EAI Endorsed Transactions
on Scalable Information Systems

Research Article ICST.ORG

Cooperative Data Caching for Cloud Data Servers

Mingcong Yang[1,*], Kai Guo[1], Yongbing Zhang[1]

[1]Graduate School of Systems and Information Engineering, University of Tsukuba, Japan

Abstract

Thanks to the advance of cloud computing technologies, users can access the data stored at cloud data centers at any time and from any where. However, the data centers are usually sparsely distributed over the Internet and are far away from end users. In this paper, we consider to construct a cache network by a large number of cache nodes close to the end users in order to minimize the data access delay. We firstly formulate the problem of placing the replicas of data items to cache nodes as a mixed integer programming (MIP) problem. Then, we proposed an efficient heuristic algorithm that allocates at least one replica of each data item in the cache network and attempt to allocate more data items so as to minimize the total data access cost. The simulation results show that our proposed algorithm behaves much better than a well-known LRU algorithm and the computation complexity is limited.

Received on 29 June 2016; accepted on 15 July 2016; published on 09 August 2016

Keywords: cloud data centers, data allocation, mixed integer programming

doi:10.4108/eai.9-8-2016.151632

1. Introduction

In recent years, the rapid development of Internet technologies has brought many new challenges to data services across the Internet. Many data-sensitive applications such as those for online video distribution and data sharing require increasingly transmission speed and wide bandwidth. Furthermore, most of or a large percentage of users located in the same or nearby areas may require the same kinds of data and the data request patterns may not change frequently. In order to reduce the data access latency and save the scarce network bandwidth, allocating the replicas of the required data at the locations close to end users is an efficient approach. However, when using a single cache node for multiple users like web cache servers [2, 8], the cache node may be overloaded and its service becomes unavailable if the cache node fails. In addition, if a data item required by a user is not stored at the cache node the user has to obtain the data item from the data center directly, resulting in a long access time and extra data traffic over the network [12]. Therefore, other researchers proposed a cooperative caching approach [4] in which a number of caching nodes are interconnected and all the caching nodes work together to provide better data services to users. Compared with the single node caching, the cooperative caching approach can reduce the processing time of data requests, improve the cache hit rates, and provide higher fault tolerance. In a cooperative caching approach, each node makes its caching decisions using the information of data access and data caching in the network, yielding better overall data access performance [10, 16].

The main disadvantage of previous cooperative caching approaches is that a node has to know the network topology and also the data cache information and therefore when the network size is large the overhead for exchanging the data cache information should be significantly high. However, when a node makes a caching decision it takes into account of only its surrounding nodes. When a data miss occurs a node has to obtain the data from a node far away or in the worst case from the data center. In this paper, we propose an overall caching approach in which data caching decisions are made by using the information of the whole nodes in the network. We formulate the data allocation problem as a mixed integer programming (MIP) problem. When the problem size becomes large,

★Please ensure that you use the most up to date class file, available from EAI at http://doc.eai.eu/publications/transactions/latex/

*Corresponding author. Email: s1520548@sk.tsukuba.ac.jp

the problem is NP-hard, and therefore we propose an efficient heuristic algorithm. In order to evaluate our proposed heuristic algorithm, we use a well-known caching approach, called least recently used (LRU), and a simple caching algorithm in which copies of data items are randomly allocated to the nodes.

The remainder of this paper is organized as follows. In Section 2, we summarize the related researches on caching technologies. In Section 3, we describe the network model considered in this paper and formulate the data replica allocation problem as a MIP problem and show it is NP-hard. In Section 4, we describe our proposed heuristic algorithm in detail. The results of our simulation experiments are given in Section 5. Finally, in Section 6, we conclude the paper briefly.

2. Related Work

Many researchers focused on various cooperative caching techniques [4, 6, 15, 16, 18, 19]. Chankhunthod *et al*. [4] proposed a hierarchical caching mechanism for web applications in which lower cache servers can resolve misses through a higher level cache servers. Researchers also proposed cooperative caching protocols such as internet cache protocol (ICP) [19], cache array routing protocol (CARP) [18], and summary cache [6]. Cooperative caching approaches can be classified into hierarchical cooperative caching and distributed cooperative caching[15, 16].

Ramaswamy *et al*. [14] proposed a cache deployment scheme based on the *expiration age* which is defined as the time difference that an object be replaced out and be cached depending on the statistical information. It shows the time difference between a document that is removed and the document that is accessed since it has been accessed last time; or the time which is the time difference between an document is removed and an document entering a cache divided by the number of hits. When a node receives a document, it determines if the document should be cached through exchanging the expiration age information with other nodes.

Bhattachrjee *et al*. [3] proposed a cache deployment scheme for self-organizing wide-area network. The authors proposed the concept of *caching radius*, which is defined by the number of hops between nodes. When a message called *reply message* is transferred between two nodes, the number of hops will gradually increase when a node receives the reply message. If the number of hops of the reply message is an integer multiple of it's caching radius, the node will store this object. however, this paper does not consider the impact of data's popularity.

Li *et al*. [10] proposed a local optimization cache deployment scheme using a modified Dijkstra algorithm to obtain an optimal caching path. Eum.S *et al*. [5] and Liu *et al*. [11] proposed a randomly cache selection approach and a dynamic linear programming approach in which the information of nodes such as node status and the demand patterns are taken into account when deciding the cache locations on the path from a cache node to the data source.

Ming *et al*. [13] proposed a cache deployment scheme based on the *age* which is defined as the distance to the server and its popularity. The further a content is from a server or the more popular the content is, the longer age it has. A data item with a longer age is to be kept longer in the cache. This paper did an overall optimization for the data distribution of all nodes on the paths from the requesting node to the data center.

The core idea of those approaches is to replace the data items according to data requests dynamically in order to improve the cache hit rate and reduce the data access delay. However, there are two important factors that are not taken into account in those approaches. One is that each node needs to maintain a *cached data table*, which contains the location information of the cached data items in the network, and the cached data table should be updated frequently. This greatly increases the computational overhead of nodes[14]. Another is that the decision of whether to cache a data item at a node is made using only the information of the transmission path from the node to the original data source, say, data center, instead of the neighboring nodes around the node[5, 10, 11, 13].

Cloud data centers generally store all the data provided to users but are usually located in locations far away from end users. On the other hand, the users residing in the same or nearby areas may access the same kinds of data frequently and furthermore the data request patterns may not change frequently. Therefore, it is a good way to place the replicas of data frequently accessed by users at locations near to the users in order to reduce the data access delay and also to reduce the workload at the data center and save network bandwidth. Due to the storage limitation of a cache node, it is beneficial to construct a cache network by a number of cache nodes so that a user can easily find the required data at a nearby cache node. This paper focuses on how to allocate the replicas of data items that are originally stored at the data centers in order to minimize the total cost for data access.

3. Model and Problem Formulation

In this section, we first introduce the network model and describe the assumptions considered in this paper. Then we formulate the data allocation problem as a mixed integer programming (MIP) problem.

3.1. NetworkModel

The network model considered in this paper is shown in Figure 1 where the set of cache nodes and the

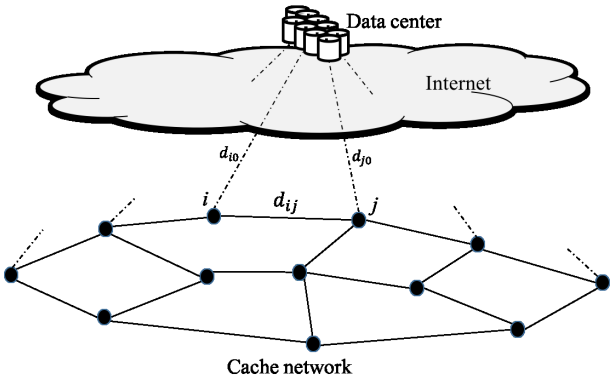

Figure 1. Data cache network.

data center are denoted by $N = \{n_1, n_2, \ldots, n_{|N|}\}$ and n_0, respectively. We let $N' = N \cup n_0$. The shortest distance between nodes i and j, which can be determined using an appropriate shortest path algorithm, is denoted by d_{ij}, and we assume that the data transmission between nodes i and j passes through the shortest path between nodes i and j. We assume that the distance from a cache node to the data center is much longer than the distance between any two cache nodes, i.e., $d_{i0} \gg d_{ij}$ $(i, j \in N)$. The set of data items stored at data center n_0 is denoted by $M = \{d_1, d_2, \ldots, d_{|M|}\}$. The demand of data item k arrival at node i is denoted by ϕ_i^k and the set of the total demands at node i is denoted by $\Phi_i = \{\phi_i^1, \phi_i^2, \ldots, \phi_i^{|M|}\}$. The size of data item k is denoted by s_k and the cache capacity of node i is denoted by A_i. We assume that $\sum_{i \in N} A_i \geq \sum_{k \in M} s_k$.The cost for data transmission per unit data per unit distance is denoted by ξ which is a positive constant value. We assume that the cost for transferring a data item from a node to another node depends only on the size of the data item and the distance between the two nodes.

We define the cost that node i requests data item s_k at node j as follows.

$$c_{ij}^k = s_k d_{ij} \phi_i^k \xi, \quad \forall i \in N, j \in N'. \tag{1}$$

We also define the caching decision variables for nodes j $(j \in N)$ by Λ_j such that

$$\Lambda_j = \{\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^{|M|}\}, \tag{2}$$

where

$$\lambda_j^k = \begin{cases} 1, & \text{if node } j \text{ caches data item } k, \\ 0, & \text{otherwise.} \end{cases}$$

We assume that a user accesses data item k from one and only one nearest cache node that keeps the replica of data time k and a variable r_{ij}^k is used to show the decision. If node i accesses data item k stored at node j, $r_{ij}^k = 1$ and $r_{ij}^k = 0$ otherwise. For node i, if there are more than one nearest nodes keeping data item k, node i can choose any node to obtain data item k. If data item k is not cached at any node, a node requesting data item k should obtain data item k from the data center. The notation used in this paper is shown in Table 1.

Table 1. Notationused in the paper

N	set of cache nodes in the network
N'	set of nodes including data center in the network
d_{ij}	shortest distance between nodes i and j
M	set of data items
s_k	size of data item k
ϕ_i^k	demand of data item k of node i
Φ_i	total demand of node i
A_i	cache capacity of node i
ξ	cost for data transmission per unit data per unit distance
c_{ij}^k	cost for requesting data item k from node i to node j
λ_j^k	caching decision variable, $\lambda_j^k = \begin{cases} 1, & \text{if node } j \text{ caches data item } k, \\ 0, & \text{otherwise} \end{cases}$
Λ_j	caching decision vector of node j, i.e., $\Lambda_j = \{\lambda_j^1, \lambda_j^2, \ldots, \lambda_j^{\lvert M\rvert}\}$
r_{ij}^k	accessing decision variable, $r_{ij}^k = \begin{cases} 1, & \text{if node } i \text{ accesses data item } k \text{ stored at node } j, \\ 0, & \text{otherwise} \end{cases}$

3.2. ProblemFormulation

Our objective in this paper is to allocate the replicas of data items to cache nodes so as to minimize the total cost for data access. We formulate the data allocation problem (DAP) as a mixed integer programming (MIP) problem as follows. Therefore, in this paper we propose a heuristic algorithm to solve the problem.

$$\min \quad \sum_{i\in N}\sum_{j\in N'}\sum_{k\in M} c_{ij}^k \lambda_j^k r_{ij}^k = \sum_{i\in N}\sum_{j\in N'}\sum_{k\in M} s_k d_{ij} \phi_i^k \lambda_j^k r_{ij}^k \xi \quad (3)$$

subject to

$$\sum_{j\in N'} r_{ij}^k = 1, i \in N, k \in M, \quad (4)$$

$$\lambda_j^k \geq r_{ij}^k, i, j \in N, k \in M, \quad (5)$$

$$\sum_{k\in M} s_k \lambda_j^k \leq A_j, j \in N, \quad (6)$$

$$\lambda_j^k \in \{0,1\}, j \in N, k \in M, \quad (7)$$

$$r_{ij}^k \in \{0,1\}, i \in N, j \in N', k \in M. \quad (8)$$

Constraint (4) means that node i retrieves the data from exactly one location, either a cache node or the data center. Constraint (5) indicates that the data request of node i cannot be served by node j unless the data replica is cached at node j. Constraint (6) shows that the total size of data replicas cached at node j can not exceed its cache capacity. Furthermore, constraints (7) and (8) are the integrality constraints on the decision variables. It is well-known [7, 9] that an MIP problem like problem (3) is NP-hard, so when the number of nodes become large, it will be difficult to calculate. Therefore, in this paper we propose a heuristic algorithm to solve the problem.

4. Proposed Heuristic Algorithm

In our proposed algorithm, we initially allocate each data item $k \in M$ to the cache nodes without considering the node capacity constraints. Then, we check whether there are any nodes exceeding their node capacities and reallocate the data copies to other nodes. Finally, we attempt to additionally allocate more data items to cache nodes that still have available caches.

4.1. Initial data item allocation

We define the cost of all the nodes accessing data item k cached at node j as follows.

$$l_j^k = \sum_{i\in N} c_{ij}^k, i \in N. \quad (9)$$

Then, we order the nodes according to l_j^k $(j \in N)$ in the ascending order. That is, we let

$$\Theta^k = \{j_1^k, j_2^k, \ldots, j_{|N|}^k\}, \quad k \in M,$$
$$\text{such that } l_{j_1^k}^k \leq l_{j_2^k}^k \leq \ldots \leq l_{j_{|N|}^k}^k.$$

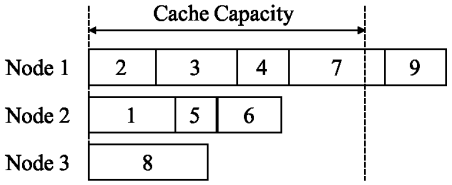

Figure 2. Data item allocation without considering node capacities.

Data item k is allocated to node j_1^k that is the head node on list Θ^k $(k \in M)$. We let G_j denote the set of data items allocated to node j and the total size of those data items may exceed the capacity of node j.

Here, we define $\Theta^k[x]$ to represent the x-th item in Θ^k and therefore the copy of each data item k is allocated to $\Theta^k[1]$. An example is shown as in Figure 2 where there are three cache nodes, denoted by 1, 2, and 3, and nine data items, denoted by $1, 2, \ldots, 9$. In this example, $G_1 = \{2, 3, 4, 7, 9\}$, $G_2 = \{1, 5, 6\}$, and $G_3 = \{8\}$. Note that some nodes may violate the node capacities, e.g., the total size of data items in G_1 exceeds the capacity of node 1.

4.2. Data item reallocation

For node j which capacity constraint is violated, we calculate the cost difference, denoted by D_k, for the allocations of data item k $(k \in G_j)$ between nodes $\Theta^k[1]$ and $\Theta^k[2]$ as follows:

$$D_k = l_{j_1^k}^k - l_{j_2^k}^k. \quad (10)$$

We order the data items in G_j by D_k in the descending order and let G_{j_sorted} denote the new set keeping the descending order. Here, we also define $G_{j\_sorted}[x]$ to represent the x-th data item in G_{j_sorted}. We will not move the data items at the head of G_{j_sorted} at node j until the node capacity is violated but attempt to move the data items at the tail to other nodes that still have available cache spaces. The data reallocation algorithm is described in detail in Algorithm 1. Since Algorithm 1 is activated only for the nodes, denoted by the set N_v, which capacity constraints are violated, therefore we should have $|N_v| \ll |N|$. The computation complexity of Algorithm 1 is governed by the sorting for l_j^k $(j \in N)$ and the data reallocation for nodes violating capacity constraints, and therefore should be bound by $O(MN \log N + M^2 \log M)$ in the worst case.

4.3. Additional data item allocation

We attempt to allocate data items as many as possible in order to minimize the total cost for data access. We

Algorithm 1 Data item reallocation algorithm.

Input: G_{j_sorted} and A_j for $j \in N$, Θ^k and s_k for $k \in M$
Output: $G_{j\_adjusted}$ $(j \in N)$ which is the set of data items allocated to nodes
$f = 1$
while $f \neq 0$ **do**
$f = 0$
for each $j \in N$ **do**
while $\sum_{k \in G_{j\_sorted}} s_k > A_j$ **do**
$k_{last} = G_{j\_sorted}[\|G_{j\_sorted}\|]$
move data item k_{last} from node j to it's suboptimal caching node based on $\Theta^{k_{last}}[2]$.
delete the first item $\Theta^{k_{last}}[1]$ in list $\Theta^{k_{last}}$.
end while
end for
for each $j \in N$ **do**
if $\sum_{k \in G_{j\_sorted}} s_k > A_j$ **then**
$f = 1$
end if
end for
end while
for each $j \in N$ **do**
$G_{j\_adjusted} = G_{j\_sorted}$
end for

order $\frac{l_j^k}{\sum_{i \in N} \phi_i^k s_k}$ $(k \in M)$ in the ascending order; that is,

$$\eta_j = \{k_j^1, k_j^2, \ldots, k_j^{|M|}\} \text{ such that } \frac{l_j^{k_j^1}}{\sum_{i \in N} \phi_i^{k_j^1} s_{k_j^1}} \leq \frac{l_j^{k_j^2}}{\sum_{i \in N} \phi_i^{k_j^2} s_{k_j^2}} \leq \ldots \leq \frac{l_j^{k_j^{|M|}}}{\sum_{i \in N} \phi_i^{k_j^{|M|}} s_{k_j^{|M|}}}.$$

We see that the data item at the head of η_j yields the least cost for data access and therefore we attempt to allocate additionally the data items at the head to node j if the data items are not cached. Here, we use $\eta_j[x]$ to denote the x-th data item in η^j. The additional data item allocation algorithm is shown in Algorithm 2. Since the data allocation at each node is performed independently, therefore the computation complexity of Algorithm 2 is $O(NM \log M)$. Therefore, the total computation complexity of the proposed algorithm in the worst case is $O(MN \log N + M^2 \log M + MN \log M)$.

5. Simulation Experiments

In this section, we describe the performance evaluation of our proposed data allocation algorithm by simulation experiments. The simulation experiment was done

Algorithm 2 Additional data item allocation algorithm.

Input: $G_{j\_adjusted}$, A_j and η_j for $j \in N$, s_k for $k \in M$
Output: G_{j_result} which is the set of the resulting data items allocated to nodes $j \in N$
for each $j \in N$ **do**
for each $k^* \in \eta_j$ **do**
if $\sum_{k \in G_{j\_adjusted}} s_k < A_j$ **then**
if $k^* \notin G_{j\_adjusted}$ and $\sum_{k \in G_{i\_adjusted}} s_k + s_{k^*} \leq A_j$ **then**
append data item k^* in list $G_{j\_adjusted}$
end if
end if
end for
end for
for each $j \in N$ **do**
$G_{j\_result} = G_{j\_adjusted}$
end for

under the environment as shown in Table 2. We compare our proposed algorithm with a cooperative least recently used (LRU) algorithm that is extended based on the well-known LRU algorithm and a random algorithm in which data items are randomly allocated to cache nodes. In the cooperative LRU algorithm, each node determines its only caching decisions independently and when a cache miss occurs the node determine whether to obtain the required data from the data center or from a neighboring node.

Table 2. Simulationenvironment

Operating system	Windows 10
Programming language	Python 3.2
Processor Type	Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz[4]
Memory size	200MB
Percentage distribution of resources	30 percent

In the simulation experiments, we considered a cache network with 20 nodes and the distance of a node pairs is chosen randomly between [1, 100] and the distances the caching nodes to the data center are all set to be 1000. The number of data items is fixed to 200. The demand for each data item is generated randomly to [1, 50].For the sake of simplicity, we assumed that all the data items have the same size in the simulation experiments.

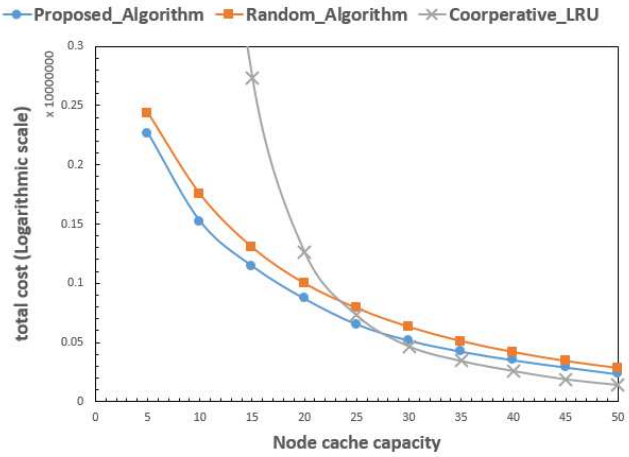

Figure 3. Effect of node capacity.

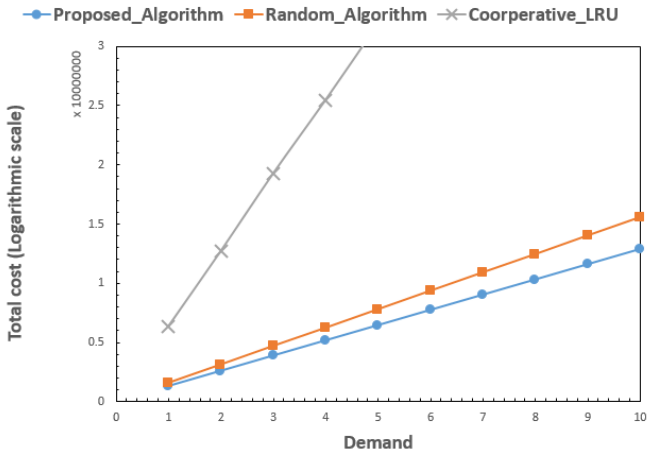

Figure 4. Effect of data demand.

5.1. Effect of node capacity

Figure 3 shows the total cost for data allocation of our proposed algorithm compared with the random data allocation algorithm and the cooperative LRU algorithm when changing the capacity of node cache. We see from this figure that our proposed algorithm performs much better than random algorithm and the LRU algorithm especially when the node capacity of the nodes is small. As we mentioned in Section 1, the capacity of a cache node is generally much smaller than that of the data center and therefore it is crucial to take the cooperation of nearby nodes into consideration in caching decisions. The LRU algorithm behaves the worst since the cache hit misses occurs frequently and to cache a new data item, a old cached data item has to be removed from the cache. The random algorithm performs worse than our proposed algorithm since it determines the data allocation without considering the demands of users.

5.2. Effect of data demand

Figure 4 shows the simulation results when multiplying the data demand of each node from 1 to 10 times that in the results shown in Figure 3 while the node capacity is fixed to be 20, that is, one tenth of the whole data items. We see from Figure 4 that when the data demand becomes large, the performance of the LRU degrades very fast and that our proposed algorithm behaves better than others. We also see that when the data demand becomes larger the superiority of our proposed algorithm over the random algorithm is more significant.

5.3. Effect of network size

Figure 5 shows the performance comparison of our proposed algorithm with the LRU and the random

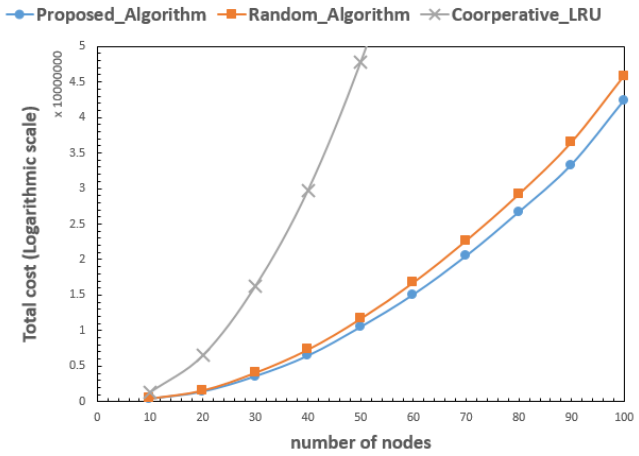

Figure 5. The change of number of nodes and data items.

algorithms. The capacity of each node is fixed to be 10, while the number of cache nodes was changed from 10 to 100. The number of data items was changed from 50 to 500 corresponding to the number of nodes, that is, on average two copies of each data item can be cached in the network. From Figure 5, we can see that when the network size becomes larger, our algorithm performs better than other two algorithms. Not surprisingly, the cooperative LRU algorithm behaves worse than others. The computation times of our proposed algorithm was 1969 seconds when the number of cache nodes was 100 and the number of data items was 500.

6. Conclusion

In this paper, we considered the problem of how to allocate the replicas of data items that are originally stored at the cloud data center to cache nodes that are deployed nearby end users in order to minimize the total data access cost. We first formulate the data

allocation problem as an MIP problem and showed the problem is NP-hard. Then, we proposed a heuristic algorithm that first allocates at least one copy of each data item in the cache network and then attempts to allocates more data items until the node capacity constraints are violated. We evaluated our proposed algorithm in comparison with a previous well-known LRU algorithm and a simple random allocation algorithm and showed that our proposed algorithm performs much better than the other algorithms over a wide range of system parameters.

Recently, many researchers have been attracted by a new Internet infrastructure called the information-centric networks [1, 17, 20] where in edge routers are equipped with disk storages to store data frequently accessed by nearby end users. Thus we can construct a cache network using those edge routers and allocate the replicas of data using our proposed algorithm in order to minimize the data access delay. As shown in Section 4.3, the computation complexity of our proposed algorithm is limited and therefore our approach should be useful even in a large-scale network.

References

[1] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[2] Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating content management techniques for web proxy caches. *ACM Performance Evaluation Review*, 27(4):3–11, 2000.

[3] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Self-organizing wide-area network caches. In *Proceedings in Conference on Computer and Communications (INFOCOM'98)*, volume 2, pages 600–608. IEEE, 1998.

[4] Anawat Chankhunthod, Peter B. Danzig, Chunk Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proceedings of USENIX Annual Technical Conference (ATEC'96)*, pages 3–13. ACM, 1996.

[5] Suyong Eum, Kiyohide Nakauchi, Masayuki Murata, Yozo Shoji, and Nozomu Nishinaga. Catt: potential based routing with content caching for icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 49–54. ACM, 2012.

[6] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3):281–293, 2000.

[7] Marshall L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12):1861–1871, 2004.

[8] Nikolaos Laoutaris, Sofia Syntila, and Ioannis Stavrakakis. Meta algorithms for hierarchical web caches. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 445–452. IEEE, 2004.

[9] Claude Lemarechal. *Lagrangian relaxation*, pages 112–156. Eds. M. Junger and D. Naddef, LNCS 2241. Springer-Verlag, Berlin Heidelberg, 2001.

[10] Wen-Zhong Li, Dao-Xu Chen, and Sang-Lu Lu. Graph-based optimal cache deployment algorithm for distributed caching systems. *Journal of Software (in Chinese)*, 21(7):1524–1535, 2010.

[11] Waixi Liu, ShunZheng Xu, Jun Cai, and Ying Gao. Scheme for cooperative caching in icn. *Journal of Software (in Chinese)*, 24(8), 2013.

[12] Jiangang Ma, Le Sun, Hua Wang, Yanchun Zhang, and Uwe Aickelin. Supervised anomaly detection in uncertain pseudoperiodic data streams. *ACM Transactions on Internet Technology (TOIT)*, 16(1):4, 2016.

[13] Zhongxing Ming, Mingwei Xu, and Dan Wang. Age-based cooperative caching in information-centric networking. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE, 2014.

[14] Lakshmish Ramaswamy and Ling Liu. A new document placement scheme for cooperative caching on the internet. In *Proceedings in International Conference on Distributed Computing Systems (ICDCS2002)*, pages 95–103. IEEE, 2002.

[15] Pablo Rodriguez, Christian Spanner, and Ernst W Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *Networking, IEEE/ACM Transactions on*, 9(4):404–418, 2001.

[16] Renu Tewari, Michael Dahlin, Harrick M Vin, and Jonathan S Kay. Design considerations for distributed caching on the internet. In *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*, pages 273–284. IEEE, 1999.

[17] Gareth Tyson, Nishanth Sastry, Ivica Rimac, Ruben Cuevas, and Andreas Mauthe. A survey of mobility in information-centric networks: challenges and research directions. In *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design-Architecture, Algorithms, and Applications*, pages 1–6. ACM, 2012.

[18] Vinod Valloppillil and Keith W. Ross. Cache array routing protocol, v1, 1998. IETF, Internet-Draft, Available at https://tools.ietf.org/html/draft-vinod-carp-v1-03.

[19] Duane Wessels and K. Claffy. Internet cache protocol (ICP), v2, 1997. IETF, RFC2186, Available at https://tools.ietf.org/html/rfc2186.

[20] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, and George C Polyzos. A survey of information-centric networking research. *IEEE Communications Surveys & Tutorials*, 16(2):1024–1049, 2014.