Designing Efficient Cooperative Caching Schemes for Multi-Tier Data-Centers over RDMA-enabled Networks [*]

S. Narravula[1] H.-W. Jin[2] K. Vaidyanathan[1] D. K. Panda[1]

[1]Department of Computer Science and Engineering
The Ohio State University
{narravul, vaidyana, panda}@cse.ohio-state.edu

[2]Computer Engineering Department
Konkuk University
jinh@konkuk.ac.kr

Abstract

Caching has been a very important technique in improving the performance and scalability of web-serving data-centers. The research community has proposed cooperation of caching servers to achieve higher performance benefits. These existing cooperative caching mechanisms often partially duplicate the cached data redundantly on multiple servers for higher performance (by optimizing the data-fetch costs for multiple similar requests). With the advent of RDMA enabled interconnects these basic data-fetch cost estimates have changed significantly. Further, the effective utilization of the vast resources available across multiple tiers in today's data-centers is of obvious interest. Hence, a systematic study of these various issues involved is of paramount importance. In this paper, we present several cooperative caching schemes that are designed to benefit in the light of the above mentioned trends. In particular, we design schemes that take advantage of the RDMA capabilities of networks and the multitude of resources available in modern multi-tier data-centers. Our designs are implemented on InfiniBand based clusters to work in conjunction with Apache based servers. Our experimental results show that our schemes achieve a throughput improvement of up to 35% as compared to the basic cooperative caching schemes and 180% better than the simple single node caching schemes. Our experimental results lead us to a new scheme which can deliver good performance in many scenarios.

1 Introduction

Clusters have easily become the most viable method to host web servers banking on their high performance-to-cost ratios. The structure of clusters, with several nodes connected with high performance interconnects, has seen the emergence of a popular web-serving systems: data-centers. With the explosive growth of the adoption of Internet and Web by today's society, the performance and scalability of these web-serving data-centers have become issues of paramount importance.

Caching of processed content in a data-center has been a long standing technique to help web systems to scale and deliver high performance. Several researchers [2] [11] have looked at various aspects of basic web caching. It has been well acknowledged in the research community that single larger cache performs significantly better than multiple smaller caches. Even though the total cache size remains the same in both cases, having multiple independent caches leads to very high localization of caching decisions and individual servers do not take advantage of the neighboring caches. Due to this disadvantage and the fact that current clusters support efficient data transfers, [14, 15, 18, 12], for example, have proposed cooperation among caches within the data-center. Several nodes in a data-center participate in this cooperative caching process and try to present a logical illusion of having a single cache. While such cooperative caching schemes show better performance than the basic case of nodes caching independently, several trade-offs exist and a through study of needs to be done and caching schemes need to be modified appropriately to reflect these.

Currently many approaches do not cautiously eliminate the possibility of duplicating cached entries redundantly on more than a single node. Some approaches can handle this in a limited manner with no direct control over the cache content [13, 7] . While this leads to better performance for the duplicated cache content, it is often at the cost of not caching other content that could have occupied this cache space. On the other hand, lack of redundant duplicates in the system necessitates the transfer of the cached entries from remote nodes on each request. In addition, cache replacement for each local cache is also considerably complicated due to the additional checks, logic and data transfers

---

[*] This research is supported in part by Department of Energy's grant #DE-FC02-01ER25506 and National Science Foundation's grants #CNS-0403342 and #CCR-0509452.

needed. A good balance between these trade offs is critical to achieve good performance. Consequently, designing these more complex protocols to deliver high performance in the light of these constraints, is a central challenge. In this paper, we present schemes to eliminate these redundant copies.

Since lack of duplication of content incurs higher data-transfer overheads for cache retrieval, traditional network hardware/software architectures that impose significant communication load on the server CPUs and memory cannot benefit fully from these. On the other hand, Remote Direct Memory Access (RDMA) enabled network interface cards (e.g., InfiniBand) are capable of providing reliable communication without server CPU's intervention. Hence, we design our cache cooperation protocols using RDMA with other one-sided operations to alleviate the possible effects of the high volume of data transfers between individual cache and sustain good overall performance.

Furthermore, current generation data-centers have evolved into complex multi-tiered structures presenting more interesting design options for cooperative caching. The nodes in the multi-tier data-center are partitioned into multiple tiers with each tier providing a part of request processing functionality. The front-end proxy nodes typically perform caching functions. In this paper, we also propose the use of the available back-end nodes to assist the proxies in the caching services. Mechanisms of providing access to the back-end nodes increases the overall complexity of the design and could potentially incur additional overheads. In our design, we handle this challenge by introducing additional passive cooperative cache system processing modules on the back-end servers. The benefits of these cache modules on the back-end servers are two-fold: (i) they provide the back-end servers access to the caching system and (ii) they provide better overall performance by contributing a certain amount resources of the back-end server when possible.

We implement our system over InfiniBand using Apache Web and Proxy Servers [10]. We further evaluate the various design alternatives using multiple workloads to study the trade-offs involved. The following are the main contributions of our work:

- Cooperative caching schemes to eliminate redundant copies and improve performance: We propose two schemes: (i) Cooperative Cache Without Redundancy (CCWR) and (ii) Multi-Tier Aggregate Cooperative Cache (MTACC).

- Detailed experimental evaluation and analysis of the trade offs involved. Especially the issues associated with working-set size and file sizes are analyzed in detail.

- Based on our evaluations with the above-mentioned schemes we propose a Hybrid Cooperative Caching (HYBCC) scheme that addresses the trade-offs associated with CCWR and MTACC and achieves the best of both the schemes.

Our experimental results show throughput improvements of up to 35% for certain cases over the basic cooperative caching scheme and improvements of up to 180% over simple caching methods. We further show that our schemes scale well for systems with large working-sets and large files.

The remaining part of the paper is organized as follows: Section 2 provides a brief background about modern network interconnects, and multi-tier data-centers. In Section 3 we present the design detail of our implementation. Section 4 deals with the detailed performance evaluation and analysis of our designs. In Section 5, we discuss current work in related fields and conclude the paper in Section 6.

2 Background

In this section, we present an overview of RDMA-enabled interconnects and multi-tier data-centers.

RDMA-enabled Interconnects: Many of the modern interconnects such as InfiniBand and Ammasso Gigabit Ethernet provide a wide range of enhanced features. InfiniBand Architecture [3] is an industry standard that defines a System Area Network (SAN) that offers high bandwidth and low latency. In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by Host Channel Adapters (HCA) and Target Channel Adapters, respectively. An abstraction interface for HCA's is specified in the form of InfiniBand Verbs. InfiniBand supports both channel and memory semantics. In channel semantics, send/receive operations are used for communication. To receive a message, the receiver first posts a receive descriptor into a receive queue. Then the sender posts a send descriptor into a send queue to initiate data transfer. In channel semantics there is a one-to-one match between the send and receive descriptors. In memory semantics, the operations allow a process to write to a virtually contiguous buffer on a remote node. Such one-sided operation does not incur software overhead at the remote side. Remote Memory Direct Access (RDMA) Read, RDMA Write and Remote Atomic Operations (*fetch-and-add* and *compare-and-swap*) are supported by InfiniBand.

Multi-Tier Data-Centers: A typical data-center architecture consists of multiple tightly interacting layers known as tiers. Each tier can contain multiple physical nodes. Figure 1 shows a typical multi-tier data-center. Requests from clients are load-balanced by the edge services tier on to the nodes in the front-end proxy tier. This tier mainly does caching of content generated by the back-end tiers. The other functionalities of this tier include embedding inputs from various application servers into a single HTML document (for framed documents for example), balancing the requests sent to the back-end based on certain pre-defined algorithms.

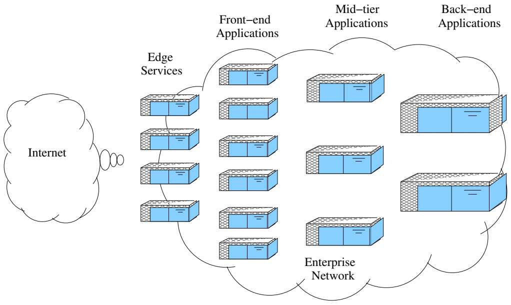

Figure 1. A Typical Multi-Tier Data-Center (Courtesy CSP Architecture design [16])

The middle tier consists of two kinds of servers. First, those which host static content such as documents, images, music files and others which do not change with time. These servers are typically referred to as web-servers. Second, those which compute results based on the query itself and return the computed data in the form of a static document to the users. These servers, referred to as application servers, usually handle compute intensive queries which involve transaction processing and implement the data-center business logic.

The last tier consists of database servers. These servers hold a persistent state of the databases and other data repositories. These servers could either be compute intensive or I/O intensive based on the query format. For simple queries, such as search queries, etc., these servers tend to be more I/O intensive requiring a number of fields in the database to be fetched into memory for the search to be performed. For more complex queries, such as those which involve joins or sorting of tables, these servers tend to be more compute intensive.

3 Design and Implementation of Proposed Cooperative Cache Schemes

In this section, we propose four schemes for cooperative caching and describe the design details of our schemes. At each stage we also justify our design choices. This section is broadly categorized into four main parts: (i) RDMA based design and implementation of basic cooperative caching (Section 3.1) , (ii) Design of a non-redundancy scheme (Section 3.2), (iii) Multi-tier extensions for cooperative caches (Section 3.3) and (iv) A combined hybrid approach for cooperative caches (Section 3.4). We first describe the common design aspects of our schemes.

External Module: The traditional data-center applications service requests in two ways: (i) by using different server threads for different concurrent requests or (ii) by using single asynchronous server to process to service requests. Catering to both these approaches used by applications, our design uses an asynchronous external helper module to provide cooperative caching support. Figure 2 shows the typical setup on each node. This module handles inter-node communication by using InfiniBand's native Verbs API (VAPI) and it handles intra-node communication with the basic data-center applications using IPC. This module is designed to be asynchronous to handle multiple overlapping requests from the data-center applications.

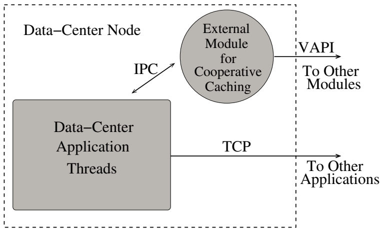

Figure 2. External Module based Design

Soft Shared State: The cache's meta-data information is maintained consistently across all the servers by using a home node based approach. The cache entry key space (called *key-table*) is partitioned and distributed equally among the participating nodes and hence all the nodes handle a similar amount of meta-data key entries. This approach is popularly known as the home node based approach. It is to be noted that in our approach we handle only the meta-data on the home node and since the actual data itself can reside on any node, our approach is much more scalable than the traditional home node based approaches where the data and the meta-data reside on the home node.

All modifications to the file, such as invalidations, location transfers, etc., are performed on the home node for the respective file. This cache meta-data information is periodically broadcasted to other interested nodes. Additionally, this information can also be requested by other interested nodes on demand. The information exchange uses RDMA Read operations for gathering information and send-receive operations for broadcasting information. This is done to avoid complex locking procedures in the system.

Basic Caching Primitives: The basic caching operations can be performed using a small set of primitives. The internal working caching primitives needs to be designed efficiently for scalability and high performance. Our various schemes implement these primitives in different ways and are detailed in the following sub-sections. The basic caching primitives needed are:

- *Cache Fetch*: To fetch an entity already present in cache
- *Cache Store*: To store an entity in cache
- *Cache Validate*: To verify the validity of a cached entity
- *Cache Invalidate*: To invalidate a cached entity

Buffer Management: The cooperative cache module running on each node reserves a chunk of memory. This memory is then allocated to the cache entities as needed. Since this memory needs to be pooled into the global cooperative cache space, this memory is registered (i.e. locked in physical memory) with the InfiniBand HCA to enable efficient memory transfers by RDMA. Several researchers [6, 5, 19] have looked at the different aspects of optimizing this limited buffer usage and have suggested different cache replacement algorithms for web caches. Our methods are orthogonal to these issues and can easily leverage the benefits of the proposed algorithms.

3.1 Basic RDMA based Cooperative Cache (BCC)

In our design, the basic caching services are provided by a set of cooperating modules residing on all the participating server nodes. Each cooperating module keeps track of the local cache state as a set of local page-tables and places this information in the soft shared state for global access.

The basic RDMA based Cooperative Caching is achieved by designing the cache primitives using RDMA operations. The communication messages between the modules are divided into two main components: (i) control messages and (ii) data messages. The control messages are further classified into (i) meta-data read messages and (ii) meta-data update messages. Since data messages form the bulk volume of the total communication load we use one-sided RDMA operations for these. In addition, the meta-data read messages use the RDMA Read capabilities. Meta-data update messages are exchanged using send-receive operations to avoid concurrency control related issues.

The basic cache primitives are handled by BCC in the following manner:

Cache Fetch involves three simple steps: (i) finding the cache entry, (ii) finding a corresponding amount of local free space and (iii) fetching the data using RDMA Read operation.

Cache Store involves the following steps: in case the local node has enough free space the entity is cached and *key-table* holding the meta-data information is updated. In cases where the local node has no free memory, the entity is stored into a temporary buffer and the local copies of all page tables are searched for a suitable candidate remote node for a possible free space. A control message is sent to that node which then performs an RDMA Read operation of this data and notifies the original node of the transfer. Once a control message is sent with a store request to a remote node, then the current entity is considered to be a responsibility of the remote node. For both these primitives, in cases where free space is not available system-wide, a suitable replacement is chosen and data is stored in place of the replacement.

Cache Validate and *Cache Invalidate* involve a meta-data read or a meta-data update to the home node respectively. As mentioned earlier, RDMA Read is used for the read operation.

Although this scheme provides a way to share cache across the proxy nodes, there may be redundancy in the cache entities across the system.

3.2 Cooperative Cache Without Redundancy (CCWR)

In this scheme, the main emphasis is on the redundant duplicates in the system. At each step of request processing, the modules systematically search the system for possible duplicate copies of cache entities and these are chosen for replacement. In aggregate, the cache replacement decisions are taken in the following priority: (i) Local free space, (ii) Remote node free space, (iii) Local redundant copies of entries cached elsewhere in the system, (iv) remote redundant copies having duplicates in the system and (v) replacement of suitable entity by removing an existing entry to make space for the new entry. We again describe the details of designs of the cache primitives.

The case of *Cache Fetch* presents interesting design options. The data from remote node is fetched into local free space or in place of local redundant copy in the priority described above. However, in case there are no free buffer spaces or local duplicates available for getting the data, remote cache entity is swapped with some local cached entity. In our design, we select a suitable local replacement, send a store message to the remote cache for this local replacement and followed by a RDMA Read of the required remote cache entity. The remote node follows a similar mechanism to decide on storage and sends back an acknowledgment. Figure 3 shows the swap case of this scheme. The dotted lines shown in the figure are control messages.

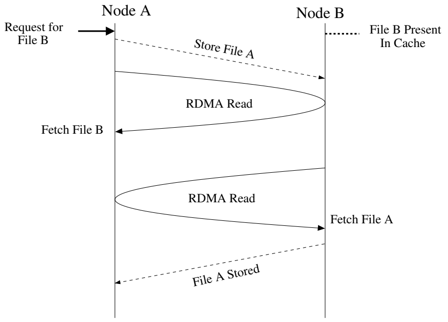

Figure 3. Cooperative Caching Without Redundancy

Cache Store design in this case is similar to the previous approach, the main difference being the priority order described above. The memory space for storing new cache entries is searched in the order of free space, redundant copies and permanent replacements.

The CCWR scheme benefits significantly by increasing the total amount of memory available for cooperative caching by removing redundant cache entities. For large working sets this yields higher overall performance.

3.3 Multi-Tier Aggregate Cooperative Cache (MTACC)

In typical multi-tier data-centers proxy servers perform all caching operations. However, the system can benefit significantly by having access to additional memory resources. There are several back-end nodes in the data-center that might not be using their memory resources to the maximum extent. In MTACC, we utilize this additional free memory on servers from other tiers of the multi-tier data-center. This provides us with more aggregate system memory across the multiple tiers for cooperative caching. Further, the involvement back-end modules in caching can be possibly extended to the caching support for dynamically changing data [11].

The MTACC scheme is designed with passive cooperative caching modules running on the back-end servers. These passive modules do not generate cache store or retrieve requests themselves, but help the other modules to utilize their pooled memory. In addition, these passive modules do not act as home nodes for meta-data storage, minimizing the necessity for cache request processing overheads on these back-end servers.

In addition, in certain scenarios such as cache invalidates and updates, the back-end servers need to initiate these operations [11]. Utilizing the modules existing on the back-end nodes, the back-end nodes can perform operations like invalidations, etc. efficiently with the help of the closer and direct access to cache to achieve significant performance benefits. Figure 4 shows a typical setup for MTACC.

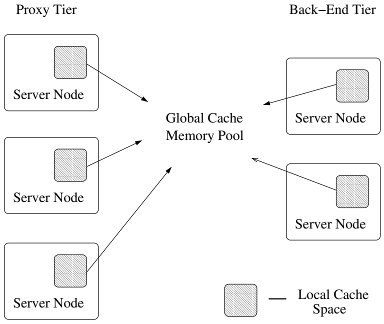

Figure 4. Multi-Tier Aggregate Cooperative Caching

3.4 Hybrid Cooperative Cache (HYBCC)

Though the schemes CCWR and MTACC can achieve good performance by catering to larger working sets, they have certain additional working overhead to remove redundant cache entities. While this overhead does not impact the performance in cases when the working set is large or when the requested file is large, it does impact the performance of the smaller cache entities or smaller working set files to a certain extent.

CCWR adds certain overhead to the basic cache processing. The added lookups for duplicates and the higher cost of swapping make up these overheads. MTACC also adds similar overheads. This larger aggregated cache system size can cause higher overheads for request processing.

To address these issues, we propose the Hybrid Cooperative Caching Scheme. In this scheme, we employ different techniques for different file sizes. To extent possible, smaller cache entities are not checked for duplications. Further, the smaller cache entities are stored and their lookups are performed on only the proxy servers without using the back-end servers. Hence, smaller cache entities are not stored on the passive nodes and are duplicated to the extent possible reducing the effect of the associated overheads.

4 Experimental Results

In this section, we present a detailed experimental evaluation of our designs. Here, we compare the following levels of caching schemes: (i) Apache default caches (AC) (ii) BCC, (iii) CCWR, (iv) MTACC and (v) HYBCC.

Experimental Testbed: For our experiments we used 20 nodes with dual Intel Xeon 2.66 GHz processors. InfiniBand network connected with Mellanox InfiniHost MT23108 Host Channel Adapters (HCAs). The clusters are connected using a Mellanox MTS 14400 144 port switch. The Linux kernel version used was 2.4.20-8smp. Mellanox IBGD 1.6.1 with SDK version 3.2 and the HCA firmware version 3.3 was used.

These nodes were setup with two web-servers and with the number of proxy servers varying from two to eight. The client requests were generated from multiple threads on 10 nodes. The web-servers and application servers used in the reference implementation are Apache 2.0.52. All proxy nodes we configured for caching of data. Web server nodes were also used for caching for the schemes MTACC and HYBCC as needed. Each node was allowed to cache 64 MBytes of data for any of the experiments.

Traces Used: Four synthetic traces representing the working sets in Zipf [20] traces were used. The files sizes in the traces were varied from 8KBytes to 64KBytes. Since the working sets of Zipf traces all have similar request probabilities, a trace comprising of just the working set is seemingly random. The working set sizes for these traces are shown in the Table 1. These present us with a number of cases in which the working sets are larger than, equal to or smaller than the total cache space available to the caching system.

4.1 Basic Performance Analysis

As an indication of the potential of various caching schemes, we measure the overall data-center throughput.

Trace	2 nodes	4 nodes	8 nodes	10 nodes
8k-trace	80M/128M	80M/256M	80M/512M	80M/640M
16k-trace	160M/128M	160M/256M	160M/512M	160M/640M
32k-trace	320M/128M	320M/256M	320M/512M	320M/640M
64k-trace	640M/128M	640M/256M	640M/512M	640M/640M

Table 1. Working Set and Cache Sizes for Various Configurations

Figures 5(a) and 5(b) show the throughput measured for the four traces. We see that the basic throughput for all the cooperative caching schemes are significantly higher than the base case of basic Apache caching (AC) - the default single node caching provided by apache.

Impact of Working Set Size: We notice that the performance improvements from the AC scheme to the other schemes show steep improvements when the cooperative caching schemes can hold the entire working set of that trace. For example, the throughput for the cooperative caching schemes for the 8k-trace for two nodes in Figure 5(a) are about 10000 TPS, where as the performance for AC is just above 5000 TPS. This shows a performance improvement of about a factor of two. This is because the AC scheme cannot hold the working set of the 8k-trace which is about 80 MBytes. Since each node can hold 64 MBytes, AC incurs cache misses and two node cooperative caching shows good performance. We see similar performance jumps for all cases where the working set fits in cache. Figure 6(a) clearly shows a marked improvement for larger traces (32k-trace and 64k-trace) for MTACC and HYBCC. This benefit comes from the fact that MTACC and HYBCC can accommodate more of the working set by aggregating cache from nodes across several tiers.

Impact of Total Cache Size: The total cache size of the system for each case is as shown in Table 1. For each configuration, as expected, we notice that the overall system performance improves for the cases where the working-set sizes are larger then the total system cache size. In particular, the performance of the 64k-trace for the 8 node case achieves a throughput of about 9500 TPS while using the memory aggregated from the web server for caching. This clearly shows an improvement of close to 20.5% improvement over basic caching scheme BCC.

Impact of System Size: The performance of the 8k-trace in Figure 6(b) shows a drop in performance for the CCWR and the MTACC cases. This is because as a result of aggregated cache across tiers for MTACC its total system size increases, hence the total overheads for each lookup also increases as compared to CCWR. On the other hand, since HYBCC uses CCWR for small cache entities and MTACC for large cache entities, its improvement ratios of HYBCC in Figure 6(b) clearly show that the HYBCC scheme does well in all cases. It is to be noted that the benefit of HYBCC will increase as the system size increases.

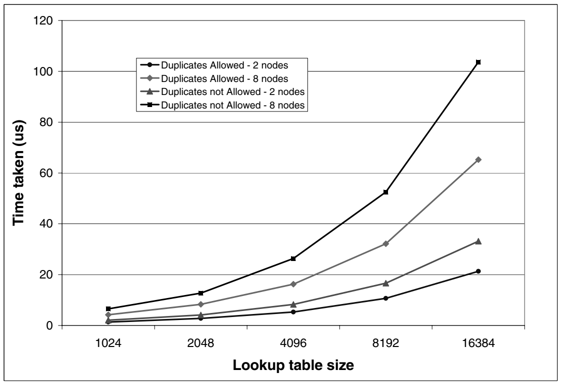

Figure 7. Bookkeeping and Lookup delay

Scheme	Fetch	Store	Validate	Invalidate
BCC	2,1	2,1	1,0	1,0
CCWR	2,2	2,1	1,0	1,0
MTACC	2,2	2,1	1,0	1,0
HYBCC	2,2	2,1	1,0	1,0

Table 2. Maximum number of messages required for each scheme (control-messages/data-messages)

4.2 Detailed Evaluation and Analysis

In this section, we discuss the performance benefits seen for each of the schemes and analyze the same.

4.2.1 Additional Overheads for Cooperative Caching

Our approaches incur different costs for lookup for different schemes. The primary difference is in the lookup times of schemes with redundancy allowed and schemes without redundancy. Figure 7 shows the worst case lookup latency for each request in steady state. We have seen that as the total size of the cache increases with number of nodes the lookup times also increase correspondingly. In addition, searching for redundant copies also incurs additional cost.

The number of network messages required for cache operations is shown in the Table 2. We see that the expected worst case number of control and data messages remain the same for all mechanisms with lower redundancy.

4.2.2 Detailed Data-Center Throughput Analysis

In the following sections, detailed analysis is presented for the each scheme to evaluate their effectiveness.

AC: These numbers show the system throughput achievable by using the currently available and widely used simple single node caching. Since all the nodes here take local decisions the performance is limited by the amount of cache available on individual nodes.

BCC: As shown by researchers earlier, the performance of the BCC scheme marks significant performance improvement over the AC scheme. These performance numbers

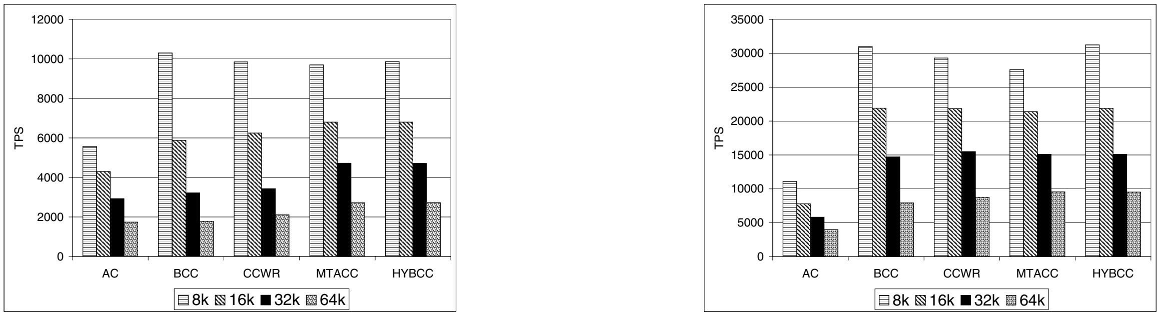

Figure 5. Data-Center Throughput: (a) Two Proxy Nodes (b) Eight Proxy Nodes

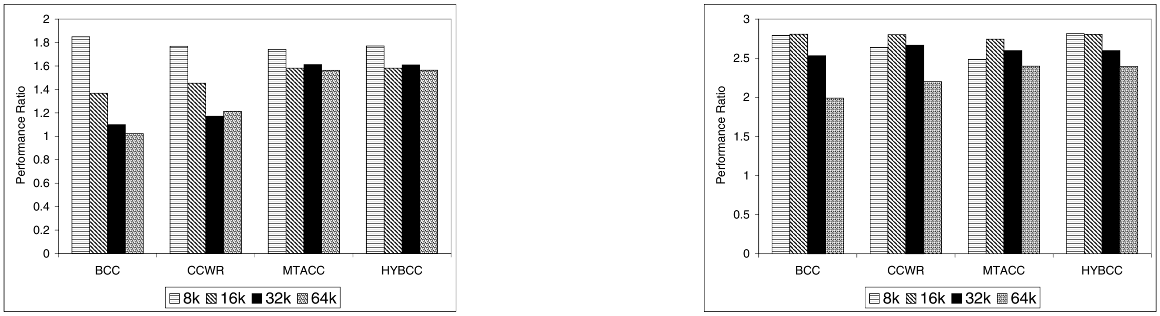

Figure 6. Performance Improvement: (a) Two Proxy Nodes (b) Eight Proxy Nodes

hence represent throughput achievable by basic cooperative caching schemes. In addition, the trends for the BCC performance also show the effect of working-set size as mentioned earlier. We see that as we increase the number of proxy servers, the performance benefit seen by the BCC scheme with respect to AC increases. The performance benefit ratio as shown in Figures 6(a) and 6(b) clearly shows this marked improvement.

CCWR: From the Figures 5(a) and 5(b), we observe that the performance for the CCWR method shows two interesting trends: (i) the performance for the traces 16k-trace, 32k-trace and 64k-trace show improvement of up to 32% as compared to the BCC scheme with the improvement growing with higher size traces and (ii) the performance of the 8k-trace shows a drop of about 5% as compared to the BCC scheme. The primary reason for this performance drop is the cost of additional book-keeping required for eliminating copies (as shown in Figure 7). We measured this lookup cost for this scheme to be about 5-10% of the total request processing time for a file of 8 Kbytes size. Since this cost does not grow with file size, its effect on larger file sizes is negligible.

MTACC: The main difference between the CCWR scheme and the MTACC scheme is the increase in the total system cache size and the total system meta-data information size. The additional system size improves performance by accommodating more entities in cache. On the other hand, larger lookup table size incurs higher lookup and synchronization costs. These reasons both show effect on the overall performance of the data-center. The 8 node case in Figure 5(b) shows that the performance of 8k-trace decreases with MTACC as compared to BCC and CCWR and the performance improves for 16k-trace, 32k-trace and 64k-trace. We observe similar trends for the 2 node case in Figure 5(a).

HYBCC: HYBCC overcomes the problems of lower performance for smaller files as seen above by using a hybrid scheme described in Section 3.4. In this case, we observe in Figures 5(a) and 5(b) that the HYBCC scheme matches the best possible performance. Also, we notice that the improvement of the HYBCC scheme over the BCC scheme is up to 35%.

5 Related Work

Several researchers [11, 8, 2, 4] have focused on the various aspects of caching. Cooperation of multiple servers is proposed as an important technique in caching [4, 9]. A popular approach of cooperative caching (e.g., [4]) uses application level redirects of requests to enable cooperative caching. This approach needs all the data-center servers to have different external IP addresses visible to the client and incurs higher overheads. On the other hand, approaches like [9, 1] use either a home node based approach for the data or use a single node for management activities. Though these approaches can be extended to have minimal redundancy, they are inherently susceptible to performance bottlenecks arising from central management mechanisms. In our approach, we use the concept of home node for just the metadata instead of the actual cached data. This alleviates the bottleneck problem to a large extent.

Further, we use an approach similar to the N-Chance approach proposed in the file-system research context in *XFS* [17]. Significant work [6, 5, 19] has been done with respect to the cache replacement algorithms. Our proposed schemes are orthogonal to these and can easily leverage the benefits of these.

In [13, 7] researchers have looked at content aware request handling which can provide controlled redundancy. These approaches can provide efficiency or optimality on a per file request basis. They redirect each request to the server holding that content. However, in HTTP 1.1 persistent connections are allowed and a client could request multiple files using the same TCP connection. This violates the per-file optimality condition. Our design is complimentary to these approaches and can handle this by sharing the cache across all nodes, thereby making all the nodes capable of handling all the files.

6 Conclusions

The importance of caching as an instrument for improving the performance and scalability of web-serving data-centers is immense. Existing cooperative cache designs often partially duplicate cached data redundantly on multiple servers for higher performance while optimizing the data-fetch costs for multiple similar requests. With the advent of RDMA enabled interconnects these cost estimates have changed the basic factors involved. Further, the utilization of the large scale of resources available across the tiers in today's multi-tier data-centers is of obvious importance.

In this paper, we have presented cooperative cache schemes that have been designed to benefit in the light of the above mentioned trends. In particular, we have designed schemes that take advantage of RDMA capabilities of networks and the resources spread across the multiple tiers of modern multi-tier data-centers. Our designs have been implemented on InfiniBand based clusters to work in conjunction with Apache based servers. We have evaluated these with appropriate request traces. Our experimental results have shown that our schemes perform up to 35% better than the basic cooperative caching schemes for certain cases and 180% better than the simple single node caching schemes.

We further analyze the performance of each of our schemes and propose a hybrid caching scheme that shows high performance in all our cases. We have observed that simple caching schemes are better suited for cache entities of small sizes and advanced schemes are better suited for the larger cache entities. As future work we propose to extend our work to support dynamic data cooperative caching.

References

[1] WhizzBee Web Server. `http://www.whizztech.com/`.

[2] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: limitations and potentials. In *Proceedings of the 4th International WWW Conference*, Boston, MA, December 1995.

[3] Infiniband Trade Association. http://www.infinibandta.org.

[4] Scott M. Baker and Bongki Moon. Distributed cooperative Web servers. *Computer Networks and ISDN Systems*, 31(11-16):1215–1229, May 1999.

[5] P. Cao and S. Irani. Greedydual-size: A cost-aware www proxy caching algorithm, 1997.

[6] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[7] Enrique V. Carrera, Srinath Rao, Liviu Iftode, and Ricardo Bianchini. User-level communication in cluster-based servers. In *HPCA*, 2002.

[8] Francisco Matias Cuenca-Acuna and Thu D. Nguyen. Cooperative caching middleware for cluster-based servers. In *Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[9] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proceedings of the ACM SIGCOMM'98 conference*, pages 254–265, September 1998.

[10] The Apache Foundation. http://www.apache.org/.

[11] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *Proceedings of System Area Networks (SAN)*, 2004.

[12] V. Olaru and W.F Tichy. Request distribution-aware caching in cluster-based web servers. In *Third IEEE International Symposium on Network Computing and Applications, 2004. (NCA 2004). Proceedings.*, 2005.

[13] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Architectural Support for Programming Languages and Operating Systems*, pages 205–216, 1998.

[14] L. Ramaswamy, Ling Liu, and A. Iyengar. Cache clouds: Cooperative caching of dynamic documents in edge networks. In *25th IEEE International Conference on Distributed Computing Systems, 2005. ICDCS 2005. Proceedings.*, 2005.

[15] J.B. Sartor, Subramaniam Venkiteswaran, K.S. McKinley, and Z. Wang. Cooperative caching with keep-me and evict-me. In *9th Annual Workshop on Interaction between Compilers and Computer Architectures, 2005. INTERACT-9.*, 2005.

[16] Hemal V. Shah, Dave B. Minturn, Annie Foong, Gary L. McAlpine, Rajesh S. Madukkarumukumana, and Greg J. Regnier. CSP: A Novel System Architecture for Scalable Internet and Communication Services. In *the Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, pages 61–72, San Francisco, CA, March 2001.

[17] Randolph Y. Wang and Thomas E. Anderson. xFS: A wide area mass storage file system. In *Workshop on Workstation Operating Systems*, pages 71–78, 1993.

[18] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.

[19] Q. Zhang, Z. Xiang, W. Zhu, and L. Gao. Cost-based cache replacement and server selection for multimedia proxy across wireless networks.

[20] George Kingsley Zipf. Human Behavior and the Principle of Least Effort. Addison-Wesley Press, 1949.