Performance Simulations of a QoS Aware Caching Method

Pertti Raatikainen [1], Mika Wikström [2], and Timo Hämäläinen[2]

[1] VTT Information Technology, Telecommunications
P.O.Box 1202, FIN-02044 VTT, Finland
pertti.raatikainen@vtt.fi
[2] University of Jyväskylä, Department of Mathematical Information Technology
P.O. Box 35, FIN-40351 Jyväskylä, Finland
timoh@cc.jyu.fi, wikstrom@mit.jyu.fi

Abstract. Research of web-servers has recently addressed the problem of content distribution coupled with quality of service (QoS). Due to the explosive growth of services offered over the Internet, novel mechanisms are needed for IP based service delivery to scale in a client-transparent way. This paper addresses the above problem considering also utilization of available processing power of servers. Many developed caching systems dedicate a fixed portion of the processing power for higher QoS services leading to lowered overall throughput of the server system. Here we introduce and simulate a QoS aware caching scheme that offers lower response delay for higher quality services and additionally optimizes utilization of the available processing power.

1. Introduction

Distribution of web-content to servers and arbitration of requests among a cluster of servers have been in focus of intense research. Location of content among the servers and service admission control are the major problems in server farm implementations. Since the same content can be located to several servers, an additional problem appears in maintaining an established connection to specific content on a known server throughout a session. A number of different schemes to locate content to servers and balance loading between them have been developed [1, 2, 3, 4, 5]. The most sophisticated ones, often called web-switches, are content aware and offer methods to maintain connection to a given server all along a session [10].

The content based switching schemes enable categorization of connections, e.g., based on the requested service, user or combination of both. Good customers or access to certain high quality services may be directed to less loaded servers enabling lower response delay. This causes skewing of the processing balance and at worst some servers are overloaded while others remain lightly loaded. Optimum loading implies that loading degree of each server can be fixed to be equal.

Caching combined with content aware switching is a technique that can be used in lowering response delay for some customers or services, while maintaining the loading balance between the servers. Since the number and size of web-files is normally large compared to the available cache size, a subset of web-pages can be located to cache

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 1202-1207, 2002.

[5, 7, 8]. To maximize the number of cache hits when the cache capacity is exceeded, a number of caching algorithms have been developed to overwrite less frequently accessed pages with more frequently accessed ones [3, 6, 7].

This paper introduces a caching method that allows to distribute web-content based on QoS requirements and simultaneously balance load among a cluster of servers to enable maximum utilization of the aggregate processing power. The objective in locating web-files to cache is to maximize cache hit-rate and thereby minimize response delay. Chapter 2 introduces the caching algorithm and related simulation model. Chapter 3 gives some simulation results and chapter 4 concludes the paper.

2. QoS Aware Caching Scheme

The objective in developing the caching scheme was to improve cache system performance measured in service response delay and utilization of the server system's processing power. Response delay is lowered by increasing the cache hit-rate and utilization of processing power is optimized by locating content randomly to servers. When the number of web-files is large, random location policy leads to uniformly distributed load among the servers.

2.1 Caching Method

Each server in a cluster is supposed to have a hard disk and cache memory of known size. The memory sizes, processing power and memory reading delays may vary from a server to another. Web-files stored on servers are categorized into a fixed number of QoS classes. In a general case, the number of files in those classes and sizes of the files are different.

The web-files are located randomly to servers and upon storing a file it is associated with one of the QoS classes. If cache is small compared to the aggregate size of files in a server system, a predetermined percentage (p_c) of files from each QoS class can be located to cache. The highest QoS classes have priority over the lower classes and the most requested files of each QoS class are selected first. Files that cannot be stored in cache are located on hard disk. If the cache is large enough to store a fixed percentage p_c of files from all QoS classes then some of the leftovers can also be placed to cache. The order in which they are stored follows the QoS class priorities and request rate intensities of the files.

2.2 Simulation Model

The simulations are carried out by applying a generic cache simulation model introduce in [9]. It allows manipulation of a number of parameters that characterize the introduced QoS based caching scheme. Performance of the system can be studied by varying the cache sizes, number and size of web-files, number of QoS classes, number of files in each QoS class, file request rates and processing power of servers.

Logically the model is divided into five decision-making blocks (see Fig. 1). At the top is a block, which decides whether the next event is related to an incoming or outgoing request. A request coming from a client is considered an incoming one and a request that has been processed and is being directed to a server an outgoing one.

In case of an incoming request, the algorithm first chooses the server that has the requested file. Different sorts of selection algorithms can be implemented based on the selected file location policy. Then the algorithm checks whether the selected server is available for service. If it is not available, the request is put to into a queue where it will stay until the algorithm enters the *outgoing* leg.

If the server is available, the simulation enters the block, which checks whether the requested file is in the cache. Different policies can be used in placing files to the cache. Here, priority is given to files of the highest QoS classes. If the requested file is in the cache, it is read and marked as the most recently accessed one. If the file is not found in the cache, it must be on the hard disk and the simulation continues to the „*Read file*" -block. The file is read from the disk and the deployed caching scheme decides how to proceed (store it to the cache or leave on the disk). This block allows the use of different caching methods and comparison of their performances.

If in the topmost block the *outgoing* leg is chosen then the server where the file is being served is located and the file is removed from service. After that, it is checked whether there is a queue for that particular service. If there is no queue, the model starts another iteration round, i.e., it starts to process the next event. If there is a queue for that service then the first request in the queue is serviced.

Furthermore, the server system keeps record of all file, their QoS classes, file locations in the server system, sizes of the caches and their degree of fullness. Access rates of the files also need to be recorded to enable request rate based location of files when the servers are running out of cache memory.

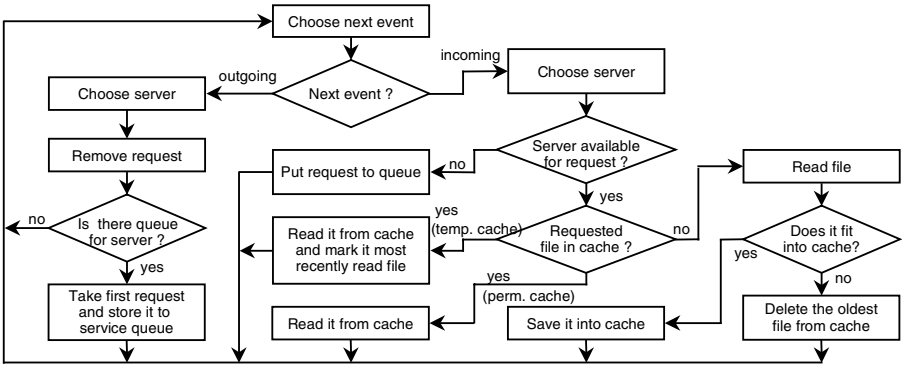

Fig. 1. Flow chart of the simulation model.

3. Simulation Examples

To demonstrated performance of the developed caching system, simulation results for a system of three servers are introduced. When a new file is inserted into the caching system, it is associated with one of four possible QoS classes and it is given a size that belongs to one of three possible categories: 1, 5 or 10 kilobytes (kb). Ten per cent (%) of the files belong to the highest QoS class (QoS1), 20 % to the second highest (QoS2), 30 % to the next (QoS3) and the rest 40 % to the lowest class (QoS4).

The simulated files were divided into two equally large groups based on their mean request rate; the higher rate was twice that of the lower one. Request rate intensities were exponentially distributed. In order to keep the simulation times reasonable, the simulated system had only 300 files. These were located randomly to servers giving approximately 100 files (about 550 kb) per server. The objective was to study performance of the proposed caching system by analyzing variation of the cache hit rates of the different QoS classes as a function of the cache size and thereby estimate the system response time. The cache sizes were varied between 0 and 550 kb, while the number of files and their sizes on each server were kept fixed.

As a comparison, performance of a non-QoS aware caching scheme, simulated in [9], was analyzed. The configuration set-up of the non-QoS aware system and the number of files, their sizes, request rates and locations were identical with those of the simulations of the QoS aware system. The only difference was the applied caching method, which did not account QoS. Instead, it exercised first-in-first-out (FIFO) discipline. At the start of a simulation, files on each server were randomly located to cache and on hard disk. The cache performed like a FIFO memory in which the most recently requested files were located at the end and less recently requested ones at the head of the FIFO queue. Each time a file (either on hard disk or in cache) was requested, it was moved to end of the FIFO. Files already in the cache were shifted towards the head of the FIFO queue. When the FIFO was full, the file at the head of the queue was removed to the hard disk.

Fig. 2 and 3 illustrate performance of the proposed QoS aware caching scheme when the proportion of files (of each QoS class) that could be located to the cache was 80 % (p_c = 0.8) and 100 % (p_c = 1.0). Fig. 4 shows corresponding results for the non-QoS aware system and Fig. 5 demonstrates the average response delay performance of these three simulated cases. The horizontal axis in all these figures gives the percentual size of the cache compared to the aggregate size of all files in the system. In Fig. 2 to 4, the vertical axis gives the percentual proportion of cache hits compared to the total number of simulated file requests. In Fig. 5 the vertical axis gives response delay normalized to file reading delay from hard disk. File reading delay from hard disk was assumed to be ten times that from cache.

Fig. 2 and 3 show that the cache hit rates of the different QoS classes follow the prefixed priorities quite nicely. The highest QoS classes reach the 100 % cache hit rate limit faster in Fig. 3 than in Fig. 2. The reason for this is that the system in Fig. 2 can store only 80 % of files of the highest QoS classes to the cache when the cache size is small. The rest of the highest QoS class files can be located to the cache only if the cache is large enough to include more than 80 % of files of all the QoS classes.

When comparing curves in Fig. 2 and 3 with those in Fig. 4, it is obvious that the proposed caching method is capable of supporting QoS. The non-QoS aware system does not show much difference between the QoS classes. The reason for the slightly differing curves is that the simulation tool assigned a QoS class, file size and request rate group randomly to every file. Thus the total size of files in each QoS class and the sizes of the two request rate groups were not exactly the above given ones.

The average response delay (see Fig 5) of the non-QoS aware system was always better than that of the QoS aware system. The reason for this is that in the QoS aware system files of the highest QoS classes have (regardless of their request intensity)

priority over the lower QoS class files in locating files to cache. This lowers the average cache hit rate and lengthens the average response delay.

Simulations have shown that the performance gap between the QoS and non-QoS aware system can be reduced by decreasing the difference between the lowest and highest request rate value or decreasing the value of p_c. Allowing p_c to change step by step with the size of cache, it is possible to share cache memory fairly between the QoS classes, offer relatively good system level response delay and still maintain QoS awareness.

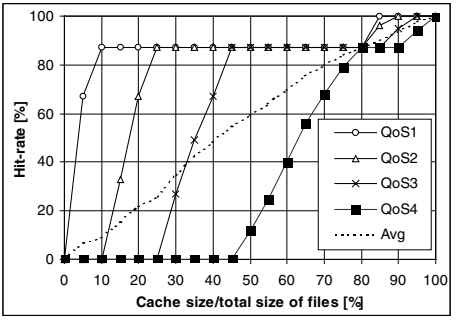

Fig. 2. Cache hit-rates of QoS aware system ($p_c = 0.8$)

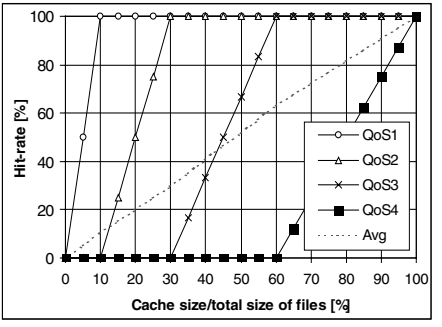

Fig. 3. Cache hit-rates of QoS aware system ($p_c = 1.0$)

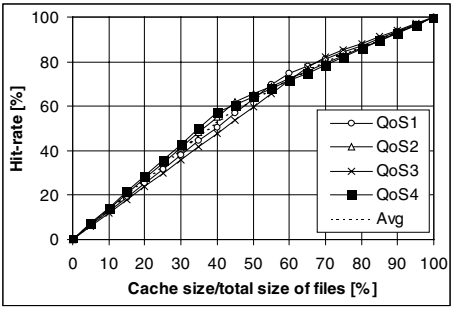

Fig. 4. Hit-rates of non-QoS aware system

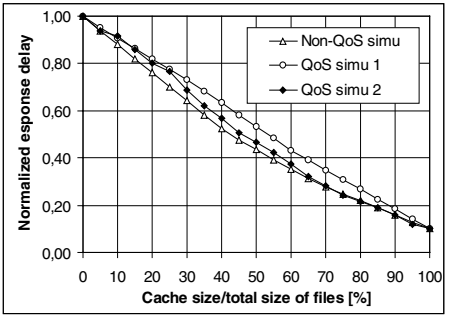

Fig. 5. Response delays of simulated systems

4. Conclusions

This paper presents a quality of service based caching scheme that allows support of different QoS classes offering shorter response delay for higher QoS class requests than for lower class ones. Processing power of the server system is utilized effectively by locating web-files randomly on the different servers and thus dividing the processing load uniformly among the servers.

The developed caching scheme was modeled by a generic simulation tool, which had previously been developed by the author, and was here used to evaluate performance of the QoS aware caching system. The model includes a number of adjustable

parameters that can be varied to find optimal performance in different simulation cases. As a comparison a non-QoS aware system was also modeled to find out possible pros and cons of the developed QoS aware scheme.

The carried out simulations showed that the QoS aware caching method is clearly able to support different QoS classes. The only drawback was found in the system level cache hit rate. Since the highest QoS class files were located first to the cache memories, the cache included also less frequently requested files and the average cache hit rate was found to be lower than in the comparative non-QoS aware system. However, the discovered difference was an acceptable one. The average response delay can be lower by decreasing the portion of the highest QoS class files that can be located to the cache memory thus giving room for the lower QoS class files.

It is for further study, to enhance the developed caching scheme to implement a more efficient content distribution algorithm. The objective is to add some feedback to the algorithm and let the allocation parameters to be adjusted dynamically to respond better to changes in the servers' conditions.

References

[1] Blaze M., Alfonso R.: Dynamic Hierarchical Caching in Large-Scale Distributed File Systems. In: Proceedings of International Conference on Distributed Computing Systems, Yokohama (Japan), June 1992, pp. 521-528.

[2] Dahlin M.D., Wang R., Anderson T. E., Patterson D.: Cooperative caching: Using remote client memory to improve file system performance. In: Proceedings of Operating Systems Design and Implementation Symposium, Monterey (USA), Nov. 1994, pp. 267-280.

[3] Dan A., Towsley D.: An approximate analysis of the LRU and FIFO buffer replacement schemes. In: ACM SIGMETRICS, May 1990, pp. 143-152.

[4] Feeley M., Morgan W., Pighin F., Karlin A., Levy H., Thekkath C.: Implementing global memory management in a workstation cluster. In: Proceedings of the 15th ACM Symposium on Operating Systems Principles, Colorado (USA), Dec. 1995, pp. 201-212.

[5] Patterson R. H., Gibson G. A., Ginting E., Stodolsky D., D. Zelenka D.: Informed Prefetching and Caching. In :Proceedings of the 15th ACM Symposium on Operating System Principles, Colorado (USA), Dec. 1995, pp. 79-95.

[6] Chou H., DeWitt D.: An evaluation of buffer management strategies for relational database systems. In: Proceedings of the 11th VLDB Conference, Stockholm (Sweden), August 1985, pp. 127-141.

[7] O'Neil E. J., O'Neil P. E., Weikum G.: The LRU-k page replacement algorithm for database disk buffering. In: Proceedings of International Conference on Management of Data, Washington D.C. (USA), May 1993, pp. 297-306.

[8] Cao P., Felten E. W., Li K.: Application-controlled file caching policies. In: Proceedings of 1994 Usenix Summer Technical Conference, June 1994, pp. 171-182.

[9] Hämäläinen T., Wikström M., Raatikainen P.: A Simulation Model for Studying of Caching Algorithms. In: Proceedings of International Conferences on Info-tech and Info-net (ICII 2001), Beijing (China), Nov. 2001, pp. 599 - 604..

[10] Apostopoulos G., Aubespin D., Peris V., Pradhan P., Saha D.: Design, Implementation and Performance of a Content-Based Switch. In: Proceedings of INFOCOM 2000, IEEE, pp 1117 – 1126.