Differentiated Caching Services; A Control-Theoretical Approach

Ying Lu, Avneesh Saxena and Tarek F. Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22903 *

Abstract

The increasing diversity of Internet appliances calls for an architecture for performance differentiation on information access. The World Wide Web is the dominant interface for information access today. Web proxy caching is the key performance accelerator in the web infrastructure. While many research efforts addressed performance differentiation in the network and on web servers, providing multiple levels of service in proxy caches has received much less attention.

This paper has two main contributions. First, we describe, implement, and evaluate an architecture for differentiated content caching services as a key element of the Internet infrastructure. Second, we describe a control-theoretical approach that lays well-understood theoretical foundations for resource management to achieve performance differentiation in proxy caches. We describe our experiences with implementing the differentiated caching services scheme in Squid, a popular proxy cache used by many ISPs today. Experimental study and analyses prove that differentiated caching services provide significantly better performance to the premium content classes.

1 Introduction

The phenomenal growth of the Internet as an information source makes world-wide information access one of its most important applications today. The increasing diversity of Internet information appliances from high end workstations to wireless devices calls for customization of information access performance. The Web today is the primary interface for information access. The key performance acceleration mechanism in the web infrastructure is web caching. To meet the increasingly diversified performance demands of appliance-heterogeneity, in this paper we investigate an architecture for performance differentiation in web proxy caches. The architecture allows different classes of content to receive different performance (i.e., QoS) levels.

The most widely-used performance metric in the context of web caching is the cache hit ratio, H. It is related directly to performance measures of interest to the clients such as average response time. For example, if the average cache hit takes T_{hit} time units and the average cache miss takes T_{miss} time units, the average response time on information access is approximately $T_{access} = H*T_{hit}+(1-H)T_{miss}$. Assuming that T_{miss} is roughly $T_{hit} + T_{backbone}$, where $T_{backbone}$ is the average roundtrip delay from the cache to an origin server, then simple algebraic manipulation yields $T_{access} = T_{miss} - HT_{backbone}$. If α is the ratio $T_{hit}/T_{backbone}$, it follows that $T_{access} = (1+\alpha-H)T_{backbone}$. From this last expression note that if the performance bottleneck is in the backbone, $\alpha << 1$, and hence caching has an important effect on reducing average service time as H increases. Conversely, if the performance bottleneck is on the side of the client, $\alpha >> 1$, and hence, performance is not affected significantly even if H is reduced. The above approximate argument demonstrates that in an environment with heterogeneous clients, different clients perceive caching performance speedup differently. An cache resource allocation policy that maximizes aggregate *client-perceived* performance should provide a different H to different client classes. Hit-ratio differentiation among different classes implies both partitioning of content and displacement of more popular content of less favored classes by less popular content of more favored classes. While the favored classes receive a better hit ratio, the overall cache hit ratio is reduced. This reduction is consistent with optimizing client-perceived performance, since the favored classes are presumably more sensitive to changes in hit ratio. It is precisely the difference in client sensitivity to changes in the cache hit ratio that motivates differential treatment of content.

We adopt a proportional differentiated services model for cache hit ratio differentiation. For example, one can specify that the hit ratio of the "premium" class

*The work reported in this paper was supported in part by the National Science Foundation under grant No. CCR-0093144.

should be twice that of the "basic" class. By avoiding absolute specification of the hit ratio, the specified performance differentiation can be enforced regardless of client access patterns. There are several immediate applications of caches with a performance differentiation mechanism. One such application is the differentiation between standard web content and the emerging wireless content. Wireless appliances require new content types for which a new language, the Wireless Markup Language (WML) was designed. Since these appliances are slow, proxies that support performance differentiation can get away with lower hit rates on WML traffic to give more resources to faster clients (that are more sensitive to network delays upon cache misses) thus optimizing aggregate resource usage. This is especially true of caches higher in the caching hierarchy where multiple content types are likely to be intermixed.

Alternatively, a web cache may choose to classify content by the identity of the requested URL. For instance, an ISP (such as AOL) can have agreements with preferred content providers to give their sites better service for a negotiated price. Our architecture would enable such differentiation to take place.

One significantly novel aspect of this paper is that we use a control-theoretical approach for resource allocation to achieve the desired performance differentiation. Digital Feedback Control Theory offers techniques for developing controllers that utilize feedback from measurements to adjust the controlled performance variable such that it reaches a given set point. By casting cache resource allocation as a controller design problem, we are able to arrive at an allocation algorithm that converges to the desired performance differentiation in the shortest time in the presence of very bursty self-similar cache load.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 describes the architecture of a cache that supports service differentiation. A control-theoretical approach is described to achieve the desired distance between performance levels of different classes. Section 4 describes the implementation of this architecture on Squid, a very popular proxy cache in today's web infrastructure. Section 5 gives experimental evaluation results of our architecture, obtained from performance measurements on our modified Squid prototype. Section 6 concludes the paper and discusses avenues for future work.

2 Related Work

Early research in the web community has traditionally been on performance optimization rather than service differentiation and QoS management. Protocol issues such as performance interactions between TCP and HTTP have been investigated at length [9, 7, 17, 15, 18]. Load balancing [12, 5, 23] and admission control [19] have often been used to improve scalability and predictability of web services.

More recently, architectures were proposed for differentiated services and QoS control. The authors of [14] proposed and evaluated an architecture in which restrictions are imposed on the amount of server resources (such as threads) available to low priority clients. In [4, 1] admission control and scheduling algorithms are used to provide premium clients with better service. In [10] a server architecture is proposed, maintaining separate service queues for premium and basic clients and thus, facilitating their differential treatment. In [2, 3] an architecture was proposed for content adaptation that supports tiered services and performance isolation among co-hosted sites.

The above efforts addressed performance differentiation at the origin server. Caches introduce a crucial additional degree of complexity to performance differentiation, namely, the ability to import and replace items on demand. On an origin server all requests are served locally. Thus, the perceived performance of the server depends primarily on the order in which clients are served. The performance speedup due to a cache, on the other hand, depends primarily on whether or not the requested content is cached. Thus, hit ratio, rather than service order, is a significant performance factor. Performance differentiation in caches, therefore, requires a new approach.

Web caching research has traditionally focused on replacement policies for optimal disk storage allocation [11, 13, 6]. Biased replacement policies were proposed to favor important users [20]. While biased replacement policies come close in spirit to our goals, prior work does not guarantee a specifiable difference between the performance levels of different classes.

3 Differentiated Caching Services

In this section we present our architecture for service differentiation among multiple classes of content cached on a web proxy. Intuitively, if we assign more storage space to a traffic class, its hit rate will increase, and the average response time of client accesses to this content will decrease.[1] If we knew future access patterns, we could tell the amount of disk space that needs to be allo-

[1] This presumes that the request traffic on the cache is not enough to overload its CPU and I/O bandwidth.

cated to each traffic class ahead of time to achieve their performance objectives. In the absence of such knowledge we need a feedback mechanism to adjust space allocation based on the difference between actual system performance and desired performance. This feedback mechanism is depicted in Figure 1 which illustrates a feedback loop that controls performance of a single traffic class. One such loop is needed for each class.

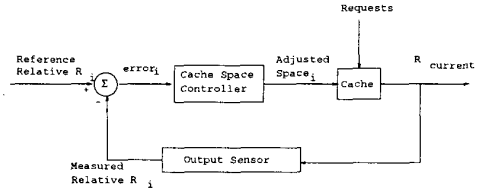


Figure 1. The Hit Rate Control loop

In Figure 1, a *service differentiation policy* dynamically determines the desired performance level for the traffic class under consideration. While we believe that our general feedback techniques can work with any performance metric that can be affected by cache resource allocation, in this paper we consider hit ratio as our metric of choice. Let there be n content (i.e., traffic) classes in the system. Let the measured average hit ratio of content class i be H_i. The differentiation policy specifies that the hit ratio of different classes should be related by the expression: $H_1 : H_2 : ... : H_n = C_1 : C_2 : ... : C_n$, where C_i is a proportionality constant or weight of class i. We define the *relative* hit ratio, R_i, of content class i to be $R_i = H_i/(H_1 + H_2 + ... + H_n)$. It determines how the class is performing relative to other classes. The desired performance of class i should be $R_{i_{desired}} = C_i/(C_1 + C_2 + ... + C_n)$. We call the difference $R_{i_{desired}} - R_i$ the performance error e_i of class i. An appealing property of this model is that the aggregate performance error of the system is always zero, because $\sum_{1\leq i\leq n} e_i = \sum_{1\leq i\leq n}(R_{i_{desired}} - R_i) = \frac{\sum_{1\leq i\leq n} C_i}{C_1+C_2+...+C_n} - \frac{\sum_{1\leq i\leq n} H_i}{H_1+H_2+...+H_n} = 1 - 1 = 0$. As we show in the next section, this property allows us to develop resource allocation algorithms in which resources of each class are heuristically adjusted independently of adjustments of other classes, yet the total amount of allocated resources remains constant equal to the total size of the cache.

3.1 Performance Differentiation Problem

We cast the performance differentiation problem as a closed-loop control problem. Each class of content i is assigned a different amount of cache storage s_i, such that $\sum_i s_i$ is the total size of the cache. Our objective is to achieve the desired hit ratio differentiation, i.e., to reduce the performance error e_i to zero for each content class. A zero error entails that the hit ratio of different classes is proportionally related by the specification $C_1 : C_2 : ... : C_n$. We reduce the error e_i to zero by adapting the storage allocation. We need to show that (i) our resource allocation heuristic converges to the desired relative hit-ratio specification, and that (ii) the convergence is bounded by a finite constant that is a design parameter. To provide these guarantees, we rely on feedback control theory in designing the resource allocation heuristic. The heuristic is invoked at fixed time intervals at which it corrects resource allocation based on the measured performance error. Let the measured performance error at the kth invocation of the heuristic be $e_i[k]$. To compute the correction $\delta s_i[k]$ in resource allocation, we use a linear function $f(e_i)$ where $f(0) = 0$ (no correction unless there is an error). At the kth invocation, the heuristic computes:

$$\forall i : \delta s_i[k] = f(e_i[k]) \quad (1)$$

The space allocation is then adjusted:

$$\forall i : s_i[k] = s_i[k-1] + \delta s_i[k] \quad (2)$$

If the computed correction $\delta s_i[k]$ is positive the space allocated to class i is increased by $|\delta s_i[k]|$. Otherwise it is decreased by that amount. Since the function f is linear, $\sum_i f(e_i[k]) = f(\sum_i e_i[k])$. In Section 3 we showed that $\sum_i e_i[k] = 0$. Thus, $\sum_i f(e_i[k]) = f(0) = 0$. It follows that the sum of corrections across all classes is zero. This property is desirable since it ensures that while the resource adjustment can be computed independently for each class based on its own error e_i, the aggregate amount of allocated resources does not change after the adjustment. This amount is always equal to the total size of the cache. Next we show how to design the function f in a way that guarantees convergence of the cache to the specified performance differentiation within a *single sampling period.*

3.2 Control Loop Design

To design the function f, a mathematical model of the control loop is needed. Consider some arbitrary content class i. Every sampling instant, an error $e_i[k]$ is measured and an adjustment $\delta s_i[k]$ is carried out by the replacement policy. The adjustment affects the hit ratio. A monotonically increasing relation is observed between cache space and hit probability. The parameter K_c is a linearization of that relation. The *expected* hit ratio at the end of a sampling interval (where expectation is used

in a mathematical sense) is determined by the space allocation and the resulting hit probability that took place at the beginning of the interval. Hence:

$$E(\delta H_i[k]) = K_c \delta s_i[k-1] \quad (3)$$

and $E(H_i[k]) = H_i[k-1] + E(\delta H_i[k])$. Remember that the relative hit ratio (the controlled performance variable) is defined as $R_i = H_i/(H_1 + H_2 + ... + H_n)$. Unfortunately, the measured $H_i[k]$ might have a large standard deviation around the expected value unless the sampling period is sufficiently large. Thus, using $H_i[k]$ for feedback to the controller will introduce a significant random noise component into the feedback loop. Instead, the measured $H_i[k]$ is smoothed first using a low pass filter. Let the smoothed $H_i[k]$ be called $M_i[k]$. It is computed as a moving average as follows:

$$M_i[k] = aM_i[k-1] + (1-a)H_i[k] \quad (4)$$

In this computation, older values of hit ratio are exponentially attenuated with a factor a, where $0 < a < 1$. Values of a closer to 1 will increase the horizon over which H_i is averaged and vice versa. The corresponding smoothed relative hit ratio is $M_i[k]/\sum_i M_i[k]$. This value is compared to the set point for this class and the error is used for space allocation adjustment in the next sampling interval, thereby closing the loop.

Next we take the z-transform of Equations (1), (2), (3), and (4) and draw a block diagram that describes the flow of signals in the hit ratio control loop. The z-transform is a widely used technique in digital control literature that transforms difference equations into equivalent algebraic equations that are easier to manipulate. Figure 2 depicts the control loop showing the flow of signals and their mathematical relationships in the z-transform. The z-transform of the heuristic resource reallocation function f is denoted by $F(z)$.

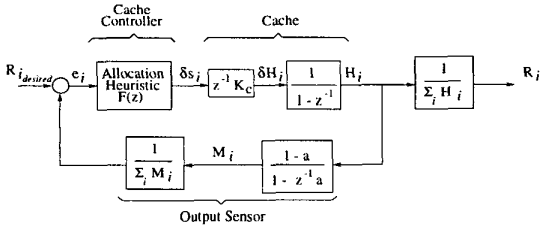


Figure 2. z-Transform of the Control loop

We can now derive the relation between R_i and $R_{i_{desired}}$. From Figure 2, $R_i = e_i F(z)G(z)$, where:

$$G(z) = \frac{z^{-1}K_c}{(1-z^{-1})\sum_i H_i} \quad (5)$$

Substituting for e_i, we get: $R_i = (R_{i_{desired}} - H_i \frac{1-a}{1-z^{-1}a} \frac{1}{\sum_i M_i}) F(z)G(z)$. Note that since M_i is a smoothed H_i, the ratio $H_i/\frac{1}{\sum_i M_i}$ that appears in the preceding equation is approximately equal to the relative hit ratio R_i. Hence, approximately, $R_i = (R_{i_{desired}} - R_i \frac{1-a}{1-z^{-1}a}) F(z)G(z)$. Using, simple algebraic manipulation:

$$R_i = \frac{F(z)G(z)}{1 + \frac{1-a}{1-z^{-1}a} F(z)G(z)} R_{i_{desired}} \quad (6)$$

To design the allocation heuristic, $F(z)$, we specify the desired behavior of the closed loop, namely that R_i follows $R_{i_{desired}}$ within one sampling time, or $R_i[k] = R_{i_{desired}}[k-1]$. The requirement translates to $R_i = z^{-1} R_{i_{desired}}$. Hence, $\frac{F(z)G(z)}{1 + \frac{1-a}{1-z^{-1}a} F(z)G(z)} = z^{-1}$, from which $F(z) = \frac{z^{-1}(1-z^{-1}a)}{(1-z^{-1})G(z)}$. Substituting for $G(z)$ from Equation (5) we arrive at the z-transform of the desired heuristic function, namely:

$$F(z) = \frac{(1-z^{-1}a)\sum_i H_i}{K_c} \quad (7)$$

The corresponding difference equation is:

$$\delta s_i[k] = f(e_i) = \frac{\sum_i H_i}{K_c}(e_i[k] - ae_i[k-1]) \quad (8)$$

The above equation gives the adjustment in the disk space allocated to class i given the performance error of that class, e_i and the aggregate of all measured hit ratios. In the actual implementation of the heuristic we substitute the smoothed aggregate, $\sum_i M_i$, for $\sum_i H_i$ to avoid noise-related problems. The resulting closed loop is stable, because the closed loop transfer function, z^{-1}, is stable and because the open loop transfer function does not contain unstable poles or zeros [22]. A method for estimating the parameter K_c which describes the relation between cache size and hit probability is described in a technical report [21].

4 Implementation

We modify Squid, a widely used proxy-cache to validate and evaluate our QoS-based resource allocation architecture. Squid is an open-source Internet cache [13] designed for high performance caching strategies implementation. This section addresses issues involved in the design and implementation of our control-loop based algorithm. The implementation closely corresponds to the control loop design. There are five modules in the QoS cache: *timer, output sensor, cache space controller,*

classifier and *actuator*. The timer sends signals to output sensor and cache space controller to let them update their output periodically. The classifier is responsible for request classification and the actuator is in charge of cache space release and allocation. The cache controller simply implements Equation (8). The output sensor measures hit ratio of each class and smoothes it using Equation (4). Hence, below we focus on the other three modules.

Timer: In order to make the control loops work at fixed time interval, we added a module in Squid that regulates the control loop execution frequency. Using the module, we could configure a parameter to let the loops execute periodically, for example, once every 30 seconds.

Classifier: This module is used to identity the requests for various classes. On getting a request this module is invoked and obtains the class of this request. The classification policy is application specific and should be easily configurable. For the sake of developing a proof of concept we are doing classification based on the IP address of the clients sending the request. In general, arbitrary classification policies based on different criteria, such as the requested site, or content type (e.g. HTML vs. GIF) are possible.

Actuator: As described in Section 3, at each sampling time the cache space controller performs the computation $s_i[k] = s_i[k-1] + \delta s_i[k]$ and outputs the new value of total space $s_i[k]$ for each class. In Squid, the cache space deallocation and allocation are two separate processes. The actuator use the output of the controller to guide the cache space release and allocation. Let $realSpace_i$ be a running counter of the actual amount of cache space used by class i. The cache scans the entries from the bottom of each class's LRU list. If the cache space assigned is less than the desired cache space for the class ($realSpace_i < s_i[k]$), entries will not be removed from that class. Whenever a page is fetched from some server, the cache will choose to save it in the disk or not based on which class requests the page and the current cache space of the class. Ideally, as a result of the above enforcement of the desired cache allocation, the cache space $realSpace_i$ occupied by each class i by the end of the kth sampling time should be exactly the same as the desired value $s_i[k]$ set at the beginning of the interval. In reality, a discrepancy may arise. For exmaple, it is possible that we want to give one class more cache space while there aren't enough requests to fill in that much space with requested pages. In order to remedy this problem, we include the difference $s_i[k] - realSpace_i$ at the end of the kth sampling interval in our computation of desired cache space for the $k+1th$ sampling period. That is, $s_i[k+1] = s_i[k] + \delta s_i[k+1] = realSpace_i + (s_i[k] - realSpace_i) + \delta s_i[k+1]$.

5 Evaluation

We test the performance of the feedback control architecture using synthetic traces. We use synthetic workload to show that our design makes the cache converge most efficiently to the specified performance differentiation under representative cache load conditions. The experiments are conducted on a testbed of nine AMD-based Linux PCs interconnected by a 100-MHz Ethernet switch. Collectively, these machines were used to run a set of web servers, web clients, and a single cache. Up to 4 of the machines ran web servers. Up to 4 generated requests, emulating the community of clients. The last machine was used to run the proxy cache. Because Apache [16] is very widely deployed in practice, we used it for our web servers. To emulate a large number of real clients accessing servers, we use Surge 2.2 (Scalable URL Reference Generator) [8], a tool that generates web references matching empirical measurements of six matrices, such as server file size distribution and request size distribution. An instance of Surge runs on each of the client workstations. By sending requests to the different apache severs, these instances collectively emulate the community of clients. There are three client classes. Our differentiation policy specifies that the hit ratio of different classes be related by the expression $H_1 : H_2 : H_3 = 1 : 2 : 3$. In order to test the performance of the cache under saturation, we configure the ratio of cache size to files population to be roughly 1 to 30 in all the experiments.

To develop a reference point against which our control-theoretical heuristic could be compared, we first use a simple linear function $f(e_i) = Ke_i$ in the control loop and determine the best cache performance over all values of K. In this case, the system reacts to performance errors simply by adjusting space allocation by an amount proportional to the error, where K is the proportionality constant. Secondly, we implement the function designed using the theoretic analysis in Section 3. By comparison, we will see that the theoretically designed function produces better performance than that of the best empirically found K, thus guaranteeing the best convergence of the cache. In this context, by performance we mean the efficiency of convergence of the hit ratio to the desired differentiation. This convergence is expressed as the normalized aggregate of the squared errors between the desired and actual relative hit ratio

achieved for each class over the duration of the experiment. The smaller the aggregate error, the better the convergence.

Figure 3 compares the aggregate error for the cache for different controller settings when the specified performance differentiation is $H_1 : H_2 : H_3 = 1 : 2 : 3$. The horizontal axis indicates the base 10 logs of the K value. The vertical axis is the sum of the square of errors ($R_{i_{desired}} - R_i$, where R_i is the *relative* hit ratio) over all classes collected in 20 sampling periods (each sampling period is 30 seconds long). The smaller the sum, the better is the convergence of the cache. We can see from the aggregate error plot in the figure, that using different values of K for the linear function $f(e_i) = Ke_i$, results in different convergence performance. In particular, smaller values of K are too sluggish in adjusting space allocation resulting in slower convergence and larger aggregate error. Similarly, large values of K tend to overcompensate the space adjustment causing space allocation (and the resulting relative hit ratio) to oscillate in a permanent fashion also increasing the aggregate error. In between the two extremes there is a value of K that results in a global minimum of aggregate error. This K corresponds to the best convergence we can achieve. We compare the performance of the simple heuristic $f(e_i) = Ke_i$ for the best K with that of the function described by Equation 8 designed using digital feedback control theory. The aggregate error computed for the latter heuristic is depicted by the straight line at the bottom of Figure 3. It can be seen that the aggregate error using the designed function is even smaller than the smallest error achieved using the simple linear heuristic above, which means that the designed function produces very good performance and successfully converges the cache.

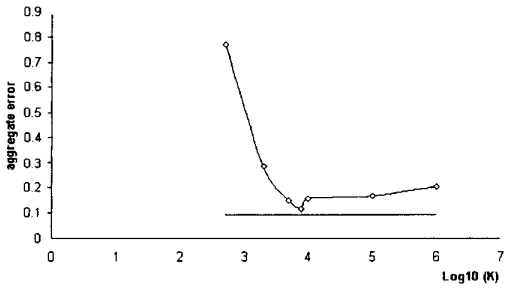


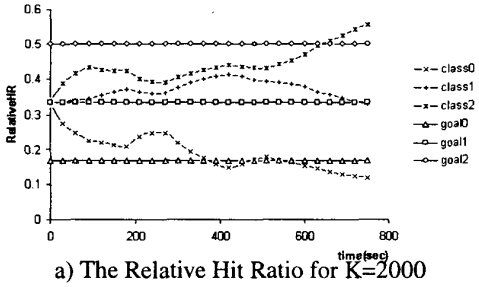
Figure 3. The Aggregate Error versus Controller Gain K

To appreciate the quality of convergence for different controller settings, Figure 4 shows plots of the relative hit ratio of different classes versus time in representative experiments. The controller is given by the simple function $f(e_i) = Ke_i$. Every point in those plots shows the data collected in one sampling period. In the figure, curve $goal_i$ is the desired performance of class i ($R_{i_{desired}} = C_i/(C_1 + C_2 + C_3)$) and curve $class_i$ is the corresponding *relative* hit ratio R_i ($R_i = H_i/(H_1 + H_2 + ... + H_n)$). Since the difference $R_{i_{desired}} - R_i$ reflects the performance error e_i of class i, we will know how well the control loop performs by comparing the two curves $class_i$ and $goal_i$. The closer the two curves, the better the control loop performs and the better is the convergence of the cache.

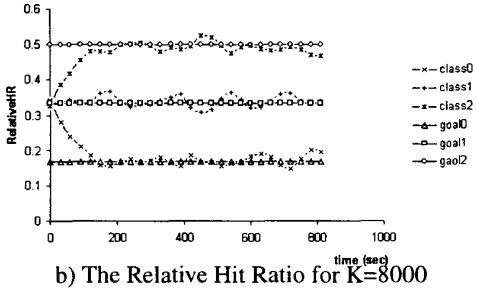
Figure 4-a depicts the *relative* hit ratio using a small value of K for the controller. From the figure, we can see that curve $class_i$ approaches the curve $goal_i$. However, the convergence is too slow. The controller is too conservative in reacting to the performance error. Figure 4-b depicts the relative hit ratio for the best possible K. The figure shows that the cache is converging quickly to the specified performance differentiation. Figure 4-c depicts the relative hit ratio for a big value K. We can see if we use the big K, the cache space adaptation is so large that the relative hit ratio overshoots the desired value. This overcompensation causes the relative hit ratio to continue changing in an oscillatory fashion, making the system unstable.

Figure 5 plots the allocated space for each class versus time in each of the preceding experiments. It can be seen that when K is small (Figure 5-a) space allocation converges very slowly. Similarly, when K is large, space allocation oscillates permanently due to overcompensation. Space oscillation is not desired since it means that documents are repeatedly evicted then re-fetched into the cache. Such cyclic eviction and re-fetching will increase the backbone traffic generated by the cache which is an undesirable effect. The optimal value of K results in a more stable space allocation that is successful in maintaining the specified relative performance differentiation.

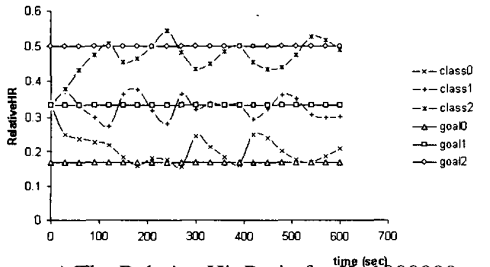
The above experiments show that controller tuning has a dramatic effect on the convergence rate and subsequently on the success of performance differentiation. In Section 3 we presented a design technique for controller tuning that computed the structure and parameters of the best heuristic function. The converge of the cache when this function is used with the analytically computed parameters is depicted in Figure 6. It can be seen that the performance is favorably comparable to the best performance we achieved by experimental tuning. Hence, our



a) The Relative Hit Ratio for K=2000

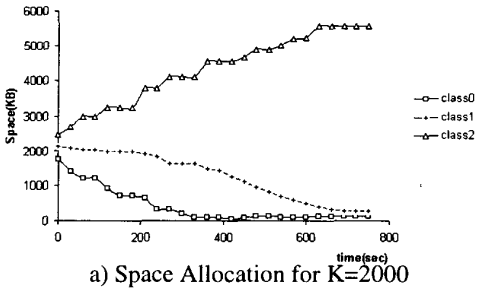


b) The Relative Hit Ratio for K=8000

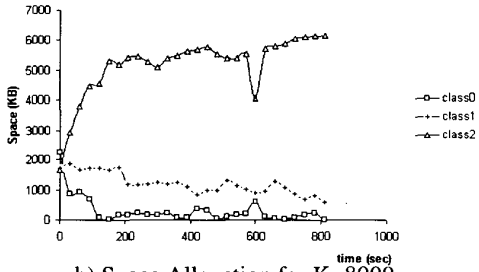


c) The Relative Hit Ratio for K=1000000

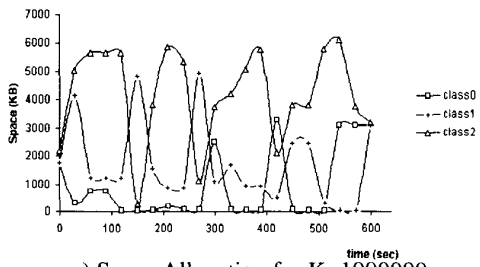
Figure 4. Relative Hit Ratio with Different Control Gains



a) Space Allocation for K=2000



b) Space Allocation for K=8000



c) Space Allocation for K=1000000

Figure 5. Space Allocation with Different Control Gains

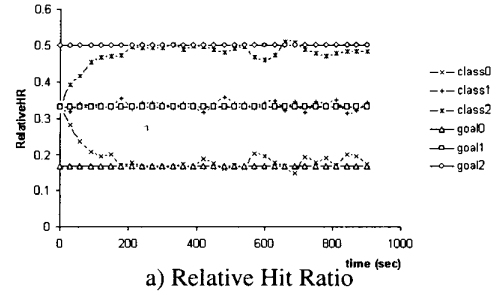
approach is successful in finding a good solution to the performance differentiation problem in a real-life web cache subjected to a realistic web load.

6 Conclusions and Future Work

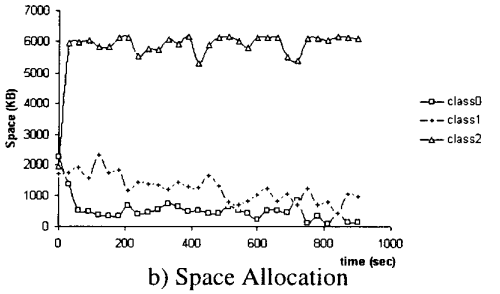
In this paper, we argued for the need for differentiated caching services in future caches in order to cope with the increasing heterogeneity in Internet clients and content classes. We proposed a relative differentiated caching services model that achieves differentiation of cache hit ratio between different classes. The specified differentiation is carried out via a feedback-based cache resource allocation heuristic that adjusts the amount of cache space allocated to each class based on the difference between its specified performance and actual performance. We described a control theoretical approach for designing the resource allocation heuristic. It addresses the problem as one of controller design and leverages principles of digital control theory to achieve an efficient solution. We implemented our results in a real-life cache and performed some preliminary performance tests. Initial evaluation suggests that the control theoretical approach results in a very good controller design compared to manual parameter tuning approaches. The resulting space controller has superior convergence properties and is successful in maintaining the desired performance differentiation for a realistic cache load.

References

[1] T. Abdelzaher and K. G. Shin. Qos provisioning with *q*contracts in web and multimedia servers. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.



a) Relative Hit Ratio



b) Space Allocation

Figure 6. Analytically Designed Controller

[2] T. F. Abdelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. In *International World Wide Web Conference*, Toronto, Canada, May 1999.

[3] T. F. Abdelzaher and N. Bhatti. Web server QoS management by adaptive content delivery. In *International Workshop on Quality of Service*, London, UK, June 1999.

[4] J. Almedia, M. Dabu, A. Manikntty, and P. C. ao. Providing differentiated levels of service in web content host ing. In *First Workshop on Internet Server Performance*, Madison, Wisconsin, June 1998.

[5] D. Andersen and T. McCune. Towards a hierarchical scheduling system for distributed www server clusters. In *Proceedings The Seventh International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.

[6] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In *Second Workshop on Internet Server Performance*, Atlanta, GA, May 1999.

[7] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz. Tcp behavior of a busy internet server: Analysis and improvements. In *IEEE Infocom*, San Francisco, CA, March 1998.

[8] P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS '98*, pages 151–160, Madison, WI, 1998.

[9] P. Barford and M. E. Crovella. A performance evaluation of hyper text transfer protocols. In *Proceedings of ACM SIGMETRICS '99*, pages 188–197, Atlanta, GA, May 1999.

[10] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, September 1999.

[11] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997.

[12] M. Colajanni, P. S. Yu, V. Cardellini, M. P. Papazoglou, M. Takizawa, B. Kramer, and S. Chanson. Dynamic load balancing in geographically distributed heterogeneous web servers. In *Proceedings of 18th International Conference on Distributed Computing Systems*, pages 295–302, Amsterdam, Netherlands, May 1998.

[13] J. Dilley, M. Arlitt, and S. Perret. Enhancement and validation of the squid cache replacement policy. In *4th International Web Caching Workshop*, San Diego, CA, March 1999.

[14] L. Eggert and J. Heidemann. Application-level differentiated services for web servers. *World Wide Web Journal*, 3(2):133–142, March 1999.

[15] T. Faber, J. Touch, and W. Yue. 'the time-wait state in tcp and its effect on busy servers. In *Infocomm*, pages 1573–1583, 1999.

[16] R. T. Fielding and G. Kaiser. The apache http server project. *IEEE-Internet-Computing*, 1(4):88–90, July 1997.

[17] J. Heidemann. Performance interactions between p-http and tcp implementations. *ACM Computer Communication Review*, 27(2), April 1997.

[18] J. Heidemann, K. Obraczka, and J. Touch. Modeling the performance of http over several transport protocols. *ACM/IEEE Transactions on Networking*, 5(5), October 1997.

[19] A. Iyengar, E. MacNair, and T. Nguyen. An analysis of web server performance. In *GLOBECOM*, volume 3, pages 1943–1947, Phoenix, AZ, Nov 1997.

[20] T. P. Kelly, Y. M. Chan, S. Jamin, and J. K. MacKie-Mason. Biased replacement policies for web caches: Differential quality-of-service and aggregate user value. In *4th International Web Caching Workshop*, San Diego, CA, March 1999.

[21] Y. Lu, A. Saxena, and T. Abdelzaher. Differentiated caching services; a control-theoretical approach. Technical Report CS-2001-03, University of Virginia, Department of Computer Science, January 2001.

[22] S. G. Tzafestas. *Applied Digital Control*. North-Holland Systems and Control Series, 1986.

[23] A. Vahadat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. Webos: operating system services for wide area applications. In *Proceedings The Seventh International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.