2012 2nd International Conference on Information Communication and Management (ICICM 2012)
IPCSIT vol. 55 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V55.5

Object Request Reduction in Home Nodes and Load Balancing of Object Request in Hybrid Decentralized Web Caching

Phongsathon Fongta and Yuthapong Somchit+

Department of Computer Engineering, Faculty of Engineering,
Chiang Mai University, Chiang Mai, Thailand

Abstract. In this paper, we propose a method to reduce the number of object requests sent to home nodes in hybrid decentralized web cache. Because when home nodes receive a large number of object requests, the hotspot problem can occur. In addition, the method also provides load balancing of object requests in delegates which are clients that can transfer web objects. The proposed method is evaluated by simulation. The results show that the proposed method can effectively reduce request loads to home node in popular object and has acceptable performance in load balancing of object requests in delegates.

Keywords: load balancing, hybrid decentralized web caching, object request.

1. Introduction

Web caching has been widely used to reduce the external bandwidth usage of the organization and to reduce latency of contents downloading of the Internet users. A centralized web caching uses dedicated machines to provide web caching. In a decentralized web caching, clients in an organization share their local web caches to provide web caching to one another. Therefore, the decentralized web caching does not require dedicated machines as the centralized web caching, which requires machines upgrading when the number of clients increases.

Squirrel [1] is one of the decentralized web caching which uses Pastry [2] for routing and object locating. Each client is assigned a node identifier called nodeID which is a 128-bit unique number calculated by hashing its IP address. When the client's web browser wants to download a web content called object in this paper, it has to find a client that it will send an object request to. The web object URL is hashed to by the same hash function given a 128-bit objectID. The client sends the object request to the client with nodeID numerically closest to the objectID using Pastry routing. The client that it sends the object request to is called home node. There are two schemes of Squirrel for object transferring that are home store and directory. In the home store scheme, home nodes store web objects and therefore can transfer requested objects to requesting clients. If home nodes do not have web objects, home nodes download web objects from the Internet. In the directory scheme, home nodes do not store web objects, but they maintain lists of clients that have already downloaded the web object. This list is called directory. The clients in the directory are called delegates. When the client sends a web object request to home node, the home node randomly selects one delegate from directory and forwards the request to that delegate. The delegate transfers the object from their local web caches to the requesting client.

Sheng and Bastani [3] proposed another scheme of Squirrel called hybrid scheme. In this hybrid scheme, the home nodes store web objects and maintain lists of delegates. When the client sends a web object request to the home node, if the home node has the object, the home node transfers the object to the requesting client. If home node does not have the object, it randomly selects a delegate from the directory and let the delegate

+ Corresponding author. Tel.: +6653942023; fax: +663942072.
E-mail address: yuthapong@eng.cmu.ac.th.

transfer web object to the requesting client through the home. The hybrid scheme has a higher hit ratio compared to home node and directory schemes. The hit ratio is the ratio of the number requests that find objects to the number of all requests.

However, in all schemes of Squirrel, the web object requests are always firstly send to home nodes regardless to which clients will transfer the objects. For the web objects with high popularity, home nodes receive a large number of object requests resulting in the hotspot problem. Because clients are users' machines which are not dedicated web caching machines, this might result in a problem of single point failures.

In this paper, we propose a method for the hybrid scheme to reduce load of object requests from home nodes of popular objects and to balance the load of object requests among delegates. The simulation results show that the proposed method can reduce the object request loads in home node of high popular web objects and show acceptable results of load balancing in delegates of high popular web objects.

The rest of paper is organized as follows. Next sections explain related work. Then, the proposed method is explained in Section 3. Section 4 explains about simulation and simulations results. Section 5 concludes this paper.

2. Related Works

There are many researched proposed load balancing in decentralized web cache or similar work [4][5][6]. However, load balancing of object requests for high popular web objects is not yet considered.

Duan and Gu [4] propose a self adaptive load balancing strategy dedicated for web cache cluster. In this approach, the front end of web cache cluster receives all incoming client requests in the system and distributes them to back ends. The front end obtains the load information from back end to balance the load among Web Cache cluster. However, this scheme does not consider the object request loads in front end. The front end has a high load of request exactly because all clients must request the objects through them.

Ananth, et al. [5] propose the load balancing algorithm for peer-to-peer system by uses the concept of virtual server. A virtual server likes a single node in the DHT but each physical node can be responsible for one or more virtual server. Load balancing is achieved by transferring virtual servers from heavily loaded nodes to lightly loaded nodes. However, this scheme has a higher movement cost for transferring between the nodes. It increases internal bandwidth usage of the organization and latency for users.

Godfrey, et al. [6] propose the load balancing that similar a concept of virtual server. Their algorithm considers the system in which data items are continuously inserted and deleted and nodes join and depart the system continuously. However, the request messages of popular object are arrived to responsible node. It has a higher load at single node.

The observation of load balancing algorithm in these proposes that specifically with system architecture and their environment. These researches do not consider the request message balancing. In our knowledge in our literature review, we did not found method for the hybrid scheme to reduce load of object requests from home nodes of popular objects and to balance the load of object requests among delegates.

3. Proposed Method

In the hybrid scheme, home nodes and delegates can transfer requested objects. However, object requests have to be sent to home nodes first. Home nodes of high popular web objects can have a problem of hotspot. With the proposed method, the requesting clients can send object requests to delegates directly instead of sending them to home node. To achieve this, there must be information of available delegates of high popular web objects distributed to clients. The performance of load balancing depends on the method to distribute this information. Clients without information of objects' delegates can still send requests to their home nodes. This section explains details of the proposed method. In section, firstly, the popular directory used in the purposed method is explained. Then, modified method for clients to sends object requests is explained. Finally, how to do load balancing of delegates is explained.

3.1. Popular Directory

We proposed the method to solve the problem that home nodes receive a large number of object requests of high popular web objects. Clients send object requests to home node using Pastry because they do not have other information of where they can get objects. Therefore, we propose the method that provides clients information of delegates of objects so they can send requests to delegates directly. With our proposed method, each client will maintain the new information called popular directory. The popular directory in each client contains objectIDs, popular count, and IP address of delegates for each objectID of high requested objectIDs with distribution count. When the client sends the object request to the home node, the home node selects information from popular directory and sends this information back to the requesting client. This information sent by home node is called popular information. The client updates its popular directory from the received popular information, and it can also distribute the update directory to other clients when it receives object requests. As a result, clients have more information of delegates of popular objects so they can send request directly to delegate instead of home nodes. The method for clients to send object requests is explained in section 3.2. Additionally, the method how home node selects popular information from popular directory is also explained in section 3.3.

The web popularity follows the Zipf-like distribution [7]. The objects in top 1% of popularity rank of objects account for 20% to 35% of number of all requests. Furthermore, the objects in top 10% of popularity rank account for 45% to 55% of number of all requests. Therefore, it is not necessary for popular to store all objectIDs. The optimal number of objectIDs that popular directory should store in the popular directory is an open issue for further research in this paper.

3.2. Web Object Requesting

The method for clients to send object requests is modified in the proposed method to reduce load from object requesting of home nodes. When the client wants to request the web object, it firstly checks from its local cache. If the object is found, the client retrieves the object from its local cache. If the object is not in its local cache, the client checks the popular directory. If the objectID is found in the popular directory, the client sends the object request to the delegate of the objectID. Conversely, if the objectID is not in the popular directory, the clients send requests to the home node of that object. When the home node receives the object request, if it has the object, it can send the object to the requesting client. Conversely, if it does not have the object, it randomly selected a delegate from its directory to forward the request to. It then updates its directory adding this client into the directory. In other words, in the proposed method, only clients requesting objects from home nodes will be added to directory of home nodes. The algorithm of clients sending request is shown in Fig. 1.

3.3. Popular Directory Distribution

Clients with popular directory with object that it wants can sends request to delegates directly. Therefore, load balancing among delegates of each object depends on popular information distributed from clients. In popular directory, each client stores objectID, popular count of the object, and IP addresses of delegates of that objectID with distribute count as previously explained. The popular count stores the number of how many times that the object has been requested from the node. If the popular directory is full, the objectID with smallest popular counts will be removed from the popular directory. The distribute count is used to count of how many times that the delegate is distributed for its objectID.

For the objectID with more than one delegate, in order to balance load from requests, when the home node distributes popular information, it selects only one delegate distributed with popular information. The delegate with smallest distribute count will be selected. If there are more than one delegate with the same smallest distribute count, the client randomly selects one of them. After the delegate is selected, its attribute count is increased by one. When any client, which is not the home node, receives popular information having the same objectID but different delegate, it will update its popular directory by replacing the delegate with the new one from popular information. By using this distribute count, the home node tries to distribute delegates of each object to clients in the organization equally. However, because information of popular directory is also re-distributed to other nodes, the number of delegates of each object in all clients might vary.

```
Requesting Client:
    check local cache;
    if local_found then
        retrieve the object;
    else
        look-up popular directory;
        if delegate_found then
            send request to delegate;
            if remote_hit then
                get object from other node;
            else
                send request to home node;
                if remote_hit then
                    get object from other node;
                else
                    send request to original web server
                    send object to home node;
                end if
            end if
        else
            send request to home node;
            if remote_hit then
                get object from other node;
            else
                send request to original web server
                send object to home node;
            end if
        end if
        cache the object;
        update popular directory;
    end if
```

```
Home node:
    receive a request message;
    check local cache;
    if local_found then
        send remote_hit and the object;
    else
        look-up directory;
        if directory is empty then
            send remote_miss;
        else
            get the object from other node;
            send remote_hit and the object;
        end if
        cache object;
    end if
    update directory;
    select popular objects and their delegate;
    if num_delegate > 1 then
        select delegate with smallest distribute_count
    end if
    update popular directory;
    send popular directory;
```

```
Delegate node:
    receive a request message;
    check local cache;
    if local_found then
        send remote_hit and the object;
    else
        send remote_miss;
    end if
    send popular directory;
```

Fig. 1: Algorithms of requesting clients, home node, and delegate in the proposed method

4. Performance Evaluation

In this section, we use trace-driven simulation implemented by Java to evaluate the performance of the proposed method. We compare the results of the proposed method to the original hybrid scheme. We first compare the number of requests sent to home. Then, we compare the coefficient of standard deviation of number of requests of each object on its. Finally, we compare the number of hit ratios.

4.1. Simulation experiments

The simulation is created under the condition shown in Table 1. The number of access of each object in each rank is generated under Zipf-law distribution with $\alpha = 0.7$. Requests are randomly generated in each client corresponding to simulation parameters in Table 1. The ten sets of simulations are run to evaluate the performance.

Table 1 simulation parameters

Environment Parameter	Value
Number of node	100
Number of object	100000
Number of request	350000
Object size	100KB
Cache size	50MB

4.2. Simulation Results

Figure 2 shows the number of requests received by home node in each object rank of proposed method compared to the hybrid scheme. The hybrid scheme here means the hybrid scheme proposed in the reference [3]. The rank is in order from highest popularity to less popularity from rank 1 to rank 2200 as they are considered to be the popular objects. The result shows that the proposed method can reduce the number of requests from home nodes especially object with high population. The loads from requests in home nodes of popular object can be reduced from 6% to 90%. The objects with very high popularity gets much benefit from the proposed method.

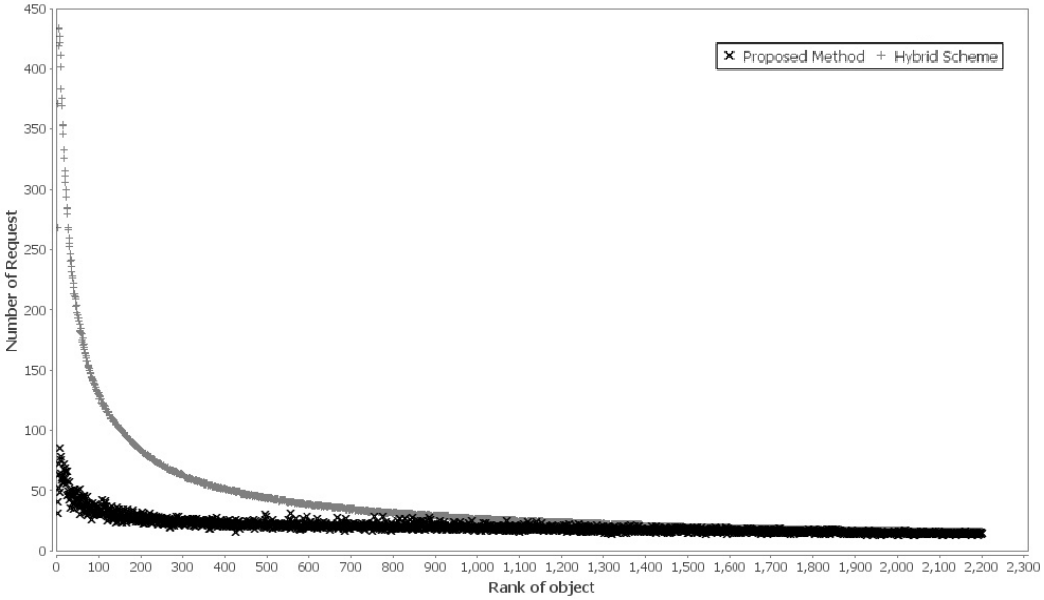

Fig. 2: number of requests received by home node in each object rank

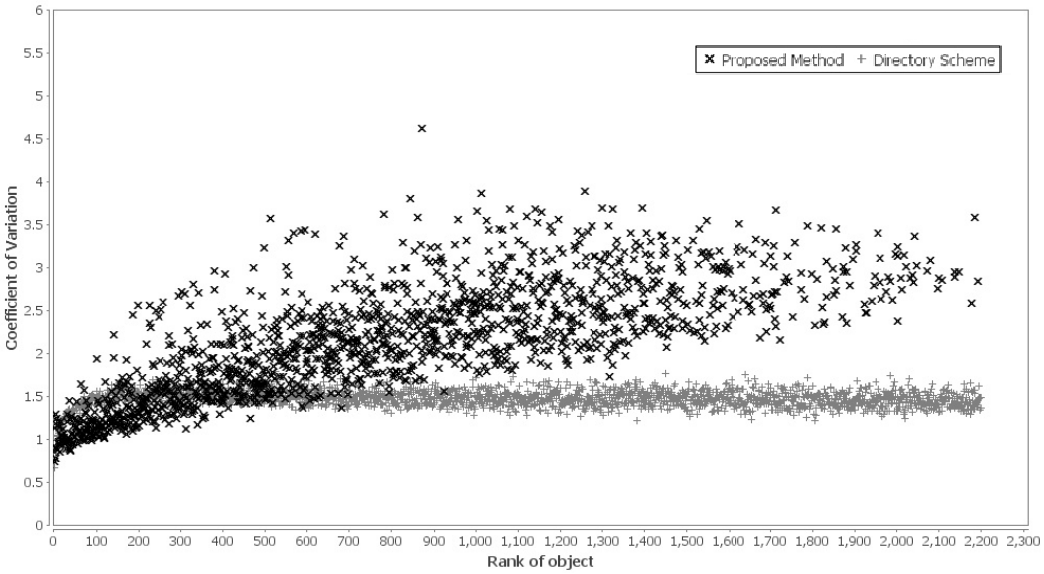

Fig. 3: coefficient of variation values of number of requests sent to delegates of each rank

The performance of load balancing of the proposed method is also evaluated. The results of hybrid scheme are calculated but are not shown in the figure. In hybrid scheme, delegates are rarely used or only few objects are downloaded from delegates. For example, objects with high popularity, home nodes transmit all objects and delegates do not receive any requests at all. This shows the problem that delegates have to store objects but they are not used to transfer objects. This gives problems from both request load and object transferring load to home nodes. Therefore, to evaluate the performance of load balancing, we compare the proposed method to directory scheme instead. Figure 3 shows coefficient of variation values of number of requests sent to delegates of each rank. The figure shows results of the proposed method compared to the directory scheme. The lower value shows the better performance. The clients that are used to be delegates but later remove objects from their caches are also counted as the number of delegates used in this calculation. For objects with very high popularity that is the first 500 ranks, the proposed method has a better performance. The performance degrades with objects with less popularity. This can be explained that in the

proposed method, delegates might be distributed into clients' popular directories unequally because clients can forwards popular information to other nodes as mentioned earlier. This affects the load balancing performance. In directory scheme, home nodes make a decision that requests should be sent which delegates. This decision is considered to be centralization. It should be noted here that the directory scheme and the hybrid scheme are not designed to make clients send request to delegates at the first place like in our method. All requests are firstly sent to home nodes so home nodes can suffer from the request loads.

In the simulations, the hit ratio of the popular objects in proposed method compared to the hit ratio in hybrid scheme is also calculated. The proposed method has hit ratio of 0.904 while the hybrid scheme has hit ratio of 0.915. The number of hits ratio in the proposed method is slightly less than the hybrid scheme. This is because sometimes when delegates replace objects from their caches, they inform home nodes to remove them from directories. However, home nodes cannot update other clients to remove immediately. This results in clients sent request to clients that no long have objects.

5. Conclusion

In this paper, we proposed the method to reduce loads of object requests in home nodes of hybrid decentralized web caching and to balance loads of requests of delegates. In the proposed method, clients use popular directory to store information of delegates of popular objects. Clients can also share this information to other clients by sending popular information. Therefore, clients can send requests directly to delegates instead of home nodes. The proposed method also tries to balance loads of requests sends among delegates for each object. The simulation results show that the proposed method has a very good performance in reducing request loads from home nodes with slightly less hits ratio compared to the hybrid scheme. The proposed method also has an acceptable performance in load balancing compared to the directory scheme, while the hybrid scheme does not use delegates worthily.

In the future research, the optimum size of popular directory should be researched. In addition, the more effective method to distribute information of delegates or information in popular directory should also be studies because it affects the performance of load balancing.

6. Acknowledgements

This research was supported by Chiang Mai University Research Fund and Graduate School, Chiang Mai University, Thailand.

7. References

[1] S. lyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. Proc. of 21st Annual Symposium on Principles of Distributed Computing. Monterey, California, USA, 2002, pp. 213-222.

[2] A. Rowstron, and P. Druschel. Pastry: Scalable decentralized object location and routing for large-scale peer-to-peer systems. Proc. of 18th International Conference on Distributed Systems Platforms. Germany, 2001

[3] B. Sheng, and F. Bastani. Secure and Reliable Decentralized Peer-to-Peer Web cache. Proc. of 18th International Parallel and Destributed Processing Symposium. Texas, Dallas, USA, 2004.

[4] Z. Duan, and Z. GU. Dynamic Load Balancing in Web Cache Cluster. Proc. of 7th International Conference on Grid and Cooperative Computing. Shenzhen, China, 2008.

[5] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P. Proc. of 2nd International Workshop on Peer-to-Peer Systems. Berkeley, CA, USA, 2003.

[6] Systems. Proc. of 23rd Annual Joint Conference of the IEEE Computer and Communications Societies. Hong Kong, China, 2004, pp. 2253-2262.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. Proc. of INFOCOM'99: Conference on Computer Communications. New York, USA, 1999.

Phongsathon Fongta was born March 9th 1988. He received bachelor of computer science from Chiang Mai University, Thailand, in 2009. His research interests are in the areas of Computer Network, Web Caching and World Wide Web Technology. Now he is a Postgraduate student of Chiang Mai University.