chapter 3.3. Here we chose threshold to be 1.3s, which means when the server response time is below 1.3 second, we accept all user requests; otherwise we reject all of them. The Figure 6 shows the oscillation when the inter-observation time equals 100 seconds, while the Figure 7 shows the similar results for the case that the inter-observation time equals 40 seconds

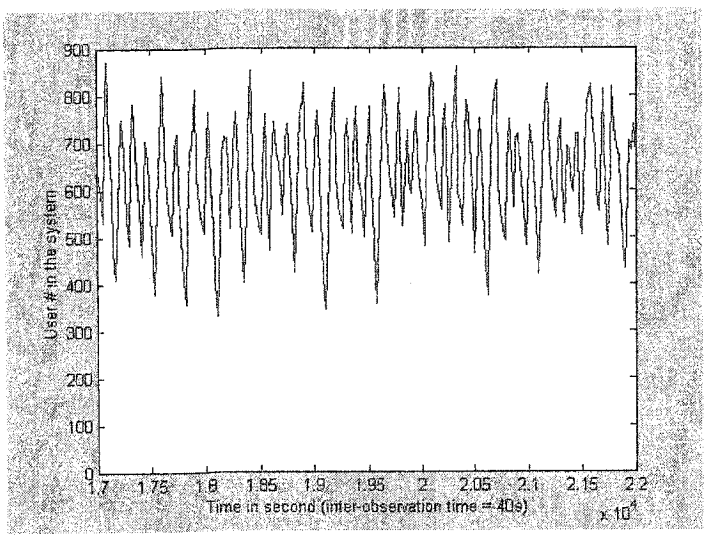


(a)

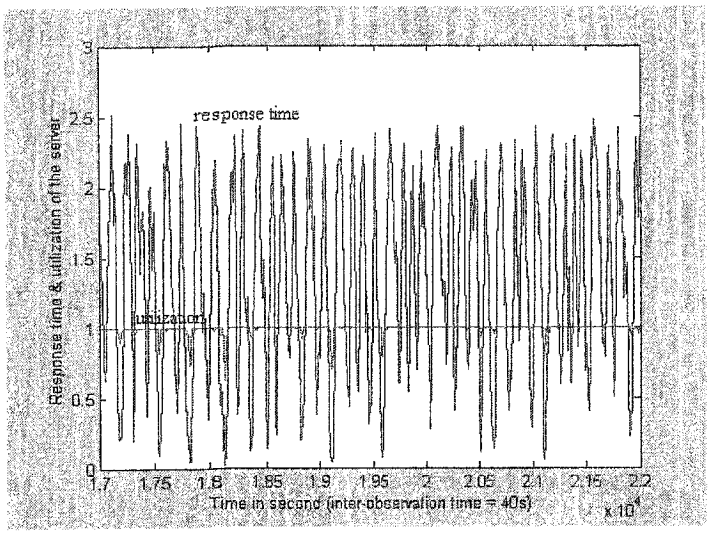


(b)

Figure 6. The oscillation of the number of users and response time when inter-observation time equals 100 seconds



(a)



(b)

Figure 7. The oscillation of the number of users and response time when inter-observation time equals 40 seconds

From the results of these figures, we can clearly observe the oscillation of the number of users, the response time, and the server utilization of both cases. Notice the difference in the amplitude of the oscillation: for 100s inter-observation time, the oscillation is exacerbated (with large amplitude and long period). We will discuss the effect of the inter-observation time on the oscillation in the following section.

What worth noticing is that, for the clients, the only quality of service of the website they are aware of is the response time. But in our later chapters, we will focus mainly on the study of the oscillation of the number of users in the system. There are several reasons that we use the number of users rather than the response time to study the oscillation of the system performance. First of all, the number of users in the system has a strong indication to the response time the system is experiencing as will be illustrated in the Section 4.4. Secondly, when the value of the response time is varying between 0.1s and 3s, the value of the number of users is changing between 100 and 800. Any variation in the response time looks just like the statistical noise due to their small value, but the oscillation of the number of users is very clear and easy to identify. What is more, it is

more convenient to model the number of user in the system rather than model the response time. So in the later chapters, we will study mainly the nature of the oscillation of the number of users and the ways to avoid it. The oscillation of response time and utilization are very similar.

4.3 System control

In this section, we will give a brief introduction to system control (see for instance [22]), because admission control is, in general, a system control. In fact, the probabilistic control proposed in this thesis is very similar to ideas that have been used for system control in other fields of applications. But so far as we know, this is the first time for it to be used on web service admission control.

System control is not something new; it is an extremely important and integral part of modern manufacturing and industrial processes. It is widely used for various automatic controllers. For example, system control is essential in the operations of controlling pressure, temperature, humidity, viscosity, and flow of process industries, and it also plays a vital role in missile-guidance systems, space-vehicle systems etc. Nowadays, the theory of system control is well understood by many engineers and scientists. In practice they use it to attain optimal performance of dynamic systems and improve productivity.

In Figure 8, we show a block diagram of a standard industrial control system. The part in the dashed-line box is the automatic controller. It detects the actuating error signal e at very low power level and amplifies it using amplifier. The amplified signal u(t) is fed to the actuator (could be a valve or electric motor etc.), which produces the input to the

plant. The output of the plant is measured by the sensor, which changes the output into a suitable variable comparable to the reference input signal. These components form a closed-loop system that is widely used in industry.

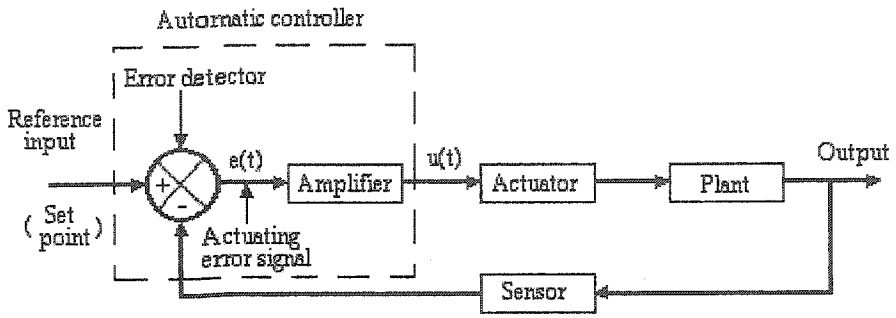


Figure 8. Industrial control system (inspired by [22])

As an example, we consider a liquid level control system, as shown in the Figure 9. In this system, the flow of liquid is controlled by a valve. The input signal to the valve is an electronic current u(t) (determined by the controller), which is converted into a pressure applied on the valve and changes its stem position. The stem position of the valve controls the amount of flow $q_i(t)$ that goes into the tank. The height of the liquid in the tank h(t) is measured in some way (we could use pressure of the liquid instead of measuring the height directly, because *height = pressure / liquid density / g*) and is fed back to the controller. The outflow of the tank is $q_o(t)$. $q_o(t)$ is a function that depends on the liquid height. To be more exact $q_o(t) = h(t)/R$, where R represents the pipe restrictance. The goal of this control system is to maintain the height/level of the liquid at a constant value. The actuating error e(t) is the difference between the actually liquid level and this expected level.

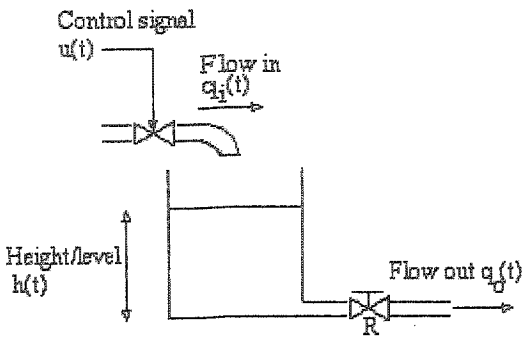


Figure 9. Liquid level control system (inspired by [22] and [23])

Roughly speaking, we can classify the industrial controllers into six categories according to their control actions, namely three basic types of controllers: two-position (or on-off) controllers, proportional controllers, integral controllers; and three controllers with combined actions: proportional-plus-integral controllers, proportional-plus-derivative controllers and proportional-plus-integral-plus-derivative controllers ([22]). Here we will make a brief introduction to the basic controllers.

Two-position or on-off controllers

In this type of controller system, the actuator has only two fixed positions. The output of the controller u(t) is either a maximum or a minimum value, depending on the reference input e(t). Such a controller is simple and inexpensive. Take the liquid level control system as an example: since the output of the controller u(t) is either a maximum or a minimum value, the $q_i(t)$ will also switch between its maximum or a minimum value, depending on whether e(t) is positive (meaning the level is below the threshold), or negative (meaning the level is above the threshold).

Proportional controllers

For a controller with proportional control action, the output of the controller u(t) is proportional to the actuating error e(t). That is $u(t) = K_p * e(t)$, while K_p is termed the proportional gain. In this case, the difference of the actual liquid level and the expected

level is amplified and directly fed to the valve.

Integral controllers

In this controller system, the output of the controller u(t) is changed at a rate proportional to the actuating error e(t). That is $du(t)/dt = K_i * e(t)$, where K_i is an adjustable constant. In this case, when the value of e(t) is double, the value of u(t) will change twice as fast.

To study control systems we must model the dynamic system and analyze its dynamic characteristics. Generally speaking, a mathematical model of a dynamic system is defined as a set of differential equations that represents the dynamics of the system. Those equations can be obtained by using physical laws governing the system, like Newton's law for mechanical system or Kirchhoff's law for electrical systems. Because, people may have different perspective on the system, the mathematical model they use may not be unique. Sometimes we may not find the absolutely correct mathematical model, but we want it to be as accurate as possible. After analyzing these equations, we can get a better understanding of the system behavior and thus optimize the control.

Take the liquid level control system as an example again. The interesting question is how we can keep the liquid height at some constant level h_m. The simple on-off approach is to set up two thresholds h_l and h_h ($h_m = h_l + \varepsilon = h_h - \varepsilon$ for some small value ε) and check the height regularly according to some inter-observation period. If the height is below the level h_l, then the controller changes $q_i(t)$ to its maximum possible value and fill the tank to the level h_h and then stops the inflow ($q_i(t) = 0$). In that case, with the control signal oscillating between on and off, $q_i(t)$ is forever oscillating between its maximum value and 0. Therefore the liquid height in the tank is also oscillating in a differential gap between h_l and h_h. Actually the curve of the height follows one of two exponential curves, one corresponding to the filling curve and the other to the emptying curve as show in the

Figure 10. This approach is referred as two-position or on-off control mentioned above. And it is the exact correspondence to the on-off approach used for admission control of the web servers.

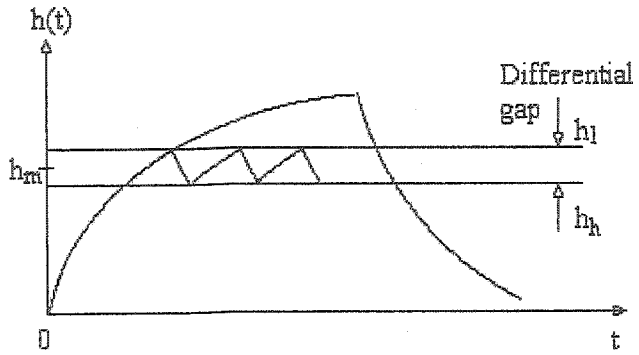


Figure 10. Liquid level oscillating with on-off control (inspired by [22] and [23])

Is there any way we could stop this oscillation? Let us model the system using physical principles. The equation governing the change in the liquid volume is

rate of change of volume of liquid = inflow -outflow

That is

$d(h(t)*A)/dt = q_i(t) - q_o(t) \quad \Rightarrow \quad A * dh(t)/dt = q_i(t) - h(t)/R$

A is a constant cross-sectional area of the tank. The above equation has one first-order derivative, dh/dt; so this system is modeled by a first-order differential equation.

Solving this equation, with the initial condition that if $t = 0$, $h(t) = h_m$, and taking $q_i(t)$ as some constant value, we can get h(t) as a function of t and $q_i(t)$.

$h(t) = q_i(t)*R + (h_m - q_i(t)*R) * e^{-t/RA}$

If we want to keep the liquid height at constant value h_m, when the whole system is in a stable state, which implies $t \to \infty$, we need to balance the following contrain.

$h_m = q_i(t)*R + (h_m - q_i(t)*R) * 0 \qquad (t \to \infty, e^{-t/RA} \to 0)$

That leads to $q_i(t) = h_m/R$. So with the initial state that the liquid level is at h_m, if we keep the inflow at h_m/R, the liquid level will remain constant. This is a so-called stable state that optimizes the system control, and avoids the oscillation of the liquid level between h_l

and h_h which occurs when we use the threshold approach. This approach is referred to as proportional control action ([22]), because the controlled inflow is proportional to the height h_m of the liquid in the tank. It also corresponds to the probabilistic approach that can be used in our server admission control algorithm, as we will see later.

In the field of network technology, we also can find examples of probabilistic approaches in controlling the traffic. A good example is the RED (Random Early Detection) algorithm used for congestion avoidance at the routers ([24]...[27]). The RED algorithm is recommended by IETF for Active Queue Management (rfc 2309), and it uses randomization in choosing which arriving packet to be dropped when the router is heavily loaded. It depends on the current level of congestion (eg. average queue length) in the router to compute the probability of packet dropping and based on this probability, part of the workload can be released, and QoS of the router is insured.

Another example is the p-persistent CSMA (Carrier Sense Multiple Access Protocols) ([28]). In CSMA, we must insure that only one station is sending out the message at one time, otherwise the message will get garbled on the bus. To reduce the chance of the collision, the p-persistent CSMA protocol senses the channel and transmits the message with probability p if the channel is idle. With a probability q = 1- p it defers the sending until the next time slot. This process will go on until the message is finally sent out or other station seizes the channel and starts transmitting. The smaller the p it is, the smaller the chance the collision will occur on the bus. In this way, the quality of service of the data transmission can be guaranteed by choosing an appropriate p depending on the load on the bus.

The example given above is a way of analysis used extensively in many fields of industrial control. Relating the theory of system control with our project, we want to

know whether it is possible that we could control the admission of the users visiting the web server in such a way that on one hand, there is no oscillation in the performance, and on the other, the server is still providing a satisfactory response time to the users that are admitted. To realize this, it is not appropriate to accept all users or reject all users suddenly, but we should be able to accept some percentage of users, so that the number of users accepted would always be exactly what the system can handle. This is exactly the idea of the proportional liquid level control.

We would like to note that we came up with the probabilistic approach used for web server admission control before reviewing the theory of system control, such as described in [22]. Only later, did we realize that similar ideas have been around in system control. And it is true (as far as we know) that this is the first time that such ideas have been used for web server admission control. In that sense, our work is still independent and original.

In the next chapter, we will introduce the probabilistic admission control approach used for the web servers.

4.4. The relationship between the response time and the number of users in the system

The admission decision is based on the observed response time on server's side. The response time is defined as the time period between the reception of a request of an object (a file) at the server and the time instance when the last byte of that object is sent out from the server. Notice that this response time is solely on the server's side, and does not

include the delay experienced in the network. As obvious as it is, the response time of the system should be related to the number of the users in the system. The more users there are, the longer the response time will be. The relationship between the response time and the number of users in the system is a very basic fact that we want to reveal, and use in our later chapters. So we discuss this before we start the study of the oscillation.

To fully study this relationship, we made a little modification in our simulation model, which keeps the number of users in the system at a constant value, and measures the mean response time of the server. We use one process to watch the number of users. It creates a new customer whenever the current number of users is smaller than a predefined value. And we measure the mean response time when the system becomes stable. Notice that we do not reject any of the customers and there is no constant customer-incoming rate either; we add one user whenever one user leaves the server. By doing this, we only want to reveal the relationship between the number of users and the mean response time on the one hand, and the server utilization on the other hand. Our measurements are plotted in Figure 11 and Figure 12:

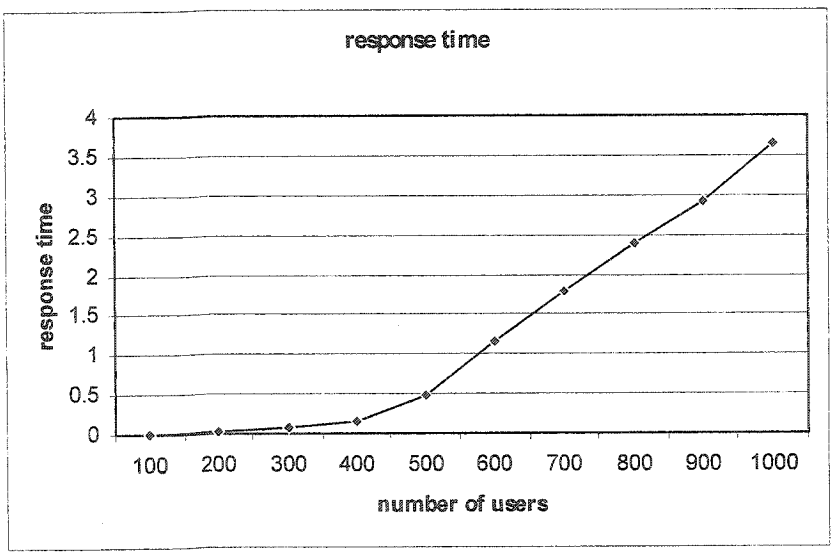


Figure 11. The response time as a function of the number of users in the system

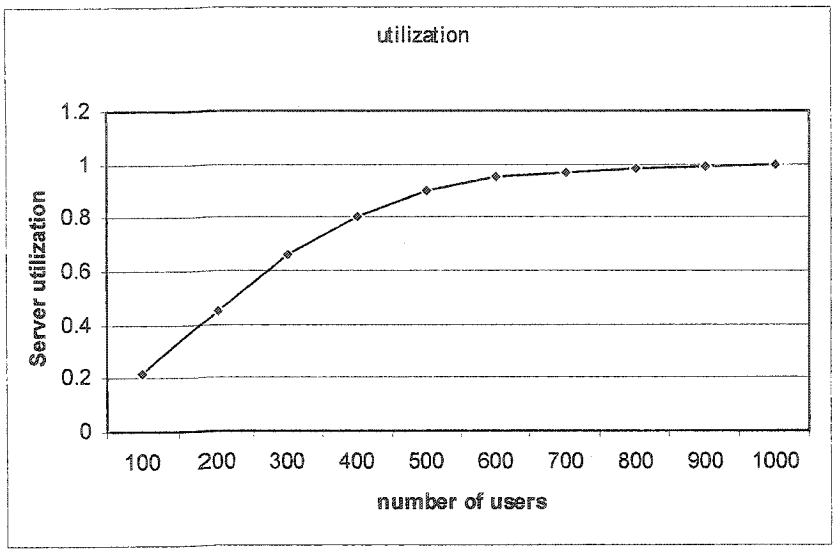


Figure 12. The server utilization as a function of the number of users admitted

From the above figure, we see that in the range between 500 and 1000 users, the response time is a linear function of the number of users in the system. And the slope of the response time in the Figure 11 indicates the ratio between the response time and the number of users in the system (response time / number of users), which corresponds to the average time the server spends on every user, or average service time. In the above figure, it is around (3.7-0.5)/(1000-500) = 0.006 second/user.

We can also combine the above two measurements to get the relationship between the response time and the server utilization. It is shown in the following figure.

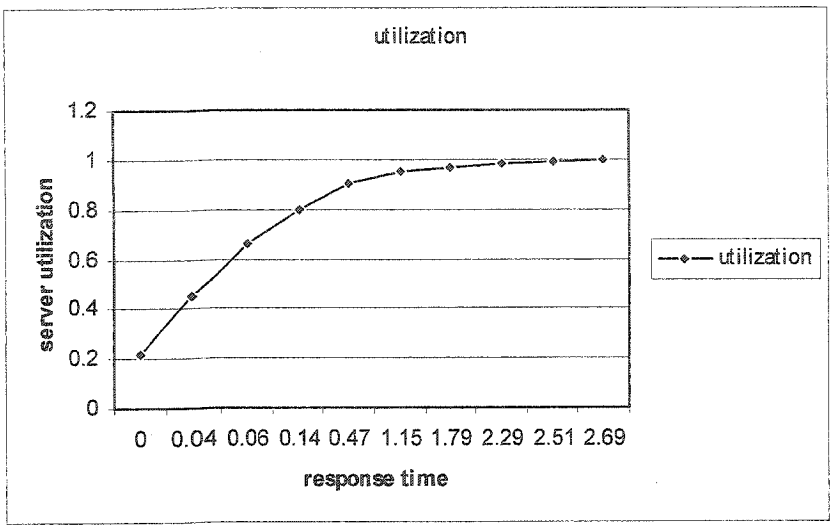


Figure 13. The utilization as a function of response time

The above discussion shows that as the number of users increases in the system, the response time will grow forever without any upper limit, while the server utilization would be 1. If the number of users could grow without any admission control, there is no way we can provide a guaranteed response time to the client. So at some point we have to choose the so-called cut-off point; below that point the server can accept more users while above that, no one would be admitted.

Now comes the question: to provide the satisfactory QoS, where should we choose the cut-off point? We can make our decision from the given knowledge of the response time and server utilizations. It is really very hard to decide solely on the response time; different customers may have different sense of what is a tolerable response time, different kinds of web services have different requirements for a reasonable response time (IP phone and video-on-demand definitely need fast response time, while web banking service maybe can tolerate slower response time etc). Generally speaking, 1.5 seconds response time is believed to be a reasonable upper bound most people agree on for recent web services.

Taking the server utilization into consideration, it is not difficult to see that if the utilization approaches 100%, there is not much improvement the server can make anymore. So in our project, we decide to choose the cut-off point before the server utilization becomes saturated. To be more specific, in the above simulation setting, we choose the cut-off point to be 620 users, which corresponding to the response time of 1.3 seconds and the utilization of 95%. In our probabilistic approach of admission control, we also take this cut-off point into consideration when we choose a probability function. In practical applications, people can choose their own cut-off point depending on the server speed, type of service etc.

4.5 The theoretical model of the oscillation of the number of users in the system

To better understand the situation of the oscillation in the system, let us consider an ideal theoretical model of the number of users in the system. Since the arrivals of the new users are considered to be independent, so they are the Poisson process. It is well know that, for Poisson process arrival events, the inter-arrival time should follow an exponential distribution. After being accepted, the users will stay in the system and is served for some period of time (the session time also follows an exponential distribution), and then leave the system. For simplification, we consider that, in every time instant there are a certain percentage of users leaving the system. The more the users are in the system, the more the users are about to leave. This is quite intuitive, and complied with the simulation outcome.

To start studying the model, we will use the following notations for convenience:

y: the number of users in the system

r_a: the arrival rate of the incoming users

p_l: the rate of users that will leave the system

t: the time

Since the change of number of users dy in some small time interval dt equals the number of incoming users minus the number of leaving users, we can easily come up with the following differential equation:

$dy = r_a * dt - y * p_l\, dt$ or $dy / dt = - y * p_l + r_a$

By solving the above differential equation, we get

$y = r_a/p_l + c * e^{-p_l t}$

Here c is some constant value, which can be computed from the initial conditions. In the initial state, for t = 0, if there are y_0 users in the system, that is $y = y_0$, we get $c = y_0 - r_a/p_l$. Replacing c by $y_0 - r_a/p_l$ in the above equation, we get the following:

$$y = f_{up}(t) = r_a/p_l + (y_0 - r_a/p_l) * e^{-p_l t}$$

In the case that $r_a = 10$ users/second, and $p_l = 0.01$, and $y_0 = 0$, $y = 1000*(1 - e^{-0.01t})$. This curve is shown in Figure 14:

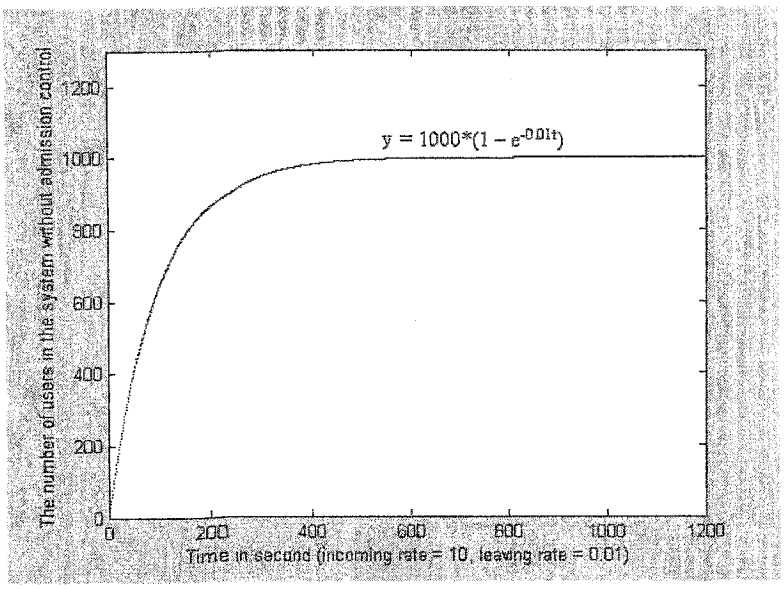


Figure 14. Theoretical model of the number of users when it increases, $f_{up}(t)$

As we can see from the above figure, when the system gets stable, or as $t \to \infty$, y reaches the value of 1000.

To compute the number of users leaving the system during the time interval of dt when there are no users coming into the system, we can set $r_a = 0$ in the above differential equation and get

$dy = - y * p_l \, dt$

By solving the above equation, we get

$y = f_{down}(t) = \quad y_0 * e^{-Pl\, t}$ (with some initial value y_0 for the number of users)

Taking $y_0 = 1000$, and $p_l = 0.01$ (the same value as above), the number of users in the system is given by $1000 * e^{-0.01t}$, as shown in Figure 15.

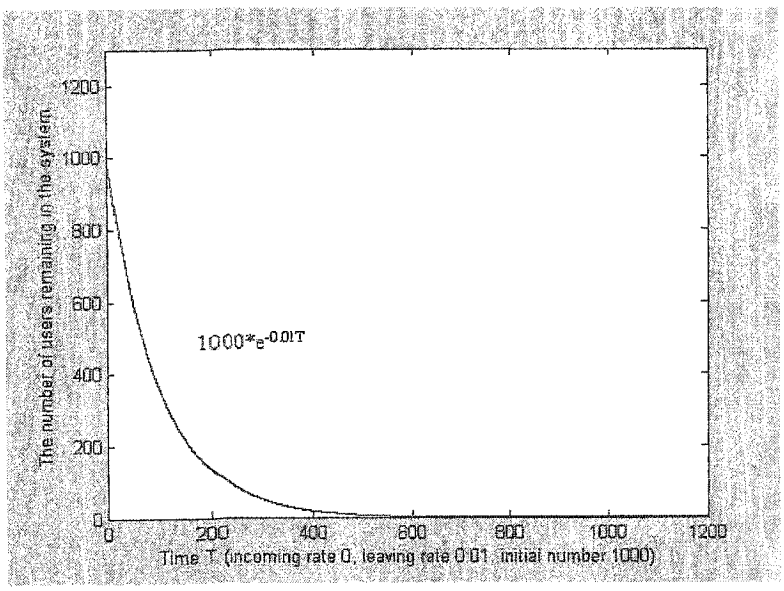


Figure 15. Theoretical model of the number of users when it decreases, $f_{down}(t)$

Equipped with the above theoretical model, we now address the problem of the oscillation of the number of users in the system using the abrupt cut-off algorithms. Suppose the cut-off point is 450 in the number of users in the system, which means we reject all new users when the number of users in the system exceeds 450; and we further assume that the inter-observation period T is very long, like 100 second; the other parameters are kept the same as above, $r_a = 10$, $p_l = 0.01$ etc. We start the admission control somewhere after the system is getting stable (the number of users is approaching 1000). See the example in Figure 16.

After the broker sees 1000 users in the system, it immediately cuts off all the incoming new users. So in the next inter-observation period T, no user will come in and the system follows the function $f_{down}(t)$ (the solid line in the following figure). From the figure of the of function $f_{down}(t)$, we can see within 100 seconds the number of users in the system can drop from 1000 to around 360. That is 640 users will leave the system in 100 second. But 360 is below the cut-off point of 450, which again turns on the admission of new users during the next inter-observation period, and the system follows the function of $f_{up}(t)$ (the

dashed line in the following figure)... This kind of oscillation will go on forever with these two functions switching back and forth.

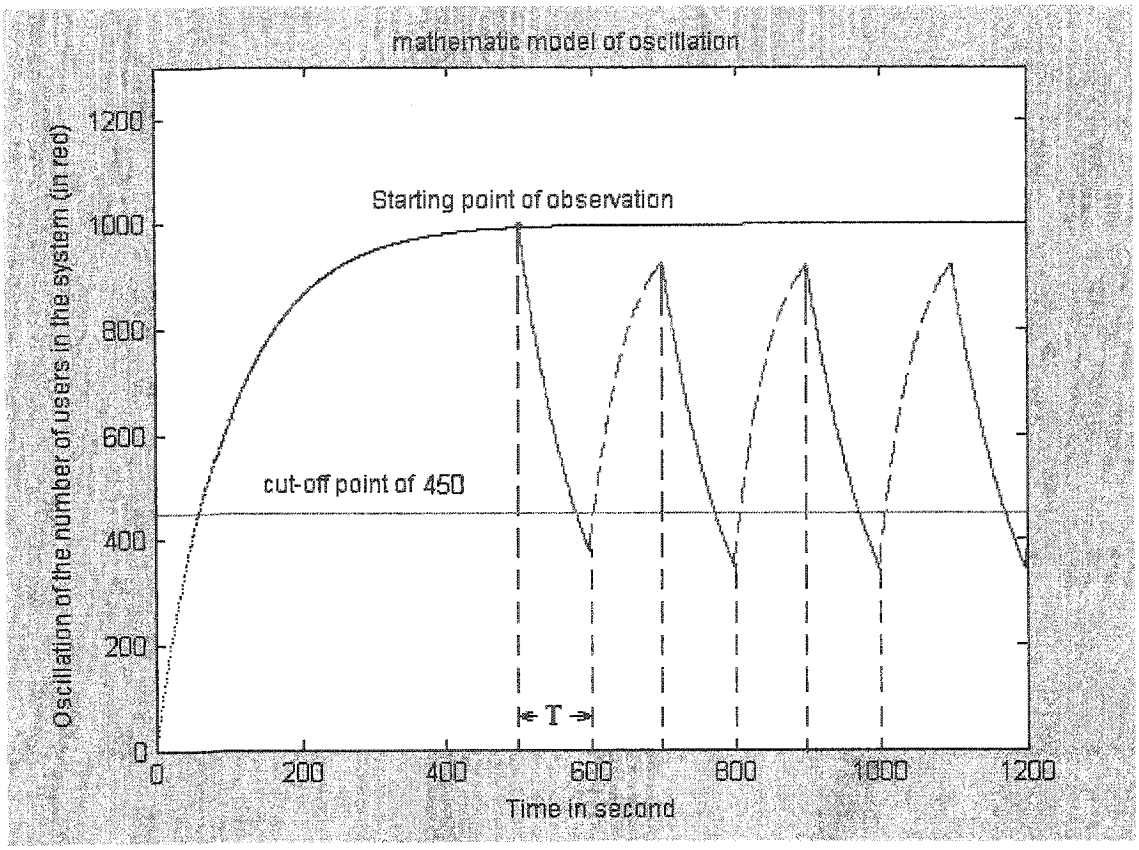


Figure 16. Theoretical model of oscillation

To compute the amplitude of the oscillation, we need to know the change of the number of users in every inter-observation period. In our ideal theoretical model, when the oscillation is getting stable, the upper bound and the lower bound of the oscillation are expected to reach their asymptotic limit. To get the value of the amplitude at this asymptotic limit, we denote the amplitude of the oscillation as A, and suppose that the oscillation is varying from L to A+L, where L is considered to be the lowest bound of the oscillation below the given threshold.

In one time interval of T, the number of users will increase by the amplitude of A; that is $f_{up}(T) - f_{up}(0) = A$ for $y_0 = L$; that is

$$(L - r_a/p_l) * (e^{-p_l * T} - 1) = A$$

And then the number of users will decrease from L+A to L in the next time interval T.

$f_{down}(T) = L$, so $(A+L)*e^{-p_l * T} = L$ (here y_0 in function $f_{down}(t)$ equals to the upper bound L+A)

By solving the above two equations $f_{up}(T) - f_{up}(0) = A$ and $f_{down}(T) = L$, we can compute A; it is given in the following expression.

$$A = r_a/p_l * (1-e^{-p_l * T})/(1+e^{-p_l * T})$$

and

$$L = r_a/p_l * e^{-p_l * T}/(1+e^{-p_l * T})$$

Taking the previous values for p_l (0.01), r_a (10), and T (100), we can compute A as 464, which coincides with what is shown in the above figure! But this formula depends on one assumption, that is, the decreasing and increasing of number of users should alternate between adjacent observation intervals. However, this depends on the threshold chosen and is not true for very high and low thresholds, as discussed below.

Notice that when $T \to \infty$, $(1-e^{-p_l * T})/(1+e^{-p_l * T}) \to 1$, so $A \to r_a/p_l$ and $L = 0$; on the other hand, when $T \to 0_+$, $(1-e^{-p_l * T})/(1+e^{-p_l * T}) \to 0$, so $A \to 0$ and $L = 0.5 * r_a/p_l$. Notice A only depends on T, r_a, p_l and does not depend on the starting point of the observation. In the following, we show an example of starting observation at the time point of 50s. As we can see, the amplitude does not change too much.

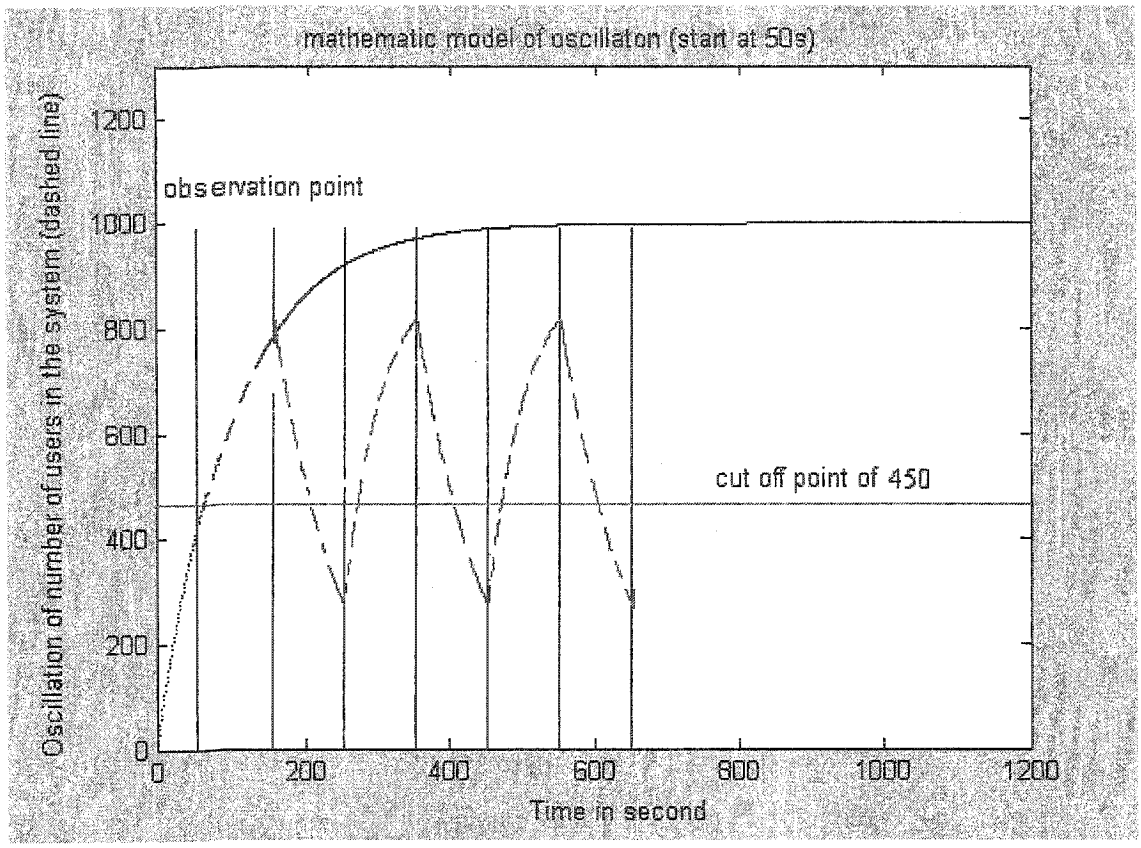


Figure 17. Theoretical model of oscillation (start at 50s)

Now we start to check how the choice of threshold may effect the oscillations. In the following figures, we show the oscillations when the threshold is 200 and 800, respectively.

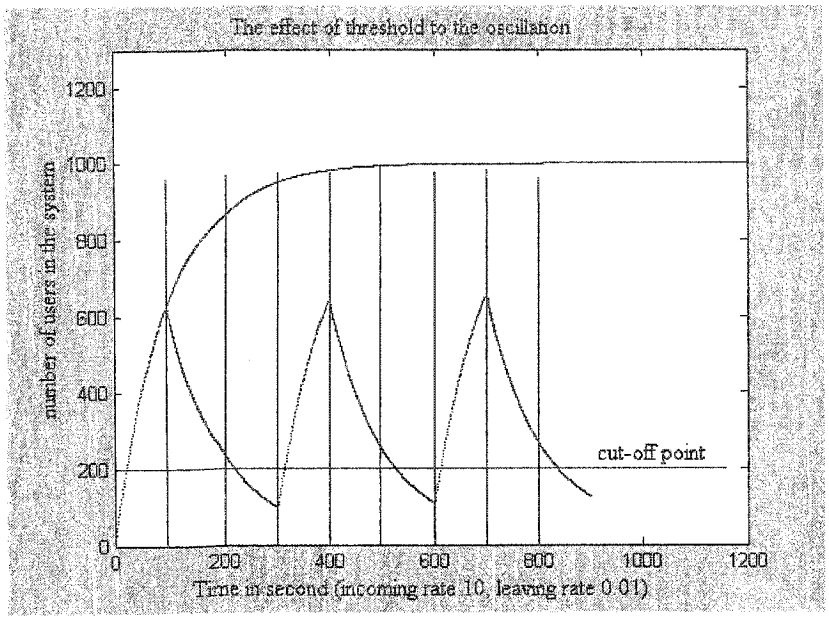


Figure 18. The effect of the threshold to the oscillation (low threshold)

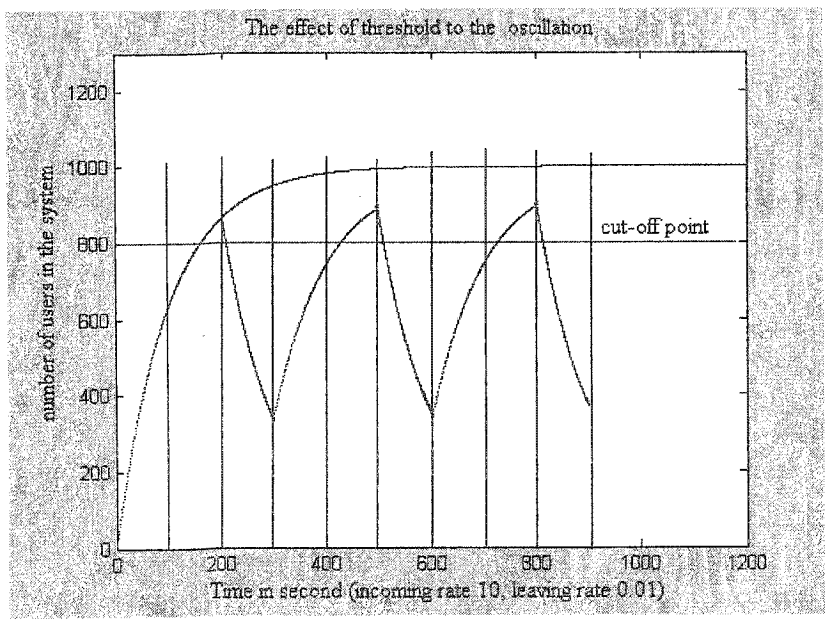


Figure 19. The effect of the threshold to the oscillation (high threshold)

The above figures show that when the threshold is chosen to be quite low, like 200, the system needs more time (twice as long as the inter-observation time) to serve users before the number of users drops below the threshold. On the opposite, when it is chosen to be

quite high, like 800, then the system needs more time to let new users coming into the system before the threshold is exceeded. In both case, the period of the oscillation becomes 3T. Notice that the period of the oscillation can be even longer (4T, 5T…) if we choose even higher/lower thresholds. The amplitude also becomes larger as the period increases.

5. Probabilistic approach of admission control

In this chapter, we introduce a probabilistic admission control for the web servers. We start with the study of the theoretical model of the probabilistic admission control and prove that it has the advantage over the on-off approach used before. And then the simulation results are presented and discussed.

5.1 Probabilistic admission control

Before we introduce the probabilistic approach, we would like to clarify one thing. That is when to use the number of users and when to use the response time as a criterion for admission. From the previous chapter, we know that there is a relationship between the number of users and the response time. This relationship also depends on the speed of the server. The faster the server is, the less time is needed to serve each user and vice versa. So there is no point of using the number of users in the system to control the admission in reality. The number of users suitable for a faster server would be far too much for a slower one. Using the number of users in the system for the admission control will apparently impair the scalability and undermine all the effect we made. But on the other hand, due to the convenience of making a theoretical model of system oscillations using the number of users, we sometimes have to use it in order to get a good understanding of the subject under investigation. So in this thesis, although we use the number of users in our theoretical model to determine whether or not to accept a new user, in reality, we use the average response time as an important index to control the admission. In this way, the algorithm does not have to care about the speed of the server, thus it can be implemented in a general environment without the knowledge of the capacity of each specific server.

In our probabilistic approach, to avoid system oscillation, each user no longer gets a yes/no answer; instead they are admitted by some probability. And this probability P is a function of the current response time r of the server. In this thesis, we assume that the function P is a piecewise linear one, like:

$$P_{a,b}(r) = \begin{cases} 1 \ (\text{if } r < a) \\ (r-a)/(b-a) \ (\text{if } a < r < b) \\ 0 \ (\text{if } r > b) \end{cases}$$

where a and b are two constants which indicate at which response time to start partial rejection and at which response time all users will be rejected, respectively.

The following figure shows this function, for a = 1.2s and b = 3.6s.

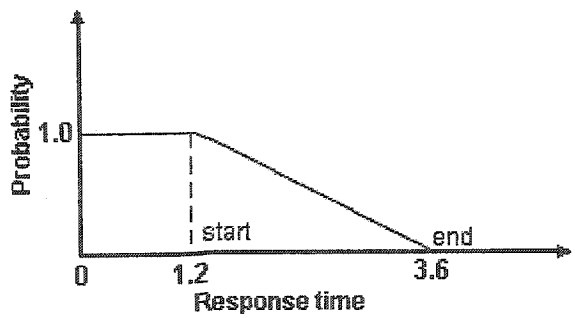


Figure 20. Probability function

With this function, we start to reject user requests when the response time exceeds 1.2 seconds. The longer the response time is, the less likely the user requests will be accepted. When it exceeds 3.6 seconds no requests can be admitted anymore. Notice that the probability function is not limited to a linear function, it could also be exponential or of any other forms. The administrator could adjust it depending on the special needs.

The broker checks the current average response time of the servers in its domain. When a client request comes, the broker picks up one server according to some criterion (like

LAS, LU in [4]), and then it calculates an acceptance probability from the current response time. Using this probability, the broker randomly grants or rejects the request.

The advantage of this approach over the previous on-off decision approach is that we can reject or accept user gradually rather than abruptly. By using this probabilistic admission control algorithm and choosing proper inter-observation time, we hope to avoid the oscillation in the system, as we will discuss later.

5.2 The study of the theoretical model of the probabilistic approach

Now we further on discuss the theoretical model of Section 4.5 where we now use the probabilistic approach to determine whether to accept users or not. Here, we again use the number of users (rather than the response time) to make admission decisions, since by using the number of users we can easily derive the mathematic formula of the workload. Besides, we already know the relationship between the response time and the number of users; it is not very difficult to translate the number of users into the response time of the system.

We use the same notations in Chapter 4. The only difference is that here we use a linear probability function P = (b-y)/(b-a) to control the admission. Here y is the number of users in the system, a and b are the integers indicating when to start rejection and when P is 0, respectively; also notice that when $y > b$, $P = 0$, and when $y < a$, $P = 1$. To make sense, a and b should be smaller than the number of users the system can reach without admission control, namely r_a/p_1 , so we have $0 < a < b < r_a/p_1$.

Again we start with the following differential equation:

$dy = r_a * P*dt - y*p_l\,dt$ for $y \in [a, b]$ ①

$dy = -y*p_l\,dt$ for $y > b$ ②

$dy = r_a*dt - y*p_l\,dt$ for $a > y$ ③

It is clear that the solutions for ② and ③ are $f_{down}(t)$ and $f_{up}(t)$ respectively, as mentioned in the Chapter 4. By solving the differential equations ①, we get

$$y = b*r_a/(r_a + p_l*(b-a)) + c * e^{-(r_a + p_l*(b-a))*t/(b-a)}$$

Here c is some constant value, which can be determined from the initial value of the equation. The initial state is at the time point where y = a; because $y \in [a, b]$ in ①. From this initial state, we can get $c = (a - b*r_a/(r_a + p_l*(b-a))) / (1 - a*p_l/r_a)^{(r_a/p_l/(b-a) + 1)}$.

Since $P \leq 1$ when y is in the range [a, b], we conclude that with the number of users within the range [a, b], the speed of the increase of the number of user with the probabilistic approach is slower than with the on-off approach.

From the solution of the differential equation ①, when $t \to \infty$, the second term in y will vanish, thus $y_{stable} = b*r_a/(r_a + p_l * (b-a))$. We call this a stable point, it can also be computed based on the fact that when a stable situation is reached, the number of incoming users equals the number of users leaving ($r_a* P = y*p_l$, where $P = (b-y)/(b-a)$). In the ideal model, the oscillation of the number of users will not go on forever, it will finally become stable at this point. It is not difficult to proof that if $p_l < r_a$, then $a < y_{stable} < b$. That is the stable point is somewhere between a and b, not necessary in the middle, (a+b)/2. The stable point is drifting between a and b depending on the workload. If the workload is really high, or $r_a/p_l \to \infty$, then $y_{stable} = b$; the stable point will be at point b.

The above equation is based on the very ideal condition, where we assume that we can check the system performance at any point in time. But in the real system, it is not possible to do so; the broker checks performance data only periodically. If at a given point, the broker computes the probability as P, then in the next time interval, P has to be used without change. Here, we take P as a variable dependent on the number of users y at the last observation time point.

Again by solving the differential equation:

$dy = r_a * P*dt - y * p_l \, dt$

we get the number of users

$$y = P*r_a/p_l + (y_0 - P*r_a/p_l) * e^{-p_l t}$$

Although observations are made only at the observation time points, the probabilistic approach will be able to suppress the oscillation, and stabilize it at the stable point. In the following figure, we show an example generated with Matlab, which shows that the oscillation is dampening out within several inter-observation time periods, and finally stays at the stable point y_{stable}.

In this example, we choose the same setting as in Section 4.5 ($r_a = 10$, $p_l = 0.01$), $a = 200$, $b = 800$. We apply the probabilistic admission control after 200 seconds. As it is show in the figure, within 10 inter-observation time periods, the oscillation is totally wiped out and the system reaches the stable point $y_{stable} = b*r_a/(r_a + p_l * (b-a)) = 500$.

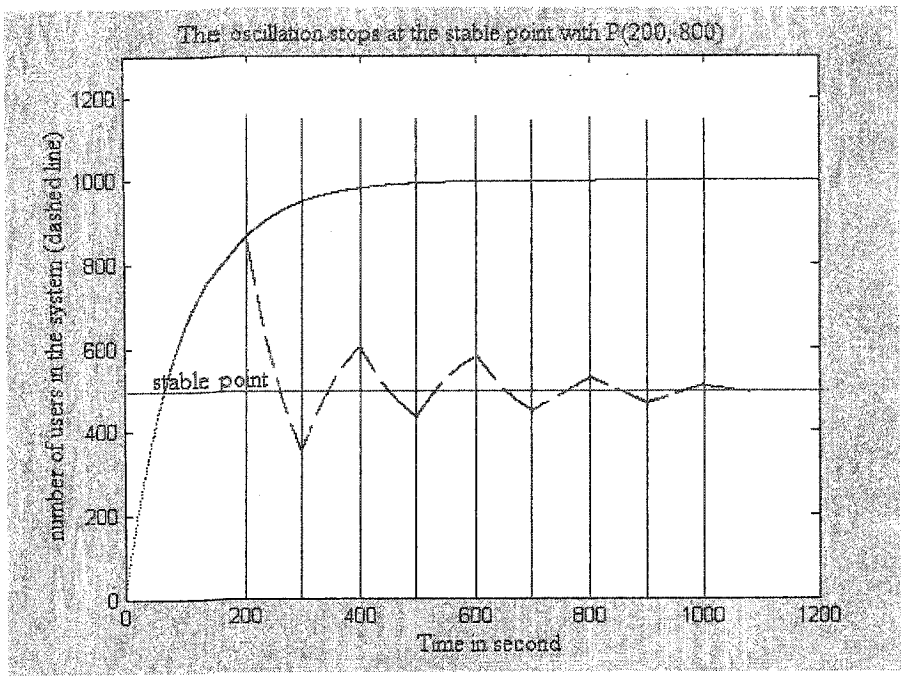


Figure 21. The oscillation gets stable at the stable point

We note that the length of the inter-observation time has to be chosen short enough such that the oscillation of the number of users will not go beyond the limits a and b. With a long inter-observation time period, the oscillation may go beyond the range of a and b, and the oscillation is unavoidable in this case. This happens if the observed number of users alternates between a value above b and a value below a, thus leading to alternate probabilities equal to 0 and 1.

Let us study the effect of oscillation on the following figure. This is a figure showing the relationship between the response time and the number of users in the system, which has been discussed in Section 4.4.

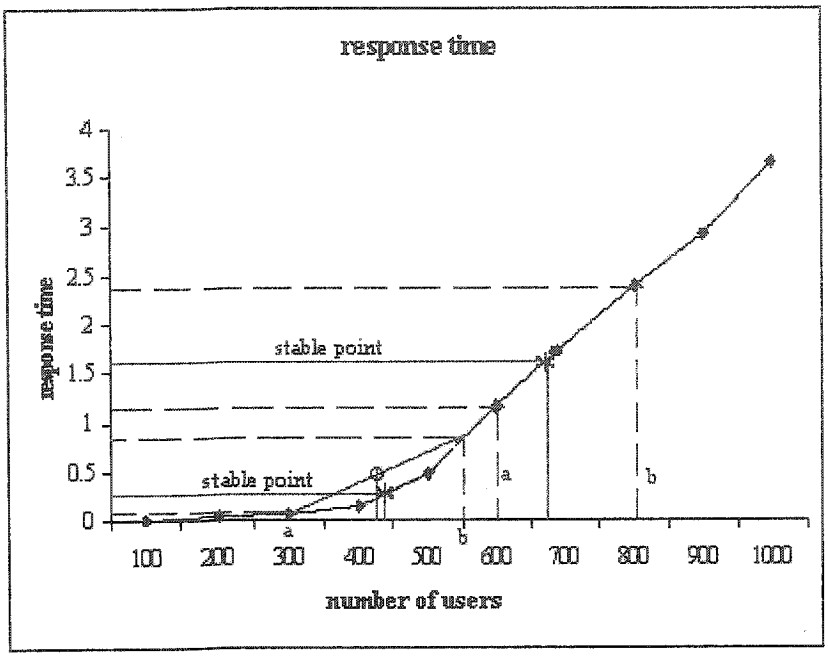


Figure 22. The stable point on the response time curve

If we choose the a and b as 300 and 550, respectively, the number of users in the system will oscillate from 300 to 550, and finally it will become stable at the stable point y_{stable} = 440 (taking p_l=0.01,r_a=10 without loss of generality). From the curve of the response time, 440 users correspond to the response time of around 0.25 seconds. We represent this stable point on the curve as an asterisk in the above figure. Consider if we use the on-off approach, then the number of users will forever oscillate between 300 and 550, and the average number of users will be around 425, which corresponds to the response time around 0.5 second (represented by a period in the figure). This is exactly where the improvement of our probabilistic approach is. It reduced the response time from 0.5s (on-off approach) to 0.25s (with probabilistic approach).

Take a look at another example where we have (a, b) = (600, 800). In this case, y_{stable} = 667; we would expect that the stable point corresponds to the response time of 1.6s (represented by a asterisk in the figure). And there is not too much improvement compared with the on-off approach, because in this range the curve is strait. We note that

in this case (with the larger values of a and b), the response time already exceeds 1.3s limit, the server is already fully loaded (server utilization equal 1), and there is no point in choosing such a large value for a and b. We conclude that, by avoiding oscillations, the probabilistic approach provides better average response time within the critical operating point when the load is close to 100%.

Needless to say, since we have proved that the stable point exists in the ideal theoretical model. In the realistic world, we would like to use the probabilistic approach, so that the stable condition can be reached, and thus oscillations can be avoided.

5.3 Simulation Result

In this section, we present the simulation experiments we have done, and their results.

5.3.1 The evaluation of different probability functions

To study the gradual probability approach for controlling the admission of new users, thus controlling the response time and server utilization of the system, we first study the behavior of different probability functions over a single group of users, and for a variety of user arrival rates, which is expressed by the mean inter-arrival time between users. The probability to accept users is a function $P_{a, b}(r)$ of the response time r, as defined in Section 5.1.

The values a and b indicate the measured response time where the broker starts to reject users, and where no user can be accepted (or P = 0), respectively. In the following

simulation test, we use 3 different probability functions $P_{0.1, 2.5}(r)$, $P_{0.7, 1.9}(r)$ and $P_{1.3, 1.3}(r)$, shown in the following figure. For the purpose of comparison, we also give one extreme case, where there is no admission control at all, which means P is always equals to 1. Notice that function $P_{1.3, 1.3}(r)$ is exactly the on-off decision approach, which was used in Salem's paper [1]. By doing a simulation, we can get the average server response time, the server utilization, and acceptance percentage for different probability functions, and we hope to find out which probability function has advantages over the others.

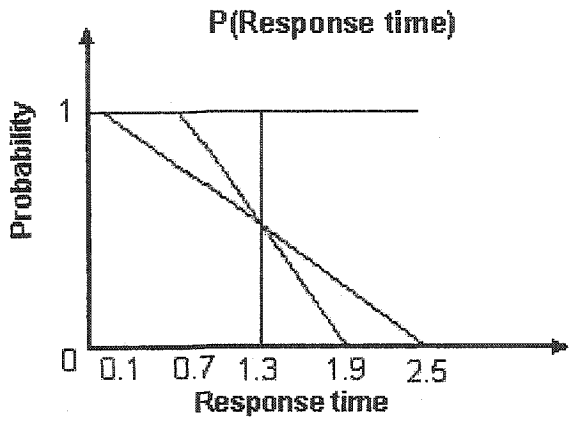


Figure 23. Probability functions

The following chart shows the simulation result for the mean response time with different admission control functions. The results are plotted against the inter-arrival time, for an inter-observation time that equals 10 seconds.

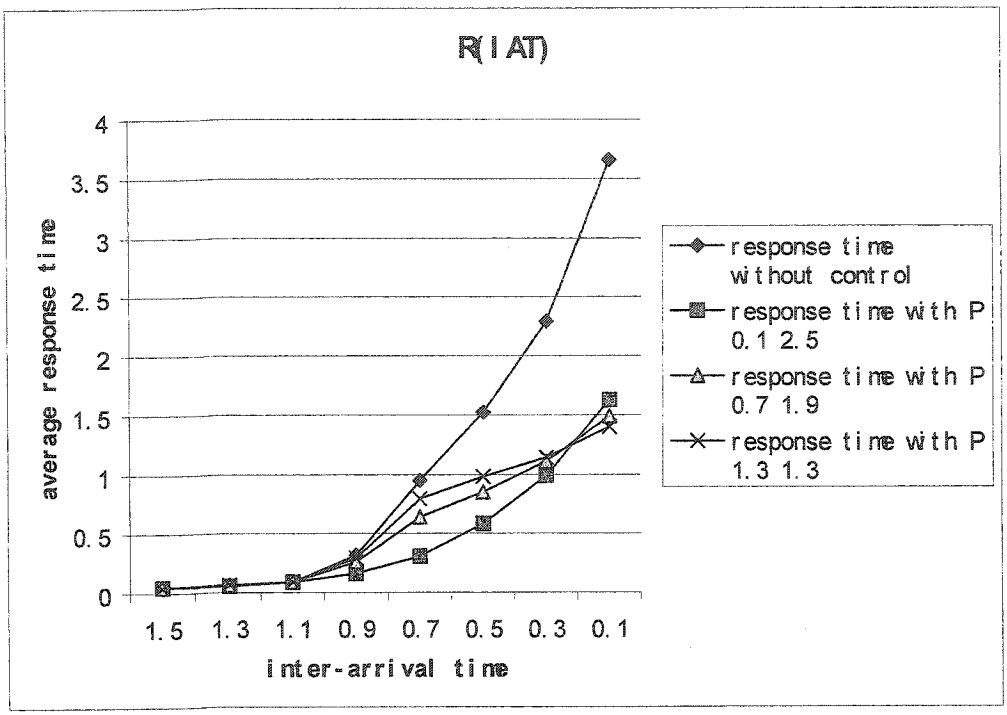


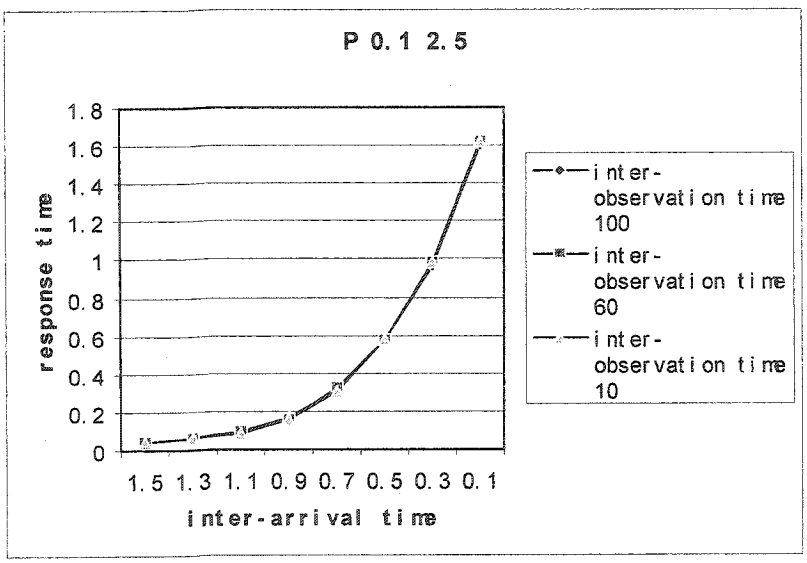
Figure 24. Average response time using different probability functions

Notice that somewhere for an inter-arrival time near 0.2s, the three lines with access control meet at a point where the response time equals 1.3s. It is no surprise to us, since this corresponds to the intersection of 3 lines in the figure of the probability functions. In other words, this intersection is caused by the fact that when the average response time equals 1.3s, all three probabilistic functions produce the same probability value - 50%. We can see that for inter-arrival times larger than 0.2s, the average response time is less than 1.3s (for all 3 probability functions), while the probability functions $P_{0.7, 1.9}(r)$ and $P_{1.3, 1.3}(r)$ reject less users than the function $P_{0.1, 2.5}(r)$, thus resulting in a higher average response time. Following the same reasoning, we can explain why for inter-arrival times less than 0.2s, the response time for the first two functions is less than the function $P_{0.1, 2.5}(r)$.

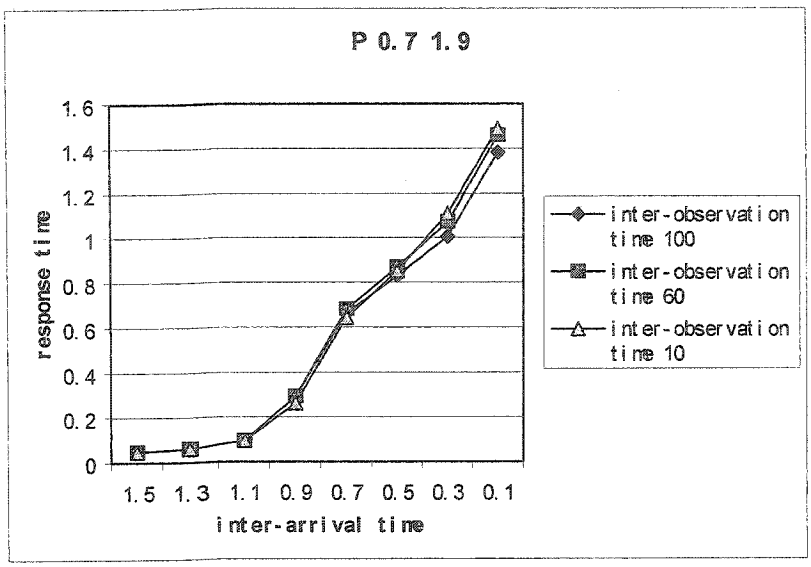
Near the point of 0.8s, there is a sudden increase of response time for the probability

functions $P_{0.7, 1.9}(r)$ and $P_{1.3, 1.3}(r)$, while the response time for probability function $P_{0.1, 2.5}(r)$ grows smoothly. This is because the first two probability functions are less gradual, and they start to reject users only when the response time approaches 1 second; so in the point around 0.8s they do not effectively reject users, resulting in a sharp increase in the response time, which actually corresponding to the situation without admission control. We can also see that the performance without admission control is really bad; the response time will grow very large, providing intolerable QoS.

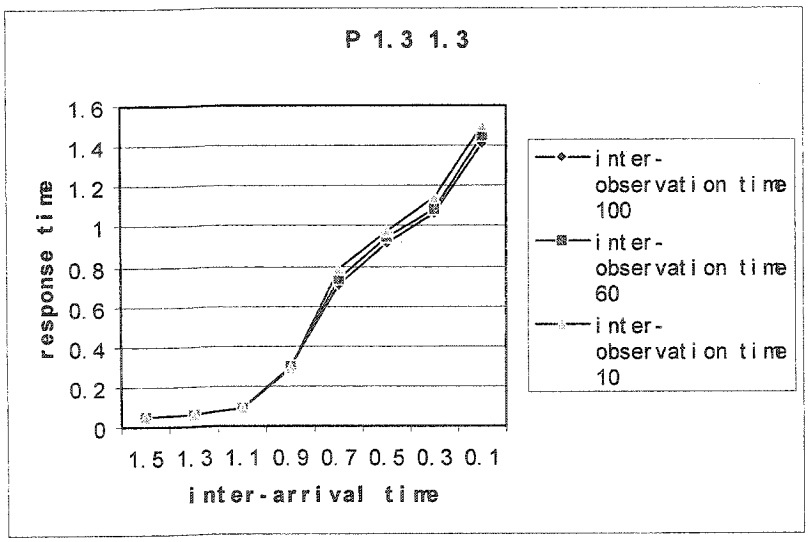
In the above test, we use the inter-observation time of 10 seconds. To understand the effect of different inter-observation times on the average response time, we have done the same simulations with different inter-observation times, namely for 10, 60, and 100 seconds. We show the resulting response times in the following figures:



(a)



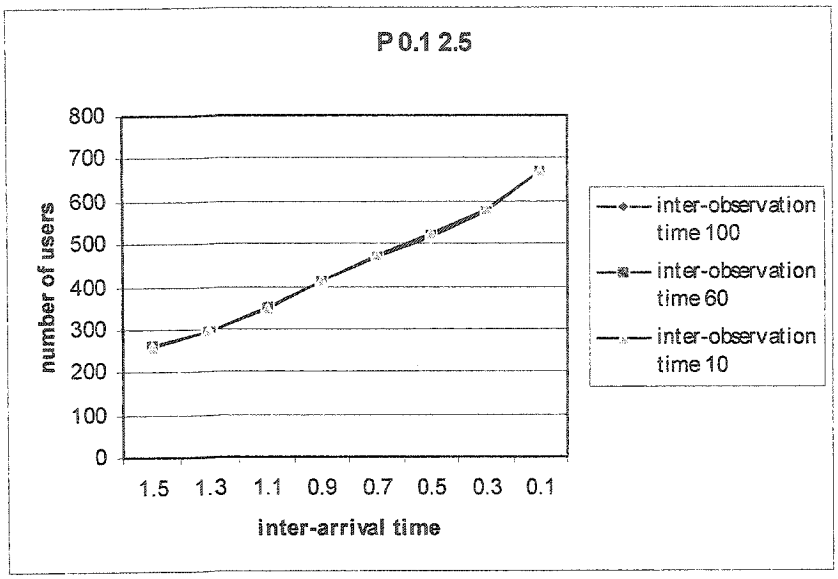
(b)



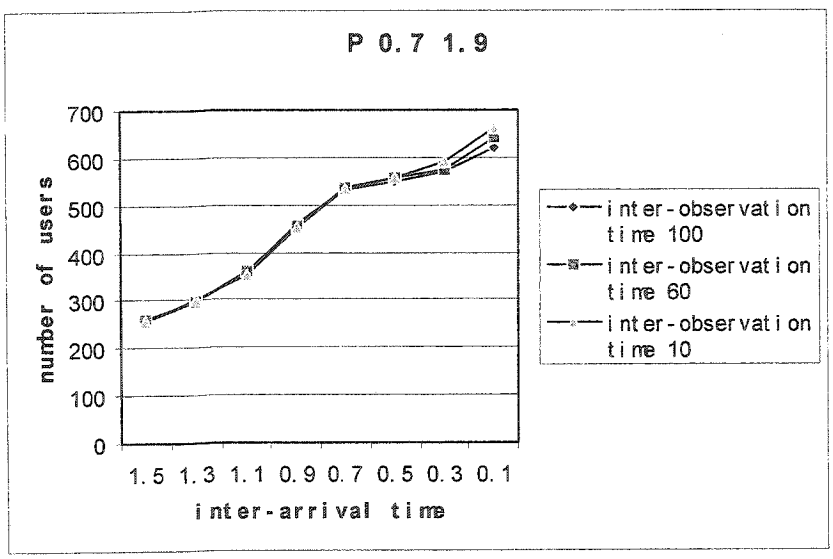
(c)

Figure 25. The average response time for different probability functions when the inter-observation time equals 10, 60, 100 seconds, respectively

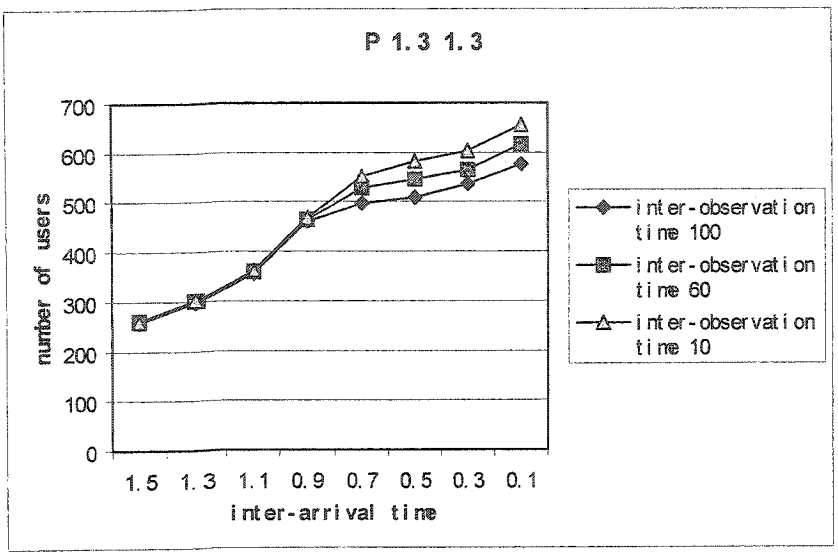
We go on checking the number of users in the system with different inter-observation time.


P0.1 2.5
800
700
600
500
400
300
200
100
0
number of users
1.5 1.3 1.1 0.9 0.7 0.5 0.3 0.1
inter-arrival time
inter-observation time 100
inter-observation time 60
inter-observation time 10

(a)


P 0.7 1.9
700
600
500
400
300
200
100
0
number of users
1.5 1.3 1.1 0.9 0.7 0.5 0.3 0.1
inter-arrival time
inter-observation time 100
inter-observation time 60
inter-observation time 10

(b)



(c)

Figure 26. The average number of users for different probability functions when inter-observation time equals 10, 60, 100 seconds respectively

By doing the same simulation for various inter-observation time intervals, we find that for different inter-observation time periods, with more gradual probability functions, $P_{0.1, 2.5}(r)$ and $P_{0.7, 1.9}(r)$, there is not much change in both their response time and the average number of the users in the system. But for the abrupt probability functions, $P_{1.3, 1.3}(r)$, with the increase of the inter-observation time, although the response time does not substantially changed, the average number of users is dropping. This means that the average service time for each user is increased. Therefore here we can conclude that the length of the inter-observation time does not have a significant effect on the performance of a more gradual probability function; but it does deteriorate the performance for the abrupt probability functions.

In the following chart, we show the utilization of the server with and without the gradual probabilistic control, plotted against the inter-arrival time, for an inter-observation time equals 10s.

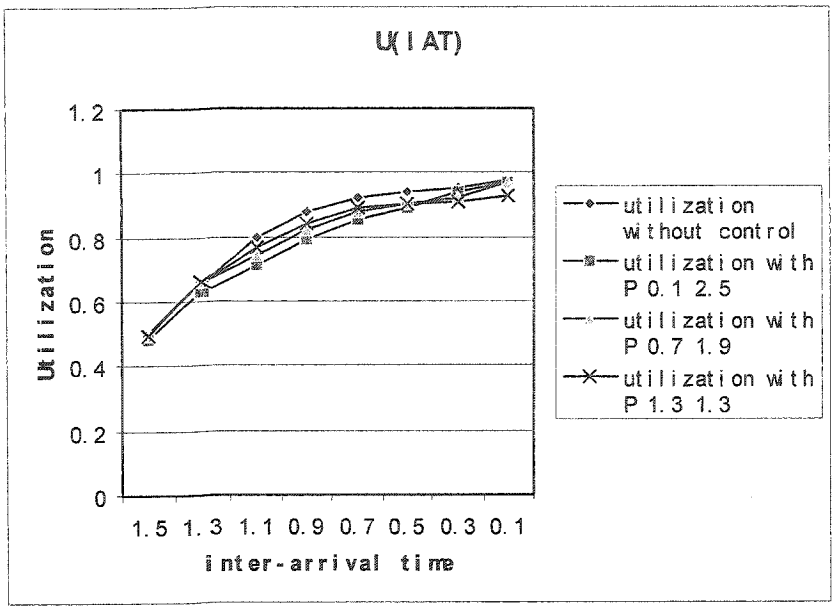


Figure 27. Server Utilization

Here we notice that in the range of inter-arrival times between 1.1 and 0.4, the server utilization with the probability function $P_{0.1, 2.5}(r)$, is a little bit lower than the utilization with the other two probability functions. This can be explained as follows: from the above Figure 24, for the inter-arrival times 1.1s~0.4s, the response time of the server is below 1.3s for all the probability functions; and within that range ([0s, 1.3s]), the probability function $P_{0.1, 2.5}(r)$ rejects more users than the other two probability functions. This results in its a lower utilization.

We can also plot the server utilization against the response time and get the following figure; these lines are very much similar.

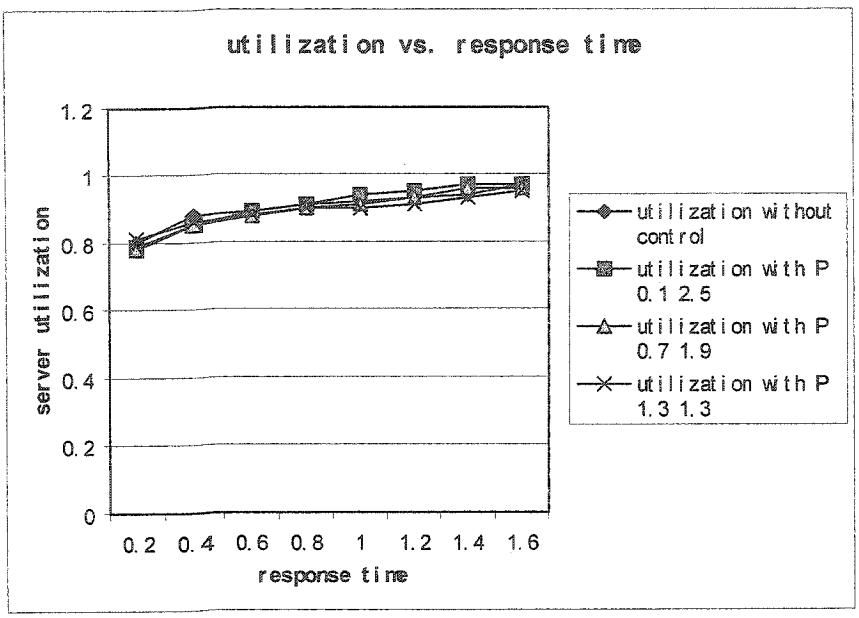


Figure 28. The utilization as a function of response time for different probabilistic functions

The following figure shows the average acceptance probability of the server with gradual probabilistic control, plotted against the inter-arrival time, for an inter-observation time equals 10s.

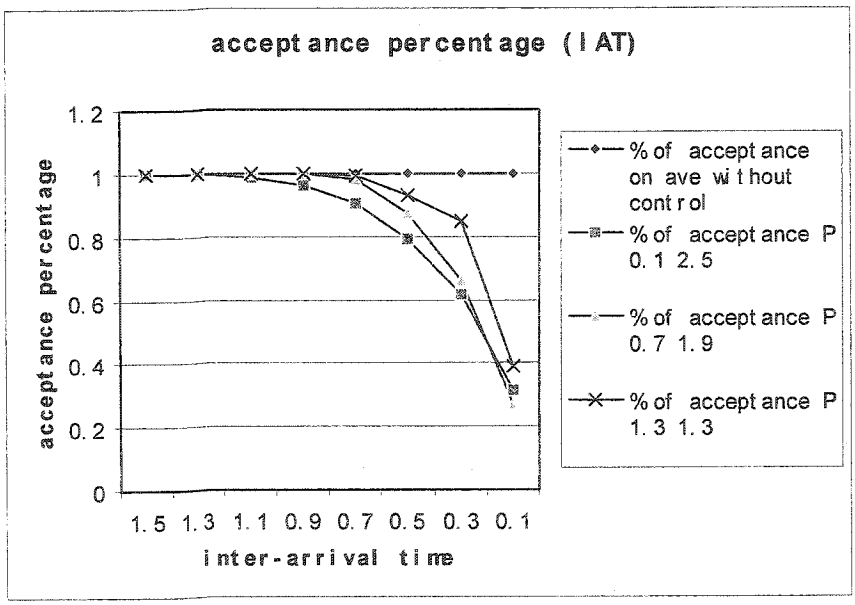
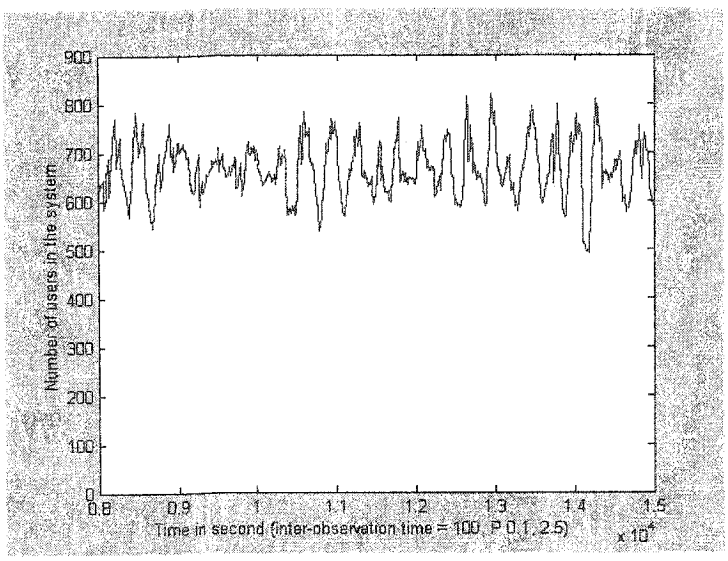


Figure 29. Acceptance percentage

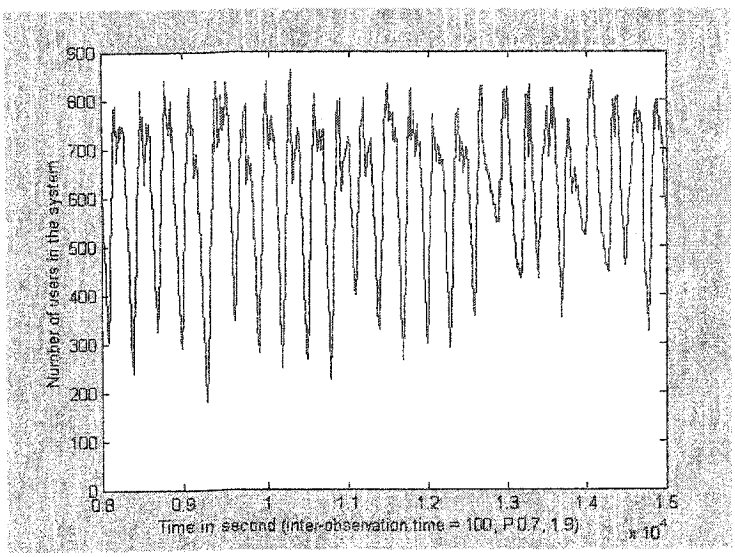
We see that the curves representing the acceptance percentage using the probability function $P_{0.7,\ 1.9}(r)$ and $P_{1.3,\ 1.3}(r)$ are a little bit higher than the percentage of the acceptance for the probability function $P_{0.1,\ 2.5}(r)$. The reason for this is that for the large range of inter-arrival times, all the functions can give an average response time lower than 1.3s, in which range the probability function $P_{0.1,\ 2.5}(r)$ will produce a smaller probability value than the other two probability functions.

5.3.2 The effect of different probability functions on the oscillations

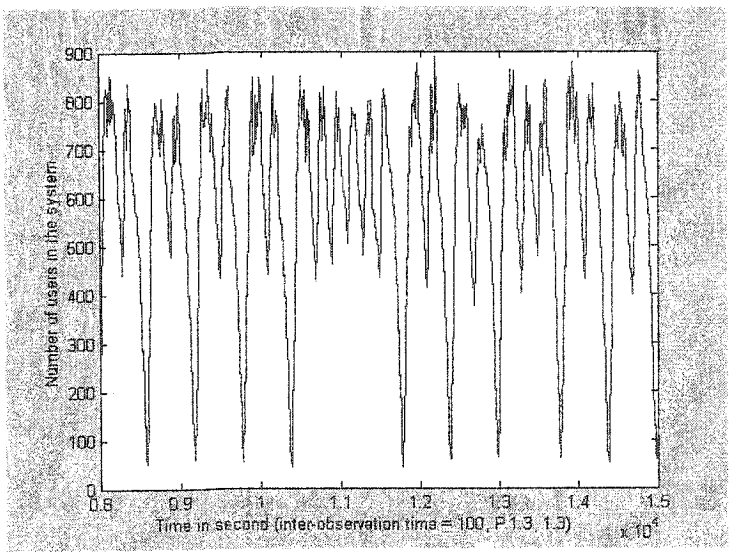
To reveal the effect of different probability functions on the oscillation of the number of users in the system, we further check the amplitude and period of the oscillation for various inter-observation times. In this simulation we set the user inter-arrival time to 0.1s and the inter-observation time equals 100s, the probability functions are $P_{0.1,\ 2.5}(r)$, $P_{0.7,\ 1.9}(r)$ and $P_{1.3,\ 1.3}(r)$ respectively. And we show the simulation result as follows:



(a)



(b)



(c)

Figure 30. The effect of different probability functions on the oscillation of the number of users

The figures above show that, as the curve of the probability function gets steeper, the oscillation of the number of users in the system increases. The amplitude of the oscillation is increased from 240 (for $P_{0.1, 2.5}(r)$) to 800 (for $P_{1.3, 1.3}(r)$).

To measure the period of the oscillation, we used the technical computing tool MATLAB (see appendix) to analyze frequency of the oscillations. After applying the FFT (Fast Fourier transform) algorithm on a large number of sample points, we get the frequency power spectrum of the signal, and the frequency of the curve is the frequency where the power spectrum gets its highest value. Other spikes in the chart can be considered as harmonics and noise, and the noise part can be smoothed out as we include more and more sample data into the calculation. After applying the FFT (fast Fourier transform) to the above sample points for the probability function $P_{1.3,\,1.3}(r)$, we can get the frequency power spectra, as shown in the following figure.

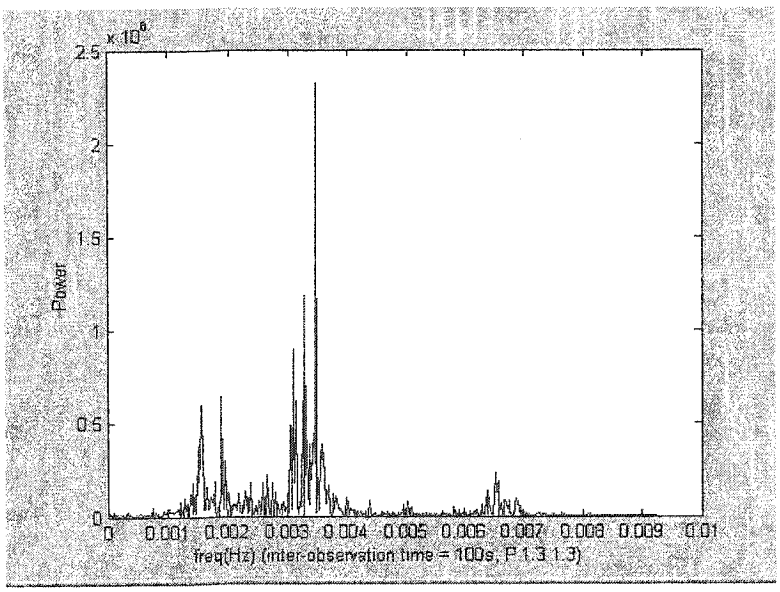


Figure 31. The power spectrums of oscillation of the number of users

The frequency where the power spectrum gets its highest value is 0.003497 in this example, so the period equals 285.98 (the inverse of the frequency).

In the following simulations, we measured the amplitude, standard deviation, mean, frequency and the period of the number of users in the system for different probability functions with different inter-observation times (60s, 40s, 20s, 10s, and 5s), when the inter-arrival time equals 0.1s, and list the simulation result in the following table:

inter-observation time = 100s					
probability function used	amplitude	standard deviation	mean	period	frequency
P 0.1 2.5	240	52.3	671	328.3	0.00305
P 0.7 1.9	560	153.4	605	292	0.00342
P 1.3 1.3	800	208.33	578	285.98	0.003497
inter-observation time = 60s					
probability function used	amplitude	standard deviation	mean	period	frequency
P 0.1 2.5	200	45.9	673	249.9	0.004
P 0.7 1.9	350	78.56	652	218.5	0.004576
P 1.3 1.3	500	117	635	177.95	0.00562
inter-observation time = 40s					
probability function used	amplitude	standard deviation	mean	period	frequency
P 0.1 2.5	200	44.32	674	229.96	0.00435
P 0.7 1.9	300	78.09	646	130.07	0.0077
P 1.3 1.3	450	115.53	627	135.5	0.0074
inter-observation time = 20s					
probability function used	amplitude	standard deviation	mean	period	frequency
P 0.1 2.5	150	41.02	674	undetectable	undetectable
P 0.7 1.9	200	54.62	652	109.67	0.00912
P 1.3 1.3	400	83.22	643	105.77	0.00945
inter-observation time = 10s					
probability function used	amplitude	standard deviation	mean	period	frequency
P 0.1 2.5	120	37.6	678	undetectable	undetectable
P 0.7 1.9	150	45.3	65.7	undetectable	undetectable
P 1.3 1.3	320	76.8	650	100	0.01
inter-observation time = 5s					
probability function used	amplitude	standard deviation	mean	period	frequency
P 0.1 2.5	97	35.1	676	undetectable	undetectable
P 0.7 1.9	140	41.3	664	undetectable	undetectable
P 1.3 1.3	220	68.7	660	undetectable	undetectable

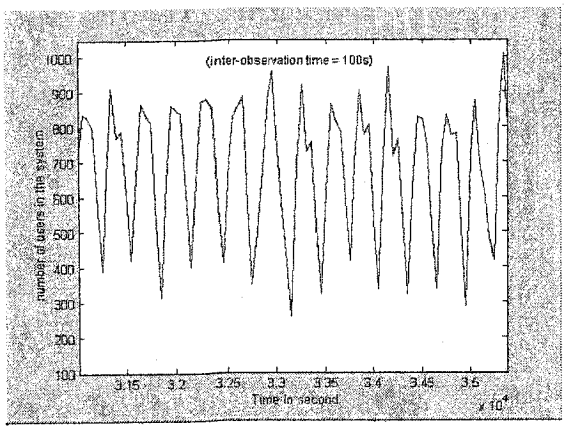
Table 1. Oscillations of the number of users in the system

We can see from the above table, for a given inter-observation time period, the more gradual the probability functions are (like $P_{0.1, 2.5}(r)$), the lower the frequency and the longer the period the oscillation will have. A more gradual probability function also leads to a smaller standard deviation, that is, a less severe oscillation. This means the oscillation is somewhat dampened out by using a gradual probability function.

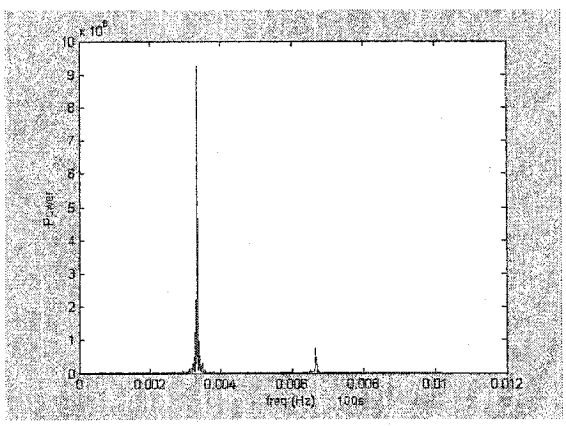
Besides, for a given probability function, the period and the amplitude of the oscillation depend on the inter-observation time. The longer the inter-observation time, the larger is the amplitude and the longer the period. As the inter-observation time decreases, the standard deviation also decreases. This is quite consistent with the decreasing of the amplitude in the second column in the above table. But the mean increases, which implies increased server utilization.

When the inter-observation time decreased to a certain value (like 20 seconds for $P_{0.1, 2.5}(r)$, 10 seconds for $P_{0.7, 1.9}(r)$, and 5 seconds for $P_{1.3, 1.3}(r)$) there is no perceivable period anymore; the whole curve looks just like statistical noise. We call this value a "stable value" for inter-observation time. If the inter-observation time is chosen to be this "stable value", the oscillation would not show any regular period anymore and we can only observe statistical fluctuations of the number of users, thus the oscillation is considered to be avoided. This "stable value" depends on the specific probability function, the more gradual the probability function is, and the larger the stable value will be. An inter-observation time smaller than the "stable value" will surely also rule out the oscillation, but the server has to notify its current response time to the broker so frequently that the broker may become the bottleneck if there are many servers involved. There is always the trade-off between how well the oscillation can be coped with and how much time the broker has to spend in collecting the performance data from the servers. We notice that even for the on-off algorithm, a decrease of the inter-observation time period will improve the oscillation. But this does not eliminate the necessity of using a probability function. Because with a more gradual probability function, the servers may report their performance data less frequently to the broker, which is a plausible improvement. With less frequent collection of the performance data, the traffic on the link between the servers and the broker will be reduced.

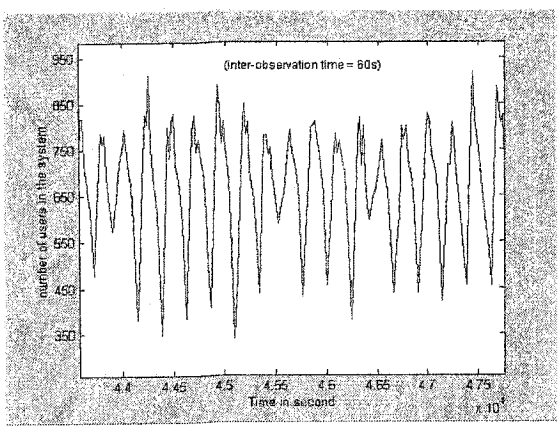
The figure below shows the oscillation of the number of users (left side) and its power spectrum after the application of the fast Fourier transform (right side) for the probability function $P_{0.7,\ 1.9}(r)$ when user inter-arrival time is 0.1 second. As we can see, when the inter-observation time decreased to 10s there is no perceived period anymore; the whole curve looks just like statistical noise.



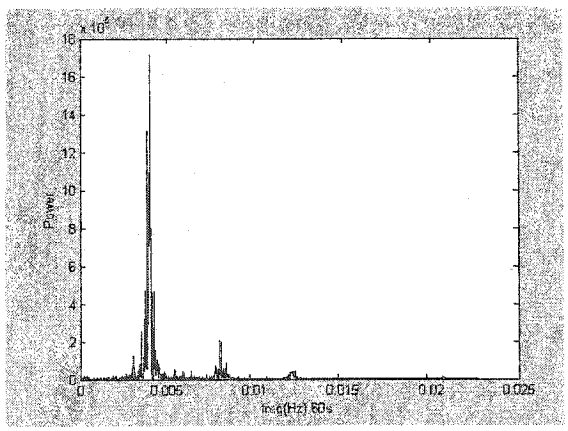
(a)



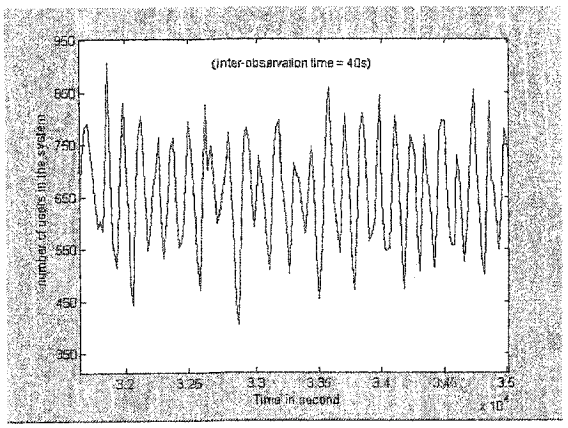
(b)



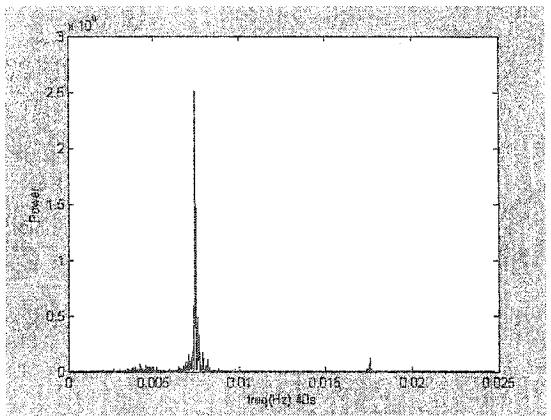
(c)



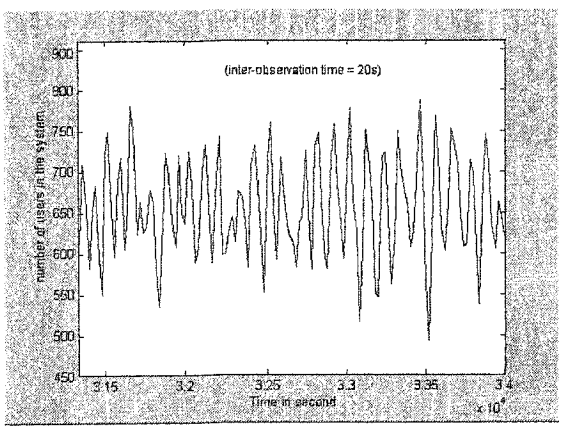
(d)



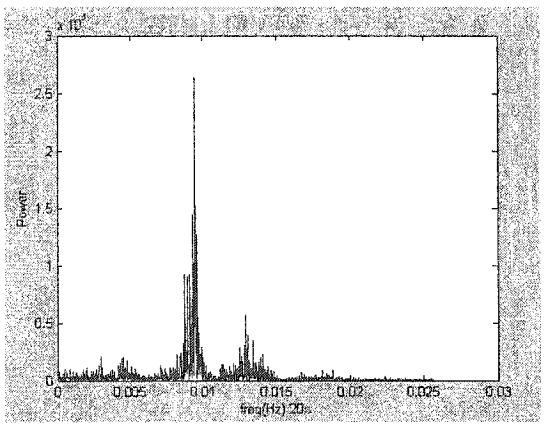
(e)



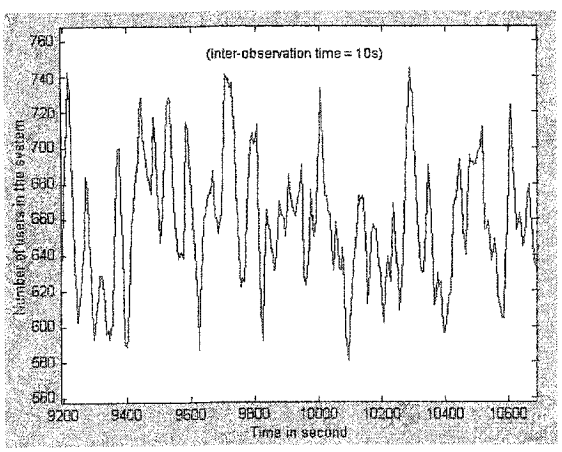
(f)



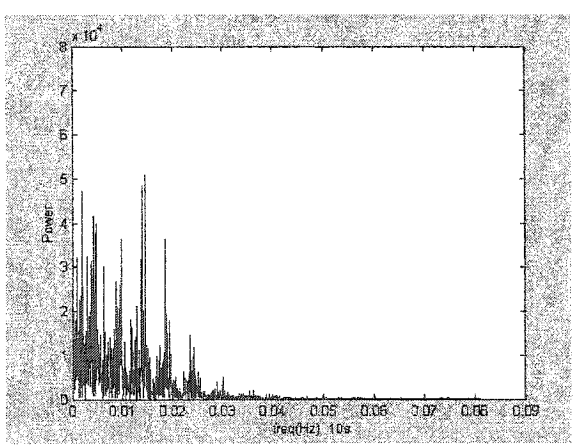
(g)



(h)



(i)



(j)

Figure 32. The oscillation of the number of users and its power spectrum

To conclude, as the above results show, the less steep probability function works better

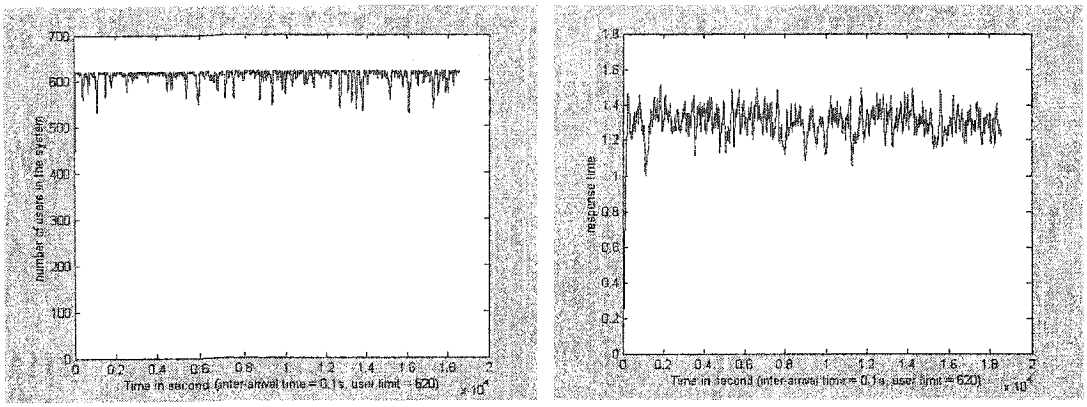
than a steeper one, in terms of its smooth control over the number of user in the system. With a gradual probability function, the oscillation will have smaller amplitude at the price of starting to reject users earlier. For the same inter-observation time, the on-off decision approach is really the worst, since it contributes more to the unstable performance, and the response time can sometimes go very high (e.g. when inter-observation time equal 100 second, for $P_{1.3,1.3}(r)$, as the user number varies between 200 to 1000, the response time varies from 0.1s to 3.5s, while under the same conditions, but for the gradual probability function $P_{0.1,2.5}(r)$, the number of users varies from 550 to 800, which corresponds to a response time between 0.8s to 2.4s).

5.3.3 Putting an upper limit to the server selection algorithm

From the previous sections, since we already know that there is some kind of relationship between the number of users admitted to the system and the response time, we wish to know whether putting an upper limit to the number of users that can be admitted would do any good for the system response time.

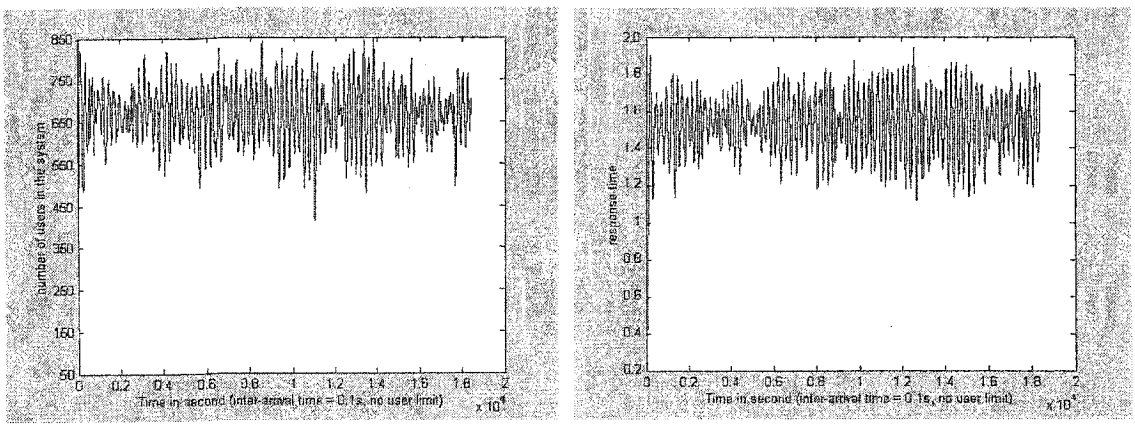
In the following simulation experiment, we still use the probability function $P_{0.7, 1.9}(r)$, the inter-observation time set to be 20 seconds and we also set up an upper limit of the number of users as 620. That means whenever the number of users in the system exceeds 620, we will reject all the newcomers. We know that when 620 users are in the system, the response time should reach around 1.3s ($R(620) = 1.3s$). In the first test, we set the user inter-arrival time as 0.1s; we get the test result in Figure 33. As expected, the average response time of the system is exactly around 1.3s; the number of users in the system is strictly below 620. Compare it with the simulation result without an upper user limit

(Figure 34), other settings being the same; the oscillation of the system with the upper limit is much better than that without any upper limit check.



(a) (b)

Figure 33. The controlled oscillation with the user limit of 620

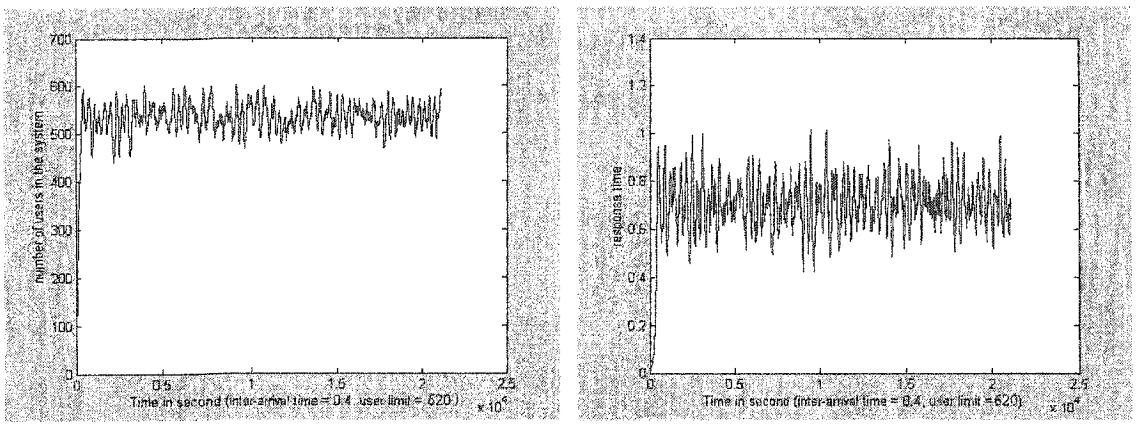


(a) (b)

Figure 34. The oscillation without user limit

When we set the user inter-arrival time as 0.4s, which is a lower incoming rate than 0.1s, we get the following simulation result (Figure 35). The oscillation of the number of users and the response time become more severe than that when inter-arrival time equals 0.1s. Obviously, this is because, with 0.4s inter-arrival time, the load is not high enough so that the upper limit of 620 can hardly be reached, therefore, this upper limit has almost no

effect on oscillation avoidance. But in this case, the oscillation is really not a big issue as long as the response time is still low.



(a) (b)

Figure 35. The controlled oscillation when the user incoming rate is low

The conclusion here is that the upper limitation of the number of users in the system has the effect of preventing oscillation only when the working load approaches or exceeds the upper limit. Below that level, although the oscillation is inevitable, the mean response time is still tolerable.

6. Probabilistic approach used on differentiated classes of users

In the previous chapter, we studied the effect of probabilistic admission control on a single group of users. In this chapter, we will study the probabilistic approach used on two groups of users, where one group has a priority over the other. There have already been some studies focusing on this research field ([32], [33], [34]), but none of them uses probabilistic approach like our approach. In the e-commerce system, we have the motivation of providing service differentiation to the different user groups. Some users are registered user, they are more likely to use the service or purchase the goods that is posted on the Web, while others maybe only casual visitors. So priority should be provided to the registered users. In this situation, we reject the users from the lower priority group while still providing satisfactory QoS to the users from higher priority group when the load is high. By doing this, we realize differentiated services for different user groups.

6.1 Using different probability functions for each of the user groups

To study the performance of our probabilistic approach for two differentiated user groups, user group A with higher priority and user group B with lower priority, we decide to use two different probability functions for each of these user groups, as defined in Section 5.1.

User A: $P_{1.05, 1.55}(r)$

User B: $P_{0.55, 1.05}(r)$

We compare this probabilistic approach with the abrupt on-off approach using two different thresholds for the two user groups (0.8s for group B and 1.3s for group A). We show these probability functions in the Figure 36

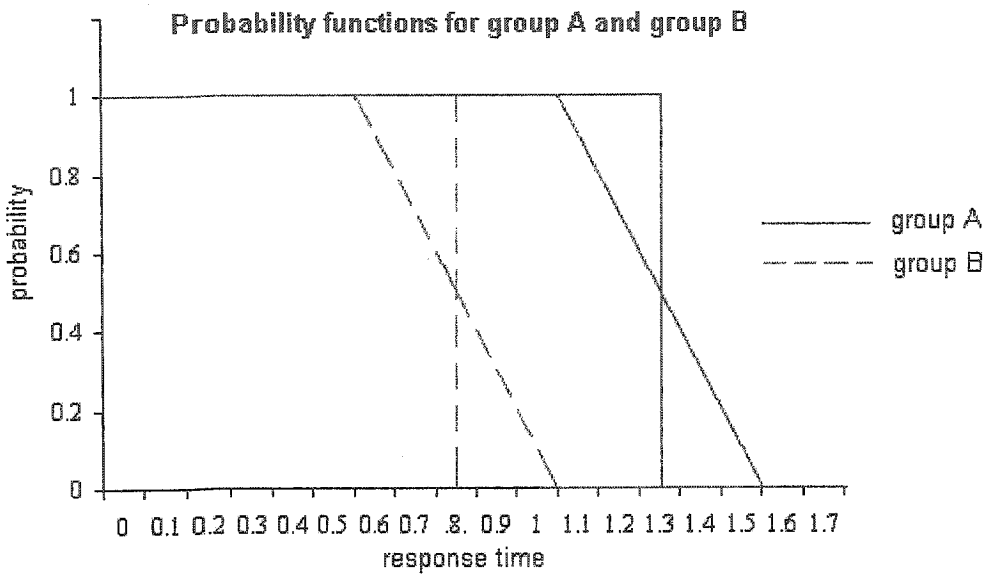


Figure 36. The probability function used for two groups of users

Since group A are the users with higher priority, also called "elite group", they should have access priority over group B users. Therefore we start to reject users in group A only after the response time exceeds 1.05s, at which moment all requests from users in group B have already been turned down (that is P = 0 for user group B). Group B is the lower priority user group; it is the first one to be deprived of the right to access the servers. We start to reject group B users when the response time exceeds 0.55s. By the time the response time exceeds 1.05s, no user from group B will be accepted anymore.

The abrupt threshold algorithm is simpler. User in group B will be rejected if the response time exceeds 0.8s; users from all user groups will be rejected when the response time exceeds 1.3s.

We test the user acceptance probability over a variety of customer arrival rates (for the customer inter-arrival time ranging from 0.7s to 0.05s, and among them, half come from

group A and half comes from group B). In this setting we use the inter-observation time 10s to reduce the oscillations and the result is shown in Figure 37. (We use the notation P a b c d to represent two probability function approach with probability functions $P_{a,b}$ and $P_{b,c}$)

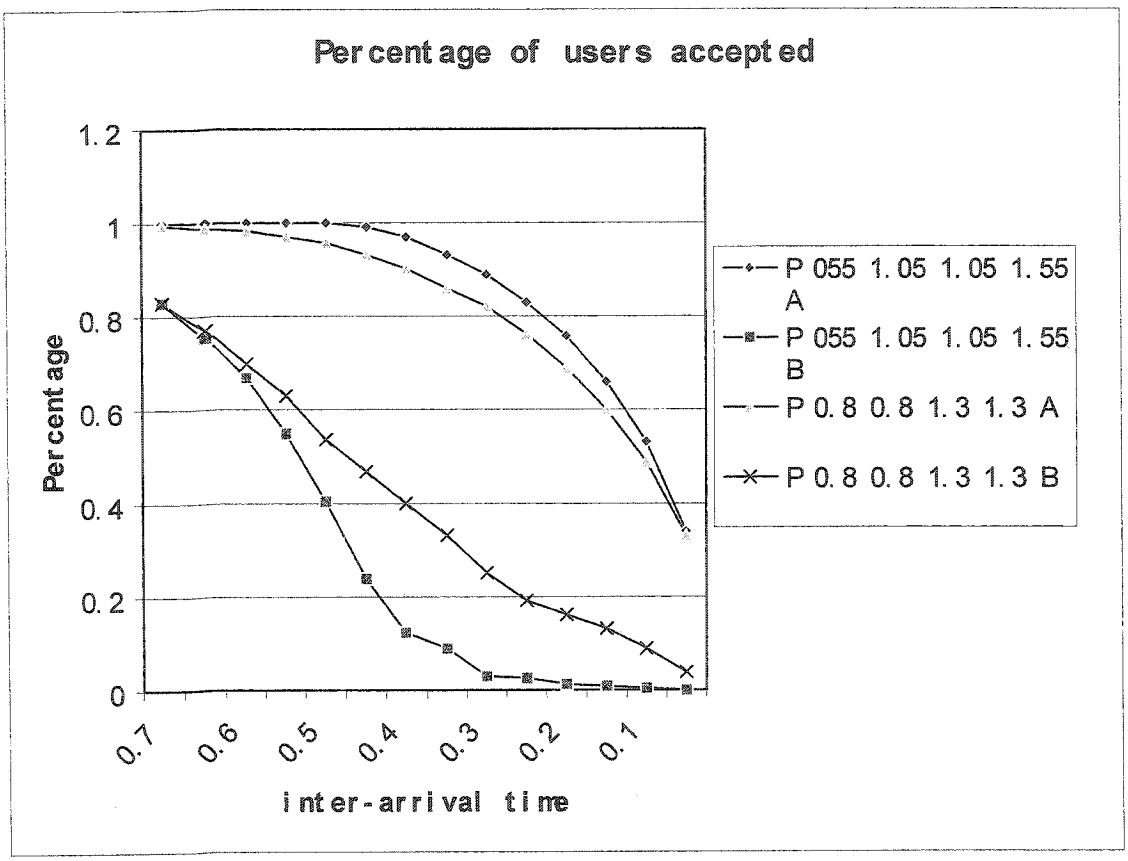


Figure 37. Percentage of users accepted

This probabilistic approach exhibits a clear the privilege for A-users over B-users in terms of the acceptance percentage. While both group enjoy the same response time provided by the system, the probability of accepting A-users is substantially higher than for B-users, especially for higher user arrival rates. For the abrupt on-off switch algorithm, the priority of A-users over B-users is not so clearly identifiable compared with the probabilistic approach. This is surely because the on-off switch algorithm does not

suppress the oscillation of the response time very well, which undermines the differentiation between the two groups of users.

We have done the same simulation for various inter-observation times (10s, 60s, 100s) for both approaches as shown in Figure 38 and 39; we find that with shorter inter-observation period, the priority of the A-users is more distinguishable. This is expected, since shorter inter-observation period means smaller oscillations.

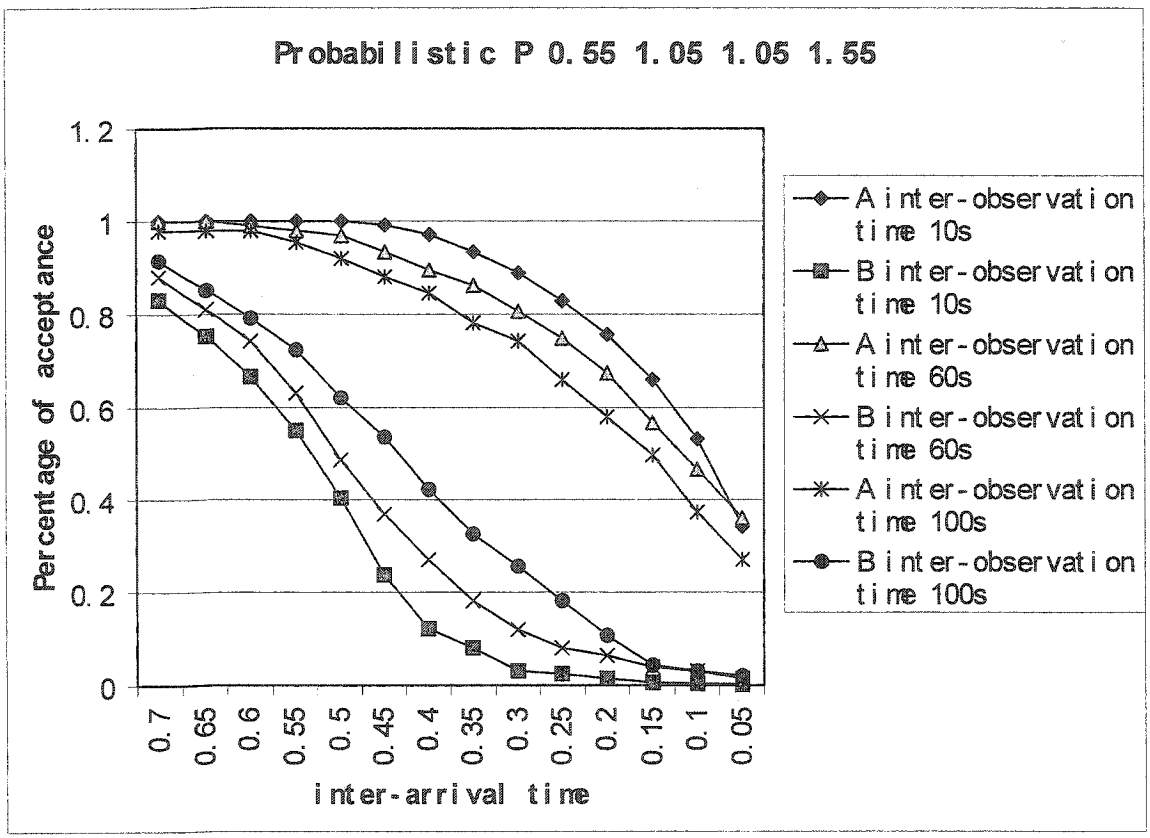


Figure 38. Percentage of users accepted of probabilistic approach for different inter-observation time

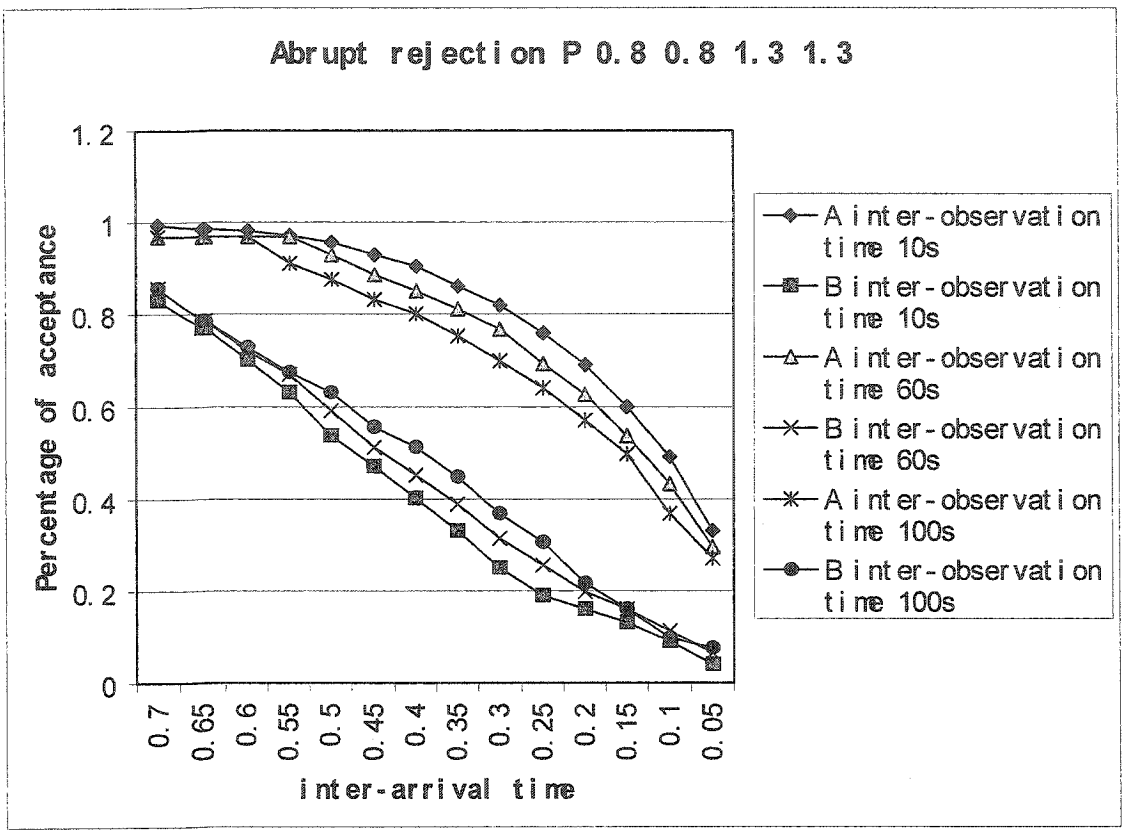


Figure 39. Percentage of users accepted of on-off approach for different inter-observation time

6.2 Using one combined probability function for several user groups

In the previous section, we used different probability function for each user group. With that approach, the system administrator should deliberately choose one probability function for each user group, and we do not have an idea of the overall performance of the system. Take the example of the previous section, even with the given probability function, and known combined rate of incoming users, it is still impossible to tell what the response time the system will be, because the response time also depends on the incoming rates of each of the user groups. In this section we introduce another approach to gradually reject users from different groups using only one probability function, while still keeping the priority of the "elite" user group. With a single probability function, one

can get easily the idea about the overall admission control of the system.

The principle of using a single combined probability function is as follows. First, we compute the probability value P of accepting incoming users (for all classes) from the current response time according to some given probability function (the so-called combined probability function). In this step we do not distinguish between different user classes. Then with the knowledge of the relative incoming rates for the different user classes, we compute the probability of accepting a user from each of the groups. Because there is no way to know in advance the incoming rate of each of the classes in the next inter-observation period, we use the incoming rate measured during the previous inter-observation period as an approximation for the next one. This means that we make an assumption that the user's incoming rates for different groups do not change frequently, which is a normal situation for most web services.

Just like in the previous section, we consider here the problem of provide differentiated service to two groups of users, namely group A with higher priority and group B with lower priority. To formally describe the algorithm, we use some notations as follows:

NA = the number of users from group A in the next observation time interval, (an estimation over the previous record)

NB = the number of users from group B in the next observation time interval, (an estimation over the previous record)

P_{all} = the total probability of accepting users, computed from the probability function $P_{a, b}(r)$, that is $P_{all} = P_{a, b}(r)$

P_A = probability to accept users in class A

P_B = probability to accept users in class B

Since the total number of users accepted in the next observation time interval equals P_{all} * (NA + NB), and it is contributed by two parts, namely the users accepted in group A (estimated number = P_A *NA) and the users accepted in group B (estimated number = P_B * NB). Therefore the following equation should hold:

P_A *NA + P_B * NB = P_{all} * (NA + NB)

Because of higher priority for class A users, P_A should always be bigger than P_B. To be more specific, when P_{all} equals 1, P_A and P_B should both be equal to 1.To keep the balance of the above equation, if P_{all} is decreasing, P_B should decrease before P_A decreases, we assume that P_A can not decrease until P_B gets to 0. More formally, we have the following constraints:

if $P_B > 0$, then $P_A = 1$

if $P_A < 1$, then $P_B = 0$

From the above equation P_A *NA + P_B * NB = P_{all} * (NA + NB), and these constraints we can compute P_A and P_B. We have the following two cases to consider:

Case 1: if NA <= P_{all} * (NA + NB)

P_A = 1 ;

P_B = (P_{all} * (NA + NB) – NA)/NB;

Case 2: if NA > P_{all} * (NA + NB)

P_A = P_{all} * (NA + NB)/NA;

$P_B = 0$;

We see that the values of P_A and P_B depend on the relative ratio of NA and NB.

Let us suppose that NA = α*NB, then the computation of the above two cases can be rewritten as:

if $\alpha/(\alpha+1) <= P_{all}$

$P_A = 1$;

$P_B = P_{all} * (\alpha+1) - \alpha$;

else

$P_A = P_{all} * (\alpha+1)/ \alpha$;

$P_B = 0$;

If the incoming rates of the two user groups are the same, that is α = 1, and the probability function is chosen to be $P_{0.55, 1.55}(r)$, then we will get for P_A and P_B exactly the functions shown in Figure 36 ($P_A = P_{1.05, 1.55}(r)$, $P_B = P_{0.55, 1.55}(r)$); in that case, we would not perceive any difference in the performance between this approach and the approach with the two probability functions in Section 6.1.

But the situations are changed when the incoming rates of the two user groups are not the same. In the following figures, the combined probability function is still chosen to be $P_{0.55, 1.55}(r)$; we show the calculated P_A and P_B when α equals 3 (when NA = 3*NB) and 1/3 (when NA = NB/3), respectively. Clearly they are quite different from Figure 36. When α equals 3, that is, the number of A-users is three times the B-users, the combined probabilistic approach tends to reject group B users very quickly, while group A users are rejected very slowly (compared with the two probability functions approach). This is quite reasonable since we have far more A-users than B-users. Not very surprisingly,

when α equals 1/3, there are more B-users than A-users; it tends to reject group B users very slowly, while group A users are rejected very quickly.

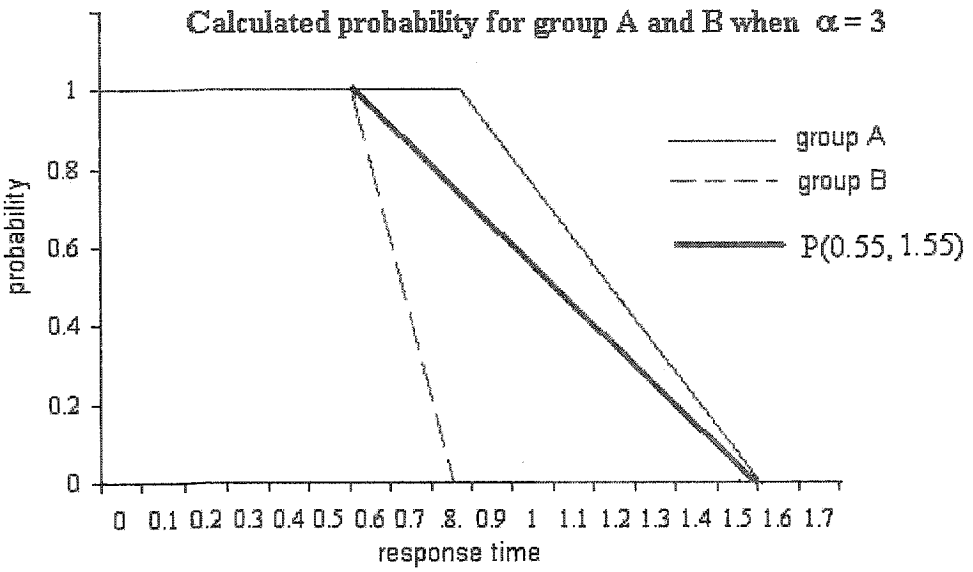


Figure 40. The calculated probability for both groups of users when $\alpha = 3$

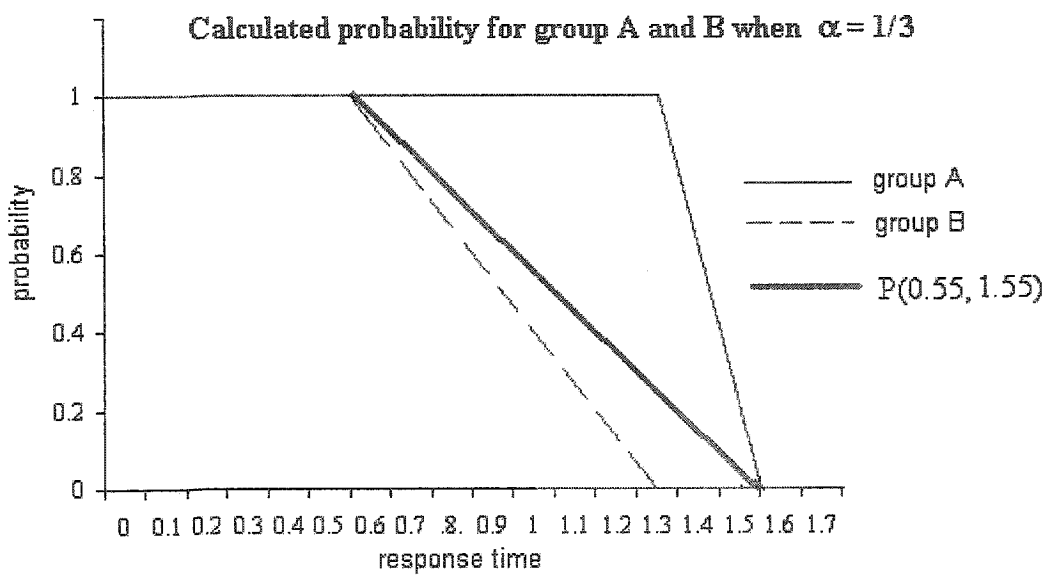


Figure 41. The calculated probability for both groups of users when $\alpha = 1/3$

In the following simulation, we test the percentage of user acceptance over a variety of customer incoming rates (for the customer inter-arrival time ranging from 0.7s to 0.05s), using a single probability function $P_{0.55,\ 1.55}(r)$; the combined probability function approach calculates the total probability P_{all} by $P_{0.55,\ 1.55}(r)$ first, and then computes the P_A and P_B. After running the same simulation for different α (α = 1, 3 and 1/3,), we get the results in the Figures 42 and 43. For comparison, we also plot the results obtained using two probability functions as defined in Section 6.1, and to avoid oscillations we set the inter-observation time as 10 seconds.

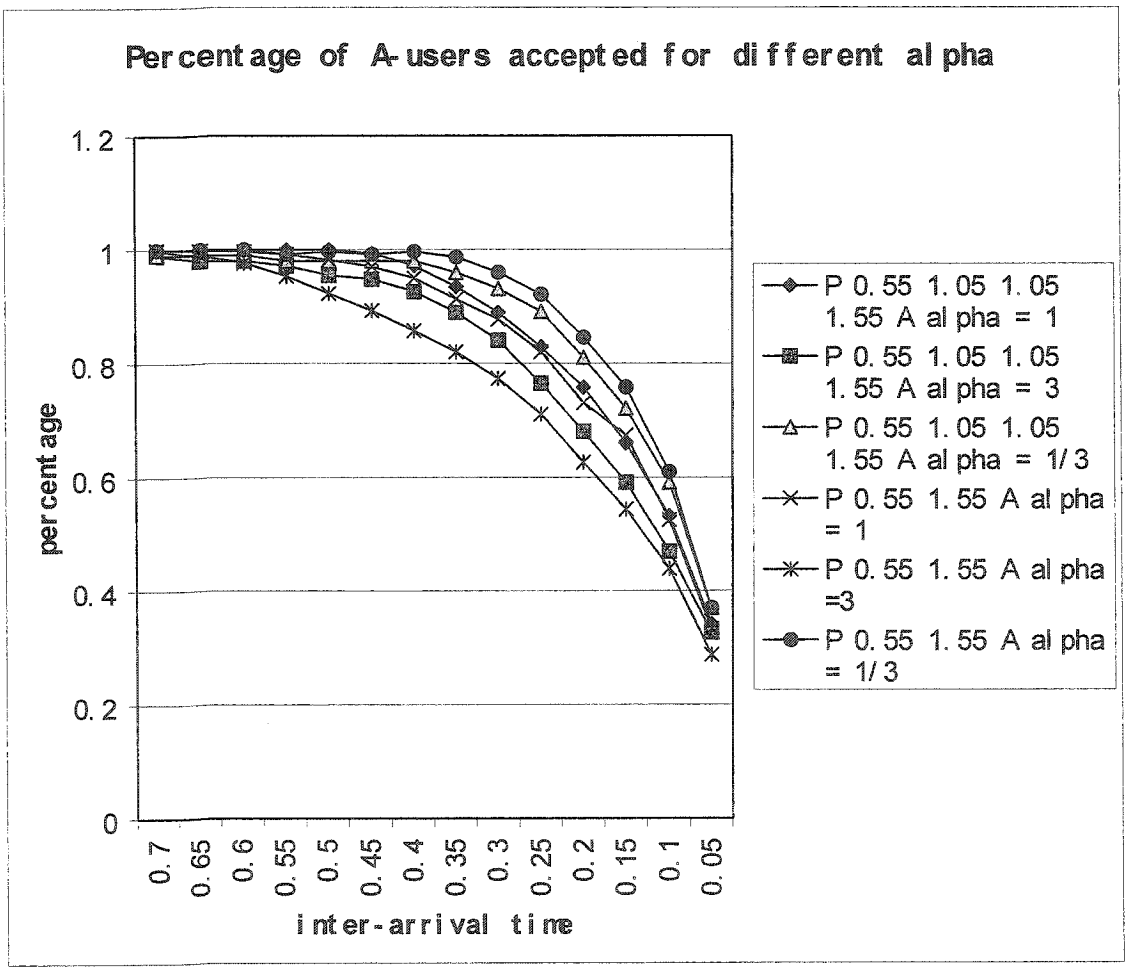


Figure 42. Percentage of A-users accepted (compared with two probability function approach)

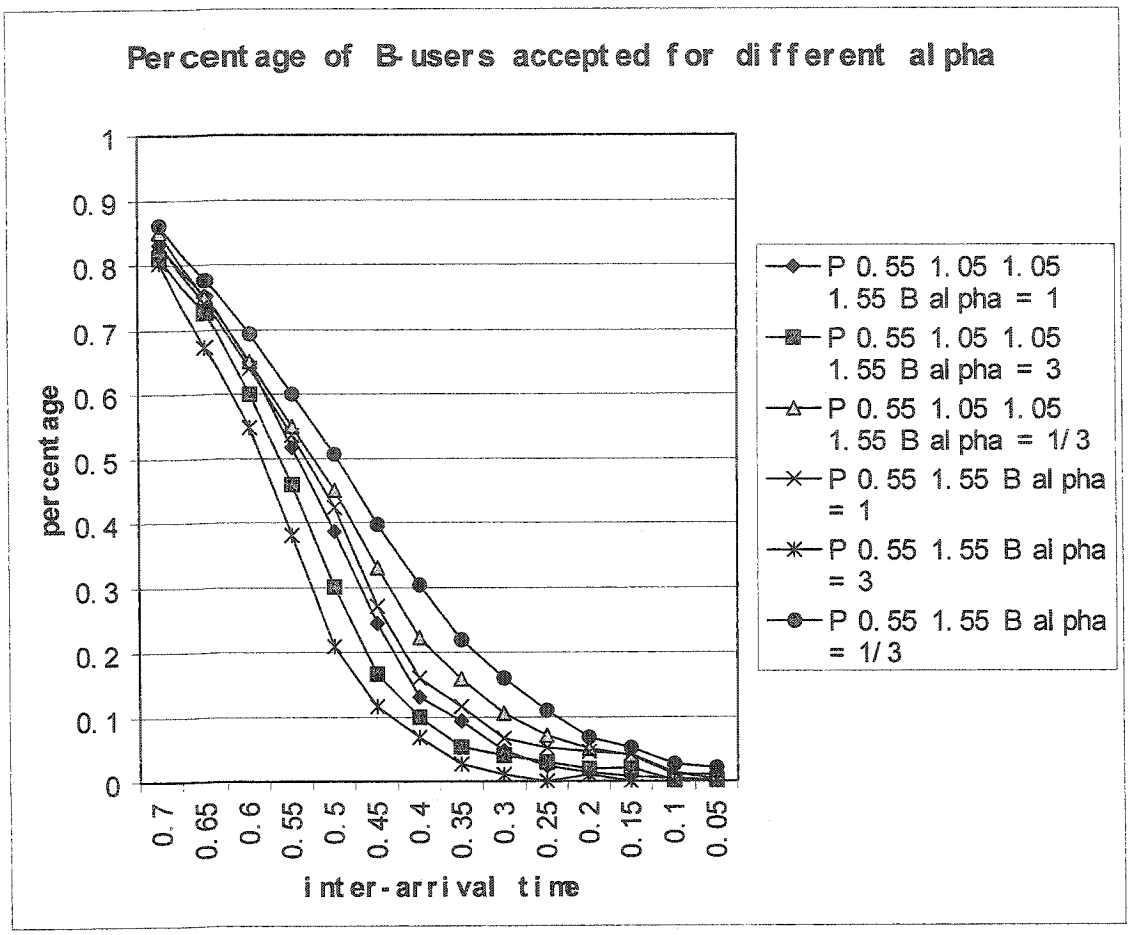


Figure 43. Percentage of B-users accepted (compared with two probability function approach)

As is shown in the above figures, when α equals 1, the percentage of acceptance using one probability function is pretty similar to the result when we use two probability functions for each of the user groups. This is because with α =1, the incoming rates of users in both group are the same, and the calculated acceptance percentages of A-users and B-users are exactly the same as given in the two probability function approach, therefore resulting in a similar performance. When α equals 3, with combined probability function, both percentages of accepting A-user and B-users become smaller due to the calculated acceptance percentages (see Figure 40). Similarly, when α equals 1/3, both percentages of accepting A-user and B-users become larger. Notice that when α equals 3

(or 1/3), with two probability functions, the percentages of accepting A-user and B-users also gets smaller (larger), but not as small (large) as using the combined probability function approach. This means that the two probability functions approach can not adjust to the workload change of different user groups as quickly as the combined probability function approach, which further effects its performance as we will see below.

To see the difference in performance between these two approaches, we compare the resulting response time between two function and combined function approaches, for various values of α. The results are shown in Figure 44.

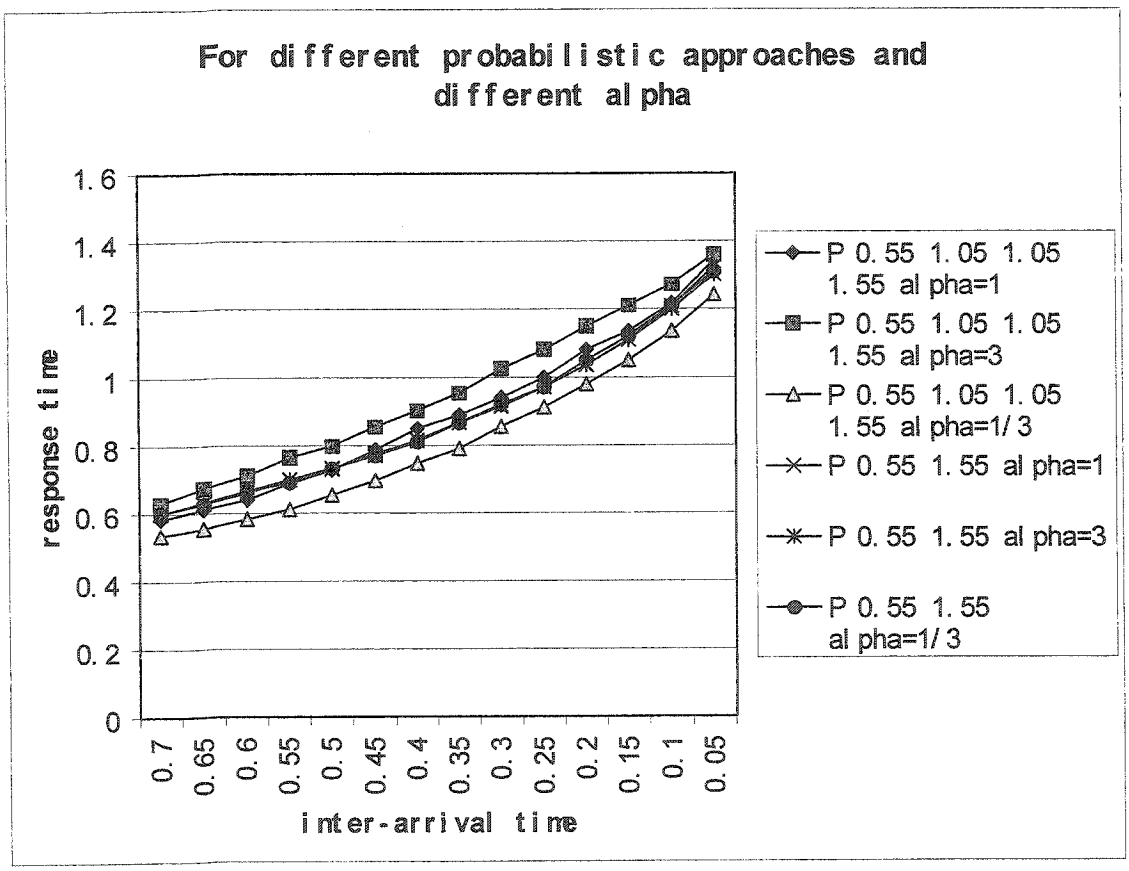


Figure 44. Mean response time of the users using different approaches

As we can see, as the user arrival rate increases, for both approaches, the response time grows very smoothly. For the combined function approach, no matter what the value of α

is, the average response time measured is almost the same. But with the two function approach, the average response time also depends on the value of α: the larger is α, the higher will be the response time of the system.

Regarding these simulation results, we can clearly see the advantage of the combined function approach: even if the arrival rates of A-users and B-users are different ($\alpha \neq 1$), the combined function approach will still result in the same response time as long as the arrival rate of users in total is the same. But for the two probability function approach, the response time also depends on the value of α. If $\alpha > 1$, we will get a slightly higher average response time, and if $\alpha < 1$, we will get a slightly lower average response time

The disadvantage of the combined function approach is: the estimation of the incoming rate of the users in the next observation period is difficult. Here we use the number of incoming users in the previous observation period as an approximation. If the incoming rate of the users changes frequently from one time interval to another, this estimation is no longer accurate; thus the efficiency of this approach may be questionable.

7. Conclusions and future work

7.1 Conclusions

Quality of service of web servers has long been a hot research area. The problem that we consider here is how we can provide satisfactory response time to the clients during the time of heavy workload. We started with the discussion of several related approaches for dealing with the problem and show that although to some extent a reasonable response time can be realized; those approaches cannot deal with the problem of performance oscillations. To solve this problem, we have improved the original server brokerage model described in Salem's paper, and invited a new probabilistic approach to reject user requests, and avoided the oscillations of the server performance.

The major results of our work are the following:

1) Based on simulations, we show the existence of oscillations of the performance of the server, the response time, the number of users in the system and the server utilization etc. We established a theoretical model for the number of users in the system, and using this model we explained that the oscillation is caused by abrupt behavior of the on-off decision-making server selection algorithm that accepts or refuses the incoming user according to some threshold.

2) Based on the theoretical model of the oscillations, we showed that a probabilistic approach that accepts users gradually, will suppress the oscillations, and eventually leads the system to a so-called "stable point".

3) We tested the effect of different probability functions on admission control. Our

simulation experiments reveal that a more gradual probability function has the advantage over a probability function with a sudden change in terms of performance stability. With a more gradual probability function, the amplitude of the oscillation is smaller, and the frequency of the oscillation is also lower.

4) We showed that for a given user incoming rate of users, and a given probability function, decreasing the inter-observation time will improve the oscillation. A smaller inter-observation time period decreases the amplitude of the oscillation. In fact, if the inter-observation time is decreased to some "stable value", there is no perceivable regular oscillation anymore, the whole curve looks just like statistic noise.

5) We explored the behavior when an upper limit to the number of users that can be accepted by the system is given. Simulations show that this approach will eliminate the oscillation only when the user-incoming rate is approaching or exceeding this limit. It has no effect on the oscillations when the workload does not reach that limit.

6) Finally, some considerations are given for this probabilistic approach for a system with several categories of users with different priorities for accessing the system. In that case, we consider two groups of users, namely group A (with higher priority) and group B (with lower priority). The goal is to provide satisfactory response time to the A-user and reject B-user when the workload is high. Our simulation results clearly show that our probabilistic approach has the advantage over the on-off decision approach in providing differentiated service to different user groups.

7.2 Future work

Despite the achievement mentioned above, we realize that there is still much work in this area waiting to be done. Some possible areas that can be improved are listed in the following.

1) We realize that when the workload is not heavy, the system does not need to check the performance of the servers very frequently. This issue is important because frequent transmission of performance data will increase the workload of the broker and the data throughput between the servers and the broker. One could therefore consider whether it is better, if we constantly change the inter-observation time depending on the current user-incoming rate. The higher the rate, the shorter the inter-observation time should be set. In this way, the frequency of the observations by the broker is adjusted to the need rather than being fixed in advance.

2) We also can change the way that the performance data is collected. The broker may collect the performance data by probing the servers instead of the performance data being pushed by the server. We still do not know which way is better. By probing, the broker has the solely control of when to collect the performance data, the servers do not have to care about this, but the workload on the broker will be increased.

8. References

[1] Mohamed-Vall M. Salem, G. v. Bochmann and J. W. Wong, "Server selection for differentiated classes of users", in the proceeding of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002), San Diego, California.

[2] G. Bochmann, B. Kerherve, Mohamed-Vall M. Salem, "Quality of Service Management Issues in E-Commerce Applications", In Electronic Commerce Technology Trends: challenges and Opportunities, Weidong Kou and Yelena Yesha, eds. IBM Press, February 2000, Chapter 14.

[3] G. v. Bochmann, B. Kerhervé, H. Lutfiyya, M. Salem and H. Ye, "Introducing QoS into electronic commerce applications", Proc. of Second International Symposium on Electronic Commerce, April 2001, Hong Kong, China, published as "Electronic Commerce Technologies", LNCS 2004, Springer Verlag, pp. 138-147.

[4] M.-V. Mohamed-Salem, J. W. Wong and G. v. Bochmann, "A scalable load-sharing architecture for distributed applications", Proc. 9th IEEE Conference on Software, Telecommunications and Computer Networks, SoftCom 2001, October 2001, pp. 747-755.

[5] Paul Barford and Marck Crovella, "Generating Representative Web Workload for Network and Server Performance Evaluation", in Proceeding of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 151-160, July 1998.

[6] Website of Mesquite Software, Inc. http://www.mesquite.com

[7] Jerry Banks "Handbook of simulation principles, methodology, advances, applications, and practice" Engineering & Management Press, ISBN: 0471134031

[8] Paul A. Fishwick "Simulation Model Design and Execution: Building Digital Worlds" Pearson Education POD; 1st edition (January 17, 1995), ISBN: 0130986097

[9] Zongming Fei, Samrat Bhattacharjee, Ellen W. Zegura, Mostafa H. Ammar "A novel server selection technique for improving the response time of a replicated service". In Proceedings of INFOCOM 98, 1998

[10] J. C. Mogul. "Network behavior of a busy web server and its clients", Technical Report WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, 1995.

[11] Martin F. Arlitt Carey L. Williamson "Web server workload characterization: the search for Invariant (Extended version)" 1996 ACM SIGMETRICS Conference, Philadelphia, PA, May 1996.

[12] Michele Colajanni, Philip S. Yu and Daniel M.Dias "Analysis of task assignment policies in scalable distributed web-server systems", published in IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 6, June 1998

[13] Computer Networks: A Systems Approach, 3rd Edition by Larry L. Peterson, Bruce S. Davie, Morgan Kaufmann, ISBN: 155860832X

[14] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, Z. Fei: "Application Layer

Anycasting", In Proceedings of INFOCOM 97, 1997

[15] Virgilio Almeida, AzerBestavros, Mark Crovella, and Adriana de Oliveira. "Characterizing reference locality in the WWW". In Proceedings of 1996 International Conference on Parallel and Distributed Information Systems (PDIS' 96), pages 92-103, December 1996

[16] M. F. Arlitt and C. L. Williamson. "Web server workload characterization: The search for invariants". In Proceeding of the ACM SIGMETRICS '96 Conference, Philadelphia, PA, April 1996.

[17] Tim Bray, "Measuring the web". In Fifth International World Wide Web Conference, Paris, France, May 1996.

[18] C. A. Cunha, A. Bestavros, and M. E. Crovella. "Characteristics of www client-based traces". Technical Report TR-95-010, Boston University Department of Computer Science April 1995

[19] M. E. Crovella and A. Bestavros. "Self-similarity in world wide web traffic: Evidence and possible causes". In Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, May 1996.

[20] G. K. Zipf. "Human Behavior and the Principle of Least-Effort", Addison-Wesley, Cambridge, MA, 1949

[21] Mohamed-Vall M. Salem, G. v. Bochmann and J. W. Wong: "Scaling Server

Selection Using a Multi-Broker Architecture," in the proceeding of the 23rd International Conference on Distributed Computing Systems Workshops, pp. 934-939, May 19-22, 2003, Providence, Rhode Island, USA.

[22] Katsuhiko Ogata "Modern Control Engineering" Prentice Hall, 4th edition (November 13, 2001), ISBN: 0130609072

[23] Jacqueline Wilkie, Michael Johnson, Reza Katebi Control Engineering "An Introductory Course" Some pages can be downloaded from Palgrave Macmillan at http://www.palgrave.com/science/engineering/wilkie/sample/0333_77129Xcha05sample.pdf

[24] S. Floyd and V. Jacobson. "Random early detection gateways for congestion avoidance". IEEE/ACM Transactions on Networking, 1(4):397--413, August 1993

[25] Vinod Sharma and A. Gupta, "Performance analysis of routers with TCP and UDP connections with priority and RED control", in the proceeding of Conf. ICCC 2002.

[26] Vinod Sharma, J. Virtamo and P. Lassila, "Performance analysis of a router with RED control", Probability in the Engineering and Informational Sciences, Vol.16, 2002, 367 - 388.

[27] Douglas Comer, "Internetworking With TCP/IP Volume 1: Principles Protocols, and Architecture", 3rd edition. PHI

[28] "Computer Networks", Fourth Edition -- by Andrew S. Tanenbaum; Prentice Hall, ISBN: 0130661023

[29] "Computer Networking: A Top-Down Approach Featuring the Internet" by James F.

Kurose, Keith W. Ross, James Kurose, Keith Ross, Addison-Wesley Publishing, ISBN: 0201976994

[30] P. Karn and C, Partridge. "Improving Round-Trip Time Estimates in Reliable Transport Protocols". In ACM SIGCOMM'87. ACM, 1987.

[31] Ludmila Cherkasova, and Peter Phaal: "Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites". In Proceedings of Seventh International Workshop on Quality of Service, IEEE/IFIP event, London, May 31-June 4, 1999.

[32] N. Vasiliou and H. Lutfiyya, "Providing a Differentiated Quality of Service in a World Wide Web Server", Performance Evaluation Review, Volume 28, Number 2, pp.22-27

[33] N. Vasiliou and H. Lutfiyya, "Managing a Differentiated Quality of Service in a World Wide Web Server", in Integrated Network Management, Volume III, May 2001.

[34] Jussara Almeida, Mihaela Dabu, Anand Manikutty and Pei Cao, "Providing Differentiated Levels of Service in Web Content Hosting", Proc. Of First Workshop on Internet Server Performance, ACM SIGMETRICS 98, June 1998.

[35] V. Cardellini, M. Colajanni, P. S. Yu, "DNS dispatching algorithms with state estimators for scalable Web-server clusters", World Wide Web Journal, Baltzer Science, Vol. 2, No. 3, pp. 101-113, Aug. 1999

[36] Daniel A. Menacse, Virgilio A. F. Almeida, Rodrigo Fonseca, and Marco A. Mendes,

"Resource Management Policies for E-commerce Servers", Second Workshop on Internet Server Performance, in conjunction with ACM SIGMETRICS 99 / FCRC, Atlanta, GA, May 1st, 1999.

[37] D. Kristol and L. Montulli: "HTTP State Management Mechanism", RFC 2109 (February 1997)

[38] C. Patridge, T. Mendez, and W. Milliken: "Host Anycasting Service", RFC 1546 (1993).

[39] V. Cardellini, M. Colajanni, P. S. Yu, "Geographic load balancing for scalable distributed Web systems", Proc. of 8th Int'l Symp. On Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'00), San Francisco, CA, Aug./Sep. 2000. IEEE Computer Society.

[40] D. Andresen, T. Yang, V. Holmedahl, O. Ibarra. "SWEB: Towards a Scalable World Wide Web Server on Multicomputers", in Proceedings of IPPS '96 - 10th Inter. Parallel Processing Symposium, IEEE. Hawaii, April, 1996.

[41] Katz. E. D., Butler, M., and McGrath, R.A. "A Scalable HTTP Server: The NCSA Prototype". Computer Networks and ISDN systems, First International Conference on the World Wide Web, Elsevier Science BV.

[41] R. Engel, V. Peris, E. Basturk, V. Peris, and D. Saha. "Using IP Anycast for Load Distribution and Server Location". In Proc. Third Global Internet Mini-Conference in conjunction with Globecom '98, November 1998.

[42] Katabi, D., and Wroclawski, J., "A Framework for Global IP-Anycast (GIA)", In Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden, August 2000.

[43] Cisco Distributed Director,
http://www.cisco.com/warp/public/cc/pd/cxsr//dd/index.shtml.