Service Differentiation in Web Caching and Content Distribution

Michal Feldman and John Chuang*
University of California at Berkeley
{mfeldman,chuang}@sims.berkeley.edu

Abstract—**Service differentiation in web caching and content distribution will result in significant technical and economic efficiency gains, to the benefit of both content publishers and service providers. Through preferential storage allocation and coordinated transitioning of objects across priority queues, we demonstrate a QoS caching scheme that achieves quantifiable service differentiation with little efficiency overhead. We develop and empirically validate a model to quantify, predict, and compare the performance of traditional and service-differentiated caching schemes.**

I. INTRODUCTION

Web caches and content distribution edge servers occupy strategic beachhead positions in the delivery chain of web content to the end users, and have therefore become indispensable to the day-to-day functioning and the long-term scalability of the WWW. With the deployment of caches by ISPs and edge servers by content distribution networks (CDNs), these services are increasingly attached with economic value. Service differentiation can result in significant gains in technical and economic efficiencies. The objective of this work is to study the resource management problem to support differentiated caching and content distribution services.

There are different dimensions of web caching and content distribution service differentiation, including object replacement [1,2], object consistency [3,4], support for dynamic and streaming data types [5], security [6], reporting [7], etc. In this work, we focus on the preferential allocation of storage resources to support priority-based caching services. We describe a differentiated caching mechanism based on priority queues, together with resource allocation and object management policies across the queues. We demonstrate its ability to provide quantifiable service differentiation in terms of performance metrics such as hit rate and object lifetime, with minimal efficiency penalties vis-à-vis traditional caches.

The emergence of CDNs in recent years has created, in essence, a two-tier caching architecture: the proxy caches deployed by ISPs constitute the "for-free" tier with best-effort services, while edge servers deployed by the CDNs constitute the "for-fee" tier with less contention and value-added services. This work is applicable to both tiers. Service differentiation can be beneficially applied to both the CDN servers and the proxy caches. Additionally, this work offers a framework that may help unify the two deployed infrastructures that have largely remained independent.

The rest of the paper is organized as follows. We introduce the object lifetime metric and describe the service differentiation mechanism in Section II. In section III we develop an analytical model for predicting object lifetime and hit ratio for the traditional and service differentiated caching schemes. The model is validated via simulation, and the results are presented in Section IV. Section V discusses related work, and Section VI concludes the paper.

II. SERVICE DIFFERENTIATION: METRICS AND MECHANISM

Existing caching solutions have largely been developed from a network-centric or client-centric perspective, with bandwidth savings and latency reductions as the primary objectives of web caches. Consistent with this approach, the most widely used performance metric is the cache hit rate. Depending on the deployment and user access patterns, caches can realize hit rates in the 30-70% range. From a publisher's perspective, however, cache hit rates do not offer any insight into the type of service received by its objects. Depending on their popularity, individual objects may receive object-specific hit rates that are significantly higher or lower than the overall hit rate.

In a service differentiated caching environment, an object-centric or publisher-centric metric becomes important. We propose *object lifetime*, the amount of time an object resides in the cache before it is purged, as one such metric. Object lifetime compliments the hit rate metric and provides a meaningful measure of service received by individual objects at caching servers. Our simulation study shows that object lifetimes can vary wildly in a traditional cache (Figure 1). Intuitively, an object's lifetime is dependent on the cache size, the traffic volume experienced by the cache, and the object's popularity. One goal of service-differentiated caches is to provide some degree of prediction or guarantee on an object's lifetime. For example, a minimum object lifetime guarantee can be provided independent of the expected or actual popularity of the object. This may be of value to an online broker who has stringent response time requirements for its premium customers, and will therefore demand greater availability of the premium (but less frequently accessed) objects. As another example, content related to rare events (e.g., disaster response information such as "what to do during/after an earthquake?") may have critical availability requirements even though they are normally unpopular [8,9].

Service level agreements based on the object lifetime metric can be provided via cache reservation or priority-based caching mechanisms. In this paper, we describe the multi-level LRU (ML-LRU) algorithm, which is a priority-based caching scheme, and evaluate it using both object lifetime and the traditional hit rate metrics. The ML-LRU algorithm is a straightforward extension of the LRU algorithm, used in most commercially deployed web caches. An ML-LRU cache consists of *M* interconnected LRU-based

* This work is supported by the National Science Foundation under Cooperative Agreement Number ITR-0085879. We thank Sumeet Solanki for his research assistance on this project.

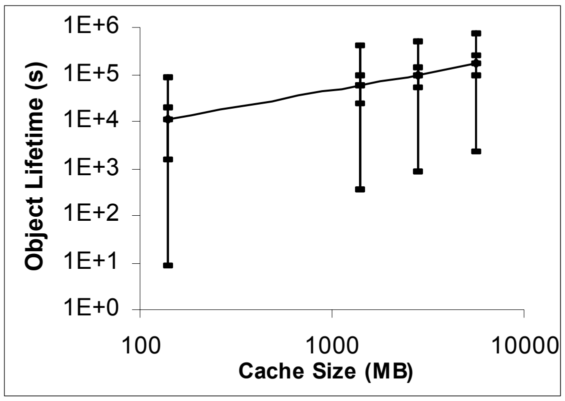


Figure 1. Objects in a traditional LRU cache experience large variations in object lifetime. In this simulation study, an object may spend anywhere between 10s and 100,000s in a 150MB cache.

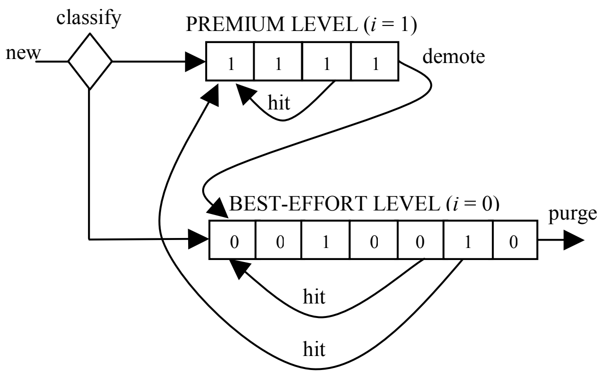


Figure 2. ML-LRU Implementation (M=2)

based queues, denoted levels i = 0, …, M-1. Each queue employs simple LRU within its own level. In addition, the queues are interconnected such that an object evicted from level i is inserted into the head of the level i-1queue. Only objects evicted from the level 0 queue are purged from the cache outright.

When a new object arrives at the cache, it is classified into one of the M levels according to some object classification policy (discussed below) and placed at the head of the corresponding queue. In the absence of cache hits, the object traverses the length of the queue until it reaches the tail and is evicted from the queue. The eviction corresponds to a demotion to the next lower level for $i > 0$, and to a cache purge for i = 0. Upon a hit, an object is transferred back to the head of its original level, independent of its current location. An object can never reach a level that is higher than its original classification.

Figure 2 illustrates the ML-LRU algorithm for M=2, which is the subject of our investigation for the rest of this paper. In this two-level LRU cache, we refer to level 1 as the "premium" level and level 0 as the "best-effort" level. The storage allocation to each level is adjustable. It is interesting to note that the best effort queue can contain both best effort and premium objects, while the premium queue contains premium objects only.

In order to support differentiated service, the cache has the ability to classify incoming objects to different queues, i.e., assigning objects to different initial levels. This is consistent with the classification mechanisms employed by web server QoS schemes [10]. Different classification policies may be used to accomplish different service differentiation and resource allocation objectives. The following list provides some examples of classification policies that can be implemented:

1. Source-based: source hostname/IP address;
2. Content-based: file type;
3. Economics-based: payment by object owner;
4. Popularity-based: object popularity;
5. Size-based: object size.

In our simulations, we classify objects in random order, but control the fraction of objects assigned to each level.

III. PERFORMANCE MODEL

A. Single-Level LRU

Consider a simple LRU cache of size S (in bytes), and objects arrive at the cache at a rate of $\lambda(t)$ bytes/second. The cache realizes a byte hit rate $H_B(S)$ and a object hit rate $H_O(S)$ which are functions of the cache size. The LRU cache is implemented using a linked list, and objects traverse this linked list, from head to tail, as new objects enter into and old objects are purged from the cache.

To begin, we derive an expression for the expected duration of an object's cache residency, or the expected object lifetime $L(t)$. First, we expect the object lifetime to be a function of the object's popularity, as captured in the object hit count f. Second, it should be inversely proportional to how quickly the object traverses the length of the linked list. This is captured in the object arrival rate $\lambda(t)$ and the object aging factor k. The expected lifetime of an object in a LRU cache can be expressed as:

$$L(t) = \frac{S}{k\lambda(t)}(1+\gamma f). \qquad (1)$$

The object hit count f is the number of times an object is requested while it is in the cache. It is object-specific and a function of the object's popularity and cache size. The object hit rate of the cache can be used as the expected value of f, i.e., $f_{\text{avg}} = H_P(S)$. Many objects may receive zero hits while in the cache (the so-called "one-timers") and so $f_{\min} = 0$. Every time an object gets a cache hit, it is promoted back to the head of the list, thereby extending its lifetime. The extension is weighted by a factor $\gamma(0 \le \gamma \le 1)$, the object's relative position in the cache when it received a cache hit. If cache hits occur to objects independent of their positions in the list, then γ= 0.5.

Next, let us consider the object aging factor k. A cached object ages or moves forward in the linked list when any one of the following events occur: (i) a compulsory cache miss, (ii) a capacity cache miss, (iii) a consistency cache miss on a less recently used object (i.e., object closer to the tail of the list), (iv) a cache hit on a less recently used object. Conversely, a cached object does not move forward in the list only when a consistency cache miss or a cache hit occurs to a more recently used object, i.e., an object closer to the head of the list than the current one. Therefore, we can express the aging factor as:

$$k = 1-\gamma[H_B(S)+M_B(S)] \quad (2)$$

where $M_B(S)$ is the consistency byte miss rate, and $\gamma(0 \le \gamma \le 1)$ is the relative position of the requested object in the linked list. Once again, if cache hits and consistency misses occur to objects independent of their positions in the list, then $\gamma = 0.5$.

By using the appropriate values for λ and f, equation (1) can be used to compute the mean and minimum lifetimes. For example, we can compute the mean lifetime L_{avg} by using the mean arrival rate λ_{avg} and mean hit count f_{avg}. Similarly, we can compute the minimum lifetime $L_{\min}$ by using the maximum arrival rate $\lambda_{\max}$ and minimum hit count $f_{\min}$. It is interesting to note that, while arrival rates are trace dependent, $\lambda_{\max}$ is also a function of the cache size. The smaller the cache, the more susceptible it is to burstiness in arrival traffic, and the larger the maximum arrival rate.

B. Multi-level LRU

Now let us consider a two-level LRU cache, where αS is allocated to level 1, and $(1-\alpha)S$ to level 0. The level 1 queue is occupied by premium objects only, while the level 0 queue is shared by both premium and best effort objects. Let β be the fraction of objects that belong to the premium level. Then the arrival rates for premium and best effort objects are $\beta\lambda(t)$ and $(1-\beta)\lambda(t)$ respectively.

We can express the *effective cache sizes* of the premium and best effort levels as S_1 and S_0. By *effective cache size* we refer to the cache sizes with relation to the arrival rate of their objects. Note that the premium level's cache size is effectively larger than the total cache size since premium objects can traverse the premium cache without the interference of best-effort objects.

$$S_1 = \left(\frac{\alpha}{\beta}\right)S + (1-\alpha)S\,; \quad (3)$$

$$S_0 = (1-\alpha)S\,. \quad (4)$$

Given the effective cache sizes S_1 and S_0, we can compute the cache hit rates for the two levels, namely byte hit rates $H_B(S_1)$ and $H_B(S_0)$, and object hit rates $H_O(S_1)$ and $H_O(S_0)$. We assume no correlation between object level and its size or popularity.

Next we compute the expected lifetimes for the premium and best effort objects. For best effort objects, the average and minimum hit counts are $f_{0\text{avg}} = H_O(S_0)$ and $f_{0\min} = 0$ respectively. The aging rate is determined by four factors: (i) level 0 consistency miss rate $M_B(S_0)$, (ii) level 0 hit rate $H_B(S_0)$, (iii) level 1 consistency miss rate $M_B(S_1)$, and (iv) level 1 hit rate $H_B(S_1)$. A cached best-effort object will not move forward in the queue for a fraction γ of level 0 hits and consistency misses, and a fraction of $(\alpha+\gamma(1-\alpha))$ of level 1 hits and consistency misses. Therefore, the expected lifetime for best effort objects is:

$$L_0(t) = \frac{(1-\alpha)S}{k_{00}(1-\beta)\lambda(t) + k_{01}\beta\lambda(t)}(1+\gamma f_0) \quad (5)$$

where

$$k_{00} = 1-\gamma[H_B(S_0)+M_B(S_0)]; \quad (6)$$

$$k_{01} = 1-(\alpha+\gamma(1-\alpha))[H_B(S_1)+ M_B(S_1)]. \quad (7)$$

Finally, the expected lifetime of premium objects is the summation of lifetimes spent in each of the two queues. In the premium queue, the cache traversal rate is influenced by the level 1 consistency miss rate and the level 1 hit rate. In the best effort queue, they experience the same expected lifetime as the best effort objects. In terms of hit count, we have $f_{1\text{avg}} = H_O(S_1)$, and $f_{1\min} = 0$. Therefore, the expected lifetime for premium objects is:

$$L_1(t) = \left[\frac{\alpha S}{k_1\beta\lambda(t)} + \frac{(1-\alpha)S}{k_{00}(1-\beta)\lambda(t) + k_{01}\beta\lambda(t)}\right](1+\gamma f_1) \quad (8)$$

where

$$k_1 = 1-\gamma\alpha\,[H_B(S_1)+M_B(S_1)]. \quad (9)$$

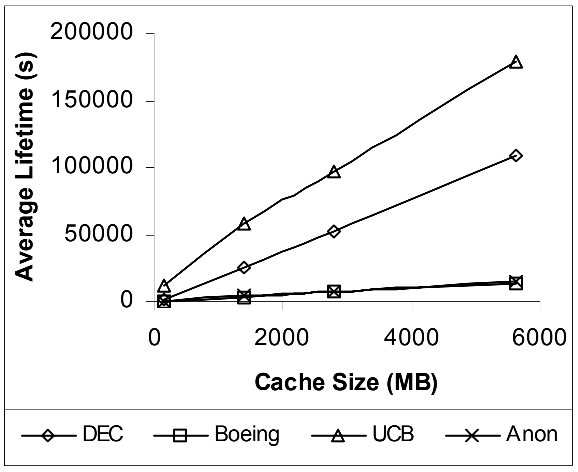
Once again, the mean and minimum lifetimes for the premium and best effort objects can be computed by using the appropriate mean and worst-case values for f_1, f_0, and λ.

IV. EVALUATION

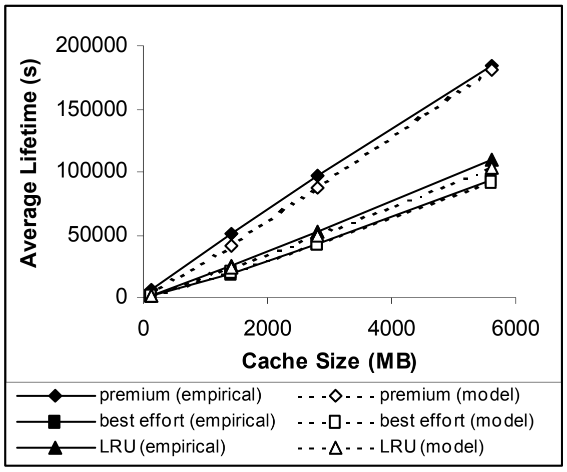
The ML-LRU algorithm is evaluated using trace-driven cache simulation. We implemented the algorithm in the WisWeb cache simulator [11] and instrumented the simulator to measure object lifetimes and hit rates for each level as well as for the entire ML-LRU cache. The simulator is exercised using three publicly available web proxy traces, namely UCB [12], DEC [13] and Boeing [14], and an additional anonymous large corporate proxy ("Anon"). Table 1 shows the summary statistics of these traces. Note that we used traces with significantly different arrival rates. All four traces contain 4-5 million object requests, but the durations range from one day (Boeing) to nine days (UCB).

Table 1. Statistics of traces used in this study.

Trace	Date	Number of requests	Duration (seconds)	Requests per second
DEC	Sep-96	5,000,000	440,916	11.3
UCB	Nov-96	4,868,539	810,300	6
Boeing	Mar-99	4,292,154	86,354	49.7
Anon	Jul-01	4,664,993	86,400	54

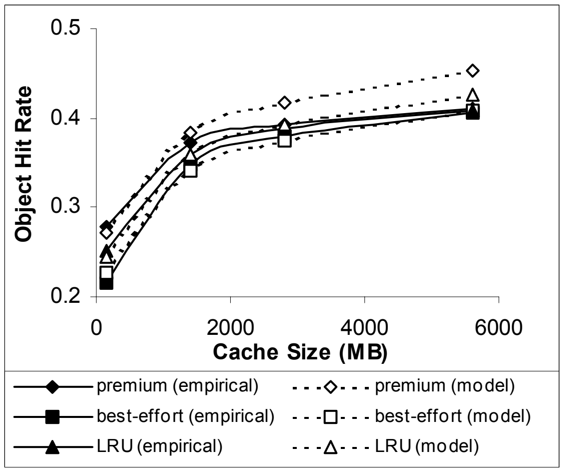


(a) traditional LRU

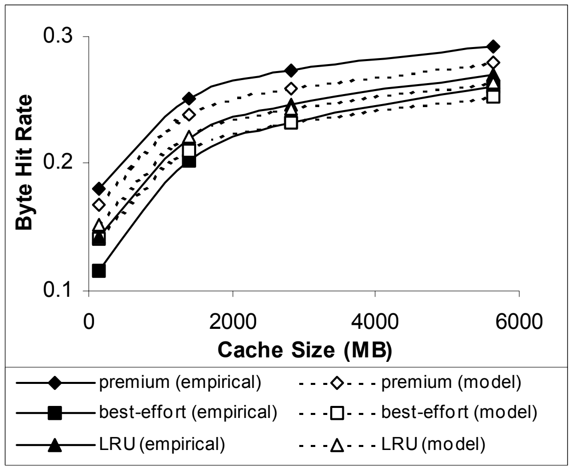


(b) LRU and ML-LRU

Figure 3. Average object lifetime vs. cache size for ML-LRU and LRU caches.



(a) Object Hit Rate



(b) Byte Hit Rate

Figure 4. Empirical measurements and model predictions for ML-LRU hit rates.

A. Traditional LRU and ML-LRU

Under the traditional LRU policy, we observe significant variance in object lifetime (Figure 1). However, we also find that, consistent with equation 1, average object lifetime L_{avg} increases linearly with cache size S (Figure 3(a)). At the same time, average object lifetime decreases with increasing average object arrival rate λ_{avg}, also in accordance with equation 1. For example, the UCB trace has lowest average object arrival rate, and the highest average object lifetime.

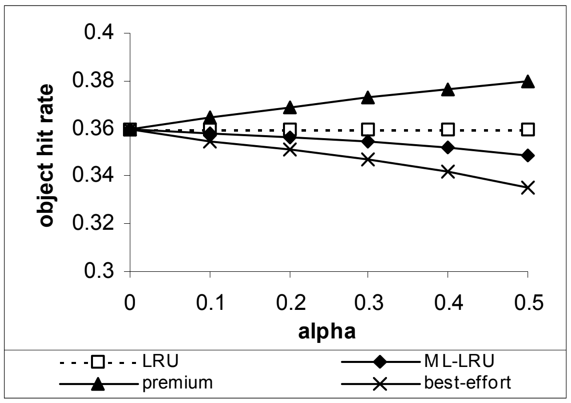
Under the ML-LRU mode, the linear relationship between average object lifetime and cache size remains valid across all object classes. Furthermore, the premium objects now enjoy a significantly higher average object lifetime than the best-effort objects and the average object in a traditional LRU cache. Figure 3(b) shows the average lifetimes for the various object classes for the DEC trace with $\alpha = 0.3$ and $\beta = 0.3$. Similar results are obtained for the other traces, and so we only present the DEC results from this point. We also see a very good fit between the predicted lifetimes from the analytic model and the actual empirical results. This suggests that the model (as embodied in equations 1, 5 and 8) is successful at predicting the object lifetimes in both the LRU and ML-LRU caches.

Figure 4 shows the predicted and measured hit rates for the ML-LRU algorithm. The model (based on equations 3 and 4) appears to provide a reasonable prediction of hit rates. In Figure 4(a), we notice that the object hit rates for all object types level off at large cache sizes, and approach the asymptotic hit rate for an infinite-sized cache. Since this asymptotic behavior is trace-specific, we do not expect the model to predict it, and it does not.

Overall, there is little efficiency loss due to cache partitioning, as the hit rates and the average lifetimes of the ML-LRU cache (averaged across all classes) are only slightly lower than those of the traditional LRU cache.

B. Multi-level resource allocation

The relative performances of different priority levels in an ML-LRU cache are strongly dependent on the allocation of storage resources among the various levels, and the classification of objects into these levels. We explore how hit rates and object lifetimes are influenced by changes in α and β.



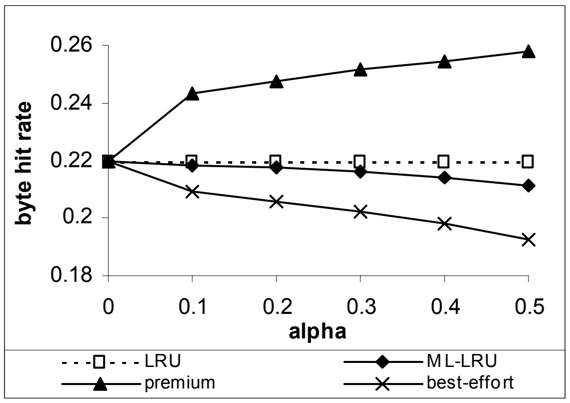


Figure 5. Object and byte hit rates vs. α for ML-LRU cache (S=1.5GB, β = 0.3)

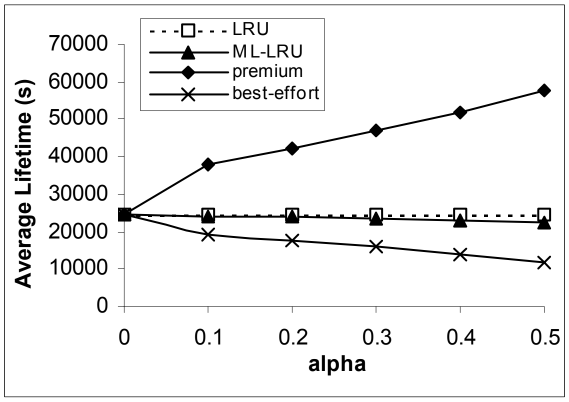


Figure 6. Average lifetime vs. α. (S=1.5GB, β = 0.1)

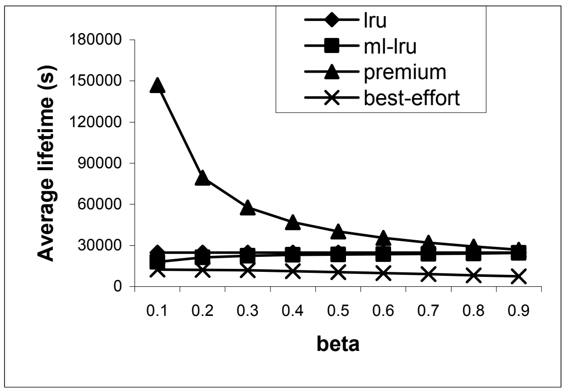


Figure 7. Average lifetime vs. β. (S=1.5GB, α = 0.5)

Figures 5 and 6 shows the effect of preferential resource allocation among service classes (via adjusting α) on the relative performance of the classes. The case of α = 0 corresponds to single level LRU, and there is no performance differentiation between the classes. As we begin to allocate more cache space to the premium class by increasing α, the performance of premium objects (hit rates and object lifetimes) increases at the expense of best-effort objects. At α = 0.5, we see a 13% and 34% difference in object and byte hit rates between the premium and best effort objects. The corresponding efficiency loss (by switching from LRU to ML-LRU) is 3% and 4% respectively. At the same time, the average premium object will experience a 380% longer lifetime than the best effort object, and a 130% longer lifetime than the typical object in an LRU cache of the same size.

The ML-LRU algorithm can be extended in future work to include a dynamic resource allocation mechanism, such that the various queue sizes can be dynamically adjusted, by tuning α, to meet high-level SLA objectives. This can be especially important if the arrival of premium and best-effort objects is bursty, such that β fluctuates significantly from its expected value. Consider the scenario depicted in Figure 7, where 50% of a 1.5GB cache is statically assigned to the premium level. If the expected fraction of premium object arrivals is 0.2, the expected average premium lifetime would be approximately 80,000s. If it turns out that only 10% of arrivals are premium objects, then the average premium lifetime would almost double to 150,000s. Conversely, if 40% of object arrivals are for the premium level, i.e., β = 0.4, then the average premium lifetime would drop to 45,000s. If the cache is committed to maintaining lifetime targets for its premium objects, it would need to dynamically increase α to maintain S_1 at its target level.

V. RELATED WORK

Service differentiation has been applied to the networking domain [15-16] and more recently, in a flurry of activity, to the web server domain [10,17-25]. In the latter case, various mechanisms and policies are proposed for delivering end-to-end QoS to the origin servers to provide preferential service to preferred clients or services.

Cache partitioning has been studied in a variety of domains. The 2Q algorithm [26], which is built upon the LRU-K algorithm [27], is presented in the database disk buffer management domain. It maintains two queues and allows transitions between them, for the specific objective of promoting the "non-one-timers." The virtual cache [28] seeks to combine the best of all replacement policies by creating multiple cache partitions and running each partition with a different replacement policy.

Kelly et al. is the first to apply QoS to the caching domain, employing a biased replacement policy that is sensitive to varying levels of server valuations for cache hits [1]. It prioritizes cache space across servers using a server-weighted LFU implementation to maximize a composite "user value" metric. [29] considers caching and replication as multiple service classes offering different degrees of QoS guarantees, but does not address the details of object replacement. Lu et al. proposes the provisioning of differentiated caching services based on control-theoretical resource allocation and is most closely related to our work [2,30]. Specifically, it enforces the desired ratios between the hit rates (but not absolute ones) of the various classes by means of per class feedback control loops. Adaptive control theory is used for automating controller tuning, eliminating the need for human interaction. The lack of interconnection between the queues prevents objects from one service class from using the resources of another under-utilized service class. This implies that the scheme has to trade off the cost overhead of frequent resource reallocation and its responsiveness to bursty traffic patterns.

VI. CONCLUSION

Service differentiation in web caching and content distribution can be effectively realized by a priority-based object management mechanism, with preferential storage allocation and coordinated transition of objects across multiple priority queues. We develop and empirically validate a model that allows us to quantify, predict, and compare the performance of traditional and service differentiated caching schemes, using both traditional and novel metrics such as hit rate and object lifetime. We find that quantifiable service differentiation can be achieved without incurring significant overhead to the overall performance of the caches.

REFERENCES

[1] T. Kelly, Y. Chan, S. Jamin, and J. MacKie-Mason, Biased replacement policies for Web caches: Differential quality-of-service and aggregate user value, in 4th Web Caching Workshop, Mar. 1999.

[2] Y. Lu, A. Saxena, and T. F. Abdelzaher, Differentiated Caching Services; A Control-Theoretical Approach, in International Conference on Distributed Computing Systems, Phoenix, Arizona, April 2001.

[3] C. Liu and P. Cao, Maintaining Strong Cache Consistency in the World Wide Web. IEEE Transactions on Computers, 47(4):445--457, 1998.

[4] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Using leases to support server-driven consistency in large-scale systems. In Proceedings of the 18th International Conference on Distributed Systems, May 1998.

[5] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents on the web. In Proc. of the IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing, Sep 1998.

[6] A. Myers, J. Chuang, U. Hengartner, Y. Xie, W. Zhuang, and H. Zhang, A Secure, Publisher-Centric Web Caching Infrastructure, in IEEE INFOCOM 2001.

[7] J. Mogul and P. Leach, 1997. Simple Hit-Metering and Usage-Limiting for HTTP. RFC 2227.

[8] M. Welsh. An Architecture for Highly Concurrent, Well-Conditioned Internet Services. Ph.D. Dissertation, UC Berkeley, 2002.

[9] L. A. Wald and S. Schwarz. The 1999 Southern California Seismic Network Bulletin. Seismological Research Letters, 71(4), July/August 2000.

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

[11] P. Cao and S. Irani, Cost-aware WWW proxy caching algorithms, in Proceedings of USENIX Symposium on Internet Technology and Systems, 193-206, Dec. 1997.

[12] UC Berkeley Home IP Web Traces, http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html

[13] Digital Equipment Cooperation Web Proxy Traces, ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html

[14] Boeing Proxy Logs, ftp://researchsmp2.cc.vt.edu/pub/boeing/

[15] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, Jun 1994.

[16] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, Dec. 1998.

[17] J. Almeida, M. Dabu, A. Manikntty and P. Cao. Providing differentiated levels of service in Web content hosting. In Workshop on Internet Server Performance, Madison, WI, June 1998.

[18] 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.

[19] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. *World Wide Web,* 2(1-2), 1999.

[20] X. Chen and P. Mohapatra. Providing differentiated service from an Internet server. In IEEE Int'l Conf. on Computer Communications and Networks, Boston, MA, Oct. 1999.

[21] V. Kanodia and E. Knightly. Multi-class latency-bounded Web services. IWQoS, Pittsburgh, PA, June 2000.

[22] D. Menasce, J. Almeida, R. Fonseca, and M. Mendes. Business-oriented resource management policies for e-commerce servers. *Performance Evaluation*, 42(2-3):223-239, 2000.

[23] R. Pandey, R. Barnes and J. F. Olsson. Supporting quality of service in HTTP servers. ACM Symposium on Principles of Distributed Computing, Puerto Vallarta, Mexico, June 1998.

[24] N. Vasiliou and H. L. Lutfiyya. Providing a differentiated quality of service in a World Wide Web server. Performance and Architecture of Web Servers Workshop, Santa Clara, CA, June 2000.

[25] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In IEEE Infocom, Anchorage, Alaska, Apr. 2001.

[26] T. Johnson and D. Shahsa. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In 20th International Conference on Very Large Databases, Santiago, Chile, Sep. 1994.

[27] E. J. O'Neil, P. E. O'Neil and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering, in Proc. of SIGMOD, Washington, DC, May 1993.

[28] M. F. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin. Evaluating content management techniques for web proxy caches. *Performance Evaluation Review* 27(4):3-11, 2000.

[29] J. Chuang and M. Sirbu. Distributed Network Storage with Quality-of-Service Guarantees. *Journal of Network and Computer Applications* 23(3): 163-185, July 2000.

[30] Y. Lu, T. Abdelzaher, C. Lu and G. Tao. An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services. International Workshop on Quality of Service, Miami Beach FL, May 2002.