A Predictive-Adaptive Algorithm for a Web Switch

Kaja Gilly[1], Salvador Alcaraz[1], Carlos Juiz[2] and Ramon Puigjaner[2]
[1]Universidad Miguel Hernández, Departamento de Física y Arquitectura de Computadores, Avda. del Ferrocarril, 03202 Elche (Spain)
[2]Universitat de les Illes Balears, Departament de Ciències Matemàtiques i Informàtica, Carretera de Valldemossa, km 7.5, 07071 Palma de Mallorca (Spain)

Abstract: A high performance and high reliable web server system is the key to the success of all Internet services. This is the main reason for Internet service providers (ISPs) to choose, more and more, running their service through a cluster of servers. Some of the current commercial products rely on constant checking of servers' health that can itself be a significant overload. Our paper proposes an algorithm for a web switch that manages a cluster of web servers, evaluated through discrete event simulation as the first step in the construction of a new web switch. The web switch model is based on the estimation of the throughput that the servers would have in the next future to dynamically balance the workload. In order to reduce the checking time, estimations are only computed in a variable slot scheduling. Simulation results have shown that a commitment between overhead and performance should be established to obtain good results in a stressed web cluster of servers

Key-Words: - Adaptive Switching, QoS, Web cluster, Load balancing, Overhead

1 Introduction

As the widespread usage of web services grows, the number of accesses to many popular web sites is ever increasing and sometimes causes the servers to be overloaded. To avoid this problem, popular service providers have to utilize either large multiprocessors or distributed network servers in order to achieve reasonable quality of service levels. The clusterization of servers could guarantee a Service Level Agreement (SLA) and therefore users perceive the expected Quality of Service (QoS). There are several commercial products for load balancing and dynamically change groups of web servers. The corresponding switch algorithms and the tactics to select the preferred server are diverse in these implementations. However, some of them rely on checking of servers' status continuously, and this can cause significant overload in the web system. In order to build a new web switch with load balancing, it would be desirable to predict the performance of the servers and to adapt the groups of servers to the current load, but also to reduce this continuous checking. Thus, the first step to achieve these goals is simulating the architecture model of a web cluster including transactions, servers and the switch to verify that the proposed algorithm accomplish these performance requirements.

The main focus of this research is the design of an algorithm for the Web switch that manages a cluster of Web servers that implement a Quality of Service (QoS) that satisfies several customer expectations. QoS referred to Internet services appears for the first time as Web Quality of Service in [3]. The term of Quality of Web Services (QoWS) that was introduced in [5], to designate different types of services that web servers can provide. We have developed an algorithm that resides in a Web switch and controls all incoming traffic of the whole web system. The web switch not only controls the status of the Web servers but also balances the workload according to the values of selected performance magnitudes.

This paper presents a simulation model for an adaptive solution of a web switch, based on the estimation of the throughput the servers will have in the next future. Estimation is done in basis of previous throughput of the servers, peak characteristics of incoming traffic and the server's current utilization. The dispatching algorithm also takes into account the burstiness factor of the web servers to control the duration of the slots. Thus, this research represents the first step in the construction of a new web switch; the performance evaluation of a model through discrete event simulation.

There are several studies on this subject. Cardellini et al. [5] propose several policies that meet QoWS principles. They do not introduce prediction in the movement of servers between users, so a decision made by the switch can be erroneous at future time. The idea of prediction is considered by Yoon et al. [20] to decide the redistribution of the tasks between the nodes of the cluster. They work with clusters composed by heterogeneous nodes, one of them being the master node. In our study, the web switch is the master node. Lu et al. [9] combine a queuing – theoretic predictor with feedback control architecture to achieve relative delay guarantees in high performance server. They based their study in M/M/1 model so no burst traffic is considered in this case.

The rest of this paper is organized as follows: we first describe our Web cluster architecture in section 2. The dispatching algorithm is described completely in section 3. Section 3.1 introduces the fundamental concepts needed to understand how this paper deals with burstiness and performance characteristics. Performance metrics used in the algorithm are introduced in section 3.2. Section 3.3 presents the throughput predictors and section 3.4, clustering mechanism. Workload conditions and simulation results are depicted on section 4 and 5 respectively. Finally, section 6 presents some concluding remarks.

2 Web Switch Architecture

A set of servers plus a switch housed together in the same physical location compose a Web cluster. The switch, considered as a layer-7 web switch, is the first device belonging to the web system that users' requests meet. The web switch could be considered a content-aware switch [6] that means that it can examine the content of HTTP incoming requests, and balance them according to the type of page requested by the user. After going through the switch, requests arrive to the selected Web Server and sometimes also to the corresponding back-end server to be serviced.

Web servers in a cluster are almost identical in terms of hardware characteristics. Moreover they have the same HTML documents stored in their physical disks and identical applications installed. Then, an incoming request arriving to the system can be redirected to any of the web servers in the cluster. Furthermore, each server controls a back-end server to compute database searches of dynamic pages running CGIs, Servlets, etc.

A VIP (Virtual IP Address) is used to identify the complete system in World Wide Web. This network address corresponds to the Web switch. Internally each server has its private IP address. The architecture of the web cluster is shown in figure 1. The switch takes over incoming requests, selects a server inside the cluster and routes the requests to the server. Therefore, the switch acts as a dispatcher of server incoming tasks. The web switch includes an algorithm that controls some non-functional parameters of the servers and also it chooses one of them basing its decision in a set of rules. Depending on the rules selected, the switch works in a complete different way

In order to prevent poor performance situations by selecting a wrong server in a given time instant, forecasting parameters are calculated in the switch during running time. Overhead produced by the additional computation required to obtain these forecasting parameters is also computed. Hence, these parameters are obtained depending on the arrival rate of incoming requests to the cluster, increasing forecasting computation frequency when peak traffic is detected and the web servers may be immediately congested. Therefore, the forecasting dispatch algorithm may be considered performance-adaptive and overhead controlled. In other words, our second goal is balancing the effectiveness of the forecasting and the price to be paid for it, in terms of added overhead.

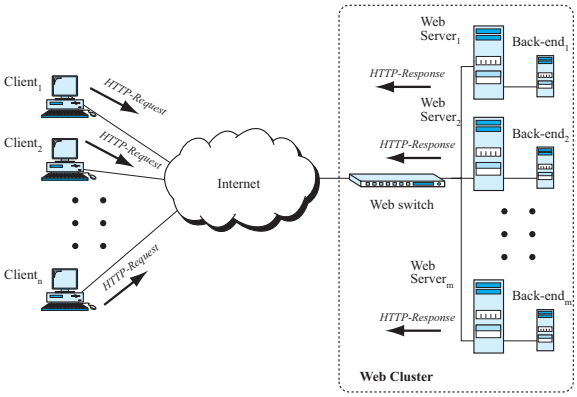


Fig. 1. Web cluster architecture

3 Dispatching Algorithm

The switch is the part of the web system that manages the connections between external users and the web servers that compose the web cluster. Users rely on web servers demanding functional requirements, e.g. information requested and the applications needed to serve it, and non-functional requirements, as guaranteed performance. These requirements indirectly define the quality of web services that users are expecting. This quality is based on the same concepts of Quality of Service (QoS) on networks but applied on web architectures. Given a web cluster that provides demanding web services, a pre-defined Service Level Agreement (SLA) has to be defined. Thus, two types of users have been determined depending on two different web service profiles: priority users and best-effort users. The switch should guarantee this level of agreement for priority class users; therefore it has to transmit their requests to best performance servers while the corresponding service level for the rest of users may not be guaranteed.

In the long run, the dispatching algorithm strategy is to select the web servers according to the SLA. The particular adaptive tactics for the selection of servers takes into account the current performance status and the future estimation of it.

Two throughput estimators of performance have been considered for each server during each slot. Due to

performance estimation values, a reorganization of servers could be required to continuously support the SLA constraints. The cluster is composed by two groups of servers, each of them attending requests for each type of user, respectively.

3.1 Burstiness and Performance

It is critical to understand the nature of network traffic in order to design a good dispatching algorithm for the Web switch. Several studies have revealed that workload in Internet traffic has self-similar characteristics ([8],[14],[11]). This implies that significant traffic variance or burstiness is present in a wide range of time scales. Therefore burstiness needs to be controlled in a web site to procure a good performance of the whole system, at least when demand is high enough. Bursty arrivals of HTTP transactions to a web cluster requiring objects of several size scales seriously perturb the quality of web services perceived by users and may collapse the cluster service.

Burstiness control has been included in present work in the form of a burstiness factor defined as a coefficient that enables the switch to restrain an eventual saturation of web servers.

Given a measurement interval T and a number of slots n, the duration of each fixed slot k is obtained as

$$D_k = \frac{T}{n} = t_k - t_{k-1} \qquad (1)$$

where every slot has identical length for every k.

The mean arrival rate for each server i during the whole measurement interval is noted as λ_i. For each slot k and each server i, $\lambda_{i,k}$ is the corresponding arrival rate. If $\lambda_{i,k} > \lambda_i$ then the slot is considered as a "bursty" slot. Burstiness factor is defined as the relation between the cumulative number of slots where $\lambda_{i,k} > \lambda_i$ called n_i^+, given n slots [12]:

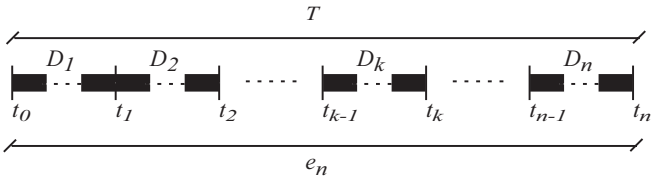
$$b_{k,i} = \frac{n_i^+}{n} \qquad (2)$$

Burstiness factor is computed during running time by incrementing the total number of considered slots, n, and also considering those slots, n_i^+, that incoming arrival frequency may produce congestion in the corresponding web server i.

In this paper we consider a variable duration slot schedule. The value of the burstiness factor on the current slot is the magnitude that defines the duration of the next slot. Thus, if burst is detected in the servers during current slot, the following testing time is reduced to check the incoming traffic. If no burst appears, the duration of the following slot is enlarged in order to reduce the overhead. By controlling the duration of testing slots, the reduction on future performance of web servers may be forecasted.

Let's consider the observation time, T, divided into several slots of variable duration. Thus, the number of slots during running time is also variable. In fact, it is different from one slot to another depending on the duration of each slot. So, let us define e_k as the current number of slots in the time t_k where D_k is the duration of the last epoch as t_k - t_{k-1} in t_k, as is illustrated in figure 2.

$$D_k = \frac{T}{e_k} \qquad (3)$$



T: time interval — D_k: duration of slot k
n: number of slots — e_k: number of slots in t_k

Fig. 2. Variable-Adaptive slotted time

Several rules have been defined to control the duration of the slots in a variable slotted time schedule. At the end of the slot k, the proposed algorithm recalculates the number of slots e_{k+1}, to use in the following slot, depending on the maximum burstiness detected among the set of servers, denoted as $\hat{b}_k$, the minimum burstiness, denoted as $\breve{b}_k$, and the number of slots previously defined e_k.

$$\begin{aligned} e_{k+1} &= e_k + \breve{b}_k \cdot e_k && \text{if } \hat{b}_k - \hat{b}_{k-1} > 0 \\ e_{k+1} &= e_k + \hat{b}_k \cdot e_k && \text{if } \begin{cases} \hat{b}_k - \hat{b}_{k-1} \le 0 \\ \text{and } \breve{b}_k \neq 0 \end{cases} \\ e_{k+1} &= e_k / n && \text{if } \hat{b}_k = \breve{b}_k = 0 \end{aligned} \qquad (4)$$

Once e_{k-1} is calculated, the duration of the following slot $k+1$, D_{k+1}, is obtained by applying the expression (3).

The duration of the first considered slot must be long enough to avoid the first transient period resulting of the initialization of the system.

The resulting dynamic computation of the slot duration will try to find the equilibrium between the amounts of tests (slots) and the detection of burstiness on time.

3.2 Performance Measurement

Incoming and outgoing traffic go through the switch, so it is possible to control the performance of the web servers and back-end servers, in terms of throughput and response time. This two-way architecture

produces more overhead than the one-way scheme although it is capable to supply performance feedback. In order to compute the throughput from the servers and conduct the algorithm tactics, indispensable measurements have to be taken. The metrics used in the algorithm are basically: the throughput of the servers, the arrival rate of HTTP transactions and the latency time of each server.

- Arrival rate: HTTP transactions arrive from outer part to the web cluster in a random behaviour. Several distributions could be considered but a clear picture of this is revealed in [2]. Since the nature of the Internet traffic is considered self-similar, the average value of this arrival rate is very useful to calculate burstiness factor as explained above.
- Throughput: The average number of HTTP transactions per second is the main metric used to feed the predictors computation. The estimate servers' behaviour relies on throughput measurements during the tests. The duration of every slot and therefore the number of slots also depends on the throughput values.
- Latency time: The Service Level Agreement (SLA) is expressed in terms of the maximum delay of dynamic transactions for priority class customers. Since two classes of users are considered requesting information to the web system, only priority users have a restricted SLA in terms of latency time. SLA contract between priority class users and the Internet Service Provider (ISP) is normally signed in terms of maximum latency time. Latency time is regulated in ISP installation, no network delays from the ISP to the client are being considered.

3.3 Throughput Forecasting

This paper considers the need of approximately calculating next future performance behaviour of web servers. This computation is done to avoid the possible congestion of web servers due to an eventual increasing of the incoming arrival rate of HTTP transactions through the switch, together with web servers' saturation. Therefore, estimation of immediately future performance level of the cluster and the burstiness or arrivals enable the switch to react in consequence. Both actions are adapted to the current web cluster circumstances. Thus, the dispatching algorithm strategy consists on detecting potential dangerous performance in both traffic ways. On one hand, the switch would check the burstiness of incoming traffic. Even short periods of burst may degrade the quality of web services perceived by the users in several scales. On the other hand, the average of service demand is approximately the same during longer periods. The switch has to manage the web cluster to these complementary (and also contradictory) expectations.

These forecasting tasks avoid the use of delayed load information, also called "*herd-effect*" [10]. This effect implies that machines that appear to be underutilized quickly become overloaded because all requests are sent to those machines until new load information is propagated. Using forecasting information, we try to predict the behaviours of the web servers in a close future time to prevent from "herd-effect".

However, testing continuously the performance of servers increases definitively the algorithm overhead and, in consequence, the latency time of transactions. Therefore, both estimations are done according to an adaptive slotted testing period. To recap, the proposed algorithm is adaptive in two different viewpoints: the transactions are processed by adapting the availability of the faster server to the current traffic, but also by dynamically considering the measurement of this traffic to the expected behaviour. Both phenomena are interrelated, so that the proposed estimation techniques are based on complementary predictions.

3.3.1 Filter Prediction

Filter prediction is based on previous estimation of throughput and real throughput measured from the servers. For a slot *k*, the filter includes the throughput measurement of current and previous *k*-1 slot for each server i, namely $X_{k,i}$ and $X_{k\text{-}1,i}$. Thus, let's define $\tilde{X}^{I}{}_{k,i}$, the mean estimated throughput at slot *k* for the server *i* as,

$$\tilde{X}I_{k,i} = A_k \cdot \tilde{X}I_{k-1,i} + (1 - A_k) \cdot \frac{X_{k,i} + X_{k-1,i}}{2} \qquad (5)$$

Thus, this estimated throughput depends on two terms: the last computed prediction and the average of the actual and previous measurements. The result of this computation is filtering throughput values based on the probability A_k [7]. This exponential smoothing places more weight on the recent historical prediction. Therefore, the *Adaptive Estimated Probability*, A_k, is defined as follows [16]:

$$A_k = \frac{2 \cdot e_k - 1}{2 \cdot e_k + 1} \qquad (6)$$

Since the average number of slots computed at slot *k*, known as e_k, is proportional inverse to the duration of the slot, D_k, the weight of A_k probability indirectly depends on burstiness (see formulas (4)). If the current slot is "long", then estimation places even more weight to previous throughput estimation. On contrast, if current slot is "short", the prediction

stresses throughput measured in servers during slots *k* and *k*-1.
This filtering provides a set of throughput estimations for the corresponding slot, one for each server in the cluster. The main effect of this estimation is smoothing traffic peaks to hold an accurate performance estimation of the servers in a long run.

3.3.2 Burst Prediction

The difference of burstiness in two consecutive slots averaged by the difference of their respective measured throughputs, for each server *i*, is represented by a *Locking Indicator* in each slot *k*.

$$\beta_{k,i} = \left(b_{k,i} - b_{k-1,i}\right) \cdot \left(X_{k,i} - X_{k-1,i}\right) \quad (7)$$

This locking indicator computes the throughput variation during the last two periods. Thus, the locking indicator measures the difference between the current and the previous burstiness factors and if it is greater than zero, then multiplies it by the difference between measured throughputs at *k* and *k*-1 slots. The resulting product of the locking indicator expresses the amount of variation of servers' performance due to the burstiness on transactions arrivals. If the burstiness detected in the slot is lower than the burstiness detected in the previous slot, then the locking factor is annulled. Therefore, the locking indicator is averaged dividing the utilization of the server, $U_{k\text{-}1,i}$, by the burstiness factor of the server at the previous slot *k*-1 [4].

$$\tilde{X}^{II}{}_{k,i} = \tilde{X}^{II}{}_{k-1,i} - \left(\beta_{k,i} \cdot \frac{U_{k-1,i}}{b_{k-1,i}}\right) \quad (8)$$

The role of this second estimator is to prevent the servers' performance computation degeneration caused by incoming traffic that shows bursty behaviour.

3.4 Web Clustering

Clustering among the servers is needed when there is no possibility to guaranteed SLA constraints for priority users. Therefore, it is necessary to control the latency time of responses addressed to priority users. In fact, dynamic requests need more computation than static due to database operations on back-end servers. So that, the latency time control of dynamic requests for priority users may ensure the servers' mean response time under the SLA value.
If latency time of dynamic requests for priority users is longer than the specified SLA then one server is selected from the best-effort set of servers and moved to the priority set of servers. Considering *P* as the set of servers that attend priority users' requests and *B*, the set of servers that attend best-effort users' requests, clustering algorithm is as follows:

$$\begin{aligned} &P{+}{+};\ B{-}{-} \quad &&if\ t_{lat}(dyn, pri\ users) > SLA \\ &P{-}{-};\ B{+}{+} \quad &&if \begin{cases} t_{lat}(dyn, pri\ users) < SLA/2 \\ and \\ t_{lat}(best-effort\ users) > SLA \end{cases} \end{aligned} \quad (9)$$

Thus, if latency time of dynamic priority requests is shorter than half SLA and the latency time of best-effort users is longer than SLA, one server is selected from *P* and moved to *B*. Each group should have at least one server.
The selection procedure of a web server to be moved from a set is done according to the best performing tactics. So that, the corresponding server is chosen to help the set of servers that need to improve their mean response time. Therefore, this tactics consist on computing the Euclidean distance from the throughput estimation of server candidates to the origin (0,0). The selected server is the one that maximizes this distance among the *m* servers belonging to the current set at slot *k*. The more distance from the origin to the two-dimension estimation point, the faster is the considered web server.

$$\max\left\{\sqrt{\left(\tilde{X}^{I}{}_{k,i}\right)^2 + \left(\tilde{X}^{II}{}_{k,i}\right)^2}\right\},\ \forall i \in m \quad (10)$$

3.5 Switching Algorithm Design

The general structure of the switch algorithm is basically comprised in two parts, the *computing* phase and the *transient* phase. During testing time of each slot, the web switch is running on the *transient* phase. At the end of each slot, the *computing* phase prepares the system for the following slot.
Transient and *computing* phases are detailed in figures 3 and 4 respectively.

```
TRANSIENT PHASE:
while TRUE {
 get_next_APPL_PDU(s);
 if request(s) then {
   //REQUEST
   if Priority(s) then {
     serverP=RoundRobin(P);
     send_req(s,serverP);}
   else {
     serverB=RoundRobin(B);
     send_req(s,serverB);}}
 else
   //RESPONSE
   modify_THROUGHPUT(i);
}
```

Fig. 3. Transient phase

```
COMPUTATION PHASE:
for i=1 to m {
   compute(XIk,i);
   compute(XIIk,i);
}
if tlat(dyn,pri)>SLA then {
   s=eucl_distance(B);
   move_server_to_P(s);
}
if tlat(dyn,pri)<SLA/2 and
   tlat(beff)>SLA then {
   s=eucl_distance(P);
   move_server_to_B(s);
}
compute_next(ek,Dk);
```

Fig. 4. Computation phase

4 Workload Conditions

The expected workload model considered in this paper includes general features of web sites. Web content is mainly composed by two classes of web pages: *static* and *dynamic* content. Static content is referred to HTML files that are included into web servers. Content on the web site is now often personalized, and therefore dynamically generated. This dynamic generation inherently produces an associated performance cost (overhead) that users could not be aware. Therefore, users have to suffer from delays not only for the latency of the network but also for the processing time of dynamic requests at server side.

In our simulation runs, we have considered five types of requests of both user classes, depending on the file size of the HTTP response. In order to compute the parameters of the simulation models of users and servers, several benchmark experiments have been performed. The laboratory scenario consists on a client and a server connected through a 10/100 Mpbs local area network. Workload has been generated by three different benchmarks: *Webstone* [19], *Httperf* [13] and *Apache Benchmark* [1].

WebStone benchmark measures the throughput and latency of each HTTP transfer and reports a useful metric derived from Little's Law. Results obtained by *Webstone* benchmark have been confronted with *Apache* benchmark and *Httperf* reports. Thus, saturation frequencies of HTTP requests (maximum throughput) per class (static and dynamic) of each server have been supplied by the tests performed with these three benchmarks. This set of saturation frequencies represents the 100% of simulated servers' utilization.

The arrival rate of incoming requests is Pareto-distributed:

$$P(x) = \frac{\alpha \cdot b^{\alpha}}{x^{\alpha+1}} \qquad (\mathbf{11})$$

with α = 1.5 and b obtained form saturation rate for each request type. On the other hand, service times for static and dynamic requests at servers are modelled according to different hyperexponential distributions where average values were obtained from benchmarking.

5 Simulation Results

A simulation model of complete web cluster system has been constructed using QNAP2 [15]. The experimental model includes six servers and six back-end servers that perform client transactions. Depending on the type of users, HTTP requests go to servers' set *P*, for priority users' requests or set *B*, for best-effort users' requests. We have considered that the percentage of priority users' requests is 50% of the overall incoming traffic and that the percentage of dynamic requests is 20%. SLA value for priority users is set to 4 seconds for the maximum expected mean response time. Three different switch algorithms have been implemented, namely, *pure static*, *pure dynamic* and *adaptive.* They have been simulated until confidence intervals arrived to the 95%. In order to confront the different performance behaviour of switch algorithms, all of them have been tested under the same workload conditions. Some interesting quality metrics have been also computed to examine the goodness of the estimations. Therefore, overhead control and throughput estimation error control are indexes to check the cost of our proposal in terms of time and error, respectively.

5.1 Pure Static Algorithm

The term static means that no movement of servers is permitted during the complete simulation interval, therefore priority and best-effort users' requests are served by static groups of servers. In this case, there is neither need of throughput prediction nor need of taking measures of server throughputs. This implies a perfect situation in terms of overhead, because the algorithm runs in the switch and no additional measurements of server throughput should be taken to control the performance of the system.

Figure 5 shows latency times for this algorithm. The horizontal axis shows the percentage of arrival rate needed to stress one server to the 100% utilization level. As the model is composed by 6 servers, 600% arrival rate level means a 100% utilization level of the whole set of the web servers system. Priority and best-effort users perceive a very similar system response with this algorithm because no movement of servers is done and both groups of user's requests are served by 50% of the web servers and back-end servers during simulation runs. SLA constraints are not guaranteed when arrival rate is at 300%, so servers cannot manage more than their 50% utilization level

5.2 Pure Dynamic Algorithm

This version of the algorithm involves movement of servers from group *B* to group *P* and viceversa when conditions specified in *computing phase* are met. So there is no need of divide simulation time under a slotted schedule because *computing phase* is executed always on the switch for every user request received.

This means that the pure dynamic algorithm achieves the best performance of the whole system because switch controls if SLA constraints are guaranteed for every request. If the SLA constraints are not guaranteed, the procedure of clustering starts. The consequence is that the switch is usually quite busy making changes in web cluster groups of servers.

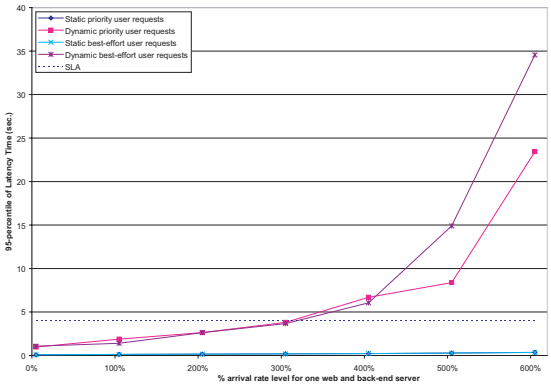


Fig. 5. Latency times of pure static algorithm

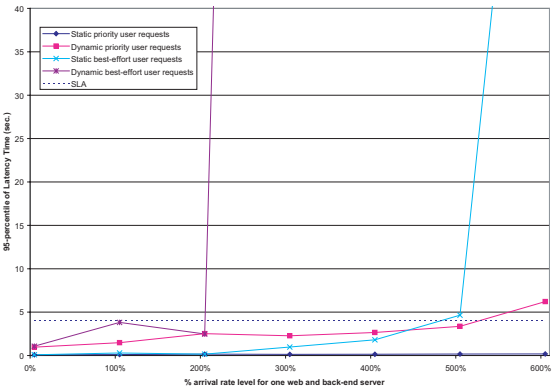


Fig. 6. Latency times of pure dynamic algorithm

The overload of the system can be calculated on basis of arrival rate level of requests. Each time a request arrives to the system, the switch take measurements of server´s throughput and checks if reorganization of the two groups of servers is needed. We have not included in this algorithm throughput predictors; hence the switch rearranges the servers depending on real throughputs measured on them. Mean response times are shown in figure 6. We have omitted the complete latency time of best-effort users to focus on the latency time of priority users. We can observe that SLA constraints are not guaranteed when web cluster is at 600% of its utilization; this is because servers are starting to be collapsed.

5.3 Predictive-Adaptive Algorithm

The proposed intermediate solution is designed focusing on controlling the performance of the system in a slotted time schedule. The last version of the algorithm includes throughput predictors introduced in section 3.3., and *transient* and *computing phases* described in figures 3 and 4. The duration of the slots varies depending on *burstiness factor*, so that, in the worst case, the overhead will be as high as pure dynamic algorithm. Our intention is to find a balanced solution: able to obtain good response time for priority users, guaranteeing SLA constraints, and reducing the overhead level of the pure dynamic algorithm.

Because of the use of throughput predictor, an inherent error is introduced in the clustering procedure. Thus, it is necessary to control this error to obtain the best performance of the web cluster. Figure 7 shows these predictor errors and their behaviour during the 3000 seconds of simulation run. It can be observed that the error on predictions, measured in terms of requests per second, is being corrected during the simulation. There is an initial startup phase where filter predictor gives the worst estimation of the throughput of the servers in the next slot. However the error ratio descends to lower levels becoming as good as the burst predictor. The error on estimations is controlled each 300 seconds period. When simulation in almost ended, at 3000 seconds, the error value of filter estimator is 3.89E-02, while for the burst estimator is 2.73E-02.

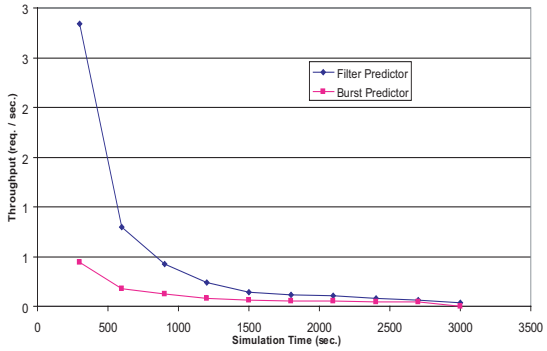


Fig. 7. Throughput error of predictors in adaptive algorithm

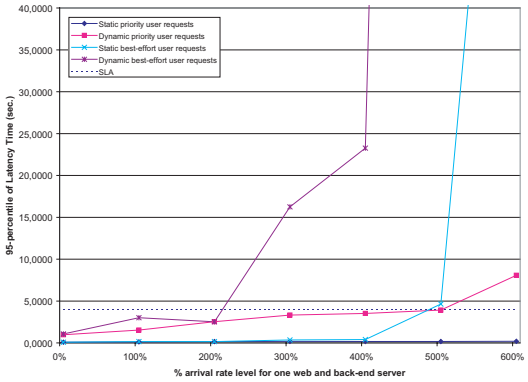


Fig. 8. Latency times of adaptive algorithm

Overhead introduced by this third version of the algorithm should be compared with the two opposite

previous versions. In pure static algorithm no overhead is introduced because there is no clustering included in the algorithm, so no check needs to be performed. In pure dynamic algorithm, overhead is proportional to arrival rate level; in fact, the computing phase is executed for each request. Therefore, throughput levels of the web servers are measured almost every time. As it would be expected, the pure dynamic version has the worst overhead level that could be. However, simulation results of adaptive algorithm show that the mean value of computing phase executions is 4569 for a total of 57592 arriving requests. Thus, the comparison of overhead level is less than 8% of the pure dynamic algorithm. Mean response time of adaptive algorithm is represented in figure 8. Complete latency time of best-effort users has been omitted to focus on the latency time of priority users.

Finally, we compare the latency times of dynamic requests for priority users in static, dynamic and adaptive algorithms in figure 9. Only the mean response times for priority dynamic requests are shown because they are the most important in terms of the performance of the web cluster. The three algorithms are simulated on identical workload conditions. All switching algorithms present a similar mean response time of web servers, whereas the mean arrival rate is less than one third of the servers' utilization due to the clustering conditions still have not been reached. However, pure static algorithm cannot react from the increasing arrival. Therefore, the servers' mean response time could be five times the one-third values when the arrival rate is close to the full utilization. On the contrary, the pure dynamic is the best switching algorithm for priority users, since the mean response time of servers is the lowest during the simulation runs. But the price to be paid for this advantage is a huge overhead that the implementation of the switch will have. The adaptive algorithm has a similar behaviour of the dynamic one. The mean response time is a bit longer, but the switching overhead is only about 8% of the pure dynamic scheduling, so that it compensates the effort of computing estimations.

6 Conclusion

This paper has presented a new proposal of adaptive switching algorithms for web clusters. The algorithm relies on throughput estimations on slotted time periods. In particular, the burst predictor detects the burstiness of traffic and the consequence of the variation on future servers' throughput. The filter predictor reduces the impact of typical traffic peaks due to the Internet nature. Both estimators are combined to select the best performance server in the cluster to guarantee the SLA. Thus, the switching algorithm is three times adaptive: in the transient phase when the slot time of testing period is variable depending on the traffic conditions, in the computing phase when the best server on cluster is selected from estimations and also moving servers from predefined set of servers among classes of users. All these procedures are done controlling the error on estimations and the overhead produced. The proposal has been confronted by simulation with pure static and pure dynamic scheduling.

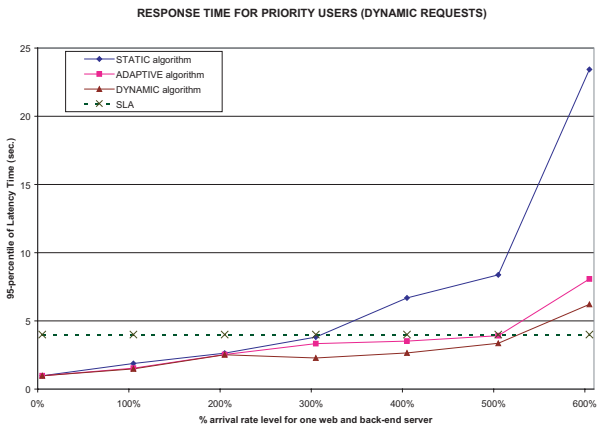


Fig. 9. Latency time for priority users dynamic requests

This paper tries to establish new ways of implementing non-functional requirements on web switching algorithms. Therefore, the continuous refinement on the web switching design including not only performability but also availability, security, etc. estimations will improve the perceived service for users.

This incipient work presented has to be developed in several ways. The overhead on computation of the predictors and the procedure of obtaining throughput measurements from servers are crucial points of this research. Other immediate future work will be to distribute the load proportionally on the servers by resource reservation mechanisms. The switch algorithm design research must be developed together with the new network protocol strategies considering the non-functional behaviour features of Internet requests.

References:

[1] Apache Benchmark. http://www.apache.org

[2] Barford, P., Bestavros, A, Bradley, A. and Crovella, M. E., "Changes in Web client access patterns: Characteristics and caching implications". World Wide Web, 2(1-2):15-28, Mar. 1999.

[3] Bhatti, N., Bouch, A. and Kuchinsky, A., "Integrating user-perceived quality into Web server design".Computer Networks. Amsterdam, Netherlands: 1999.

[4] Buzen, J. P., “Operational Analysis: an Alternative to Stochastic Modelling”, in Performance of Computer Installations. North Holland, Jun. 1978, pp. 175-194

[5] Cardellini, V. and Cassalicchio, E., Colajanni, M., Mambelli, M. “Web Switch Support for Differentiated Services”. ACM Performance Evaluation Review, 29, 2001.

[6] Casalicchio, E., Colajanni, M., “Scalable Web cluster with static and dynamic contents”. Proc. IEEE Int'l Conf. on Cluster Computing, Chemnitz, Germany. Dec, 2000.

[7] Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. and Wang, R., “TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links2”, In Proceedings of ACM MOBICOM ’01, Jul. 2001.

[8] Crovella, M. and Bestavros, A., “Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes”, Proceedings of SIGMETRICS'96: The ACM International Conference on Measurement and Modeling of Computer Systems. Philadelphia, Pennsylvania. 1996.

[9] Lu, Y., Abdelzaher, T., Lu, Ch., Sha, L. and Liu, X. “Feedback Control with Queueing-Theoretic Prediction for Relative Delay”. Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2003.

[10] Dahlin, M. “Interpreting Stale Load Information,” Proceedings of the 19th International Conference on Distributed Computing Systems, May 1999.

[11] Kant, K., "On Aggregate Traffic Generation with Multifractal Properties", Proceedings of GLOBECOM 2000, Rio de Janeiro, Brazil.".

[12] Menascé, D.A., Almeida, V.A.F "Scaling for E-Business". Prentice Hall, 2000.

[13] Mosberger, D. and Jin, T. “httperf: A Tool for Measuring Web Server Performance”. First Workshop on Internet Server Performance. in WISP, pp. 59 - 67, Madison, WI, June 1998, ACM

[14] Pitkow, J.E. "Summary of WWW characterizations". Computer Networks and ISDN Systems, vol.30, num.1-7, pp.551-558, 1998

[15] Simulog Corp, The QNAP Reference Manual. V.4.

[16] Unger, S., Gansterer, W. and Juiz, C., “Simulation and Extensions of the VTP Protocol”. Submitted to the Networking 2004 Int. Conf., Athens, Greece.

[17] Vazhenin, D., Vazhenin, A. On-line WWW-Monitor. 3rd WSEAS Int.Conf. on Software Engineering, Parallel & Distributed Systems. (SEPADS 2004)

[18] Vink, B, Bruneel, H. “Multi-Server Queues Subject to Server Interruptions”. 5th WSEAS Int. Conf. on Applied Mathematics (MATH 2004)

[19] Webstone Benchmark. http://www.mindcraft.com/webstone/

[20] Yoon, W.O., Jung, J.H. and Choi, S.B. “Dynamic Load Balancing Algorithm using Execution Time Prediction on Cluster Systems”. The 2002 International Technical Conference On Circuits/Systems, Computers and Communications. Phuket, Thailand, July, 2002