Improving Web-Scale Cache Performance through Adaptive Policies

Nathan Beckmann ○ beckmann@cmu.edu ○ http://nathanbeckmann.com
Assistant Professor ○ Carnegie Mellon University ○ School of Computer Science
5000 Forbes Avenue ○ Pittsburgh PA 15217
Google Sponsor: George Kurian ○ gkurian@google.com

Abstract

Caches are a critical component of Internet applications. Caching the results of user requests (e.g., database queries or dynamic web content) significantly reduces the latency of subsequent requests and lowers the load on backend servers. However, as datasets have grown exponentially in recent years, caches have grown as well, so that they now consume thousands of datacenter servers. Improving their performance can thus significantly reduce the operating costs of Internet services. Yet caches today are managed using simple caching policies, such as least-recently used (LRU) or random replacement, that leave significant performance on the table.

We propose to borrow ideas from processor caches, where achieving high performance with limited resources has received years of attention, to improve cache performance in Internet services. In particular, our recent work has developed an adaptive, statistical replacement policy for processor caches. The key insight is a novel and intuitive metric that, without any tunable parameters, ranks objects for eviction.

Preliminary results are promising: On a real production trace of memcached requests, we more than halve the number of misses, achieving a hit rate of 90\% vs. just 78\% for LRU. An LRU cache would need to be 8× larger to match this performance. Many opportunities remain to support the new features of Internet services (e.g., varying replacement costs) or offer new user features (e.g., differentiated quality-of-service).

Introduction

Caches are a fundamental building block of computer systems. Caches allow frequently used objects to be accessed quickly, but at the cost of only holding a limited number of objects. The critical question is: what objects should be kept in the cache? This is determined by the cache’s replacement policy, which chooses objects to evict when space is needed for new objects.

Internet services rely on caches to store intermediate results, e.g., in the form of distributed key-value stores like memcached [1, 9] or in distributed file systems like BigTable or GFS [4, 6]. Caches satisfy repeated requests to the same item, decreasing the latency seen by the user and lowering the load on backend servers.

Common uses of web caches include storing database queries, dynamic web content, static assets like images, or file system metadata and blocks. Often several types of objects co-exist from different applications, and, to prevent these applications from interfering with one another, system administrators explicitly partition the cache into separate pools running on different machines. Caches consume an enormous amount of resources: e.g., thousands of servers at Facebook [1, 9]. Using them well is thus critical to the performance and scalability of Internet services.

A cache’s performance is generally measured by its hit rate (i.e., the percentage of requests it satisfies), and the main challenge in improving hit rate is uncertainty about when objects will next be requested. Indeed, with perfect information about future requests, cache management becomes a scheduling problem where simple policies are known to be optimal. For instance, Belady’s MIN [3] evicts the candidate that is requested furthest in the future, and it is optimal for caches with uniformly sized objects. However, since perfect information about the future is unavailable in practice, caches instead often use simple heuristics.

Least-recently used (LRU) is most common (e.g., memcached uses LRU), but other standard policies include least-frequently used (LFU) or random replacement. The problem. Unfortunately, these heuristics fail on common request patterns. For example, consider a request stream that scans repeatedly across $N$ objects: i.e., requesting objects $1, 2 \ldots N, 1, 2 \ldots N$, and so on. If the cache can hold $N - 1$ objects, then the cache should get a hit rate of nearly 100\% for large $N$, and indeed MIN achieves this performance. But LRU gets zero hits on this request pattern. Similar pathologies hold for other common heuristics, and although prior work has considered many ways of combining LRU and LFU (e.g., ARC [8], LRU-$k$ [10], LIRS [7], and many others [11]), such policies still rely on simple heuristics with pathological request patterns. The problem is that these policies encode assumptions about the request stream in their heuristics or parameters, and their performance suffers when these assumptions do not hold.
To address this performance gap, we propose that caches should instead use adaptive, statistical caching policies that perform well across many different request patterns. Our solution applies techniques from our prior work, but makes major improvements to address the unique features of caches in Internet services.

**Approach and insights.** We recently developed an adaptive, statistical policy for processor caches called EVA [2]. EVA is a general metric for cache replacement that performs well on arbitrary request patterns. Unlike prior policies, EVA makes minimal assumptions about request behavior. Instead, EVA draws upon Markov decision processes (MDPs) to guide decision-making without any tunable parameters. EVA adapts to applications so that prior heuristics like LRU arise naturally when appropriate, but without directly encoding any of these heuristics. As a result, EVA outperforms state-of-the-art heuristics across diverse workloads.

The key insight is that EVA views each object in the cache as an investment and decides which object to evict through cost-benefit analysis. The cost-benefit analysis includes the opportunity cost of caching each object, which increases in proportion to how much space the object consumes, i.e., how long it spends in the cache. EVA thus ranks objects by estimating the probability they will hit (benefit) and how long they are likely to spend in the cache (cost). Specifically, EVA “charges” objects for the time they spend in the cache at a rate equal to the average hit rate, since this is the long-run opportunity cost of consuming cache space. Hence, an object’s net benefit, or economic value-added (EVA), is

\[ \text{EVA} = \text{Hit probability of object} - \text{Cache's hit rate} \times \text{Expected time object will spend in cache} \]

The object with the lowest EVA is evicted. The challenge is to accurately estimate the probabilities when ranking each object. Rather than making assumptions about request behavior, EVA relies on online monitoring and statistical inference to adapt itself to the request stream. This lets EVA work well on many different request behaviors and avoid the pathologies discussed above.

In our prior work in processor caches, hardware severely constrained implementation complexity and hence the sophistication of this statistical inference. We also focused on processor caches where objects have uniform size and uniform replacement cost, and we did not consider how behavior varied across applications. These are important limitations in the context of Internet services that we address in this proposal.

**Preliminary results.** To explore the potential of this proposal, we have improved EVA to account for each object's size by scaling opportunity costs in proportion to object size. We evaluate policies on a real production trace from Memcachier (http://www.memcachier.com), a provider of multi-tenant, distributed key-value stores. EVA more than halves cache misses over LRU, achieving a hit rate of 90% vs. 78% for LRU. (LRU’s hit rate is similar to that reported for memcached at Facebook [9].) An LRU cache would require 8x as much cache space to match EVA’s performance. To our surprise, EVA even outperforms MIN’s hit rate of 88%—although MIN does not account for object size, it has perfect information about future requests and so often outperforms even carefully tuned heuristics. Finally, by judiciously pre-computing probabilities, EVA adds negligible overhead: just a few table lookups and arithmetic operations per replacement. We believe that these early results show promise of our approach on caches for Internet services.

**Proposal**

We propose to apply EVA to caches throughout Internet services to improve their performance and add new user features. These caches introduce many new challenges: e.g., non-uniform object sizes, non-uniform replacement costs, and differing behavior across different applications sharing the cache. Fortunately, with more resources available than in hardware caches, Internet services also introduce new opportunities to address these challenges. Finally, we propose to integrate this new caching policy in a high-performance implementation, e.g., a key-value store like memcached or MemC3 [5].

**New caching considerations introduced by Internet services.** Caches in Internet services differ from processor caches in several important respects. Their objects vary widely in size, from a few bytes to hundreds of MBs. Similarly, some objects are relatively inexpensive to replace, whereas others (e.g., complex database queries) are very expensive. The replacement policy should account for these differences when choosing which objects to evict. EVA gives a natural framework for this, since the object size and replacement cost simply change the costs and benefits of retaining the object.

**Sophisticated statistical inference.** In Internet services, EVA can afford to use more sophisticated statistical inference than is possible in processor caches. For example, these caches are often shared by distinct types of objects, e.g., objects from different applications. Since different object types behave differently, this
information should be used to improve predictions about the costs and benefits of retaining each object. We propose to exploit these correlations to further improve replacement decisions.

In fact, there are many dimensions on which to categorize objects: size, application, time since reference, frequency of reference, etc. Moreover, Internet services exhibit regular diurnal request patterns. Ideally, all of this information would be brought to bear when predicting an object’s behavior. However, this is a complex inference problem, since data are sparse and which dimension is the best predictor may change over time or in different contexts.

At a minimum, we will characterize applications and identify the dimensions (e.g., object size, application id) that are reliably good predictors of object behavior. We will also explore sophisticated statistical or machine learning techniques that leverage all dimensions to better predict how objects behave.

**Differentiated quality-of-service.** With modest changes, EVA can give system administrators additional control over how caches are used. For example, application priorities can be implemented by giving different benefits for hits from different applications. With this simple change, EVA then implicitly manages the cache to favor particular applications. EVA is also naturally “scan resistant,” so that high-churn applications that access the cache frequently with few hits do not interfere with other applications. As a result, system administrators may not need to explicitly split the cache into separate pools and tune each pool for each high-value application, e.g., as is done at Facebook [9]. We propose to explore differentiated quality-of-service in EVA, including its impact on throughput and fairness.

**Real-world implementation.** We propose to integrate EVA into a real, high-performance caching system, letting EVA be easily deployed in production systems. We propose to develop a high-performance implementation, borrowing techniques from EVA’s hardware implementation for efficiency, and study the tradeoff between replacement overheads and hit rate. We plan to first implement EVA in a key-value store, e.g., memcached or MemC3 [5]. We are also interested in exploring other contexts, e.g., distributed file systems [6].

**Expected outcomes.** We expect that this proposal will produce a caching system with hit rates significantly higher than the state-of-the-art and comparable request throughput, while providing new features to system administrators. Altogether, this proposal should substantially reduce the resources needed to achieve a given level of cache performance. We also hope to contribute new scientific insights by accounting for the unique features of Internet services and by employing sophisticated statistical inference.

**Opportunities for collaboration with Google.** There are numerous opportunities for collaboration with Google. Caches are pervasive throughout Internet services, and their characteristics may vary depending on context. Our current research infrastructure uses production traces from Memcachier on distributed key-value stores. We would be very interested to apply these ideas to caches at Google [4, 6] through a joint research collaboration. One promising avenue of collaboration could be, either during or upon completion of this work, having a student intern at Google to explore these ideas.

**Budget**

The PI requests $81,475 for one student-year of support. The detailed breakdown is: $32,724 for student stipend; $44,374 for PhD tuition; $2,877 for computing costs; and $1,500 for student travel.

**References**

Nathan Beckmann

School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213
(818) 388-7289
beckmann@cmu.edu
nathanbeckmann.com

RESEARCH INTERESTS
Computer systems, computer architecture, operating systems, parallelism, and performance modeling & analysis.

EDUCATION
Massachusetts Institute of Technology .................................................. Sep 2015
Ph.D., Electrical Engineering and Computer Science.
Thesis: Design and Analysis of Spatially-Partitioned Shared Caches.
Supervisor: Daniel Sanchez.
Massachusetts Institute of Technology .................................................. Sep 2010
S.M., Electrical Engineering and Computer Science.
Thesis: Distributed Naming in a Factored Operating System.
Supervisor: Anant Agarwal.
University of California, Los Angeles .................................................. Mar 2008
B.S. Computer Science. Summa cum Laude.

AWARDS
George M. Sprowls Doctoral Thesis Prize .............................................. 2015
Best doctoral thesis in computer science at MIT.
William A. Martin Memorial Thesis Award ........................................... 2010
Best master's thesis in computer science at MIT.
UCLA Bachelor of the Year in Computer Science .................................. 2008
UCLA Rose Hills Foundation Science and Engineering Scholarship (2×) .......... 2007 & 2008

PROFESSIONAL EXPERIENCE
Carnegie Mellon University ................................................................. Jan 2017 - Present
Assistant Professor in the Computer Science Department of the School of Computer Science. My research focuses on hardware-software co-design to improve the performance and energy-efficiency of future systems. By working across many layers of the system stack, from theory to hardware, I design systems that dynamically and transparently optimize themselves to active applications. In particular, one major problem facing computer systems today is that applications are increasingly bottlenecked on accessing data. My recent work focuses on two ways to make data accesses inexpensive: scheduling data in caches near the cores that access it; and using analytical performance models to manage cache space more efficiently.
Massachusetts Institute of Technology .................................................. Sep 2015 - Jan 2017
POSTDOC with Prof. Daniel Sanchez; worked on well-behaved, high-performance memory systems for parallel processors.
Massachusetts Institute of Technology .................................................. Sep 2012 - Sep 2015
RESEARCH ASSISTANT to Prof. Daniel Sanchez; worked on scheduling data across caches in parallel processors.
Massachusetts Institute of Technology .................................................. Sep 2008 - Sep 2012
RESEARCH ASSISTANT to Profs. Anant Agarwal, Frans Kaashoek, and Nickolai Zeldovich; worked on distributed operating systems (fos project).
Nvidia .......................................................... Summer 2007
SOFTWARE INTERN in the embedded division; worked on OpenGL ES 2.0 and optimizing customer applications.
Symantec Research Labs ................................................................. Summers 2005 & 2006
RESEARCH INTERN at Symantec Research Labs; prototyped an early design of an extrusion detection system.
University of California, Los Angeles .................................................. Sep 2003 - Mar 2008
UNDERGRADUATE RESEARCHER with Profs. Glenn Reinman and Miodrag Potkonjak; worked on cache organization for physics simulation and statistical analysis of sensor networks.

PEER-REVIEWED PUBLICATIONS
Cache Calculus: Modeling Caches through Differential Equations ................... CAL 2016
Nathan Beckmann, Daniel Sanchez
Whirlpool: Improving Cache Management with Application-Level Data Classification ........... ASPLOS 2016
Anurag Mukkara, Nathan Beckmann, Daniel Sanchez
Acceptance rate: 22%
Modeling Cache Performance Beyond LRU .......................................................... HPCA 2016
Nathan Beckmann, Daniel Sanchez
Extended technical report: MIT CSAIL, April 2015.

Harshad Kasture, Davide Bartolini, Nathan Beckmann, Daniel Sanchez
Acceptance rate: 22%

Nathan Beckmann, Daniel Sanchez
Acceptance rate: 22%

CDCS: Scaling Non-Uniform Cache Architectures with Computation and Data Co-Scheduling ........ HPCA 2015
Nathan Beckmann, Po-An Tsai, Daniel Sanchez
Acceptance rate: 22%

Jigsaw: Scalable Software-Defined Caches ....................................................... PACT 2013
Nathan Beckmann, Daniel Sanchez
Acceptance rate: 22%

The Case for Elastic Operating System Services in fos ....................................... DAC 2012
Lamia Youseff, Nathan Beckmann, Harshad Kasture, Charles Gruenwald III, David Wentzlaff, Anant Agarwal
Acceptance rate: 23%

An Operating System for Multicore and Clouds: Mechanisms and Implementation ........ SOCC 2010
David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, Anant Agarwal
Acceptance rate: 19%


ATIC: Improving Performance and Programmability with On-Chip Optical Networks ........ ISCAS 2010
James Psota, Jason Miller, George Kurian, Henry Hoffmann, Nathan Beckmann, Jonathan Eastep, Anant Agarwal
Acceptance rate: 45%

A Unified Operating System for Clouds and Manycore: fos ................................... CAOS at HiPEAC 2010
David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, Anant Agarwal
Acceptance rate: 17%


Graphite: A Distributed Parallel Simulator for Multicores .................................. HPCA 2010
Jason Miller, Harshad Kasture, George Kurian, Charles Gruenwald III, Nathan Beckmann, Christopher Celio, Jonathan Eastep, Anant Agarwal
Acceptance rate: 18%


Hardware-based Public-key Cryptography with Public Physically Unclonable Functions .......... IH 2009
Nathan Beckmann, Miodrag Potkonjak

ADDITIONAL TECHNICAL REPORTS

PIKA: A Network Service for Multikernel Operating Systems ................................ MIT CSAIL, Jan 2014
Nathan Beckmann, Charles Gruenwald III, Charles Johnson, Harshad Kasture, Filippo Sironi, Anant Agarwal, Frans Kaashoek, Nickolai Zeldovich

Efficient Cache Coherence on Manycore Optical Networks ................................. MIT CSAIL, Feb 2010
George Kurian, Nathan Beckmann, Jason Miller, James Psota, Anant Agarwal

Core Count vs Cache Size for Manycore Architectures in the Cloud ....................... MIT CSAIL, Feb 2010
David Wentzlaff, Nathan Beckmann, Jonathan Eastep, Anant Agarwal

Jason Miller, James Psota, George Kurian, Nathan Beckmann, Jonathan Eastep, Jifeng Liu, Mark Beals, Jurgen Michel, Lionel Kimerling, Anant Agarwal

POSTERS

CDCS: Computation and Data Co-Scheduling ..................................................... Cloud Workshop, MIT, 2014
Po-An Tsai, Nathan Beckmann, Daniel Sanchez
Best student poster.

Jigsaw: Software-defined Caches ........................................................................ MIT CSAIL Industry Affiliate Program, 2013
Nathan Beckmann, Daniel Sanchez

Scalable Applications on a Factored Operating System .......................................... ASPLOS 2012

Applications on a Factored Operating System .................................................... EuroSys 2012
Charles Gruenwald III, Nathan Beckmann, Harshad Kasture, Chris Johnson, Barry Kasindorf, Larry Stewart, Anant Agarwal

Distributed Parallel Network Stack for Multicore ................................................. NSDI 2011
Charles Gruenwald III, Nathan Beckmann, David Wentzlaff, Harshad Kasture, James Ward, Anant Agarwal
TEACHING
6.823: Computer System Architecture ................................................................. Spring 2014
Teaching Assistant ................................................................................................. Massachutes Institute of Technology

PATENTS
Miodrag Potkonjak, Nathan Beckmann

Autonomous, non-interactive, context-based services for cellular phone ........ US Patent 8744429, June 2014
Miodrag Potkonjak, Nathan Beckmann

Nathan Beckmann, Miodrag Potkonjak

Scott Schneider, Nathan Beckmann (at Symantec Research Labs)

Semantic compression ................................................................. US Patent pending (filed Apr 2010)
Nathan Beckmann, Miodrag Potkonjak

PERSONAL

Last updated on September 7, 2016.