Cooperative Cache implementation using Dynamic Hash Scheme in Partitioned Networks

Mehdi Mohtashamzadeh
Department of Computer Engineering
Soosangerd Branch, Islamic Azad University
Soosangerd, IRAN

ABSTRACT

Caching is a popular mechanism for enhancing performance of memory access speed. To achieve such enhancement, cached data should have enough locality in time. Locality is an inseparable property of data in standalone computers, but do network traffic patterns have sufficient locality so that caches can take advantage from? Although individual IP addresses do not show significant locality, IP prefixes in a network (especially close to hosts which comprise hot documents) are dominantly repetitive and local, so caches may work based on IP prefixes instead of entire IP addresses.

Considering the locality of IP prefixes, there is still another challenge which should be overcome. According to memory constraints hosts are faced with, caches can include limited proportion of information that they really need to. To acquire non-cached information, nodes have to communicate with corresponding servers which is time and resource consuming. To effectively deal with such issue networks can be enhanced with cooperative caching by which every node provides its cache contents to others when they submit requests.

This paper outlines an improved cooperative caching method which functions based on IP prefixes to divide network to several partitions. Experiments indicate that the proposed mechanism would noticeably improve cache hit, cache load, and CPU load parameters.

Keywords

cooperative caching; IP prefix; Static and dynamic hashing.

1. INTRODUCTION

The main duty of caches is to store data so that future requests for that data can be served faster. The greater the number of requests that can be served from the cache, the higher the overall performance becomes. Since quantity of entries which can be cached is limited, caches can cooperate with each other to improve overall performance of networks. Cooperative caching aims to form a unified cache in the file cache hierarchy and has been proposed to reduce disk accesses [1].

Technology advances in hardware, namely processor design and network technology have significantly highlighted the importance of cooperative caching. In the meantime, disk technology has not experienced such giant improvements. In other words the performance gap between disk speed and processors is increasing unpleasantly. A case in point is transferring blocks of data between hosts in an Infiniband or 10 Gigabit Ethernet network which can be done two orders of magnitude faster than reading such blocks from nowadays disks.

To have efficient cooperative caches several issues should be considered. The geographical location of caches, the number of caches to form a group, allocating caches to appropriate groups, and cooperation among groups are the main challenges which should be tackled.

2. Related works

This research has taken advantage of two distinct concepts, cooperative caching and Route prefix caching. Related works in each area are listed below.

2.1 Related works on cooperative caching

Dynamic web content caching, which aims to move data and application closer to internet users, has become a controversial field to research communities and several contributions have been made [2, 3, 4]. In [5] a thorough analysis of different caching approaches and pros and cons of each has presented. Cooperation among caches was first discussed in [1, 6, 7, 8]. Shah et al. [9] have proposed a dynamic data propagation mechanism among cooperative caches, in which each data is allocated a dissemination tree. The server circulates the update to the data item through this tree. The main focus of last two mentioned researches was on consistency maintenance of caches.

Authors of [10] have proposed a dynamic hash-based cooperation scheme for efficient document lookups and updates. They have also presented a document placement and replacement scheme to place documents among caches of a cache cloud. In this research, each cooperative cache group consists of several sub-groups. An entry which is supposed to settle in the sub-group K may be hosted by different caches (in sub-group k) in different periods based on the capacity and load of caches in the sub-group.

Furthermore, researchers have worked on various challenges related to dynamic caching, namely reducing consistency function overheads and performance analysis of caching schemes [2, 11, 12, 13].

2.2 Related works on route prefix caching

IP router needs to inspect every packet header for destination IP address, then it will lookup routing table for next hop address in order to determine where to forward the packet. This process may become a bottleneck since it must be done for each packet. In contrast, if a router be capable of performing Longest Prefix Matching (LPM) based on the destination IP address then the process might be lighter and faster. In this case routers need to maintain a table of prefixes rather than exact IP addresses. Upon address lookup, the longest prefix that matches the beginning of the IP address is chosen.

The idea of using IP prefixes is based on following ideas [14]: first, the possible number of IP routing prefixes is much smaller than the number of possible individual IP addresses. A check of routing prefixes from The university of Oregon Route View project [15] showed 61832 entries as compared to up to 2^32 possible distinct IP addresses. Secondly, Web content servers are increasingly being hosted in central data centers. With World Wide Web dominating Internet traffic it is not surprising that a large portion of the traffic will be moving towards data centers. These centers host thousands of web sites, but only share one single routing prefix. Last but not least, Caching routing prefix can benefit from routing table compaction techniques [16].

Considering above facts, usage of IP prefixes can undoubtedly lead to better results. Current research has taken advantage of IP prefixes to direct various packets to different parts of network.

3. Architecture of the cooperative cache and design ideas

Network can comprise several partitions. These partitions are supposed to be located in parts of network which are selected to deploy caches. Hence, each partition has its own caches. There may be k main caches inside each partition which are responsible for initial caching of data. In addition, there are few caches (four caches in our implementation) for each main cache in four directions of partitions (North, South, East, and West) which are called boundary caches in this research. These caches are used to maintain copies of data items in main caches to achieve efficient response times and fault tolerance.

Simulation results show that each partition deals with limited number of IP prefixes. Putting forward this view, networks can be configured in a way that packets with different destination addresses may be directed to various partitions for processing. The belief that such data separation may efficiently improve cache performance, since caches in a specific partition would deal with limited IP addresses. In this case experiencing repeated requests (which their info exists in caches) by the partitions is highly probable.

Each partition has a central manager and some boundary cells. Boundary cells are responsible for incoming traffic diffusion. Shortly after network startup, boundary cells start to save IP prefixes for a while. In the next step, collected data would be sent from boundary cells to their corresponding central managers. Managers can extract IP prefix ranges by analyzing received data. Finally, each manager should send the acquired range to its boundary cells and other partitions' managers.

Each packet's IP prefix would be checked on its entrance to a partition. The packet can enter the partition if a boundary cell evaluates its IP related to IP ranges of the current partition; otherwise the packet would be rerouted to another partition. However, cache lookup process will start for received requests. The dynamic hash function by which an appropriate cache is selected to perform lookup process is described in the next section. Afterwards, the specified cache (by the mean of the dynamic hash function) will be searched using complete IP address and in case of a hit data can be extracted from this cache. Since data may also exist in one of the other four boundary caches related to the current cache, in order to eliminate congestion inside the partition, data items would be retrieved from one of these caches instead of the main cache. Suitable boundary cache to access is the nearest to the entry point of the request. However, in case of a miss data can be fetched from corresponding servers to a main cache which the hash function shows.

Table 1. Applied algorithm for boundary cells

```
Pick a packet from router queue;

                         PART A
If (a normal packet for routing is picked) then
  Extract "IP Prefix";
  If ("IP Prefix" is not related to the current partition) then
      Reroute packet to an adjacent partition;
  Else If (manager has declared congestion and has asked for
  temporary rerouting) then
      Set "rerouted packet" flag in the packet to 1;
      Set "source partition number" field in the packet with the
      number of current partition;
      Reroute packet to an adjacent partition;
  Else
      Commence cache lookup process;
  End if;
                         PART B
Else if (a "congestion status request (sent from manager)" is
picked) Then
    If (congestion controller detects congestion) then
          Set "congestion status Response" flag to '1';
    Else
          Set "congestion status Response" flag to '0';
    End if;
          Send "congestion status Response" packet to manager;

                         PART C
Else if (a "Reroute request (sent from manager)" is picked)
then
              Set "Rerouted packet" flag of all packets in router
              queue to 1;
              Set "Source partition Number" field of all packets in
              the queue with number of current partition;
              Reroute packets in the queue to adjacent partition;

End if;
Restart the algorithm;
```

Algorithm which Boundary cells have taken advantage of is presented in Table 1. As it can be seen in "PART B" of the algorithm, a manager can get aware of congestion status in different parts of its partition and in case of congestion detection manager may ask specific boundary cells to reroute requests to adjacent partitions temporarily (PART C).

As stated before, there are four caches for each main cache in four geographical directions of partitions. Each entry in the main cache has four counters, each for one of the boundary caches (N counter, S counter, E counter, and W counter). Each time data retrieval is performed using a boundary cache instead of the main cache, one of these counters would be increased according to geographical position of the used boundary cache. If one of the counters exceeds the threshold, from then on all requests for this entry must be served from corresponding boundary cache considering the fact that next requests would enter the partition from this geographical direction; as a result if such requests could be served in the borders of partition, congestion inside the partition could be plummeted. Such decisions is made and broadcast to all boundary cells by managers. Hence, entering plenty of requests, boundary cells can do lookup operation in one of the boundary caches and refer to the main cache only in case of misses.

It is worth noting that if data in servers are changed, these changes could be announced to main caches. In the meantime, boundary caches can get their contents consistent with main caches.

3.1 Dynamic hash function

A basic solution to cache assignment is to use a hash function which uniquely maps a requested URL to a specified cache in a collection of caches. This method is known as static hashing. The main drawback of this mechanism is that it cannot provide fair cache load balance since performance parameters such as cache load, CPU capacity, and network load are not considered.

On the other hand, dynamic hashing can uniformly disperse incoming requests in the cache group. “CPU Load”, “cache capacity”, and “workload” on the cache are dynamic parameters in our implementations since they can largely affect cache performance. In this work caches were configured to periodically compute and send their current CPU load, cache capacity, and cache load information to central manager. Analyzing received info, central manager can determine the node which is less busy and can handle more requests. Manager broadcasts this node’s address to others. Firstly Caches are hashed using following hash function:

MD5 (requested URL) mod k

Requested URL information should be fetched from main servers if they do not exist in the hashed cache. Before putting the fetched data into the hashed cache “CPU Load”, “cache capacity”, and “workload” parameters are checked. If the values have not exceeded their thresholds, it is permitted to assign cache entries to the fetched data; otherwise, an indirect caching process should take place. In this case, the cache which is recently introduced by central manager is designated to hold the fetched data. Towards this end, the “indirect” flag of the hashed cache entry (the entry which was supposed to be assigned to the fetched data) would be set to ‘1’; the address of designated cache should be saved in the mentioned entry of the hashed cache and the fetched date should be sent to designated cache. In case of future requests for the discussed URL, the cache is again hashed by “MD5 (requested URL) mod k” function. Lookup process would retrieve the entry which comprises the indirect flag equal to ‘1’; hence the address of designated cache would be extracted and used to fetch relevant data.

When the manager broadcasts address of a cache (to be used as designated cache by other caches), all caches look for indirect cached data in their entries (which their source is the same as the address broadcast by manager) and send found items back to its main source. Take for example “cache A” is introduced by manager and an entry which in fact is related to “cache A” is found in “cache B”. “Cache B” then tries to send back the indirect cached data to “cache A” considering the fact that from now on “cache A” can serve its own requests and there is no need for indirect cache accesses anymore.

4. Simulation results

Three cache configurations have been considered in simulations. Firstly, cache architecture without network partitioning and with static hashing mechanism, secondly, cache architecture with network partitioning and static hashing and finally, cache architecture with partitioned network and dynamic hashing function. Furthermore, requests were all saved in a file and induced into the network in specified timelines. Presented results in the following are related to 10000 requests (cache update and lookup) from the trace file.

Fig. 1 depicts the percentage of cache hits for three discussed configurations. X-axis presents number of caches. It is evident that despite from quantity of caches, hit ratio for the non-partitioned network with static hashing is noticeably lower compared to two other cases. Not only network partitioning improves cache hits (according to IP prefix disaggregation and directing each packet to an appropriate region) but also usage of dynamic hashing and increase in number of caches enhance the cache hits.

It can be seen in Fig. 1 that hit ratio of the non-partitioned network with static hash function which deploys hefty 20 caches is almost the same as the hit ratio of partitioned scheme with dynamic hash function with 5 caches.

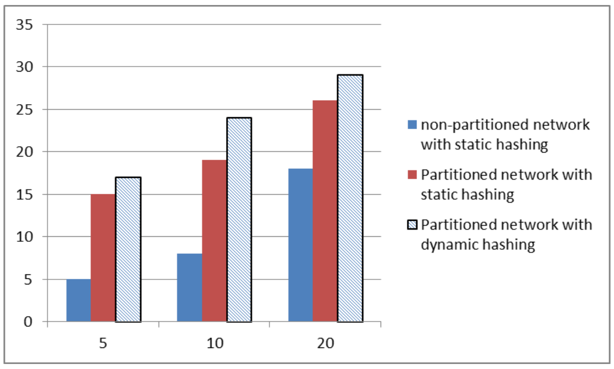

Fig. 1: Percentage of hit rates for 5, 10, and 20 caches and three cache configurations. X-axis and Y-axis present number of caches and hit ratio percentage, respectively.

In next step, a network with 5 caches was simulated and three above mentioned cache configurations were tested. For simulations network was partitioned to two regions. One of the partitions which included first, second, and third caches (on X-axis) dealt with 70% of entire requests, while the other region absorbed 30% of cache requests. Results of this simulation are presented in Fig. 2. Cache requests consist of lookup and update functions. Static hash function in the non-partitioned network uniformly spread requests among all caches, since the cache load parameter for all caches revolves around 2000. The partitioned static hashed network has experienced unfair load balance in both regions since it efficiently defuses packets according to their IP prefixes but disseminates requests entered a partition in an inefficient static manner.

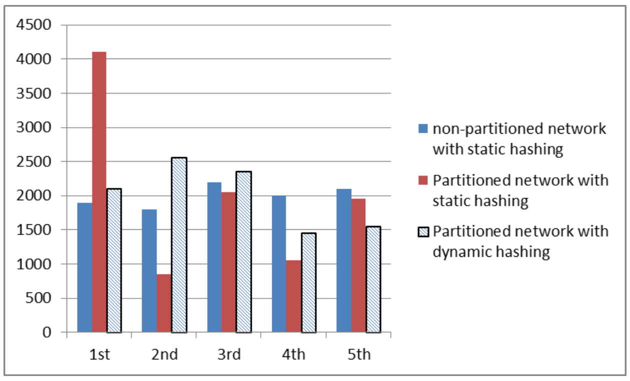

Fig. 2. Cache loads on 5 caches in three cache configurations. X-axis and Y-axis are cache numbers and quantity of requests received by each cache, respectively.

Lastly, the partitioned dynamic hash based network has shown more adequate cache load results compared to the previous approaches since it provides a closely uniform dispersal of cache loads in each partition. Although loads of requests are fairly shared among caches in the first scheme, it could not be considered as a suitable mechanism according to the results presented in Fig.1 and considering the fact that this mechanism cannot consider time-variant dynamic properties of caches.

5. Conclusion

In this research a cooperative caching method has been introduced. The proposed method takes advantage of IP prefixes to divide networks into some partitions, assuming that each partition deals with few IP addresses. Considering this fact, caching improves and requests are processed faster. Experiments indicate that the proposed mechanism would noticeably improve cache hit, cache load, and CPU load parameters.

6. REFERENCES

[1] M. Dahlin, R. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In OSDI'94, 1994.

[2] Edge Side Includes - Standard Specification. http://www.esi.org.

[3] IBM WebSphere Edge Server. http://www3.ibm.com/software/webservers/edgeserver/.

[4] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Application Specific Data eplication for Edge Services. In *WWW-2003*.

[5] C. Yuan, Y. Chen, and Z. Zhang. Evaluation of Edge Caching/Offloading for Dynamic Content Delivery. In *WWW-2003*.

[6] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web Caching with Consistent Hashing. In *WWW-8*, 1997.

[7] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet. In ICDCS-1999.

[8] A. Wolman, G. M. Voelkar, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In SOSP 1999.

[9] S. Shah, K. Ramamritham, and P. Shenoy. Resilient and Coherence Preserving Dissemination of Dynamic Data Using Cooperating Peers. IEEE-TKDE, July 2004.

[10] L. Ramaswamy, L. Liu, and A. Iyengar. Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks. 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05).

[11] W.-S. Li, W.-P. Hsiung, D. V. Kalshnikov, R. Sion, O. Po, D. Agrawal, and K. S. Candan. Issues and Evaluations of Caching Solutions for Web Application Acceleration. In VLDB-2002.

[12] D. A. Menasce. Scaling Web Sites Through Caching. IEEEInternet Computing, August 2003.

[13] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering Web Cache Consistency. ACM-TOIT, August 2002.

[14] H. Liu, "Routing prefix caching in network processor design," in Proc. ICCCN, Phoenix, AZ, 2001.

[15] University of Oregon Route Views Project, http://www.antc.uoregon.edu/route-views/

[16] H. Liu, "Reducing Routing Table Size using Ternary-CAM", Proc. Hot Interconnect, Stanford, 2001.