ASA: An Adaptive Space Allocation Algorithm for Cache Management in Multi-level Cache Hierarchy

Li Ou, Karthik Sankar,Xubin (Ben) He,
Department of Electrical and Computer Engineering
Tennessee Technological University
lou21@tntech.edu, kksankar21@tntech.edu, hexb@tntech.edu

***Abstract*— Multi-level cache hierarchies are widely used in high-performance storage systems to improve I/O performance. However, traditional cache management algorithms are not suited well for such cache organizations. Recently proposed exclusive caching works well with single or multiple-client, low-correlated workloads, but suffers serious performance degradation with multiple-client, high-correlated workloads. In this paper, we propose a new cache space management algorithm, Adaptive Space Allocation (*ASA*), which implements both exclusive and inclusive caching and manages storage caches by providing optimal inclusive cache space adaptively according to the characteristic of input workloads. Ours results show that *ASA* increases the cumulative cache hit ratios dramatically for both high and low-correlated workloads.**

I. Introduction

Caching is a common technique for improving the performance of I/O systems. Researchers have developed many algorithms to manage the buffer cache, such as LRU [1], LFU, 2Q [2], LIRS [3], and ARC [4]. These algorithms were designed for local cache replacement because they do not need any information from other caches. They worked well for a single system. In a distributed I/O environment, buffer caches are mostly organized as multi-level cache hierarchies residing on multiple machines. We refer to upper level storage client caches as L1 buffer caches and lower level storage caches as L2 buffer caches [5]. The access patterns of L2 caches show weak temporal locality [5] after filtering from L1 caches, which implies that a cache replacement algorithm, such as LRU, may not work well for L2 caches. Additionally, local management algorithms used in L2 caches are inclusive [6], which keep blocks that have been cached by L1 caches, and waste aggregate cache space.

Several attempts have been made to improve cache performance of multi-level buffer caches for distributed I/O systems. Recent research [6], [5], [7], [8], [9] characterizes the behavior of accesses to L2 caches, and introduces multiple algorithms based on the characteristics to improve the L2 cache hit ratio. Except for multi-queue replacement [5], all other algorithms try to achieve exclusive caching [6] through quick eviction of duplicated blocks in L2 caches. Implementing aggressive exclusive caching may get a high hit ratio in case of a single storage client, but multiple-client systems introduce a new complication: the sharing of data among clients. It may no longer be a good idea to discard a recently read block from the L2 cache after it has been sent to a client cache, because the block may be referenced again by other clients in the recent future. Real workloads show behavior between two extremes: disjoint workloads, in which the clients each issue references for non-overlapping parts of the aggregate working set, and conjoint workloads, in which the clients each issue exactly the same references in the same order at the same time [6]. Nearly disjoint workloads are *low-correlated* workloads, and nearly conjoint workloads are *high-correlated*. For low-correlated workloads, aggressive exclusive caching is effective, but for high-correlated workloads, since the same blocks may be referenced by multiple clients within a relatively short time period, inclusive caching is more attractive. Thus, for a multiple-client system, it is important to design an algorithm which balances between aggressive exclusive caching and inclusive caching according to workload characteristics.

In this paper, we propose a new cache space management algorithm, Adaptive Space Allocation (*ASA*), for multi-level I/O systems to provide high cumulative hit ratios in multiple storage client cache systems, for both high-correlated and low-correlated workloads. The main idea of ASA is to implement exclusive caching in storage servers to increase utilizations of aggregate cache spaces, while allocate a small inclusive cache to improve local hit ratios for blocks frequently referenced by multiple clients. The size of the small inclusive cache needs to be tuned carefully to balance between high-correlated and low-correlated workloads. *ASA* manages storage cache and provides optimal inclusive cache space adaptively according to the characteristic of input workloads. We compare the *ASA* algorithm with the traditional LRU and other typical multi-level cache management algorithms such as exclusive caching [6], [5], 2Q [2], and SLRU [6], using simulations under different workloads. The results show that compared to LRU, *ASA* can dramatically increase the overall cache hit ratios.

II. Design of ASA

Exclusive caching increases cache hit ratios for single client systems or multiple-client systems with low-correlated workloads, while inclusive caching provides better performances for multiple-client systems with high-correlated workloads. One way to achieve high hit ratios for both single client and multiple-client systems is to divide cache space into two part and implement both inclusive caching and exclusive caching, but the size of the two caches need to be adjusted dynamically according to the characteristics of input workloads. Furthermore, in a multiple-client system, a higher correlation of workloads means that it is more likely that a block requested by one client is found in caches of other clients, because a block used by one client may have been or will be referenced by other clients within a limited time period, so cooperative client caches [10] could be used with the inclusive cache of storage servers to improve hit ratios of high-correlated workloads.

In our design, we implement an exclusive cache in the storage server to improve hit ratios of low-correlated workloads, and use cooperative client caches to provide high hit ratios for high-correlated workloads, which means requests do not result cache hits in the storage server could be redirected to a client cache. We define *cumulative hit ratios* as hit ratios provided by both the storage cache and cooperative client caches, and *local hit ratios* as hit ratios provided only by the storage cache. Combination of exclusive caches in the storage server and cooperative client caches may achieve high cumulative hit ratios, but suffers local hit ratio degradation in the case of high-correlated workloads. We use a small inclusive cache, like LRU cache, in the storage side, to improve local hit ratios by caching blocks used frequently by multiple clients, but the size of the small inclusive cache needs to be tuned carefully to avoid sacrificing cumulative hit ratios too much, since duplicated blocks exist in both the inclusive cache of storage server and client caches. *ASA* try to maintain high cumulative hit ratios while increase local hit ratios by adjusting the size of the inclusive cache and exclusive cache in the storage server dynamically according to the characteristic of input workloads.

In high-correlated workloads, *ASA* increases the size of the inclusive cache because the number of reused blocks is relatively large. In low-correlated workloads, *ASA* decreases the size of the inclusive cache because blocks are seldom reused by multiple clients. In the case of single client system, or disjoint workloads, since there is no reused blocks at all, *ASA* try to allocate the entire storage cache space to exclusive caching as quickly as possible. In our design, we use a LRU cache as the inclusive cache in the storage side. One hit of the LRU cache will increase its size by one block, and one hit of the exclusive cache will shrink its size by one block (at the same time, the size of the exclusive cache increases by one block). Since we need to maintain high cumulative hit ratios, a ghost cache which simulates a totally exclusive storage cache is implemented to provide a reference for each moment of accesses. We compare the actual cumulative hit ratio and the reference provided by ghost cache periodically. If the current cumulative hit ratio is too low compared to that of the ghost cache, the LRU cache size will be reduced. Fig. 1 outlines the *ASA* algorithm.

```
/* procedure to be invoked upon a reference to block b */

blockGet(block b)
{
  if b is in exclusive cache {
    remove b from exclusive cache;
    increase size of exclusive cache by 1;
  }
  else if  b is in LRU cache {
    remove b from LRU cache;
    increase size of LRU cache by 1;
  }
  else
     read b from disk;
  put b to tail of LRU queue;
}

/* procedure to be invoked upon every 1000 references to
cache */

adjustcache()
{
   calculate cumulative hit ratio of real cache and ghost;
   if hit ratio of ghost is much larger than real cache
       decrease size of LRU by n; // n is a tunable parameter.
}

/* procedure to be invoked upon a eviction of block b */

blockPut(block b)
{
  if b in LRU cache
     remove b from LRU cache;
  put b into exclusive cache;
  put b into ghost cache;
}
```

Fig. 1. ASA algorithm.

TABLE I
CHARACTERISTICS OF TRACES

Trace	Clients	IOs (millions)	Volume	Correlation
Cello92	1	0.5 per day	10.4GB	high
HTTPD	7	1.1	0.5GB	high
DB2	8	3.7	5.2GB	low

III. Simulation Methodology

We use trace-driven simulation to compare cumulative L2 cache hit ratios of *ASA* and other algorithms, including LRU, 2Q [2], exclusive caching [6], and SLRU [6]. We have developed a simulator to simulate two-level buffer cache hierarchies with multiple clients and one storage system. LRU is used as the replacement algorithm in the L1 caches, and multiple algorithms mentioned before are implemented in the L2 cache. Thus in our simulations, when referring to LRU, we talk about *LRU-LRU* (L1-L2 caches). In our study, we use 4KB as the cache block size for our experimental evaluation. We have examined other block sizes, with similar results. Three traces are chosen to represent different types of workloads: high-correlated and low-correlated. Table I shows the characteristics of traces.

The *HP Cello92* trace was collected at *Hewlett-Packard* Laboratories in 1992 [11]. It captured all L2 disk I/O requests in *Cello*, a timesharing system used by a group of researchers to do simulations, compilation, editing, and e-mail, from April 18 to June 19. We use the trace collected on April 18 as the workload for the single client simulation. Since requests of the traces collected in different days access the same data set, we also use them as workloads for the multiple-client simulation: each trace file collected within one day acts as the workload of one client. These workloads are high-correlated. The *HTTPD* workload was generated by a seven-node *IBM SP2* parallel web server [12] serving a 524MB data set. Multiple http servers share the same files. We use the *HTTPD* workload as the high-correlated workloads for the multiple-client simulation. The *DB2* trace-based workload was generated by an eight-node *IBM SP2* system running an *IBM DB2* database application that performed join, set and aggregation operations on a 5.2GB data set. Each *DB2* client accesses disjoint parts of the database. No blocks are shared among the eight clients. We use the *DB2* workload as the low-correlated workload for the multiple-client simulation.

Since L1 buffer cache sizes clearly affect an L2 cache's performance, we carefully set the L1 buffer cache sizes for the three traces to achieve a reasonable L1 hit ratio. The cache size of *HP 9000/877* server is only 10-30MB, which is very small by current standard. The *Cello92* trace and the *HTTPD* trace show high temporal locality, and a small client cache may achieve a high hit ratio. In the simulations, we assume the cache size of each client is 16MB for the *Cello92* traces, and 8MB for the *HTTPD* trace, providing an L1 hit ratio of approximately 50%. The *DB2* trace shows very low temporal locality, and a 512MB client cache just provides an L1 hit ratio of no more than 15%. But if the cache size increases to 600MB, the L1 hit ratio suddenly increases to 75%, because *reuse distances* [5] of most blocks are less than 150KB(600MB divided by block size 4KB). To reserve enough cache misses for L2 caches, we assume the cache size of each client for the *DB2* trace is 512MB. Since the number of compulsory cache misses in the *DB2* trace is large, we use approximately 10% of the requests to warmup the cache space.

IV. Simulation Results

A. Single client and low-correlated traces

We use the *Cello92* trace as the workload of a single client system, and *DB2* trace as low-correlated workload for a multi-client system. Actually, *DB2* trace is a disjoint workload since there is no block reused among multiple clients. Fig. 2(b) and Fig. 2(d) show the hit ratios under those two workloads.

It is obvious that there is almost no difference between hit ratios of *ASA* and purely exclusive caching under both workloads. It is the results exactly we expect, since in single client or disjoint workload, *ASA* should quickly allocate entire cache space to exclusive caches. We record the percentage of exclusive cache in the storage cache space under different time of simulation for the single client *Cello92* trace. The result is shown in Fig. 3(a). At the beginning of the simulation, since there are no eviction operations (the client caches are not full), *ASA* allocates most cache space to inclusive caches. After clients begin to evict blocks to the storage cache, *ASA* quickly reallocate cache space to the exclusive cache, and only within the time of 2% of total simulation, the size of exclusive cache is already more than 98% of the entire cache space, then reaches almost 100% quickly and maintains until the end of the simulation.

B. High-correlated traces

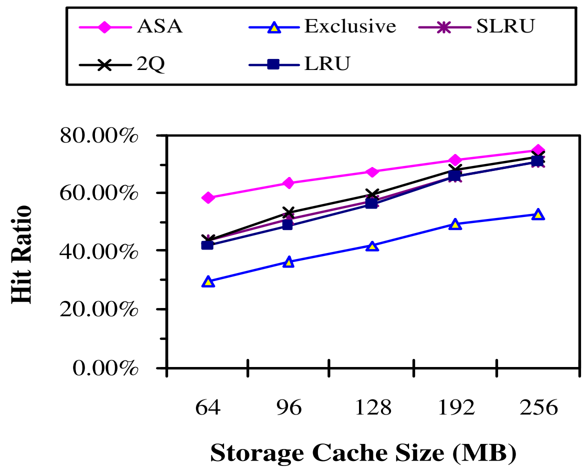
We use the *Cello92* trace and the *HTTPD* trace as multiple-client high-correlated workloads. Fig. 2(a) and Fig. 2(c) shows the hit ratios of different algorithms. The *ASA* always provides the best hit ratio among all the algorithms. LRU provides a relatively high hit ratio because each block in an LRU cache has a long life before it is discarded, and thus has a high possibility to be referenced again and again by different clients with high-correlated workloads. The gain of *ASA* becomes smaller as the storage cache is larger, since a large cache size retains a block for a long enough time, within which it is accessed by most clients. Exclusive caching suffers serious performance degradation even compared to LRU, because discarding a block immediately after it is referenced once causes many cache misses for following references from other clients.

ASA balances well between the local storage hit ratio and the cumulative hit ratio. Fig. 4 compares the cumulative and storage hit ratios of LRU, exclusive caching, and *ASA* under various configurations. We intentionally change replacement algorithms for exclusive caching and LRU to add cooperative client caches and provide cumulative hit ratios. We have mentioned before that entirely exclusive caching in storage with cooperative client caches provides maximal cumulative hit ratio, but very low storage hits in high-correlated workloads. When compared with exclusive caching, we find that cumulative hit ratio of *ASA* is almost same, while local storage hit ratio is much higher. We also find that even with cooperative client caches, LRU can not provide satisfying cumulative cache hits, because most blocks in the storage cache and cooperative client caches are same, and aggregate cache space is wasted. *ASA* provides both satisfying local hit ratios, almost same to typical inclusive caching, and high cumulative hit ratios.

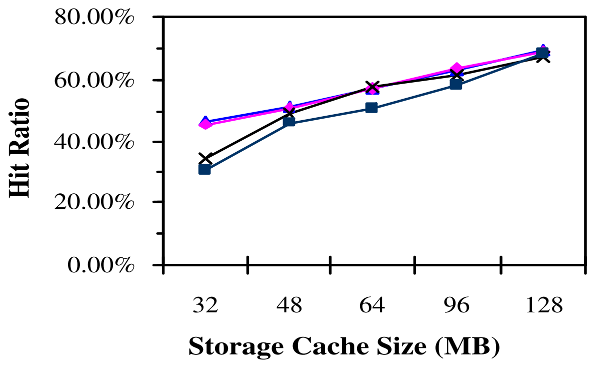
In high-correlated traces, *ASA* should maintain the balance between the size of the exclusive cache and inclusive cache, according to the characteristic of input workloads. We also record the percentage of exclusive cache in the storage cache space under different time of simulation for the 7 client *HTTPD* trace. The result is shown in Fig. 3(b). After the warm up of the cache space (about 10% of total simulation), the size of the exclusive cache remains at about 70% of the storage cache space. *ASA* allocate about 30% of the total cache space to the inclusive cache for keeping most requently reused blocks. This is why *ASA* could provide both satisfying local hit ratios and high cumulative hit ratios.

V. Related Work

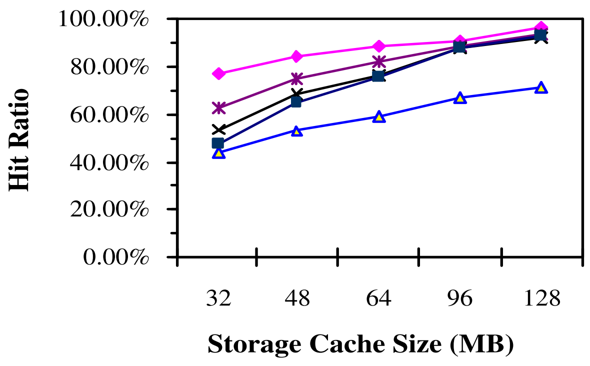
L2 caches have poor hit ratios as demonstrated in [5] and [6]. Many new algorithms have been proposed recently to improve cumulative hit ratios, such as MQ [5], Demotion-based algorithm [6], Global L2 buffer cache management [5], X-Ray [8], and client-controlled cache replacement [9]. Chen *et al.* [7] classified all those algorithms into two types: hierarchy-aware caching, and aggressively-collaborative caching, and compared the performance among typical algorithms belonging to the two types. Ari *et al.* proposed ACME [13] to adaptively select the best replacement policy for each cache-level to achieve high accumulative hit ratios. Our work in multi-level cache hierarchies builds upon but is different from previous studies because the *ASA*



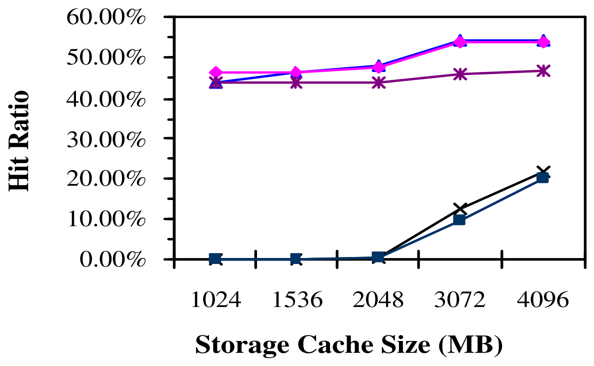
(a) 4 clients under Cello92 trace



(b) Single client under Cello92 trace

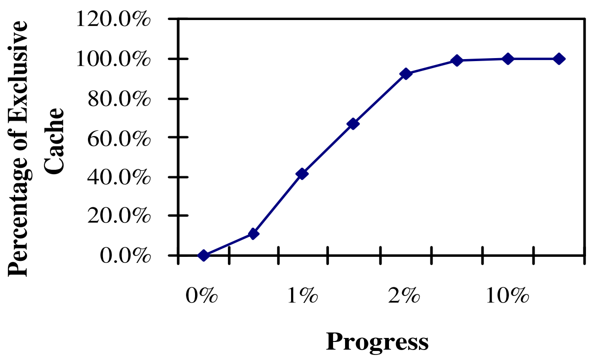


(c) 7 clients under HTTPD trace

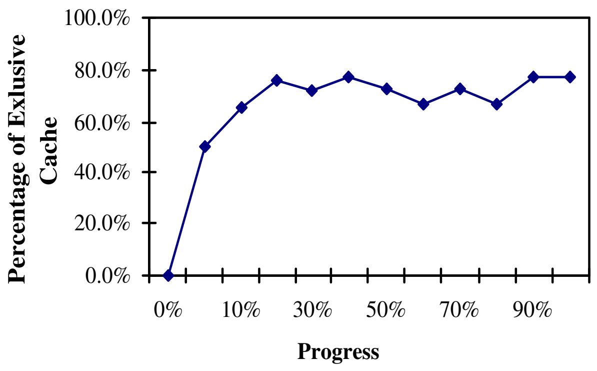


(d) 8 clients under DB2 trace

Fig. 2. L2 cache hit ratio with various number of clients under the Cello92 traces, the HTTPD trace, and the DB2 trace.

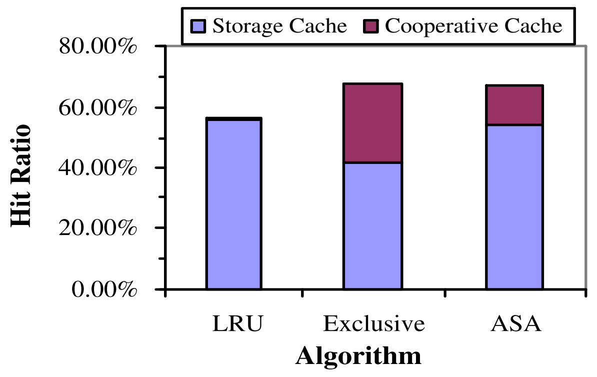


(a) Single client under Cello92 trace

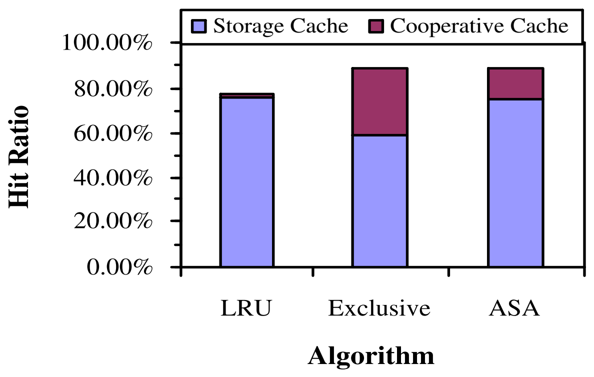


(b) 7 clients under HTTPD trace

Fig. 3. The percentage of the exclusive cache in the storage cache space according to the progress of simulation under various workloads.



(a) 4 clients under Cello92 trace



(b) 7 clients under HTTPD trace

Fig. 4. Comparison of cumulative and storage cache hit ratio among LRU with cooperative caches, exclusive caching with cooperative caches, and ASA under the Cello92 trace in 128M storage cache size (a), and the HTTPD trace in 64M storage cache size (b).

algorithm is adaptive to multiple-client systems, with either high-correlated workloads or low-correlated workloads.

Researchers have considered using adaptive algorithm to adjust the cache size according to input workloads. Megiddo and Modha [4] dynamically set size of two cache queues to quickly evict cold blocks, while keep frequently used blocks. Wong and Wilkes [6] adaptively insert evictions and loading blocks into various points of cache space to improve hit ratios of storage server caches in multi-client systems. Our work is related to but different from those previous algorithms, because we combine exclusive caching, inclusive caching, and cooperative client caching together. *ASA* allocate cache space dynamically to the inclusive cache and exclusive cache to balance cumulative and local hit ratios.

VI. Conclusions

In this paper, we propose a new buffer cache management algorithm: *ASA*, to improve performance of L2 caches in multi-level cache hierarchies for both single client systems and multi-client systems with high-correlated and low-correlated workloads. *ASA* combines both exclusive caching in storage caches to improve hit ratios for low-correlated workloads, and cooperative client caching to improve hit ratios for high-correlated workloads. *ASA* allocates cache space dynamically between the inclusive caches and exclusive cache to maintain cumulative hit ratios and improve local hit ratios according to various workloads.

We have evaluated our *ASA* algorithm and other typical multi-level caching algorithms using simulations under both high-correlated and low-correlated workloads. The results show that *ASA* achieves high hit ratios for both low-correlated high-correlated workloads, and dramatically increases the cumulative cache hit ratio over LRU.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation under a REU grant SCI-0453438, Research Office under a faculty research grant, and the Center for Manufacturing Research at Tennessee Technological University.

References

[1] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *ACM SIGMETRICS*, May 1990, pp. 143–152.

[2] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," in *Proc. Twentieth International Conference on Very Large Databases*, 1995, pp. 439–450.

[3] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in *Proc. ACM SIGMETRICS*, 2002, pp. 31–42.

[4] N. Megiddo and D. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. Second USENIX Conf. File and Storage Technologies*, 2003.

[5] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *IEEE Transactions on Parallel Distributed Systems*, July 2004.

[6] T. Wong and J. Wilkes, "My cache or yours? Making storage more exclusive," in *Proc. USENIX Ann. Technical Conf.*, 2002.

[7] Z. Chen, Y. Zhang, and Y. Zhou, "Empirical evaluation of multi-level buffer cache collaboration for storage systems," in *ACM SIGMETRICS*, 2005.

[8] L. N. Bairavasundaram, M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "X-RAY: A non-invasive exclusive caching mechanism for RAIDs," in *Proc. 31th Annual International symposium on Computer Architecture*, June 2004, pp. 176–187.

[9] S. Jiang and X. Zhang, "ULC: A file block placement and replacement protocol to effectively exploit hierarchical locality in multi-level buffer caches," in *Proceedings of the 24th International Conference on Distributed Computing Systems*, Mar 2004.

[10] M. Dahlin, R. Wang, T. Anderson, and S. Patterson, "Cooperative Caching: Using remote client memory to improve file system performance," *Operating Systems Design and Implementation*, 1994.

[11] C. Ruemmler and J. Wilkes, "Unix disk access patterns," in *Proc. Winter 1993 USENIX Conf.*

[12] E. D. Katz, M. Butler, and R. McGrath, "A scalable HTTP server: The NCSA prototype," *Computer networks and ISDN systems*, vol. 27, no. 2, pp. 155–164, Nov 1994.

[13] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. E. Long, "ACME: Adaptive caching using multiple experts," in *Proc. in Informatics*, vol. 14, 2002, p. 14158.