Scheduling Strategy to Improve Response Time for Web Applications

Ludmila Cherkasova

Hewlett-Packard Laboratories,
1501 Page Mill Road, Palo Alto, CA 94303,
e-mail: {cherkasova}@hpl.hp.com, fax: 1-650-813 3381, phone: 1-650-857 3753

Abstract. We propose a tunable scheduling strategy that lies between FIFO and shortest-first, based on the value of a coefficient Alpha. If Alpha is set to zero then this strategy is just FIFO. Larger Alpha gets us closer to shortest-first strategy which is known to provide optimal response time. However, unlike the shortest-first, proposed scheduling strategy is starvation free. This scheduling strategy, called Alpha scheduling with no preemption, allows to improve overall response time per HTTP request more than 3 times under heavy load. We demonstrate our results with a simple simulation model using SpecWeb96 to generate representative WWW workload.

1 Introduction

With the growth of the World Wide Web, HTTP (Hyper Text Transfer Protocol [FGMFB97]) use has increased dramatically. It became the dominant method to transfer the data over Wide Area Networks. Some web sites became extremally popular and to function efficiently they needed fast, high-performance http servers. There are several factors limiting performance of Unix HTTP servers. The two basic and most likely ones are the connection rate (defined by the CPU power) and the network bandwidth [PDGWW96]. Thus, scheduling HTTP requests for the efficient usage of two basic shared resources on a web server such as CPU and shared network interface can have a significant impact on the performance of this web server.

A new scheduling technique for jobs competing to be serviced using shared resources is proposed. A scheduling strategy, called Alpha scheduling with no preemption, allows to improve overall response time per HTTP request more than 3 times under heavy load.

Using SpecWeb96 benchmark [SpecWeb96] to generate representative WWW workload and a simple simulation model, we show how this scheduling strategy can be applied to the HTTP requests queue. Different HTTP requests retrieve the files of different sizes which occupy shared system resources for different amounts of time: small files consume less CPU amounts as well as network bandwidth than the big ones. The requests are stored in a priority queue with a key, which is computed using a function of "clock" and the size of the requested file. Alpha scheduling logically looks similar to a shortest-first strategy due to a part of the key formula which depends on the size of the requested file. However, it allows to avoid the inevitable for shortest-first strategy starvation problem due to a part related to the "clock".

Proposed scheduling strategy allows to reorder the requests queue in such a way that it significantly reduces waiting time in a queue, and as a result significantly improves overall response time per request. Thus under the same load, a server using Alpha scheduling will have shorter queue of requests than a server operating under FIFO scheduling. This leads to better system resources utilization, and as a result to improved system throughput.

The remainder of this paper is organized as follows: Section 2 introduces the basic scheduling algorithm, called Alpha scheduling with no preemption; Section 3 describes how the proposed algorithm can be used to reorder HTTP requests for efficient processing; Section 4 presents the SpecWeb96 benchmark description; Section 5 presents the performance analysis study results for Alpha scheduling with no preemption using SpecWeb96 benchmark as a workload generator; Section 6 and Section 7 generalizes Alpha scheduling with no preemption to correctly handle requests from different classes.

Acknowledgements: My thanks to Tomas Rokicki with whom I had pleasure to work on related ideas for packet-switched interconnect [CR94], and to my colleagues Peter Phaal and Sekhar Sarukkai for useful discussions and remarks. My special thanks to colleagues from New Jersey Lab: Pranav Naik who has implemented Alpha scheduling in Apache code and proved that it works, and Hari Pulijal and Ashok Nadkarni who appreciated the idea and supported this work.

2 Alpha Scheduling with No Preemption

There is set of known scheduling algorithms proposed in the past such as round robin, different priority queue based algorithms (with aging and without, which are actively used in OS), Alpha scheduling [CR94], etc. They were proposed for different purposes: some of them intend to optimize overall service response time, some of them are targeting fair access in sharing the critical resources.

All of the algorithms mentioned above are strongly based on the assumption that a job can be partitioned in "chunks" and the scheduling dictates which job "chunk" is executed next. Often additionally, they have one more assumption that a short job can preempt the execution of the longer one.

Two well known scheduling disciplines which do not have above assumptions are FIFO (first-in, first-out) and "shortest job first". Typical current scheduling of HTTP queue is FIFO (with obvious extensions for high priority jobs). With such a strategy, starvation is impossible; each request will eventually be serviced. The maximum time waiting in the queue is proportional to the sum of the requested file lengths of all the requests in the queue. For example, if we have the following three requests in a queue:

- first request requires 1000 (conditional) time units to be serviced;
- second request requires 10 time units to be serviced;
- third request requires 20 time units to be serviced;

If these three requests are processed in FIFO order then

- first request response time is 1000 time units;
- second request response time is 1010 time units;
- third request response time is 1020 time units;

In such a way, two "short" requests processed after the "long" one, observe this "long" request processing time and include it in their response time. Resulting average response time is 1010 time units per request.

"Shortest job first" discipline provides the optimal response time. If shortest-first strategy is used to serve three requests above then

- as a first request will be served those that requires 10 time units;
- as a second request will be served those that requires 20 time units;
- as a third request will be served those that requires 1000 time units;

Resulting average response time is 343.3 time units per request.

"Shortest job first" is not used in practice because it leads to starvation problem of long jobs in presence of continuous arrivals of shorter jobs.

We propose a scheduling strategy that lies between FIFO and shortest-first, based on the value of a coefficient Alpha, and which is starvation free.

Consider requests $r_i (i \geq 0)$ competing to be serviced using shared resource S (or set of shared resources).

Let $cost(r_i)$ be a function reflecting amount of time the request r_i occupies shared resource S such that if request r_j occupies shared resource S less than request r_k then $cost(r_j) < cost(r_k)$.

To optimize the average response time per request r_i, which includes both waiting time in a queue and service time using shared resource S, we are going to schedule requests queue in the following manner.

Requests are stored in a priority queue.

Request r_i is inserted into the priority queue with a priority of

$$c + \alpha * cost(r_i).$$

where

- The parameter c is a running queue "clock" that starts at zero and increments for each serviced request r_i by the $cost(r_i)$.
- The tuning parameter α controls the balance between fairness and latency minimization; it can range from 0 to ∞

Requests with the lowest priorities get processed first.

If $\alpha = 0$ then this strategy is simply FIFO .

If $\alpha = 1$ or some other finite positive value, then this scheduling will not allow any single request to be delayed indefinitely by the other requests, no matter how the request stream looks like. For finite α every request is eventually delivered. This is because c increases with each request sent, so eventually every new request will be inserted in the priority queue after a given request. That is, starvation is impossible. Larger α provides better average response time because it allows larger key gap for insertion of "cheap" requests in front of "expensive" ones; smaller α provides better fairness among requests of different cost.

Let us call this scheduling discipline as Alpha scheduling with no preemption.

Values of parameter c and element keys are easily kept bounded, either by resetting c to zero whenever the request queue empties, or by performing a scan through the priority queue and decreasing all the priorities and c by the priority of the head request on those rare occasions when the clock (or lement key) is about to exceed some maximum.

3 Reordering HTTP Requests Queue Using Alpha Scheduling with No Preemption

An http server is a fileserver that uses TCP/IP to reliably transport data over Wide Area Networks. CPU service time and requirements for network bandwidth are directly proportional to the size of the requested file.

Using this observation we are going to reschedule HTTP requests queue in the following manner.

Let $cost(r_i) = length(r_i)$, where $length(r_i)$ is a file size retrieved by the request r_i.

Request r_i is inserted into the priority queue with a priority of

$$c + \alpha * length(r_i).$$

REMARK: In case of persistent connections [FGMFB97], the requests belonging to the same connection have to be serviced in order they arrive. In this case, the generalization of Alpha scheduling described in Section 6 has to be used.

4 SpecWeb96 Benchmark

We use SpecWeb96 benchmark [SpecWeb96] parameters to generate representative WWW workload, which is an industry standard benchmark for generating HTTP requests and measuring Web Servers performance.

File mix is defined by the files (requests) distribution from the following four classes:

– 0 Class: 100bytes - 900bytes (35%)
– 1 Class: 1Kbytes - 9Kbytes (50%)
– 2 Class: 10Kbytes - 90Kbytes (14%)
– 3 Class: 100Kbytes - 900Kbytes (1%)

The average file size for SpecWeb96 workload is 14,675 bytes.

We will show the impact of Alpha scheduling with no preemption on average response time for HTTP requests generated by SpecWeb96 workload.

There are two major contributors to the request response time:

– Request Response Time = waiting time in queue + processing and transfer time.

Our main strategy is to minimize waiting time in the queue.

5 Simulation Results of Alpha Scheduling with No Preemption Using SpecWeb96 Benchmark

To justify the proposed scheduling, the following simplified simulation model was built. For example, one can imagine a server which can serve X requests/sec as an upper limit. Applied to SpecWeb96 terminology it is X OPS/sec or X connections per second. We will apply load of 5% of X, 10% of X, ..., 95% of X to see the impact of servicing requests queue using the Alpha Scheduling with no preemption against the original FIFO strategy.

We have set a capacity of our server to 100 OPS/sec.

Figure 1 shows average response time for SpecWeb96: FIFO vs Alpha-scheduling with no preemption (α = 1,10,20,30).

The performance improvement for SpecWeb96 under Alpha scheduling with no preemption is extremally good. Parameter $\alpha = 30$ is close to optimal.

The interesting question is how has the performance improved for requests from different classes.

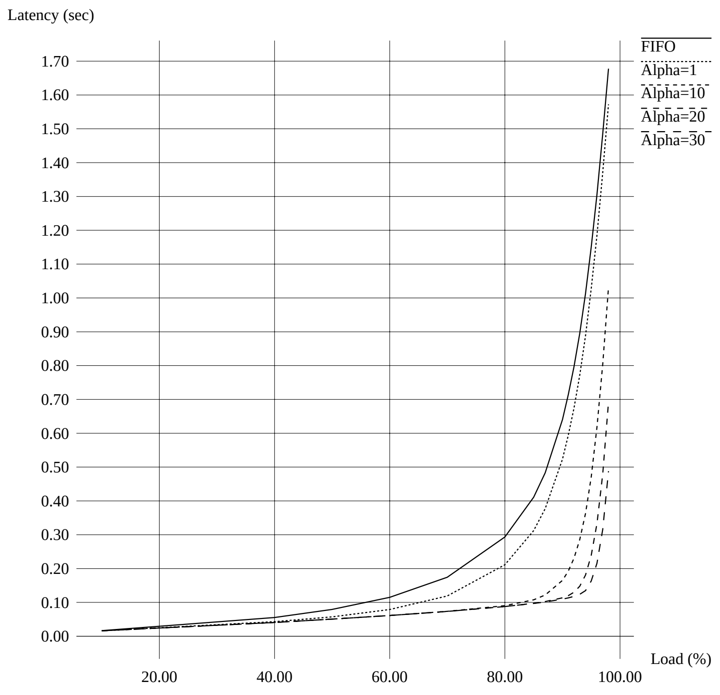

Fig. 1. Average Response Time for SpecWeb96: FIFO vs Alpha Scheduling with No Preemption.

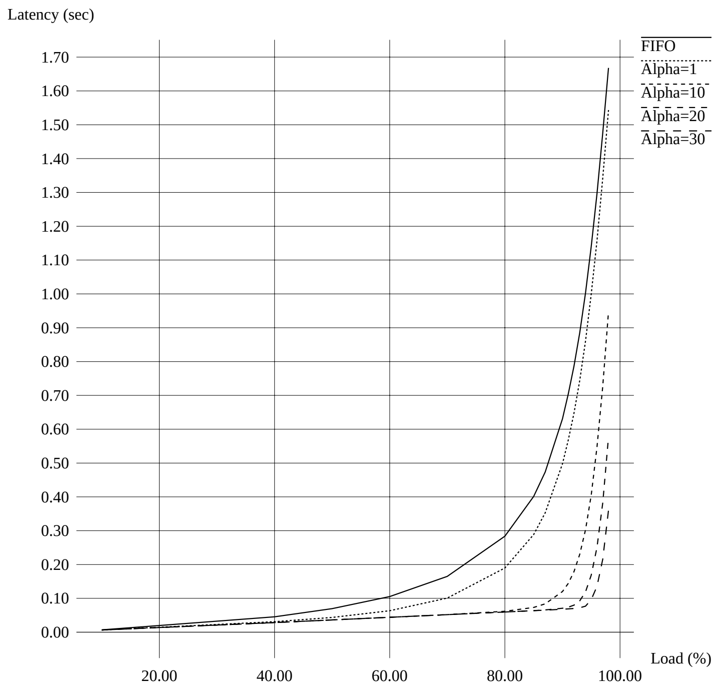

Fig. 2. Average Response Time for Class0 files in SpecWeb96: FIFO vs Alpha Scheduling with No Preemption.

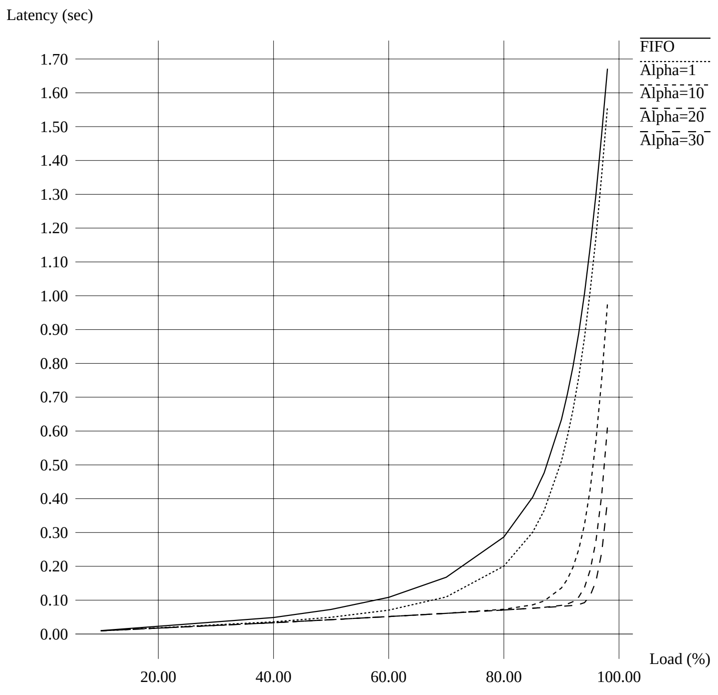

Fig. 3. Average Response Time for Class1 files in SpecWeb96: FIFO vs Alpha Scheduling with No Preemption.

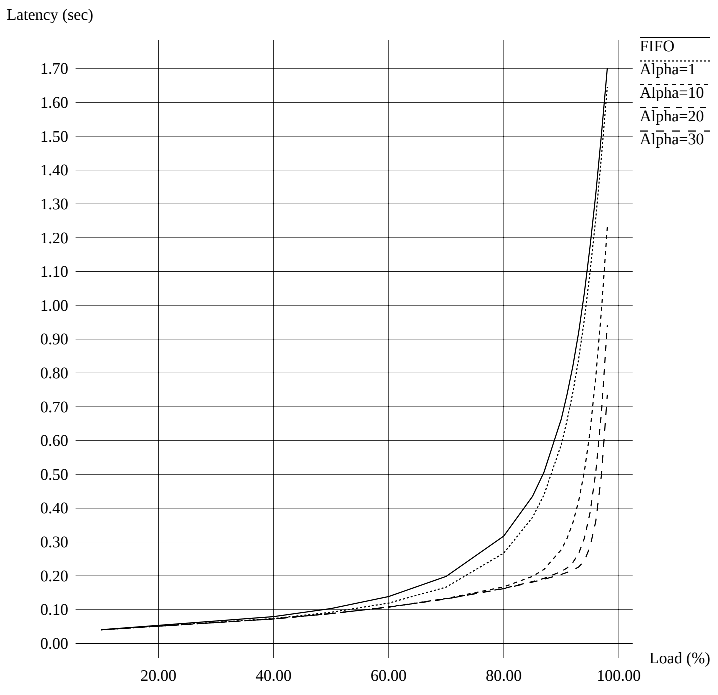

Fig. 4. Average Response Time for Class2 files in SpecWeb96: FIFO vs Alpha Scheduling with No Preemption.

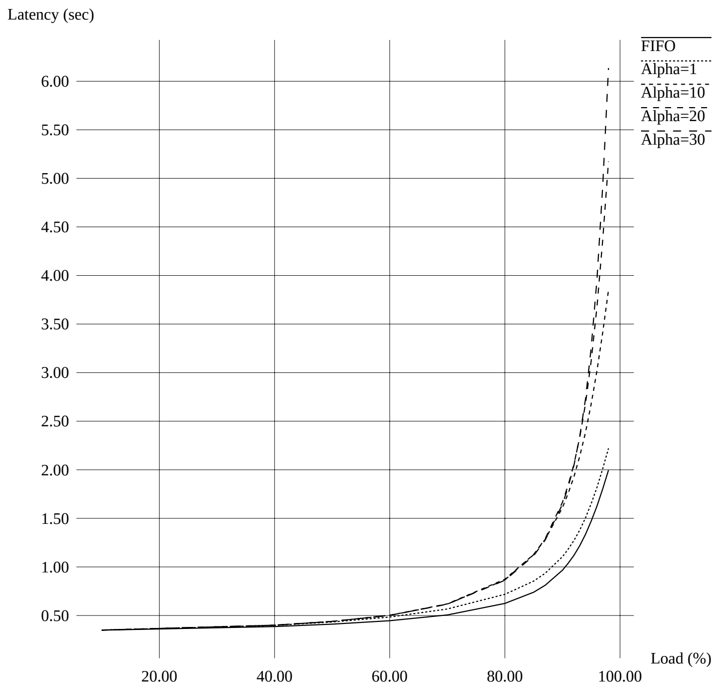

Fig. 5. Average Response Time for Class3 files in SpecWeb96: FIFO vs Alpha Scheduling with No Preemption.

Figures 3, 4, 5 show the average request response time for SpecWeb96 classes 0,1,2,3: FIFO vs Alpha scheduling with no preemption.

We can clearly see, that average response time of classes 0,1,2 (which make 99% of all requests) is significantly improved compared with average response time of these classes under FIFO strategy.

Only the response time of class 3 (1% of all requests) became worse. It should be expected. Proposed scheduling is favoring the “short” requests by letting them to overpass the “long” ones. However, the performance penalty for requests of class 3 is reasonable. It is about 2-3 times worse in a heavy loaded area.

6 Generalization of Alpha Scheduling with No Preemption to serve Different Job Classes

Consider requests $r_i (i \geq 0)$ competing to be serviced using shared resource S. Let these requests belong to different job classes $J_k (k = 1, ..., K)$. Additionally, there is a requirement that jobs from the same class should be executed in order they arrive.

REMARK: In case of persistent connections introduced by HTTP 1.1, there is a requirement that the requests belonging to the same open connection have to be serviced in order they issued.

In this case, the reordering can be done only among the requests belonging to different classes since the requests withing the same class should be processed in the same order they arrive. We will show a simple generalization of Alpha scheduling to handle this situation.

Let $cost(r_i)$ be a function reflecting amount of time the request r_i occupies shared resource S such that if request r_j occupies shared resource S less than request r_k then $cost(r_j) < cost(r_k)$.

Requests are stored in a general priority queue.

Let us introduce two additional parameters for each class $J_k (k = 1, ..., K)$:

- $cur\_queue\_length(J_k)$ to reflect the current number of requests in the general priority queue belonging to the class J_k and
- $cur\_pr(J_k)$ to reflect the priority which was assigned to the last request from the class J_k when $cur\_queue\_length(J_k)$ is not equal to zero. Otherwise, $cur\_pr(J_k)$ is zero.

Initially, both of these parameters are set to zero.

Request r_i from the job class J_k is inserted into the general priority queue with a priority $pr(r_i)$ calculated in the following manner.

- 1). First, the $pr(r_i)$ is calculated using usual formula from Section 2:
$$pr(r_i) = c + \alpha * cost(r_i)$$
- 2). If $pr(r_i) \geq cur\_pr(J_k)$ then request r_i is inserted into the general priority queue with the priority $pr(r_i)$ computed above, and $cur\_queue\_length(J_k)$ and $cur\_pr(J_k)$ are recomputed by the following formula:
$$cur\_queue\_length(J_k) = cur\_queue\_length(J_k) + 1$$
and
$$cur\_pr(J_k) = pr(r_i).$$
- 3). If $pr(r_i) < cur\_pr(J_k)$ then request r_i is inserted into the general priority queue with a priority
$$pr(r_i) = cur\_pr(J_k) + 1.$$
If this takes place, $cur\_queue\_length(J_k)$ and $cur\_pr(J_k)$ are recomputed by the following formula:
$$cur\_queue\_length(J_k) = cur\_queue\_length(J_k) + 1$$
and
$$cur\_pr(J_k) = pr(r_i).$$

Requests with the lowest priorities get processed first. Whenever the head request r_i, belonging to the job class J_k gets serviced, the following actions take place:

- The parameter c (a running general queue "clock" that starts at zero) is incremented by the $cost(r_i)$.
- $cur\_queue\_length(J_k) = cur\_queue\_length(J_k) - 1$.
- If $cur\_queue\_length(J_k)$ becomes equal to zero then $cur\_pr(J_k)$ is reset to zero too.

The tuning parameter α controls the balance between fairness and latency minimization in the same way as described above in Section 2; it can range from 0 to ∞.

The proposed modification of Alpha scheduling allows to service requests from the same job class in order they arrive, while optimizing the overall response time via efficient rescheduling of the requests from the different job classes.

7 Extention of Alpha Scheduling with No Preemption to serve Job Classes of Different Priorities

Let the job classes have different priorities. If J_m job class has higher priority than J_n job class then it means that requests from J_m has to be served before requests from J_n.

The simplest way to optimize response time for those jobs is to introduce a priority queue (with its own parameters) per priority class and schedule requests within their priority class (queue).

Service starts with highest priority queue, and it is switched to next priority queue only when highest priority queue becomes empty, etc.

REMARK: There is an equivalent way to serve requests from job classes having different priorities using one priority queue. It can be achieved by splitting the range of general priority queue keys into disjoint sections (ordered the same way as correspondent priority classes) and using this section of a priority queue (with its own parameters) to serve the requests of correspondent priority class.

8 Conclusion

In this paper, we have introduced a new scheduling strategy, Alpha scheduling with no preemption, and using SpecWeb96 benchmark we have shown how it can significantly improve the overall response time per HTTP requests by simply scheduling the requests appropriately.

Reordering the HTTP requests assumes that the HTTP request processing is split into two phases: parsing the request where the header of the request is parsed and the size of the requested file is recognized, and completing the request where the file is moved from memory or disk to network.

There is another place inside the Web Server where Alpha scheduling with no preemption can be used. Proposed scheduling can be applied to reorder the replies for processed HTTP requests, i.e. to reorder the files at network interface for their efficient network transfer. This will have similar performance benefits and will be simpler to implement, since at network interface the size of file to transfer is known and it covers all types of HTTP requests.

In both cases, reducing overall response time per request will lead to better system resources utilization, and as a result to improved system throughput.

References

[CR94] Cherkasova, L. and Rokicki, T.: Alpha Message Scheduling for Packet-Switched Interconnects. HP Laboratories Report No. HPL-94-71, August, 1994.

[FGMFB97] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. Internet Proposed Standard Protocol, RFC-2068. URL http://www.ics.uci.edu/pub/ietf/http/.

[PDGWW96] Prefect, F., Doan, L., Gold, S., Wicki, Th., Wilcke, W.: Performance Limiting Factors in HTTP (Web)Server Operations. In Proceedingds of COMPCON96, Santa Clara, California, February, 1996, p.267-272.

[SpecWeb96] The Workload for the SPECweb96 Benchmark. URL http://www.sepcbench.org/osg/web96/workload.html