IEICE TRANS. INF. & SYST., VOL.E97–D, NO.7 JULY 2014

LETTER

CRRP: Cost-Based Replacement with Random Placement for En-Route Caching*

Sen WANG†, *Nonmember*, **Jun BI**†a), *Member*, *and* **Jianping WU**†, *Nonmember*

SUMMARY Caching is considered widely as an efficient way to reduce access latency and network bandwidth consumption. En-route caching, where caches are associated with routing nodes in the network, is proposed in the context of Web cache to exploit fully the potential of caching. To make sensible replacement and placement decision for en-route caching, traditional caching schemes either engage computation-intensive algorithm like dynamic programming or suffer from inferior performance in terms of average access latency. In this article, we propose a new caching scheme with cost-based replacement and random placement, which is named CRRP. The cost-based replacement of CRRP introduces probing request to timely perceive cost change and the random placement is independent of current caching state, of $O(1)$ computational complexity of placement decision. Through extensive simulations, we show that CRRP outperforms a wide range of caching schemes and is very close to the traditional dynamic-programming-based algorithm, in terms of average access delay.

key words: *Web caching, en-route caching, cache policy*

1. Introduction

Caching has been shown as an efficient way to reduce network bandwidth consumption and propagation latency [3]. The authors of a previous work [5] first tried to provide caches in a hierarchical arrangement in the network in order to significantly reduce the network bandwidth. In such a hierarchical architecture, N. Laoutaris et al. [1] proposed a series of meta algorithms which determine how to place an object along its request path. The proposed LCD (Leave Copy Down) performs very well against the de facto LCE (Leave Copy Everywhere) and others, under all studied scenarios [1]. However, these meta algorithms are independent of object replacement policy, and no cache cooperation was considered.

Going further on along the direction of hierarchical caching, a new caching architecture is developed, which is called en-route web caching [6]. In en-route web caching, each router has been enabled caching capability and is able to cache files passing through it. Requests are routed directly towards the source, and en-route caches check if a copy of the requested file is present at the local cache, and respond directly if found. Tang et al. [4] showed that, in the en-route web caching system, object management could be deduced to a cooperative optimization problem where object replacement is integrated with object placement. A dynamic programming algorithm is provided to solve the optimization problem. The server (or intermediate cache) executes it for each request with the computational complexity of $O(k^2)$ where k is the number of nodes on the request path. Due to the complexity of solving the optimization problem, the applicable scope of this scheme is limited. K. Li et al. [7] extends the previous work of Tang et al. [4] from the linear array along the request path to the tree rooted at the file server. The proposed scheme significantly increases computational complexity since a server needs to compute the optimal placement for the entire tree. However, the performance gain is quite limited. W. Li et al. [3] proposes a more general theoretical model to analyze the data access cost in cooperative caching systems, and provide both a division-based algorithm and a heuristic greedy algorithm to reduce the computational complexity of the dynamic programming algorithm proposed in the previous work [4]. The proposed division-based algorithm is applicable only within some conditions. The performance of the heuristic greedy algorithm is quite inferior to that in the previous work [4].

In this study, in order to lower the computational complexity and maintain high performance as well, we propose a new caching scheme with cost-based replacement and random placement, named CRRP, for en-route caching. The cost-based replacement of CRRP involves probing request to perceive cost change, and the random placement is independent of current caching state, of $O(1)$ computational complexity of placement decision. In the rest of this article, we begin with a description of the system model concerned in the following sections. Section 3 presents the design details of CRRP. In Sect. 4, we evaluate CPPP by extensive simulations on NS3.

2. System Model

For Web caching, the network can be modeled with a undirected graph $G(V, E)$, where V is the set of nodes (routers) enabled with caching capability and E is the links between nodes. Every server or client is attached to a node in V. Requests are issued by clients for Web objects (or content objects) maintained by servers. We assume that each web object is served by exactly one server. Requests are forwarded to servers, along their regular routing paths. Routing paths

Manuscript received November 29, 2013.
Manuscript revised February 25, 2014.

†The authors are with the Institute for Network Sciences and Cyberspace and Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing, 100084 China.

*This research is supported by the National High-tech R&D Program ("863" Program) of China (No.2013AA010605) and the National Science Foundation of China (No.61161140454).

a) E-mail: junbi@tsinghua.edu.cn (Corresponding author)

DOI: 10.1587/transinf.E97.D.1914

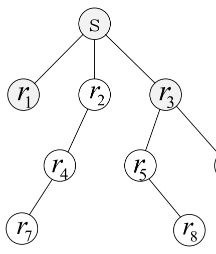

Fig. 1 The spanning tree rooted at a server.

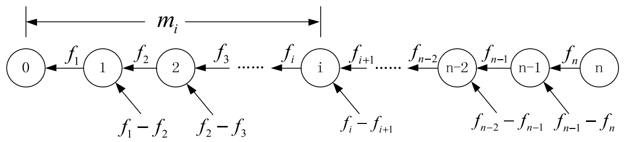

Fig. 2 System model for en-route caching.

from all the nodes to a given server form a tree topology like the topology shown in Fig. 1.

In en-route caching, request is always served by the first node caching the requested object along the routing path. For example, a request issued by a client connected to node r_8 is forwarded along the path $\{r_8, r_5, r_3, s\}$ to the server s. The dark nodes are those nodes which contains the requested object of the request. Therefore, when the request arrives at node r_3, it would be responded with the cached copy in r_3's local cache. We assume the content object goes back to the client along the same path of the request. While the content object goes back to the client, there are two decisions to be made, namely: 1) Which nodes along the path should cache it (object placement problem); 2) If it is needed to evict some objects at a certain node to make space for this object, which objects should be evicted (object replacement problem). Traditionally, these two problems are considered separately. For the first problem, the default LCE, LCD [1] and MODULO [6] are among the proposed options; for the second problem, LRU (Least Recently Used) and LFU (Least Frequently Used) are two alternative object replacement policies.

To solve the two problems for a perspective of optimization, we need to take into account many factors, such as object access frequency and communication costs between nodes. Since the two problems are only about those nodes along request path, the topology concerned is line structure as Fig. 2 shows. In the linear topology, node n is where a request first enters the network, and node 0 is where the requested object is first found along the routing path of the request. For object O, each link is associated with a value $c(e, O)$ which represents the cost of sending a request for object O and the object back on the link e. The cost could be interpreted into such as network latency, bandwidth consumption and so on. In the case of network latency, every node measures the network latencies of all its links. Suppose the network latency of link e is $d(e)$, the cost $c(e, O)$ of object O is set to $2 \cdot d(e)$, considering both the transmissions of the request and object back. In the case of bandwidth consumption, the cost $c(e, O)$ of object O is the size of the object, denoted by $s(o)$, neglecting the bandwidth consumption of request packet. The cost of sending object O along a path is defined simply as the sum of the costs of the links of the path. For example, sending object O from node 0 to node i, the path cost is represented as $m(O) = \sum_{j=0}^{i-1} c(e_{j,j+1}, O)$. The access frequency of object O perceived at node i is denoted as $f_i(O)$. We define a placement P as a subset of the n nodes $\{1, 2, \ldots, n\}$ along the path. The size of the placement P is denoted as k, and the elements are represented as $1 \leq p_1 \leq p_2 \leq \cdots \leq p_k \leq n$. The *cost saving* of caching object O at node i is calculated by the production of the access frequency $f_i(O)$ and the path cost from node 0 to node i which is denoted as $m_i(O)$, namely $f_i(O) \cdot m_i(O)$. Suppose that it is needed to evict those objects $\{O_1, O_2, \ldots, O_j\}$ to make space for caching object O at node i, then the *cost loss* of caching object O is computed as the sum of the cost savings of those objects, namely $\sum_{k=1}^{j} f_i(O_k) \cdot m_i(O_k) \equiv l_i(O)$. Concerning the total *cost saving* of a placement P, it should be noted that these nodes are dependent with each other in a sense that placing an object in one node would result in no request of this object going further to upstream nodes. The total *cost saving* of a placement P is given by $\sum_{j=1}^{k} (f_{p_j}(O) - f_{p_{j+1}}(O)) \cdot m_{p_j}(O)$. The $f_{p_{k+1}}(O)$ is involved and set to be 0 for brevity of exposition. The total *cost loss* is given by $\sum_{j=1}^{k} l_{p_j}(O)$. The goal is to maximize the total *cost gain* $\sum_{j=1}^{k}((f_{p_j}(O) - f_{p_{j+1}}(O)) \cdot m_{p_j}(O) - l_{p_j}(O))$.

3. Cost-Based Replacement with Random Placement

In the previous work of Tang et al. [4], a dynamic programming solution is first introduced to solve the joint object placement and replacement problem formulated in the last section. As mentioned previously, due to the complexity of solving the optimization problem, the applicable scope of this scheme is limited. However, this work provides an insightful and inspirational theorem to our approach, which says that the optimal placement with locations $p_1, p_2, \ldots, p_k$ satisfies the following inequalities:

$$f_{p_j}(O) \cdot m_{p_j}(O) \geq l_{p_j}(O)$$

Interested readers are referred to [4] for its proof. The left part of the inequalities is the *cost saving* at node p_j. The right part is the *cost loss* for evicting other objects at node p_j, in other words, the sum of cost savings of those objects. This theorem can be interpreted into that a node is in the optimal placement only if placing the object in this node is locally beneficial which means the *cost saving* of the object outweighs the *cost loss* concerning the single node.

Based on this insight, we try to reduce the complexity of solving the joint placement and replacement problem by proposing a decoupled solution. We propose a cost-based replacement algorithm with cost-probing mechanism for individual caches. The cost-probing mechanism allows individual caches to timely perceive the cost change of each object. It gives our scheme an advantage in comparision with the dynamic programming solution [4]. For the dynamic programming solution [4], once a decision is made, there is no mechanism to perceive the changes of the conditions which result in the decision. For example, an eviction of upper placement will not trigger reevaluations of lower placements. In term of placement problem, we compare a series of low-complexity placement algorithms and find that the random placement works best with the aforementioned cost-based replacement algorithm. We name the combination of the cost-based replacement and random placement CRRP. In the following subsections, we present respectively the two essential parts of CRRP, namely the cost-based replacement and random placement.

3.1 Cost-Based Replacement

To decide which objects to replace for a newly-coming object, we adopt a similar replacement algorithm like the one first proposed in the context of single Web caching [2]. It is based on access frequency and sensitive to the change of access frequency which is crucial to counter the inter-impact between replacements of different caches. The function $\frac{f(O)\cdot m(O)}{s(O)}$ ($s(O)$ is the size of the object) is used as the only criterion to select the candidates to evict. Specifically, the proposed replacement algorithm greedily selects the objects with the smallest $\frac{f(O)\cdot m(O)}{s(O)}$ to evict until there is enough space for a newly-coming object. In order to assessing the access frequency of an object, we use the same sliding-window approach proposed in [2] which can timely perceive the change of access frequency. Specifically, for each object O, up to N most recent reference times are recorded, and the access frequency is calculated by $f(O) = \frac{N}{t-t_N}$ where t is the current time, and t_N is the Nth most recently referenced time. In our simulations, N is set to 3 as suggested in the previous work of Scheuermann et al. [2].

As mentioned earlier, placements at different nodes affect each other in terms of *costing saving*. For example, in Fig. 2, caching object at node i could reduce the access frequency perceived by upstream nodes (e.g., node 1). In the meanwhile, it reduce the *costing saving* at downstream nodes (e.g., node n) due to the reduction of $m(O)$. In order to make replacement decision locally, it is crucial to timely perceive the changes of $m(O)$ and $f(O)$. At each entering point of request, say node n in Fig. 2, we mark a request as a probe request with a probability of q to perceiving the change of $m(O)$ of nodes all along to the content server. A marked request will go further to the content server even if there are some intermediate nodes having the requested object. When the object travels back, it would not be placed at any node. Instead it would record the $m(O)$ from the last node which has the object and inform nodes on its way the current $m(O)$. Specifically, a variable is associated with the object and set to 0 initially when it is first generated at content server. As the object travels through a link, the variable is added by the cost of the link. When it arrives at a node, the node uses the variable as the *cost saving* of the object. If the node has the object cached locally, the variable is set to 0 before leaving.

3.2 Random Placement Strategy

We propose a random placement strategy to work with our cost-based replacement policy. This placement strategy is independent of current caching state, of $O(1)$ computational complexity and easy to implement. In term of performance, an advantage of the random placement is that it gives every node along the request path the same caching opportunity in a less aggressive way than those of other schemes such as LCE. As described before, this placement strategy may not result in an optimal solution. We resort to our local cost-based replacement policy with cost-probing mechanism to correct the suboptimal placement decision made by the random placement strategy.

The proposed random placement strategy works as follows. Along the path back to the requester, one node is randomly selected as the place to cache the object. A variable with an initial value of 0, called *HopCount*, is attached to each request to record the hop number the request has passed. When the request is responded by the content server or an intermediate router, a random number within the range [1, *HopCount*] is selected and embedded in the response packet to indicate which node should cache the object along the path back. As the requested object goes back, the hop number it has passed is recorded. When it arrives at the indicated node, it is cached there. Besides, like the probing process, a variable is associated with the object to record the $m(O)$ from the starting node in order to inform the indicated node. This random placement strategy has two advantages. The first is that it gives the same opportunity to each cache on the path to test the eligibility of the object via competition on *cost saving*. The second is that it is more conservative than other strategies such as LCE since only one node is selected. This placement strategy together with the proposed cost-based replacement policy is named CRRP.

We also try two other placement strategies to compare with CRRP. The first placement strategy is the LCE. As the requested object O goes back to the requester, the object is inserted into every cache it traverses. The *costing saving* $m(O)$ is set to the cost of the last-hop link, since the last-hop upstream node has cached the same object. We named this strategy as CR-LCE. The second placement strategy is the placement strategy proposed in the previous work of Bhattacharjee et al. [6] which is a simple placement optimization to LCE. On the path from the cache (or server) to the requester, the object is cached at the nodes that are a fixed number (called cache radius) of hops apart. Thus, an object ends up being distributed in concentric "rings" centered on the content server where it resides. The cache radius is set to 3 in our simulations like that in the previous work [6]. We name this strategy CR-MOD.

4. Evaluation

Besides CR-LCE and CR-MOD, we compare by simulation the proposed CRRP with existing other schemes, such as

Table 1 Parameters of simulations.

Parameter	Quantity
Total number of nodes	240
Ratio of WAN nodes and MAN nodes	1:1.4
Number of network links	344
Number of content servers	140
Number of clients	140
Average link latency between WAN nodes	0.79 second
Average link latency between MAN nodes	0.12 second
Number of objects	1000 object per provider
Access frequency distribution	Provider: $1/i^{\alpha}, \alpha = 0.8$ Object: $1/i^{\beta}, \beta = 0.8$

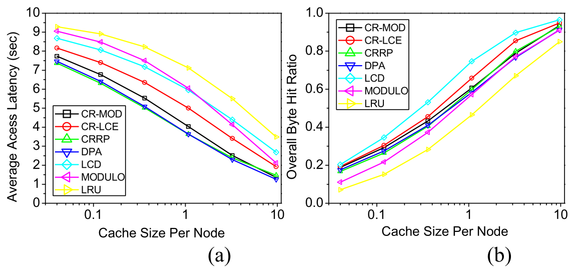

Fig. 3 Cache size vs. average access latency and overall byte hit ratio.

LRU (with default LCE), LCD [1], MODULO [6] and the dynamic programming algorithm proposed in [4] that is referred as DPA in following figures. We use a network topology randomly generated by the Tiers program in our simulations. In the topology, each MAN node is associated with one client and one content server. Some parameters of our simulations are listed in Table 1.

The object size distribution follows a uniform distribution ranging from 5KB to 10MB. The arrival of request follows the Poisson Process, and the distribution of average arrival intervals for clients follows a uniform distribution from 0.1 second to 0.9 second. The access frequencies of the content servers as well as the objects maintained by a given server both follow a Zipf-like distribution, like that in [4]. Specifically, the probability of a request for object O in provider s is proportional to $1/(i^{\alpha} \cdot j^{\beta})$ where s is the ith most popular server and O is the jth most popular object in s. We empirically set the probing probability p to 0.005.

In Fig. 3, the horizontal axis is the relative cache size per node that is the percentage between the cache size per node and the size of the total content population. As Fig. 3 (a) shows, CRRP is very close to the DPA in most cases in terms of the average access latency, and even outperforms slightly DPA when cache size is small. It outperforms others significantly in most cases. As Fig. 3 (b) shows, CRRP is close to DPA but underperform LCD significantly in terms of the overall byte hit ratio. As our results show, LCD is the best in terms of the overall byte hit ratio but underperform significantly CRRP in terms of the average access latency because it does not take into account the cost factor.

To investigate the impact of the skewness parameter α of Zipf distribution, we vary it from 0.2 to 1.4 and fix the relative cache size to 0.36. Figure 4 shows the consequential results. As Fig. 4 (a) shows, the differences between various caching schemes increase, as parameter α rises. In all the cases, CRRP is close to DPA and outperforms others. Interestingly, the overall byte hit ratio does the opposite as shown in Fig. 4 (b). The differences among various caching schemes decrease as parameter α rises. In terms of the overall byte hit ratio, CRRP outperforms DPA when α is small, and LCD does the best.

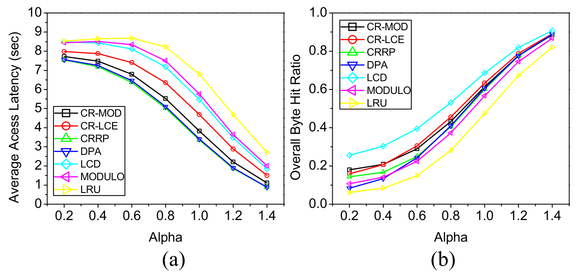

Fig. 4 Impact of parameter alpha.

5. Conclusion

In this article, we propose CRRP for en-route caching, which randomly places object along the request path and integrates cost-based replacement policy with cost probing. Extensive simulations show that CRRP outperforms a wide range of caching schemes and is very close to the conventional dynamic-programming-based algorithm, in terms of average access delay. As a future work, we will conduct trace-driven simulation to evaluate CRRP further.

References

[1] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," IEEE International Conference on Performance, Computing, and Communications, pp.445–452, 2004.

[2] P. Scheuermann, J. Shim, and R. Vingralek, "A case for delay-conscious caching of web documents," Computer Networks and ISDN Systems, vol.29, no.8-13, pp.997–1005, Sept. 1997.

[3] W. Li, E. Chan, G. Feng, D. Chen, and S. Lu, "Analysis and performance study for coordinated hierarchical cache placement strategies," Comput. Commun., vol.33, no.15, pp.1834–1842, 2010.

[4] X. Tang and S.T. Chanson, "Coordinated en-route web caching," IEEE Trans. Comput., vol.51, no.6 pp.595–607, 2002.

[5] P.B. Danzig, R.S. Hall, and M.F. Schwartz, "A case for caching file objects inside internetworks," SIGCOMM Computer Communication Review, vol.23, no.4, pp.239–248, 1993.

[6] S. Bhattacharjee, K.L. Calvert, and E.W. Zegura, "Self-organizing wide-area network caches," Proc. INFOCOM '98, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, vol.2, pp.600–608, 1998.

[7] K. Li, H. Shen, F.Y. Chin, and S.Q. Zheng, "Optimal methods for coordinated enroute web caching for tree networks," ACM Trans. Internet Technology (TOIT), vol.5, no.3, pp.480–507, 2005.