Distributed Caching Proxy Servers Design and Implementation

Neeraj Kumar
neerajkumar.cse06@itbhu.ac.in

***Abstract*—Proxy servers serve as an intermediate agent between clients and server systems. They relieve organizational networks from heavy traffic by storing the most frequently referenced web objects in their local cache. Design and development of such servers therefore is very critical for performance of network systems. In this paper we propose a distributed implementation of a caching proxy server. We also provide an algorithm for cache management in distributed proxy servers and conclude with implementation details for the same.**

***Index Terms*—Cache Management, Hashing, Proxy Servers, MPI**

I. INTRODUCTION

WITH the growth of modern communication systems the world wide web has grown exponentially. Internet applications are consuming more bandwidth than any other communication system. The issue of efficient management of internet resources is therefore of prime importance.

A Proxy Server is an application gateway which acts as an intermediate agent between client and server systems with primary objective to improve response time by caching frequently accessed data. It operates at the HTTP[5] layer.

The work of a proxy server is similar to a web server in transferring client requested files. In addition, it is able to interpret and modify the HTTP request before forwarding it to the end web server. Ideally, when a user requests for a file the proxy server first looks for it in its local cache. If there is a cache hit, the object is returned back to the user otherwise the request needs to be forwarded. The server can forward the request either to the web server or query a neighboring proxy. The idea that a proxy server can forward the request to its peer results in development of distributed proxy systems. The cache handling is improved and it is desired that most of the times it is CPU and not the network is involved in furnishing the requested object.

II. RELATED WORK

The idea of developing a distributed proxy server was studied and several methods were proposed in this regard.[2] Some of them make use of a hierarchical scheme. Here proxy servers are organized in levels. A server when requested, first searches for files in its local cache and if not found it could query a neighboring proxy until it reaches the root. This involves unnecessary forwarding of requests in case the requested object was not cached. Hashing forms the base of most of the distributed proxy server methodologies. A particular page is mapped to an unique proxy server from an array of proxy servers. Tsui,Kaiser, Liu [4] proposed Adaptive Distributed Caching (ADC) Algorithm for distributed cache management. They present self-organized cache management without use of any broadcasting protocol. The algorithm required a learning period before it could compete with other hashing based solutions.

The base of the algorithm proposed here is also hashing and hierarchical scheme but with due care to other issues like optimum request forwarding and efficient cache management.

Neeraj Kumar is a third year undergraduate student at Department of Computer Science and Engineering, Institute of Technology, Banaras Hindu University, Varanasi India-221005.

III. THE ALGORITHM

A. Related Terminology

In simple LRU based caching mechanism, any file which is accessed is cached by replacing the least recently used file in the cache. To ensure that a page is cached only if it has been accessed frequent enough and removed only if it has not been accessed recent enough, we use three tables viz *Primary table, Secondary table* and *Caching table.*The primary table stores those page/file urls which have been accessed at least once. Entries are added and removed on simple LRU basis. The secondary table has entries similar to primary table. An url is inserted in the secondary table from primary table if it has average request time lesser than the worst average request time in the secondary list. The caching table holds the list of files that are actually cached. A page requested after an entry for it is made in secondary table is considered for moving to caching table.

Sample primary/secondary table

Resource Id	**Last Accessed**	**Avg. request time**	**Size(kbs)**
abc.com/foo	1752	12	256

Caching table contains an additional field for proxy server which actually stores the requested resource on the network.

To ensure that old entries are removed from the cache an age parameter is introduced, defined as :

$$T_{age} = T_{avg}(1 + transfer\_rate/object\_size) \qquad (1)$$

$$T_{avg} = T_{avg} + (T_{now} - T_{last\_access})/2 \qquad (2)$$

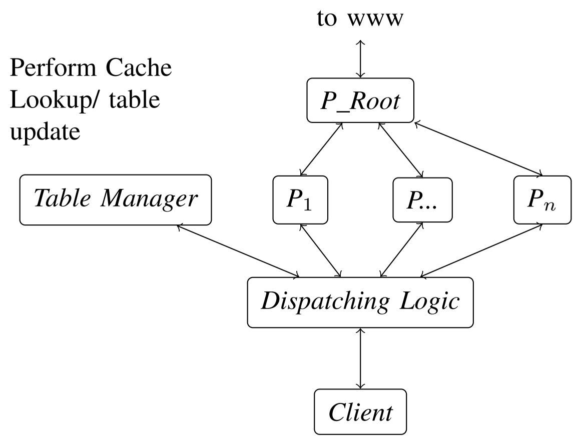
B. Algorithm

It is apparent that two compute-intensive tasks in a proxy system are Cache Lookup/Update and read/write from/to cache. In our proposed design, the focus was to distribute these tasks so as to attain maximum speedup. The proposed design has four major elements, the *dispatcher,* the proxy nodes, the root node (connected to world wide web) and the *TableManager*.
We maintain a two-level hierarchy. At the root level we have a proxy server connected to the internet(WAN). The proxy servers at the lower level of hierarchy are connected to the root node.
The TableManger is a distributed application responsible for managing the primary, secondary and Caching tables. The proxy nodes need not know about these tables, however these nodes need to maintain a mapping of resource-id to the local path at which the resource is stored.The dispatcher acts as an interface between the TableManager and the proxy nodes.
The step sequence is:

1) Client makes a request for resource.
2) The dispatcher creates a thread to handle this request. In this thread, the dispatcher queries the TableManager to check if the requested resource is cached or not.
3) If the requested resource is cached, the TableManager updates the tables with new timestamp for the resource and returns the machine name of the host which has cached the object, say $\mathrm{P_i}$.
4) The dispatcher then forwards the request to $\mathrm{P_i}$ and returns the response to the client.
5) If the requested resource is not cached, the TableManager looks up the primary and secondary table.
6) If a corresponding entry is present in secondary table, the resource needs to be cached. The TableManager then selects the host with minimum entries in caching table say $\mathrm{P_{min}}$. The Caching Table is then updated with an entry for this resource being cached on $\mathrm{P_{min}}$. The dispatcher then forwards the request to $\mathrm{P_{min}}$, instructing it to cache the response and then returns it to the client.
7) else If a corresponding entry is present in primary table, the tableManager checks if it can be moved to secondary table. If yes, then the secondary table is updated with this new entry.
8) If no entry is present in primary table, the tableManager simply returns. If no valid host is returned to the dispatcher, the dispatcher selects a proxy node(in round robin fashion) and forwards the request to this node. Note that the node will not cache the response as no entry for this request exists in the secondary table. The dispatcher checks the response to see if it is cacheable or not. If yes, it requests the TableManager to update the primary table with this request. The response is then returned back to the client.

The TableManger is an independent distributed application which populates/queries the cache tables based upon its interaction with the dispatcher. We have implemented this application using Message Passing Interface(MPI) in C and executed it on SGE ALTIX 450 server. An interesting idea will be to implement the same using MapReduce framework.

C. Line Diagram



IV. CONCLUSION AND FUTURE WORKS

In this paper we presented a design and implementation of a distributed caching proxy server. The proxy server design presented here provides an elegant solution to cache management with minimal response time. It should be noted that the request forwarding in any case is not greater than 2. Larger sized data are cached for a longer period of time.This reduces the time taken for service to minimum if they are found in cache. Furthermore the caching and removal of data from cache process is well managed to ensure that unnecessary objects are not cached.
The results with our implementation were found to compete well with other hashing based techniques. An interesting idea that needs to be tested is to also use the proxy nodes for Table Management. This way we will be able to further decrease the response time by adding more nodes to the network.

V. ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. Rajeev Srivastava, Associate Professor, Department of Computer Science and Engineering,IT-BHU, Varanasi (India) for his valuable guidance with this project.

REFERENCES

[1] Tsui, Kaiser, Liu *Distributed Proxy Server Management:A Self-Organized Approach* 2003.
[2] Law, Nandy, Chapman *A scalable and distributed WWW proxy System* 1997.
[3] Chiang, Li, Liu, Muller. *On request forwarding for dynamic web caching hierarchies.* In Proceedings of the 20th International Conference on Distributed Computing Systems, 2000.
[4] Tsui, Kaiser, Liu *Adaptive distributed caching* 2002.
[5] RFC for HTTP `http://www.faqs.org/rfcs/rfc2616.html`.