



UNIVERSITAT POLITÈCNICA DE CATALUNYA

Escola Superior d'Enginyeria de Manresa

The Problem of Peak Loads in Web Applications and its Solutions

June 7, 2019

bachelor's thesis submitted by

CARLA PRAT GARCÉS

in partial fulfillment of the requirements for the

DEGREE OF ICT SYSTEMS ENGINEERING

Advisor: Sebastià Vila Marta

Abstract

In this thesis we will analyse the problems that peak overloads cause in web applications and what their possible solutions can be. Overloads have become more common as the Web has extended to different areas and Internet becomes more and more accessible. E-commerce being one of the worst scenarios and the most common where overloads can happen. This analysis will be based in some of the methods known nowadays. The main purpose of this thesis is to collect and compare them with each other in order to obtain a guide that will help to choose which one would adapt better to a specific application type as well as to show tests done in some of the cases.

Resum

En aquesta tesi analitzarem els problemes que els pics de demanda causen en les aplicacions Web i quines possibles solucions podem trobar. Les sobrecàrregues s'han convertit en un problema recurrent en diferents àrees a mesura que Internet s'ha anat fent més accessible. Les pàgines de comerç *online* són un dels pitjors casos on es poden produir sobrecàrregues. L'anàlisi que es durà a terme estarà basat en els diferents mètodes que es coneixen en l'actualitat que tracten aquesta problemàtica. L'objectiu principal

d'aquesta tesi és recolectar i comparar aquests mètodes de forma que s'en pugui fer una guia per triar quin d'ells és més adequat segons el tipus d'aplicació que tinguem. A més també podreu trobar els tests que s'han realitzat en alguns d'aquests casos.

Contents

Abstract	i
Resum	i
1 Introduction	1
1.1 Objectives	2
1.2 Preliminary concepts	3
1.2.1 Types of Web applications	3
1.2.2 E-commerce Applications	4
1.3 Overview	6
2 Admission Control	9
3 Session-based admission control	11
3.1 What is a session?	11
3.2 SBAC characteristics	12
3.3 Comparing an overloaded server to one with SBAC implemented .	13
4 Hybrid Admission Control	19
5 Predictive Admission Control	21

6 Profit-Aware Admission Control	23
6.1 Requests Classification	23
6.2 Defining the Profit-Aware Modules	24
6.3 Comparisons between four different test beds	27
7 Combined LIFO-Priority Scheme for Overload Control	31
7.1 A Comparison between Single Queue FIFO, 8 Queue Always LIFO and 8 Queue LIFO-Priority	32
8 Load Balancing	35
8.1 How to load balance	36
9 Comparing the methods	39
10 Conclusions	43

1 Introduction

This thesis aims to survey the techniques available to manage peak overloads on web applications. The chosen topic intends to give light to a subject about which population does not talk much. The effects a load peak has on users, and for that matter, on applications, can be crucial. In a web application, if an unexpected load hits the server, it is most likely that the services offered by the application start to fall. Even if an important transaction, such as paying, is being done. As users ourselves, when these kind of problems arise, we can not help but think that our payment will be processed wrongly or even worse, processed twice. This recurring problem happens especially in well known commercial web sites as, for example, ticket sellers sites. This made us wonder why companies had not tried to find a better solution to this recurring problem. This issue is most commonly solved by hiring extra server space or buying more servers in order to redirect requests and avoid overloading a single server. But sometimes this is not enough and one of our goals is to find a way to solve these peaks without hiring extra space.

Elseways, there are also repercussions on the server end. If a web application has a periodicity on its overloads, it will probably be equipped with the necessary amount of extra servers in the event of a peak happening. However, not all web applications cover the same public and might not face this situations often. Each application is unique depending on the public it reaches. One might have peaks

every time a product is launched and another one when tax payments need to be done. For this reason, analysing the Web application we are working with is crucial. It allows us to rethink the existing methods in order to avoid renting new space and making the most out of the one we already have. Which is why our aim in this thesis is to find a way for them to exploit the maximum capacity of their server.

1.1 Objectives

Our main objective in this thesis is to create a guide that includes some of the most developed methods and algorithms for load management, in order to make it easier for web application developers to find better alternatives to traditional load managing algorithms. This has been of important relevance as finding valuable articles about load managing techniques has been a complicated task due to the enormous amount of information online.

To better dissect our main objective, here is an expanded list of the main bullet points we want to achieve in this project:

- Build a simple yet detailed enough explanation on how each method works.
- Compare the methods in order to establish pros and cons for different applications.
- Establish a criteria to be able to choose the best method for a certain type of application.
- Collect some methods in one place for easy access to information.

1.2 Preliminary concepts

According to [1], *a web application or web app is a client–server computer program in which the client (including the user interface and client-side logic) runs on a web browser*. As time has gone by, web sites have evolved into web applications that is why we are not to confuse them. The main differences between a dynamic web site and a web application are, most of the time, where they run, whether it is on the server or browser side. When a web site is considered a web applications is due to its similarities with a desktop or a mobile application.

1.2.1 Types of Web applications

Web applications can be divided in different types in order to reach towards different areas of business. As seen in [2], web applications can be categorised in six different types. However, this thesis will only be considering the following three:

- *E-commerce applications*. In charge of the process of buying and selling by electronic means, e-commerce applications refer to both online retail as well as electronic transactions, as [3] details.
- *Portal web application*. [4] states that portal web applications offer Web-accessible, interactive tools on a secured website that delivers both related and unrelated applications, services and links. They also provide data in an easy understandable format. As well as modify or manipulate said data, and communicate with companies or individuals about it.
- *Content management systems*. Applications that are used to manage web content and that also allowing multiple contributors to create, edit and publish are considered content management systems; as [5] declares.

From the aforementioned, we will mostly work with e-commerce as it is the more complex when talking about request management. Nonetheless, if a method cannot be applied to e-commerce it might be suitable for portal web applications or content management systems. More information on E-commerce will be handled in the next section.

1.2.2 E-commerce Applications

E-commerce applications manage both the process of buying and selling online. It includes online auction sites, internet banking, online ticketing and reservations, business to business (B2B) transactions and business to consumer (B2C), etc.

E-commerce has grown to a certain point where when a load higher than expected enters the server, the consequences can be fatal. One of the most important parts in e-commerce apps are payment transactions. Operating those correctly so that the user does not feel like something could go wrong is a delicate situation. This is where quality of service comes in.

Quality of Service on E-commerce Applications

Quality of service (QoS) is the description or measurement of the overall performance of a service, particularly the performance seen by the users of the network, as stated by [6]. Referencing report [7], quality of service is an important factor for user experience and also for the different system resources used by E-commerce applications. *The goal for QoS management is to assure the controlled sharing of these resources among different users according to certain management policies.*

To quantitatively measure quality of service, several related aspects of the network service are often considered. They differ depending on the user or the inner point of view.

From a user's perspective, this are the different parameters that change QoS:

- *Response time*

Time between the request is sent and the time a response is provided to the user.

- *Availability*

Percentage of time the server is available during an observation period.

- *Servability*

Percentage of time the server is available and can accept the user's request. To guarantee a certain quality of response time, the system will refuse new requests when the response time of the system exceeds a certain limit.

- *System throughput*

Number of user requests handled by the system per second. This is important as the response time of a given system increases as the system throughput increases.

On the other hand, the QoS parameters from the different system components within our architecture are the following:

- *Network propagation delay*

Time between sending a packet to it being received by the destination.

- *Network access capacity*

Determined by the network access link, it is the maximum throughput by which a computer can send data over the network.

- *Effective network maximum throughput*

Maximum throughput that can be obtained between two computers over a network. It is limited in most cases by the flow control mechanism of the TCP protocol.

- *Response time of the query process and throughput*
Number of queries processed per second.
- *Processing delay, queuing time and throughput in Web server and application process*
- *Availability of different servers and system components in them*

1.3 Overview

This thesis is going to be structured in a way we can talk about both theoretic and/or practical methods. Not all the methodologies contained in this document will have been tested in order to be implemented. Regardless, all of them will include a theory tear-down on each of them along the lines of how requests are managed and how overloads will be controlled. We will also touch on comparisons between the different types of Web application and how we can discern an advantage for a certain type of application but a disadvantage for another type.

The structure of the document will start by introducing some basic concepts and explaining the initial goals of this thesis in chapter 1 and then admission control and its policies in chapter 2 will be explained for further use in the next chapters. From there the methodologies will begin. In chapter 3 you will find theory about Session-Based Admission Control and some test graphs explained. Followed by Hybrid Admission Control in chapter 4 and Predictive Admission Control also with some test graphs in chapter 5. More on Profit Aware Admission control will be found in chapter 6 with some test graphs too preceded by Combined LIFO-Priority in chapter 7. And to end the methods, Load Balancing will be explained in chapter 8. To conclude the document, a comparison between all the methods will be found in chapter 9 followed by the conclusions of this project in

chapter 10. The bibliography used can be found at the end of this document for further information on the discussed topics.

2 Admission Control

As [8] states, admission control (AC) is a validation process where a check is performed before a connection is established to see if current resources are sufficient for the proposed connection. AC has two important strategies: responsiveness and stability.

Responsiveness, or a fast reaction, is very important when during previous time intervals, the server's load has been consistently high and has exceeded its capacity. In the events of the previous behaviour happening, it is important the admission control policy switches as soon as possible to reject and control the new incoming traffic.

Withal, if the new received traffic is still under a manageable load, a slow reaction, also called stable policy will be a desirable property. As it takes into account some of the load history, it helps to maximize the server throughput and will not reject the new incoming traffic.

Unfortunately, these properties are contradictory. Responsiveness makes a more restrictive policy as it aims to reduce the number of aborted sessions and also wants to achieve higher service levels at a lower server session throughput. Especially when the server manages heavy loads but is yet to be overloaded. However, stability takes into account the history behind the server's load. If the total server load has not surpassed the server capacity, stability will allow better server session throughput. But as a less restrictive policy on rejection, the rate

of aborted sessions will grow and we will start getting poorer session completion characteristics.

Bearing this in mind, in the following chapters, we explain in depth some of the methods to control peak loads. They might vary the way they want their admission control to be performed.

3 Session-based admission control

The session-based admission control (SBAC) was introduced by L. Cherkasova and P. Phaal in [9] and [10] in order to improve web QoS for commercial web servers. This admission control mechanism is based on the server CPU utilization and works with sessions instead of requests.

3.1 What is a session?

A session is composed by multiple requests, all coming from the same user and transaction. Sessions are commonly lengthier in time than individual requests especially if they involve purchase transactions. This comes from analysing how a user interacts with an e-commerce web site. For example buying a product. To realize that transaction many requests need to be sent in order to select the product, ask for shipment and total costs, pay for the products and get a confirmation saying everything was processed correctly.

Let's consider we were to realize this transaction in a session with unrelated requests. The first steps of the transaction are already processed and a load that exceeds the server capacity is to be processed. This load includes, among others, our next request. The server, seeing as it can not process all of the requests, starts refusing connections. One of the dropped requests was our attempt to pay for the selected product, and as the server did not know whether it was a new or

an already existing connection, our session will have been aborted.

If our requests were not involved in a payment transaction, dropping those requests would not be as much of a deal. But that is not our case. A quality of service needs to be provided in order to assure the users, if they were to use our site, that transactions would be completed correctly.

3.2 SBAC characteristics

The way SBAC works is by measuring the server utilization during predefined time intervals (ac-intervals). To compute the observed utilization we will use the last interval's measured utilization and some data that characterizes the utilization of the server in the recent past. If the observed value surpasses a specified threshold, in the next interval, the admission controller will start rejecting new sessions and will only accept requests from already started sessions. When the value drops below the threshold, the server will change its policy and the next time interval will begin to admit new sessions again.

From now on the word workload will be appearing frequently making reference to a certain amount of sessions or requests that are used as a set of data for a simulation or the actual quantity of sessions and requests arriving to the server. As well as the word load, which refers to the amount of requests a server receives in order to process them later.

As mentioned before, one QoS requirement on session-based workloads is the fair chance of completing any accepted session regardless of its length. Another desired requirement would be to not accept a request when the session has just started instead of doing so in the middle of it. This will make the server's work more focused in completing already begun sessions instead of wasting it on sessions that could potentially be aborted half way through.

Prioritizing sessions is also an option that benefits the server when a peak load happens. By doing so, it helps us decide which user will be more beneficial to the site. If a user has visited the website before, they will have a higher priority than one who is there for the first time. As well as a customer that is completing a purchase will receive a higher priority than one that is just browsing the website. This service-level policies can be applied only at specific times, such as when traffic is increasing to a heavy level or when a peak happens.

In report [9], in order to establish a correct workload space the server access logs from a specific site were analysed. When analysing the session length distribution, it clearly showed that sale sessions are much longer than normal sessions. Being the average length of sale sessions 2.5 times longer than a normal ones. Which is why longer sessions should be taken into account instead of being marginalized, as they significantly impact sales in commercial sites.

3.3 Comparing an overloaded server to one with SBAC implemented

In order to analyse the server performance, a simulation was developed where different situations were put to the test by tweaking different parameters. The next graphs show the characteristics of an overloaded web server. In the Figure 3.1a we can see the throughput percentage of the server in completed sessions for a range of different load percentages. As well as in Figure 3.1b the server utilization for a range of different load percentages too. Both of them comparing what would happen with different session lengths. For a load percentage higher than 100 percent, it would imply the load is over the server capacity. The average session lengths used will be of 5, 15, and 50 requests per session.

While observing graph Figure 3.1a, the reason we see shorter sessions have

such a drop in throughput has a lot to do with the fact that shorter sessions are more likely to be completed. But as the sessions are shorter, it means much more of them will get aborted in order to obtain a much quantitative value of the throughput in exchange of low quality in session acceptance.

In the next graph Figure 3.1b, the percentage of server utilization does not differ much from longer to shorter sessions. As the load grows we see a decrease in server utilization. If the session aborted is long, it can create enough unused resources to serve several shorter sessions. But if it is a short session, we would often need several of them to be aborted before we had enough unused resources to complete a new session.

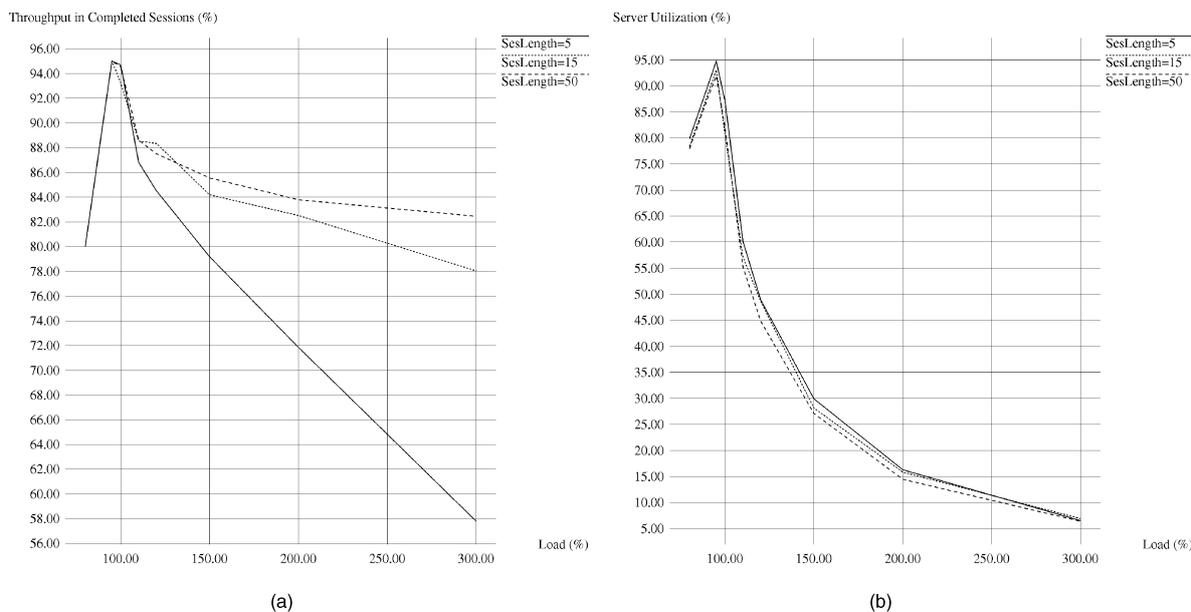


Figure 3.1: (a) Throughput in completed sessions for an overloaded server. (b) Server useful utilization of processing sessions which end up completed.

When this same analysis is done in a server that has SBAC using ac-responsive

strategy. The results get significantly better, specially for longer sessions. The session length of 5 is not representative in a e-commerce environment as even the shorter sessions tend to be longer, but it was included in order to understand possible limitations in the method. The same can be said about loads of 300 percent. If a web server has to deal with loads bigger than 200 percent it would already be a better idea to increase the sites capacity with additional servers. However, to understand the general behaviour and compare it to the results above it will be included.

If we focus on the first graph Figure 3.2a we can see a throughput improvement on longer sessions compared to shorter ones. It is worth mentioning that in order to get such results, some sessions needed to be rejected. That is what Figure 3.2b shows us with the percentage of aborted sessions for different load percentages. We see how the abortion level on longer sessions does not increase and stays steadily at zero percent. This is due to the requirement of no sessions aborted out of the accepted ones. Unfortunately for short sessions, the mechanism still admits more sessions than it can process and, as a consequence, we experience that growth on the percentage of aborted sessions. This is due to the average session length, the shorter it is, the higher the number of sessions that will be generated by the clients and therefore, accepted by the server during the ac-interval.

For example, if our ac-interval was of 5 seconds, our listening queue was of 1024 requests and our web server was accepting new sessions for a load of 300 percent, during a second it would accept around 600 new sessions, not forgetting the ones already being processed.

The amount of aborted sessions in this scenario are due to the listening queue of the server overflowing. If we reduced the ac-interval of the admission control mechanism to 0.5 seconds we would see a drop of 7.5%. This scenario could be fixed by reducing the ac-intervals to assure that the number of aborted sessions

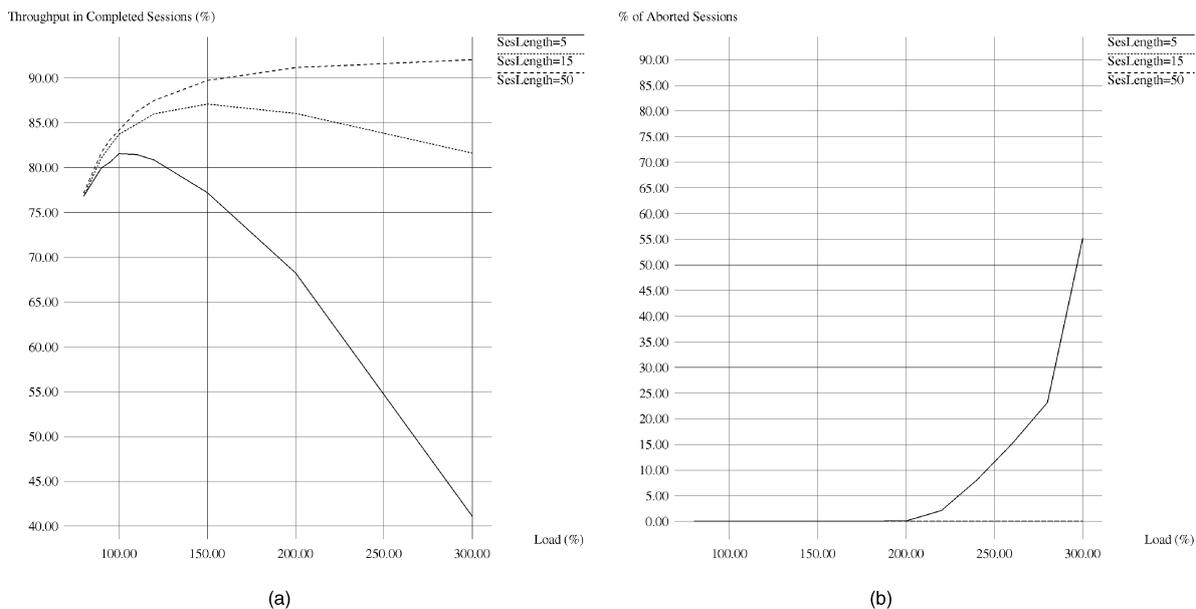


Figure 3.2: Web server performance with SBAC: (a) Throughput in completed sessions. (b) Percentage of aborted sessions admitted from those admitted for processing.

will not grow in case the load is twice as big as what the server can handle. However, we can not always reduce the ac-interval as some CPU utilization is updated on a base of a 5 second interval. However, as the graph below Figure 3.3 shows, an ac-interval of 5 seconds is unsatisfactory for workloads of different average session lengths.

This demonstrates that CPU utilization-based implementation of SBAC can brake under certain circumstances not working properly. The decision to accept or reject a session is made within the limits of the ac-intervals. Which is why when a decision is made, we can not change the mode until the next interval comes. In this slot of time we could have a very high load coming and the number of accepted sessions would be much greater than the server capacity could support. This leads to future aborted sessions and poor session completion characteristics.

This can be avoided by estimating the number of sessions a server is able to

process adapting it to a specific e-commerce site as well as the specific pattern of mixed transactions a customer would execute in the site. Also the number of sessions should be fixed during the time interval and no more sessions should be admitted.

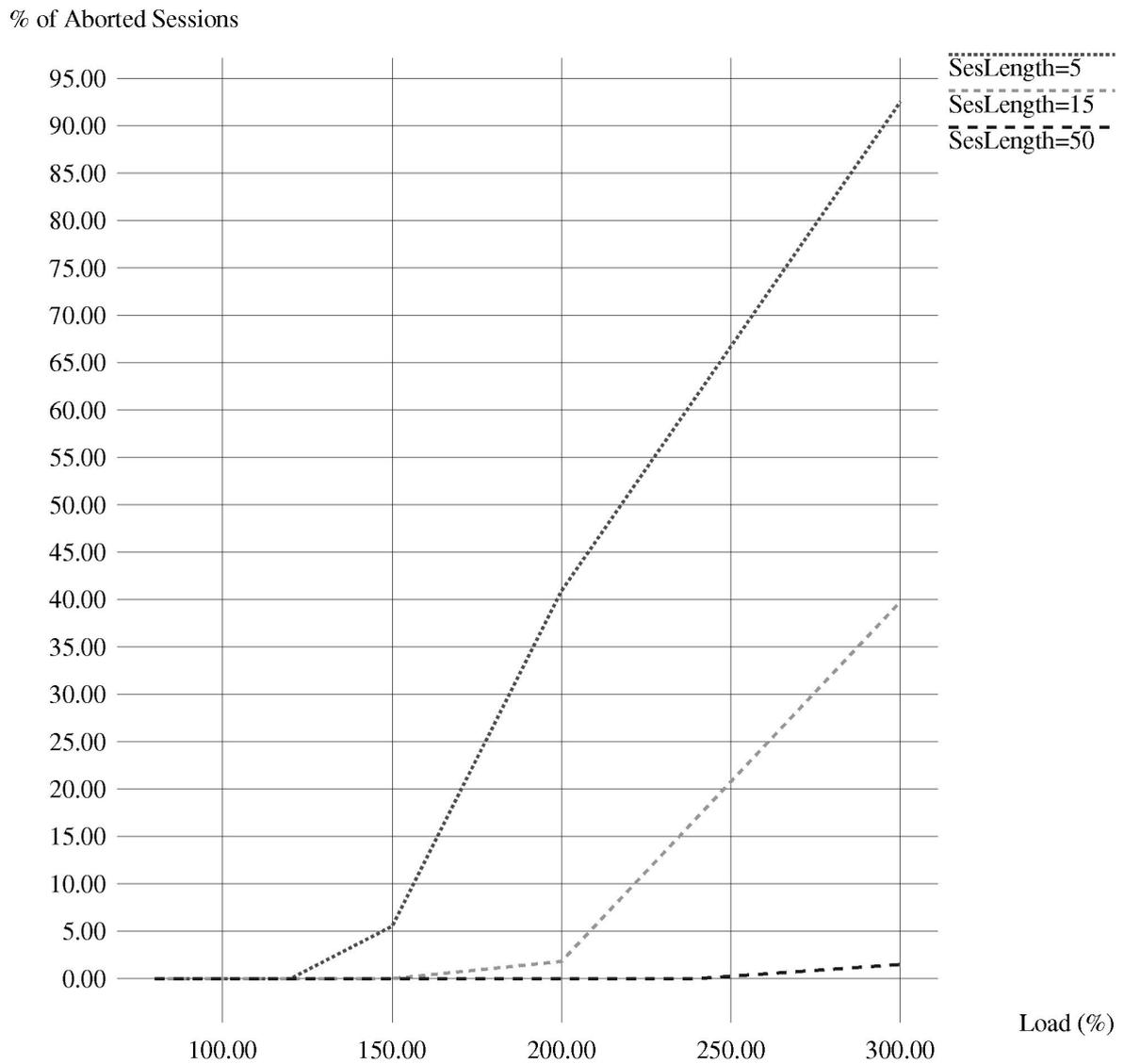


Figure 3.3: Percentage of aborted sessions from admitted for processing for server with admission control, AC-Interval = 5sec, workload with average session length of 5.

4 Hybrid Admission Control

The hybrid ac-strategy was designed to answer how "goodness" and efficiency could be measured once a web server is augmented with an admission control mechanism. Also introduced by L. Cherkasova and P. Phaal, [9, 11], this method uses two important values: the percentage of aborted requests and the percentage of refused-connection messages sent by the server when the listen queue is full. Aborted requests indicate a unsatisfactory level of service and can also lead to aborted sessions, which could be useful as a degrading server performance warning sign. The server can determine this aborted requests based on the client-side closed connections. Differently, refused connections are a sign of overload in the server and poor session performance.

When both the number of aborted requests and refused-connection messages are zero, the admission control mechanism has used an adequate ac-function that is able to cope with the current workload and traffic rate. Contrarily, when the number grows it reflects that the ac-function needs to be more responsive. However, if the percentages are zero it could also mean the ac-function is too restrictive and the server is rejecting sessions that could be handled.

The idea behind the hybrid ac-strategy involves adjusting the admission strategy to be more responsive when aborted requests or refused connections are observed. During a time long enough to observe change, the strategy is consistent until there are no aborted requests or refused connections. Then the strategy is

adjusted to be less responsive. This adjustment to the admission strategy tries to move closer to ac-stable until the necessity to move to ac-responsive is signaled by the number of aborted requests or refused connections growing.

It is interesting to mention that in real life, all the aborted requests might not happen due to the low level of service. There is a certain percentage of aborted requests with no reason as to why they were aborted. This strategy should be defined bearing in mind this percentage and making it react above it.

5 Predictive Admission Control

Predictive admission control's goal is to minimize the number of aborted sessions by rejecting to connect to them when the server will not be able to support them. This method is also introduced by L. Cherkasova and P. Phaal on [9, 10], as a way to achieve a better solution to the problem in graph Figure 3.3 of the SBAC method.

As stated in section section 3.3 the only way to avoid the problem of ac-intervals being way too long or not being able to be as short as we would need them, we need to find the estimated rejection overhead and the number of sessions a server is capable to process per time interval. To correctly estimate this number we have first need to calculate the rejection overhead. We can find this with the following formula:

$$RejectionOverhead(\%) = \frac{S_r \cdot (Load - 1)}{SesLength - 1} \quad (5.1)$$

Being S_r the request rate a server can process per second, $Load$ the session arrival rate and $SesLength$ the average session length.

The rejection overhead varies depending on the average session length and applied load. The higher our load is and the shorter the session length, the higher the rejection overhead. This formula will hold while the values of $SesLength$ and $Load$ fulfill this condition: $Load - 1 \leq SesLength - 1$.

Once we have an estimated value for the rejection overhead we will be able to

predict the number of sessions a server can process per interval.

Predictive admission control works in the following way. For each admission control interval of time (ac-interval), it predicts the amount of sessions the server will be able to handle. The server will accept that number of requests and reject any new session that goes above that number. This is done with the following formula:

$$y = \frac{S_r \cdot (SesLength - Load)}{SesLength \cdot (SesLength - 1)} \quad (5.2)$$

Obtaining the maximum request rate S_r that a server can support under a specific workload is easy. A counter of accepted requests divided by a counter of accepted sessions will give us the average session length $SesLength$. And by counting the number of arriving sessions we will have the $Load$.

With this equation we will be able to decide weather to use one or another admission control strategy. To find the server capacity in sessions S_s we will just need to do the following division:

$$S_s = \frac{S_r}{SesLength} \quad (5.3)$$

The efficiency of this strategy will be subject to the accuracy of our prediction. If inaccuracies are taken care of this strategy works much better. Managing the rounding of fractions can better the efficiency. Although a more serious inaccuracy is the erroneous prediction of the load as the prediction is based on the previous ac-interval. To solve this, the ac-interval has to be reduced or increased in order to stop it from happening in the next interval. But on this ac-interval we might have accepted some sessions we should not have.

6 Profit-Aware Admission Control

Profit-aware admission control is described by Chuan Yue and Haining Wang on [12] as a way to protect E-commerce Web sites from overloads. The key feature of this method is to keep track of inter-session purchase records of clients and utilize them to admit new sessions. The inter-session record of a client is the information about said client having previous purchases in a site or not. The metric they use in this method is the buy-to-visit (B2V) ratio. This ratio gives a different percentage depending on the history of a client with a web site. For example, a client without a history of purchases in a site will have a lower percentage of B2V ratio, but a client that has already purchased something will have a higher percentage.

6.1 Requests Classification

The major challenge when designing this method is how to classify customers in an efficient and reliable manner. E-commerce Web sites rely on login authentication as a method to identify a user but this is also one of the motives of high cart abandonment. That's why in some cases registration and login are optional. We can also use per-session cookies that get stored in memory to identify requests from the same user within a session (persistent cookies), although these can get cleared and or disabled.

Even though those were great ways to identify clients and classify them, the

risk of not "rewarding" those clients due to the clearance of cookies or lack of registration can affect the main goal of this method. That is why using IP addresses to identify customers is more efficient. IPs come in each request and this way we do not have to worry about the client doing any modification or extra step on their side.

One of the worries of using IPs is the inaccuracy of dynamic addressing, NAT (Network Address Translation) boxes and proxies. However there are studies, like [13], which demonstrate that dynamic renumbering from DHCP (Dynamic Host Configuration Protocol) happens on the order of days. In addition, even if DHCP changes the IP address of the end-host it will not change the network identification. Even if proxies group a larger amount of geographically diverse users, some techniques have been developed in order to detect the proxy in real-time and let the web site make more accurate decisions on the client identification.

It might seem a large amount of IP's that need to be stored, but the profit-aware mechanism only needs to store the IP's of those customers that ended up purchasing in the Web site. That is why, network identification prefixes will be stored to significantly condense the IP address space into aggregated clusters. The IPs and purchase behaviours will be stored in a hash table, as well as the network identification prefixes and their purchase behaviours will be stored into another hash table.

6.2 Defining the Profit-Aware Modules

The profit-aware admission control mechanism consists of two modules: the identification profiling module and the admission decision module. The first module is always on as it has to record customer purchase behaviours. The profiled IP will be stored in the first hash table and then the network identification prefix

will be added onto the second table. Updating then, both of the tables with the behaviour. The second module will only turn on when an overload circumstance is happening.

In this method we will also be benefiting of the E-commerce unit, sessions. To know more about them follow section section 3.1.

On one hand we have the identification prefix module. Different to other methods, the identification profiling module only benefits from the inter-session relationship for customer classification. It records the IP address of the customer only when the customer confirms a payment. This makes it possible for the module not to introduce a big overhead and can always be active. The profiled identification information will be stored in two hash tables, which will be used for making admission decisions.

This tables are called Individual Hashtable and Prefix Hashtable. Individual Hashtable keeps the 32-bit individual IP addresses of the purchasing customers and also records the most recent payment behaviours of said individual IP addresses providing short-term information for making admission decisions. Prefix Hashtable records the network identification prefixes of all customers and captures the aggregated payment behaviours of said network identification prefixes, supplying long-term information for the posterior admission decisions. The IP addresses will be maintained by the identification profiling and later used by the admission decision module.

Obviously, the tables cannot increase indefinitely. Which is why in case of reaching the capacity limit, some of the existing elements will be removed in favor of most recent customers. This is why the clock replacement algorithm [14] will be implemented in the Individual Hashtable. This algorithm has basic requirements such as: no replacing IPs that are frequently purchasing and making the replacement cost very low.

In the Prefix Hashtable, the network identification prefix is stored instead of the customer's IP. Then, for each identification prefix we will find the number of committed purchases, which increases by one each time a customer, within the same prefix, confirms a new purchase. When the network identification is not in the table, a new entry is created and set to one. Once the network identification prefix is in the table, we will keep track of the number of sessions for the network identification prefix as well.

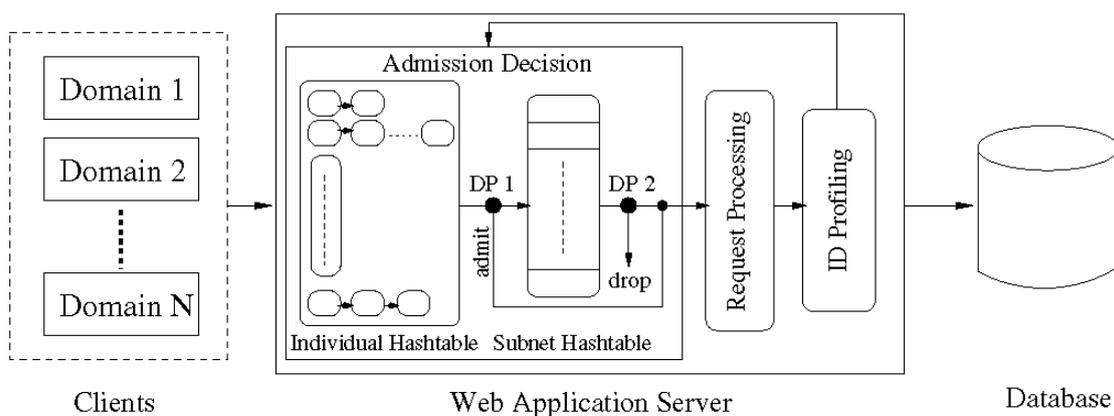


Figure 6.1: Profit-aware admission control architecture.

On the other hand we have the admission decision module. This module is only in action when the site is overloaded. Two decisions will be made, DP1 and DP2 as can be seen in Figure 6.1. Under activation circumstances, the IP address will be searched in the Individual Hashtable, and the IP's session will be accepted if the IP has been found in the hash table. Then the IP will be assigned a probability of one. We have to assume that the server will be able to support all its hashed customers. Otherwise, the site should increase its processing capacity or apply probabilities that are less than one. This decision is made at DP1. In

case the IP was not found in the Individual Hashtable, the network identification prefix would be looked for in the Prefix Hashtable, where if found the admission probability of the new session would be computed based on the B2V ratio of the matched networks identification prefix.

Elseways, the minimum admission probability of the B2V ratio interval will be applied. The B2V value will be different depending on the number of network identification prefixes. The larger the B2V value the higher the admission probabilities. To make admission control easier, the computation of admission probability is based on a certain number of B2V ratio intervals. The admission probability of each B2V ratio interval is set proportional to its corresponding B2V ratio range upper bound. For any network identification prefix given, its B2V ratio is mapped into a B2V ratio interval. Then the admission probability of the network identification prefix is set to that B2V ratio interval. The admission decision happens a DP2 comparing the assigned admission probability to a random number to choose if it will be accepted or not. Some pseudo code provided by the report can be seen next:

6.3 Comparisons between four different test beds

In the next graphs Figure 6.3 and Figure 6.4 we are going to see four different methods tested: no admission control, random admission control and profit-aware admission control with 100 and 30 percent knowledge of the previously purchasing customers. The first four show the throughput of Web interactions per minute and the next four the throughput of payments per minute.

In graph Figure 6.3a we can see in blue the customers who have purchased something in the past and in black the browsing customers. As the number of users in the site grow we can see both types of customers decreasing. An

```
Algorithm: Admission Decision  
newIP = the IP address of the session  
if ( newIP is in the Individual Hashtable )  
    set its reference bit to 1  
    admit this new session  
else  
    newNetID = the network ID prefix of the new session  
    if ( newNetID is in the Prefix Hashtable )  
        map its B2V ratio to a B2V ratio interval  
        admitProb = the admission probability  
        of the B2V ratio interval  
    else  
        admitProb = the minimum admission probability  
        of all B2V ratio intervals  
    endif  
    if ( random()  $\leq$  admitProb )  
        admit this new session  
    else  
        reject this new session  
    endif  
endif
```

Figure 6.2: The algorithm for new session decision.

improvement is observed in graph Figure 6.3b when using random admission control on both types of users but we would want the blue line to grow and the black one to decrease. This is what we see in both the profit-aware admission control at 100 and 30 percent in graphs Figure 6.3c and Figure 6.3d. The blue line gets more stable which means users who are more likely to purchase will get their sessions accepted.

If we check Figure 6.4 a progress towards more purchases is made from the graph Figure 6.4a all the way to graph Figure 6.4c. The number of purchases increases as the customers who previously bought in the Web site get prioritized. Even with a 30 percent of knowledge in graph Figure 6.4d, the profit-aware admission control gets a much better number of payments as the number of users accessing the Web site increases.

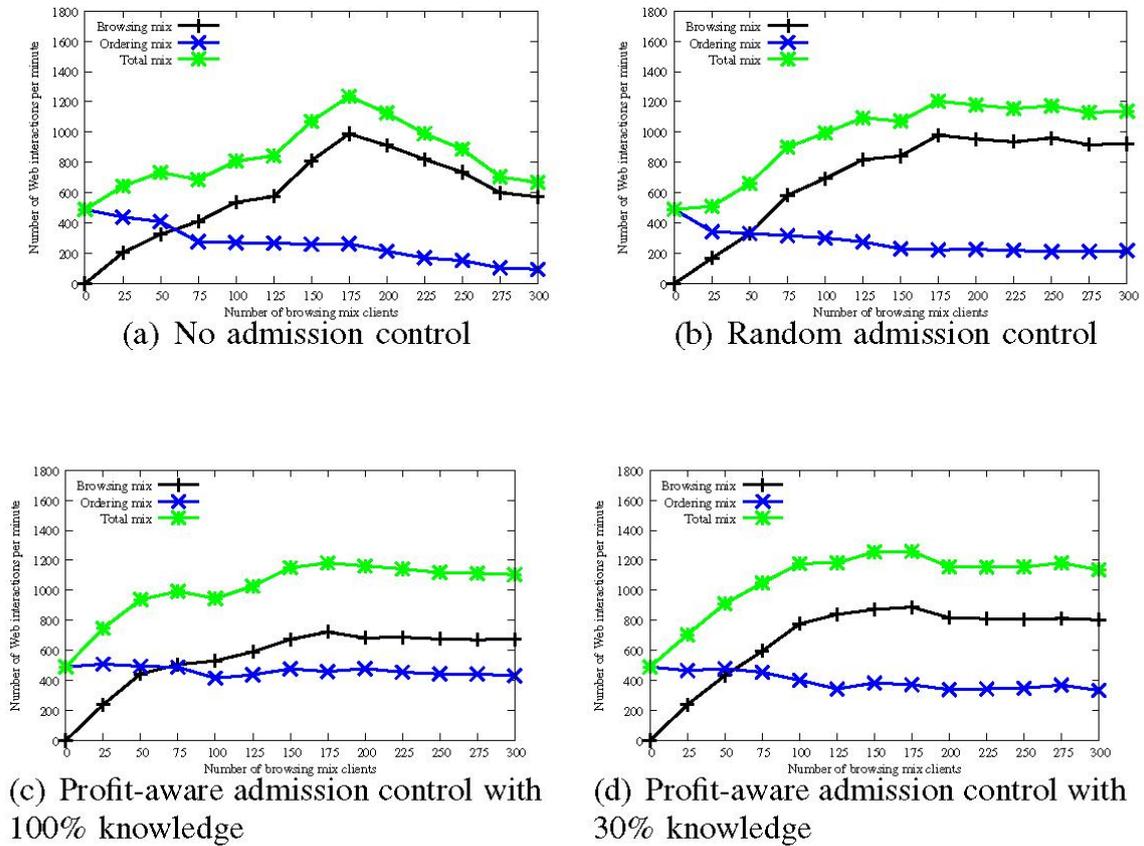
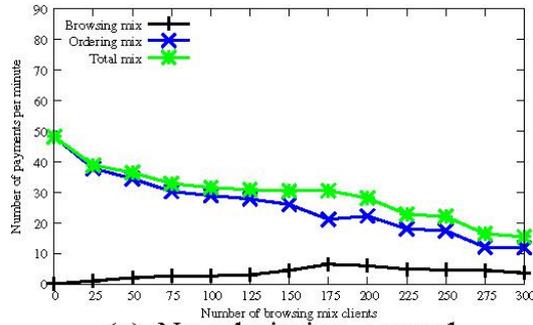
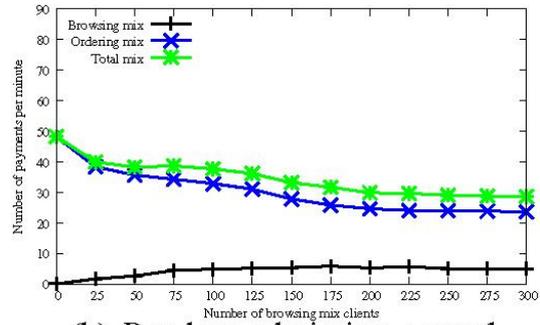


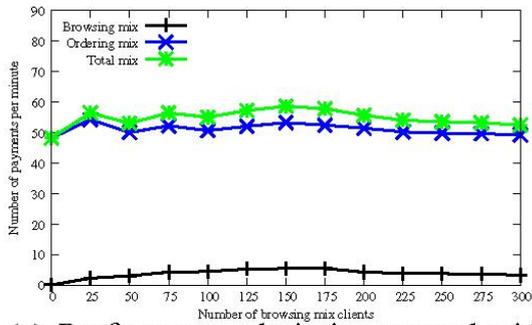
Figure 6.3: Throughput in terms of number of Web interactions per minute as a function of the system load for the four sets of experiments. In each sub-figure the Browsing mix curve stands for the throughput for basic customers, the Ordering mix curve stands for the throughput for previously purchasing customers, and the Total mix curve is the overall throughput for the two kinds of customers.



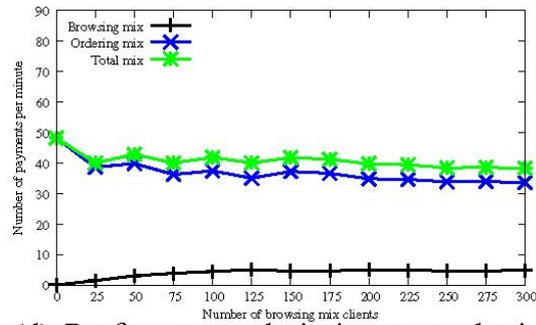
(a) No admission control



(b) Random admission control



(c) Profit-aware admission control with 100% knowledge



(d) Profit-aware admission control with 30% knowledge

Figure 6.4: Throughput in terms of number of payments per minute as a function of the system load for the four sets of experiments. In each sub-figure the Browsing mix curve stands for the throughput for basic customers, the Ordering mix curve stands for the throughput for previously purchasing customers, and the Total mix curve is the overall throughput for the two kinds of customers.

7 Combined LIFO-Priority Scheme for Overload Control

Combined LIFO-priority scheme for overload control is described in report [15] by N. Singhmar, V. Mathur, V. Apte and D. Manjunath. This methodology has two objectives: maximise the throughput of revenue-generating more requests while improving overall throughput of the server during overload.

As a way to achieve the first goal, a priority mechanism is employed. Separate queues will be maintained for each type of request: transactions and browsing. The transaction requests queues are given a simple non-preemptive priority over the browsing queues, making the assumption that if a transaction request is waiting to be served a browsing request will not be processed. Between transaction queues, the queue for the last request of a transaction has the highest priority.

To complete the second goal, a load-based LIFO mechanism is proposed. A FIFO policy during normal loads and a LIFO policy during overload. LIFO based policies provide a better throughput and delay performance at overloads compared to FIFO. The high variance of delay works in our favor as the mean delay at overload is high. This gets us more requests that do not time out as it would happen with FIFO. In the report, the assumption of the browsing requests causing the overload, is made. That is why it will employ LIFO on overload for browsing queues and FIFO always for transaction queues. It also proposes

a dynamic priority mechanism to select browsing requests that could potentially end up in transaction requests.

LIFO-priority uses CPU utilization for the browsing requests. If the utilization crosses a predefined upper threshold, then it starts serving browsing requests according to LIFO. It will continue with that method until the CPU utilization is above a lower threshold.

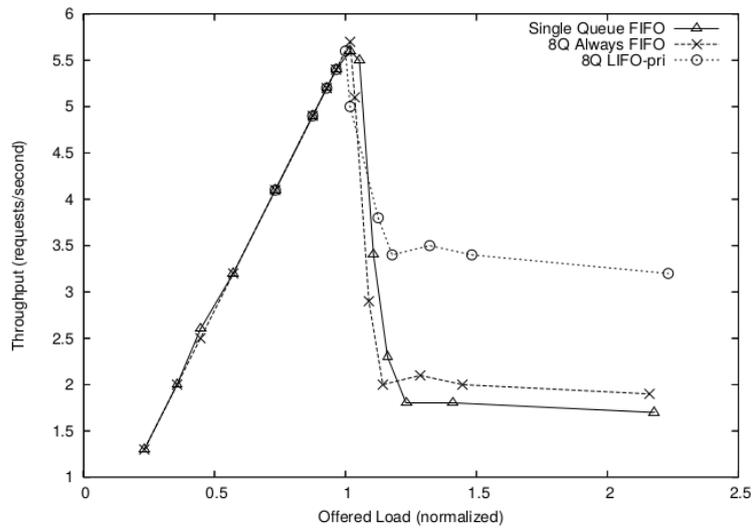
7.1 A Comparison between Single Queue FIFO, 8 Queue Always LIFO and 8 Queue LIFO-Priority

In the workload model for this comparison, the maximum capacity of the server is 5.6 requests per second. The server is configured with eight queues: four for browsing requests and four for transaction requests. Single queue FIFO stores all requests and serves them using FIFO and having an assumed capacity of 100 requests. Eight queue always LIFO stores each different type of request in a different queue and always serves them in FIFO. Browsing queues have an assigned capacity of 50 requests and transaction queues of 25 requests. In eight queue LIFO priority stores requests the same way as 8Q always FIFO but serves the requests with LIFO at overload for browsing queues and FIFO for transaction queues, using the same capacities as 8Q always FIFO.

The graph in Figure 7.1 shows the overall throughput as a function of the offered load. The offered load is normalized where a 1 is a load of 100 percent, meaning the number of requests received is the same as the servers capacity. We can see how at offered load 1 all three methods have similar throughput. However this changes as the load grows. We can see how at a approximated load of 1.3 the 8 queue LIFO priority stops decreasing and gets stabilized at a throughput between 3.5 and 3 requests per second. A much better mark than the other FIFO methods

as they keep falling until reaching a throughput between 2 and 1.5 requests per second.

In the table from Figure 7.1 we observe the number of dropped requests per method at 85 percent of the load and at 140 percent. At 85 percent everything works perfectly fine for each method but things go downhill at 140 percent, which is where requests timeout and generate retries as a reaction to the load exceeding the server capacity, and that increases the offered load to the server. However, after some of those requests get cancelled, some sessions get aborted too and the session's requests do not retry again. This produces some reduction in the load. Although having almost a 60 percent of completed requests, the number of dropped requests is of almost a 20 percent. Also, the number of aborted sessions, described as not generated, shows a better percentage compared to the other methods.



ρ	$\rho = 0.85$			$\rho = 1.4$		
Case	SQ	8Q-AF	8Q-LIFO-Pri	SQ	8Q-AF	8Q-LIFO-Pri
Completed	100	100	100	29.9	36.6	57.5
Timed out	0	0	0	36.8	29.9	7.5
Dropped	0	0	0	10.6	13.1	18.2
Not Generated	0	0	0	22.8	20.4	16.8

Figure 7.1: Overall throughput versus load.

8 Load Balancing

Load balancing is a mechanism that sits in-between the client and the hosts that provide the services the client wants to use. A host is a physical server which contains one or more services available, like for example HTTP, DNS, etc. This positioning of the host is not a strict rule but a good practice. Using the example explained in [16], the load balancer is going to be configured with a virtual server pointing to a cluster of hosts consisting of two service points, as shown in Figure 8.1. In this scenario, it is common for the hosts to have a return route back to the load balancer, so that the return traffic can be processed through it on the way back to the client.

Following Figure 8.1, let's imagine a client attempts a connection to the service. The load balancer will accept the connection and will decide which host will receive the connection. Then it will change the destination IP to match it to the service's one of the selected host. The host will accept the connection and will respond back to the original source through the load balancer and back to the client. The load balancer will intercept the return packet from the host, change the IP to match the clients and forward the packet. Lastly, the client will receive the packet believing that it came from the virtual server.

In the events of an overload, what might happen is that a client has all its data stored in one of the hosts, which implies that client can not be moved from that host risking losing the information built up until this moment. When this

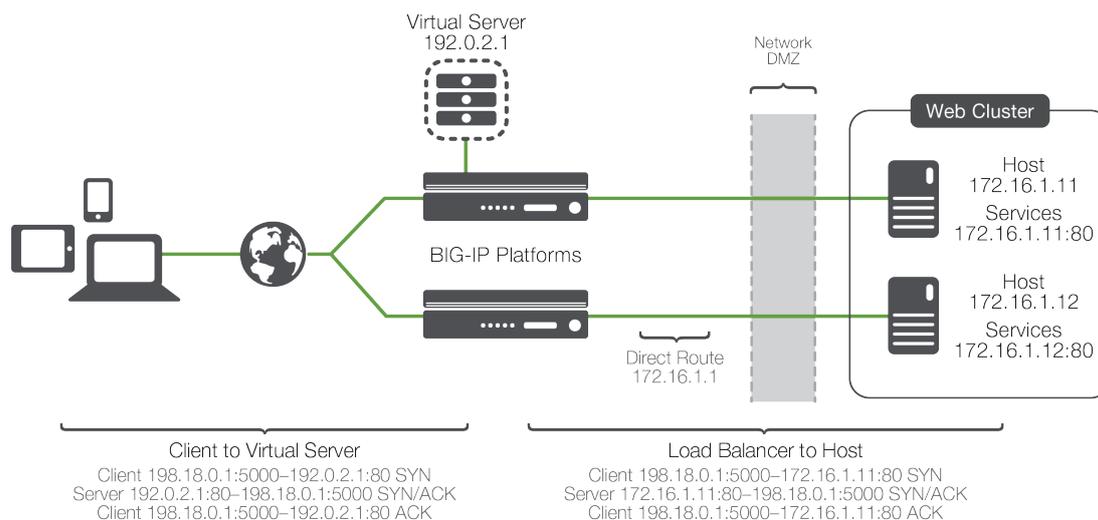


Figure 8.1: A basic load balancing transaction.

happen, only new connections can be moved into the other host to facilitate the work on the first one and hoping it has not accepted much more requests than it could handle so the users do not realize a change has happened.

8.1 How to load balance

Algorithms to decide to what server the load balancer is going to send the requests include least connections, dynamic ratio and simple round robin where the load balancer simply goes down the list starting at the top and allocates each new connection to the next host. When it reaches the bottom of the list it starts again at the top. A server will not always be active. That can happen when a big load enters the site and the requests flood one of the server and another has to take over some of them to avoid having aborted requests.

Least connection policy selects the server with the least amount of connections to attend a request. When the server reaches the maximum number of requests it

can handle, the load balancer automatically changes the server status to disabled and stops sending requests to that server and moves to the next one with the least amount of connections.

Most load balancers include health monitoring services in order to know if a host is or is not responding so they can send requests to it or not. Some higher level health monitors can help the load balancer understand that maybe one service is unavailable, but other services on the same host might still be working and should be considered as valid destinations for user traffic.

The easiness in which a load is balanced goes far beyond the hosts and the algorithms used. It also affects the data bases and how that data is shared between the different hosts. Having a shared database would imply moving the problem down into the chain. If our host is overloaded and we decide to add a new host into the equation, at the same time we are adding a new server making moves on our database, which saturates it and then a new problem surfaces. This is why analyzing what would happen in certain scenarios guarantees that we will be able to balance our services in order to avoid this.

Load balancing is one of the most common methods used as it is effective and easy to implement. Getting extra server capacity to help the existing server makes it an at-reach solution for big and small web applications. If the server needs support, renting extra servers will help during the time of the inconvenience and then the application can go back to just the original server. This used to be an expensive move to make but as cloud technologies have been normalized, the use of virtual servers is a much affordable option and it keeps getting cheaper as new ways of renting appear in the market.

9 Comparing the methods

After the explanation of those six methods in the previous chapters, we are ready to start pointing out the benefits and the disadvantages of each method and establish certain guidelines in order to structure and categorize the methodologies.

Session-Based, Hybrid and Predictive Admission Control

These three methods are strongly related for the sake of obtaining the best possible version of a working method. Starting by reshaping requests into sessions to give importance to longer transactions, finishing in a mechanism capable of deciding what policy is better in each load situation.

This technique makes a great candidate to be implemented in servers used for online shops and high demand sites as a ticketing service could be. Unfortunately despite the tests in the reports, these methods have not been tested nor implemented on real web application servers.

The benefits they could provide to E-commerce sites are massive. Loads are never as we can expect them to be due to their human factor. Which is why policies will not always perform amazingly in different situations. Stepping into a multiple policy realm is already a huge improvement, but if we could act before the problem has started, it would be even better. Here is where prediction comes in. Predicting overload behaviours allows us to be one step ahead of the problem in most cases, and the more knowledge we have of our site, the better these

predictions will be able to stop the server from overloading.

For that reason, these three methods are better compared to others when they are considered combined, in lieu of being separate entities.

SBAC techniques and Load balancing

One of the best parts of the SBAC group of techniques, is that it can be used in two different ways. One of them consisting on just utilizing a server and making the best we can for it to manage overloads and work through them with as much normality as possible. And the other, for far more demanded web servers, which would consist on getting helped by load balancing. Using SBAC as a load balancing policy allows us to make sure the server resources will be available to more users and therefore, overloads will not be such a weight on a single server.

Load balancing embraces buying or renting new server capacity in order to prevent big loads from overloading our original server. Some years ago, renting new capacity was expensive but as time has passed, virtual servers have taken over the sector, and so has cloud computing. This makes load balancing, even though being the most "non server thoughtful" method, a good and cheap alternative for quick fixes. It does not matter that your peak overloads are periodical, spontaneous or happen in a pattern, load balancing can have your back as different renting and buying options have hit the market. SBAC method is a way to get the best throughput out of your server if the overload peaks happen in a pattern or spontaneously, but not often. Outside these characteristics SBAC may not be able to help in the same amount load balancing would.

Session-Based and Profit-Aware Admission Control

Between these two methods the differences start growing. Whereas SBAC seems to look for the commerce as well as the user, in profit aware admission control, the

business comes first. Making a profit from the customers is the main concern, to the point of, a part of prioritizing recent buyers, if a new customer comes in but the algorithm does not choose it as a possible candidate that might end up buying, in case of an overload that user will be kicked out. While this kind of philosophies might make sense in some online business such as shops or banks, it might not make sense in other contexts such as university servers that manage registrations and browsing at the same time.

Sessions allow to this last categories a way to prioritize transactions that have a purpose in the moment, regardless of the history of the user in the Web page. Even if profit-aware uses sessions, their goal is to provide IP information on the user instead of information on the transaction.

Profit-Aware and LIFO-Priority

Compared to profit-aware admission control, LIFO-Priority is not as good of a match to E-commerce than profit-aware is. While profit-aware admission control has a native instinct of accepting historically appealing customers requests, LIFO priority concentrates on retaining the benefits of assigning high priority to a transaction request. LIFO priority also improves throughput performance due to enhancing response time performance when abandonments start occurring. As stated earlier in this project, improving response time reduces the number of abandonments which at the same time causes less session abandonments while increasing overload. But it is also worth mentioning that LIFO priority implies high unfairness and variability, but the abandonment rates during overload make it an interesting choice.

After laying down all the points, the following comparative table Table 9.1 is presented in order better visualize what has just been exposed.

Methods	Pros	Cons
SBAC	Sessions allow transactions to be more fluid.	In overload situation, if a request from a session is cancelled, the whole session is aborted.
Hybrid AC	Introduces a way to use two different ac-strategies in the same situation	It can only change from a strategy to another when the change in behaviour has been detected.
Predictive AC	Uses observed behaviours, to predict when to change ac-strategy before an overload happens.	The predictions can not be based on different sites. Each site has unique behaviours that will help the predictive system work.
Profit-Aware AC	Does not need session information. Uses IP and network ID prefixes.	By giving new customers a random probability, possible purchasers might be discarded by the algorithm.
LIFO-Priority	Frequent purchasing clients are prioritized. Low session abandonment rate. Multiple queues for differentiated browsing and transaction requests.	High unfairness and variability.
Load Balancing	Accessible solution for different web application sizes. Easy way to increase server capability without modifying the server's admission control policies.	Rough solution, does not try to fix what is already there.

Table 9.1: Pros and cons of the methods discussed.

10 Conclusions

To end this thesis we should evaluate if we were able to fulfill the requirements proposed at the start of the document. We have created a small guide to the solutions of overload peaks that helps us understand how will those methods help if they were to be implemented in a web server. By studying and explaining those methods in this guide we got a better understanding of the admission control world that exists within the web servers behaviours.

A personal goal by the end of the project, was to be able to determine if there was something better than load balancing and why was it the easy and trustful solution in most overload problems. The response to this question is as easy as it can be: servers have a limited capacity. Even if our Web application was in a server with great capacity, if the loads visiting our web page were to be bigger than it could support, we would need another server. There would be no other solution for us, other than to have multiple servers which could be later optimized by using one of the described methods in this project, or another found in reports online. But also, an important point of view that was learnt is that load balancing is not the only solution. For smaller web applications that do not need extravagant amount of server capacity, there are many ways to control possible peak overloads without investing money in more capacity. But as seen, nowadays money is not even a problem anymore with the cloud and virtual servers.

Future work could include testing and implementing this methods to different

types of web applications in order to obtain a much broader view on how each of them behaves. Also compare and even merge strategies in order to obtain more powerful methods.

To gain further insight it is recommended to read the reports for each method as they provide more mathematical information on the algorithms used.

Bibliography

- [1] Wikipedia contributors. *Web application* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 28-Apr-2019]. 2019.
- [2] B2B Quotes distributors. *What are the Different Types of Wep Apps?* — *B2B Quotes*. [Online; accessed 28-Apr-2019]. 2019.
- [3] E-commerce Platforms distributors. *E-commerce Definition. What is Ecommerce? E commerce Explained for 2019* — *E-commerce Platforms*. [Online; accessed 14-Mar-2019]. 2019.
- [4] Techopedia distributors. *What is Portal Application?* — *Definition from Techopedia*. [Online; accessed 28-Apr-2019]. 2019.
- [5] Optimizely distributors. *Content Management System* — *Optimizely*. [Online; accessed 28-Apr-2019]. 2019.
- [6] Wikipedia contributors. *Quality of service* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 08-Mar-2019]. 2019.
- [7] Gregor V. Bochmann et al. “Introducing QoS to electronic commerce applications”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 2040. Springer Verlag, 2001, pp. 138–147. ISBN: 3540419632. DOI: 10.1007/3-540-45415-2_11.

- [8] V. Elek, G. Karlsson, and R. Ronngren. *Admission control based on end-to-end measurements*, pp. 623–630. DOI: 10.1109/INFCOM.2000.832236. URL: <http://ieeexplore.ieee.org/document/832236/>.
- [9] L. Cherkasova and P. Phaal. *Session-based admission control: a mechanism for peak load management of commercial Web sites*. 6. 2002, pp. 669–685. DOI: 10.1109/TC.2002.1009151. URL: <http://ieeexplore.ieee.org/document/1009151/%20https://ieeexplore.ieee.org/abstract/document/1009151>.
- [10] Ludmila Cherkasova and Peter Phaal. *Hybrid and Predictive Admission Strategies to Improve the Performance of an Overloaded Web Server*. 1999. URL: <https://www.hpl.hp.com/techreports/98/HPL-98-125R1.pdf>.
- [11] L. Cherkasova and P. Phaal. *Predictive admission control strategy for overloaded commercial Web server*, pp. 500–507. DOI: 10.1109/MASCOT.2000.876577. URL: <http://ieeexplore.ieee.org/document/876577/>.
- [12] Chuan Yue and Haining Wang. *Profit-aware Admission Control for Overload Protection in E-commerce Web Sites*. June 2007, pp. 188–193. DOI: 10.1109/IWQOS.2007.376566. URL: <http://ieeexplore.ieee.org/document/4262470/>.
- [13] Martin Casado and Michael Freedman. *Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification*. Jan. 2007.
- [14] F J. Corbato. “A paging experiment with the multics system”. In: (July 1968), p. 20.
- [15] Naresh Singhmar et al. *A Combined LIFO-Priority Scheme for Overload Control of E-commerce Web Servers*. 2006. URL: <https://arxiv.org/pdf/cs/0611087.pdf>.

- [16] Jr KJ (Ken) Salchow. *Load Balancing 101: Nuts and Bolts*. [Online; accessed 19-Mar-2019]. URL: <https://web.archive.org/web/20170430085805/https://f5.com/Portals/1/Cache/Pdfs/2421/load-balancing-101-nuts-and-bolts-.pdf> (visited on 06/03/2019).