A collaborative caching strategy in content-centric networking

Qing Hu^{1,2}, Chengming Li^{2,}, Touhidul Hasan², Chengjun Li¹, and Qingshan Jiang²*

¹School of Information Engineering, Wuhan University of Technology, Wuhan, China

²Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China

Abstract. Content-centric Networking (CCN) is one of the most promising network architectures for the future Internet. In-network caching is an attractive feature of CCN, however, the existing research does not consider the off-path nodes, or gives a large communication overhead for cooperation, which makes the caching utilization lower, and hard to achieve comprehensive performance optimization. To reduce the data redundancy and improve the caching utilization, we propose a Regional Hashing Collaborative Caching Strategy (RHCCS). According to calculate the importance of nodes in the network topology, we divide the network into the core area and edge area. In core area, we select the relevant nodes for cooperation, store the block in the off-path nodes with the hashing algorithm, and add a new table in original data structures for routing in the collaborative areas. As for edge area, we deploy the on-path reversion scheme. By simulating in ndnSIM and comparing with the basic caching strategy in CCN, experimental results indicate that the RHCCS can effectively reduce data redundancy, routing hops, requesting delay, and significantly increase the hit rate.

1 Introduction

Nowadays the smartphones are popular for information sharing and access. In order to share contents, we have to deploy IP networks. However, IP communication does not support mobility for the mobile network. In addition, the growth of network traffic, especially the video is much faster than Moore's Law [1], which causes a lot of duplicate redundant transmission on the IP network and led to tremendous network transmission pressure [2]. The emergence of the internet of things (IOT) leads to the lack of IP network address. Therefore, the Content-centric Networking was proposed to solve the mobility of IP networks and lack of IP address [3]. Compared with the traditional IP network model, CCN pays attention to the content of users' request, and it uses the name of content as the symbol of the network transmission instead of the current IP address, where it changes the transmission mode from "push" to "pull". It makes the data routing directly by the content's name, which meets the needs of the future development of the internet [4]. In CCN, one of the most important feature is the support of node caching [5]. It stores data at the nodes, and

* Corresponding author: cm.li@siat.ac.cn

when the same request is issued, the node responds directly without forwarding the need to the server and achieves better performance.

The caching technology of network originates from Web Cache, CDN, and P2P system. However the cache system is relatively closed, and different applications can not share the resources, which is no longer in line with the application-independent nature of caching in CCN. Some new features have emerged in CCN, such as transparency based on unique flags, ubiquity of caching functionality, and fine-grained content block level. The limited caching space in CCN is not enough compared with the massive amount of internet information. Therefore, the caching utilization rate of nodes becomes a significant challenge for the CCN.

The default caching strategy in CCN represents Leave Copy Everywhere (LCE) [6], and it saves data on each router of the content transmission path which causes a waste of resources. Against this, Laoutaris [7] proposes the Leave Copy Down (LCD) strategy which caches content on the nodes downstream of the cache-hit and reduces the redundancy. However, LCD reduces the hit rate. Psaras [8] introduces the probability caching strategy which caches the data with probability, it reduces the data redundancy, but the data distributes randomly, so it can not provide a good service for the users. Consequently, Kim [9] proposes a cache policy based on node capacity where it caches data on the node with the largest cache space, but it does not consider the interaction of the nodes, which will lead to the deterioration of network performance. Chai [10] proposes a caching strategy based on centrality. This strategy places the buffer on the nodes with the largest centrality, which reduces the redundancy and improves the hit rate. However, the cache space of nodes with small centrality is almost unusable.

Conversely, Ming [11] proposes an age-based cooperation caching strategy. However, when the content age in a node is over, the content hit rate will be reduced. At the same time, the signaling overhead is large, which brings extra expenses. Wang [12] proposes a global hash-based caching strategy, but it uses global strategy for all nodes, which has complicated and high computational overhead.

Some of the proposed methods store the content in the on-path nodes, as for the off-path nodes, the storage spaces are wasted. Because of the small storage capacity, the contents are replaced frequently. The others give some cooperative strategy, but they have high computational overhead and bring a lot of extra expenses. All of that leads to the data request delay increasing and gives users a bad experience in internet communication.

In order to solve the above problems, this paper proposes a Regional Hashing Collaborative Caching Strategy (RHCCS), which takes the topology structure of the network into full consideration. In the proposed method, we calculate the centrality of each node according to the Dijkstra algorithm and select the nodes whose centrality has reached a certain value as the core nodes. For the core nodes we use K-spilt algorithm to divide them into some clusters. And use the hash algorithm to cache each fine-grained content block in one cluster cooperatively, which ensures that only one copy of each piece of content is cached in the cluster. As for the edge nodes, we apply the on-path reversion scheme to increase the overall performance. The proposed strategy takes full consideration of the capacity and importance of the nodes, which is effectively reduced data redundancy and relieves the congestion and load of the network.

2 Implementation of RHCCS

In this section, we proposed a Regional Hashing Collaborative Caching Strategy (RHCCS), which consisted of a collaborative domain generation method and routing model. With the caching strategy, we can get better performance in the network caching.

2.1 The Generation of Collaborative Domain

A collaborative domain is a region formed by a group of related nodes. In this collaborative region, the sum of the node's caching capacity is the caching capacity of the domain. The use of nodes to form a collaborative domain can take full advantage of each node's cache space and increase the throughput capacity. The generation of the collaborative domain consists of calculating the importance of nodes and clustering nodes.

We measured the importance of a node by centrality. Calculating centrality can classify nodes precisely and provide initial samples for clustering. In network topology, the centrality of node v refers to the sum of the ratio of the number of paths traversed by the source node s to the destination node t from the total shortest path number. and the calculation formula is as follows:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (1)$$

where V represents all the nodes in the whole topology, σ_{st} represents the number of all shortest paths. $\sigma_{st}(v)$ represents the number of shortest paths which is through the node v . The Dijkstra algorithm is used for calculating the shortest path between nodes.

Clustering is a process of dividing nodes into some classes which is made up by similar nodes. After clustering, the generated cache space is the collaborative cache domain. Particularly, we use K-split algorithm to cluster the nodes into clusters. In the K-split clustering algorithm, the similarity between two nodes is determined by their topology distance: $Dist(r_i, r_j) = d_{r_i r_j}$, the $Dist(r_i, r_j)$ is the number of hops between two nodes. The specific clustering process is as follows:

Step1: Divide all nodes into edge and core nodes, and select the core nodes as the sample for clustering.

Step 2: Randomly select k core nodes which have bigger centrality as the cluster center.

Step 3: Calculate the distance for all the sample points to each cluster center and classify all the sample points to the nearest cluster center.

Step 4: Calculate the minimum distance to all other nodes for each sample point in the cluster, and make the node with the minimum value which becomes the new cluster center.

Step 5: Back to the step 3, until the center of the clusters no longer change.

In this paper, we use the typical topology GEANT2012 as the experimental topology. After calculating the centrality of each node, we select the nodes whose centrality is more than two as the core nodes. And we apply K-split algorithm to group all the core nodes into five clusters. After this, the nodes in the clusters will be re-number preparing for hash routing.

2.2 Routing Model

Routing refers to the data from source to destination and the decision of its process through the path node. For hash routing in the cluster, we create a new routing model by adding a Hash Value Table (HVT) into the three original data structures (Pending Interest Table, PIT; Forwarding Information Base, FIB; Content Store, CS) in CCN. The hash routing model is shown in figure 1.

HVT consists of the content prefix, hash label, and next hop interface. Different packets have different names and using the names in the hash algorithm we can dynamically update HVT. In order to distinguish whether the routing node is the core node or edge node, avoid the repeated request packet brought by the hash map. A flag is added to the original package in CCN. The communication in CCN is driven by the users so that it includes two kinds of data types in the process of transmission, the interest packet, and the data packet.

We adds processing flags to the original data type, and stores centrality in the data field of each node.

Fig. 1. The routing model.

Fig. 2. Routing in the cluster.

To explain how the hash routing operates in the cluster, we apply the example depicted in figure 2. In this topology, E_1 and E_2 are edge nodes, N_1-N_5 are core nodes, and N_1-N_5 form a collaborative domain. When the request is coming to E_1, it will go to N_1 and hash. Let assume that, we get a hash mapping value of 3. The HVT table in N_1 stores the hash mapping labels and forwarding interfaces in the cluster, and according to look up the HVT. The interest packet is forwarded to the correspondingly numbered node N_3 via N_2. If the requested content is stored in N_3, the content returns directly. Otherwise, the interest packet is forwarded to N_4 by querying the HVT. And by querying the FIB of N_4, the interest packet is forwarded to the edge node E_2, E_2 requests data from the data source. When the data source returns content to the cluster, the content will be stored in the node N_3 and returns to the requester. After hash mapping, the content will be uniformly cached in the domain. The specific forwarding process of the interest and data packet on each node will be described later.

Algorithm 1:Forwarding Process of Interest Algorithm 2:Forwarding Process of Data

For each node receives a interest packet
iterates over local storage space (CS)
if there is cache in CS **then**
 Discard it and return data
else look up the PIT
 if there is not matching entry in PIT **then**
 Look up the node flag
 if it is in the cluster **then**
 look up HVT
 if it match in HVT **then**
 Forward the packet to the matching node
 else look up FIB
 if it match in FIB **then**
 Forward the packet to next hop
 else discard it
 else look up FIB
 if it match in FIB **then**
 Forward the packet to next hop
 else discard it
 else increase the interface in PIT

For each node receives a data packet
iterates over local storage space (CS)
if there is cache in CS **then**
 Discard it
else look up PIT
 if it match in PIT **then**
 Look up the node flag
 if it is in the cluster **then**
 Get the home domain from HVT
 Store data in the domain by hash
 Delete the entry in PIT
 else store the content in CS
 and delete in PIT
 else delete in PIT

In RSCCH, when an interest packet arrives at a node in the network, it will search the CS entry first, if the request data is stored in the CS, it will discard the interest packet

directly. If there is no corresponding data in the CS, it will search PIT. When it gets a match, that means there are other users requested the same data, so add the interface to PIT entry, and discard the interest packet. If there is no matching entry in the PIT, the node flag will be queried. If the flag indicates that the node is an edge node, it will look up FIB, and forward the request. Otherwise, the HVT is looked up. The HVT stores the information of all the nodes in the current cluster, and it indicates that the content of the request has been stored in the cluster when it gets a match. According to the HVT routing entry, the interest packet will be sent to the corresponding nodes in the cluster. If there are no matching entries in the HVT, it will look up FIB, and use the best routing policy to forward the interest packet to other nodes or clusters. At the same time record the interface in the PIT entry, the process shown in Algorithm 1.

The data packet forwarding process is shown in Algorithm 2. When a data packet arrives at a node, it will look up the CS. If the data packet already exists in CS, it will be discarded directly. Otherwise, the data packet will look up the PIT. If it gets a match, the node's flag will be queried. If it is edge node, the content will be stored in the CS, and it will be forwarded. If the node is in clusters, the HVT will be looked up. Through the HVT, we obtain the clustering domain of the current routing node, and use the hash algorithm to co-cache each fine-grained content block, then sends the data packet to the corresponding port. If there is no corresponding entry in the PIT, the data packet will be discarded directly.

3 Simulations and Analysis

The ndnSIM [13] and Ubuntu16.04 are used for the simulation. In the comparison, we achieved the LCE and Probability caching strategies. As for network topology, the GEANT2012 is applied. For the simulation, we set one producer (server) and nine consumers (client). The arrival of the node's request obeys the Poisson distribution, and the probability of content request obeys Zipf-Mandebrot. In addition, we set the replacement policy to least recently used (LRU). The simulation time is selected as 1000s, where the sampling period is 2s. We install AppDelayTracer, L2RateTracer, L3RateTracer, and CsTracer for the simulation, and use average request delay $\alpha(t)$, cache hit rate $\beta(t)$, and the average number of routing hops $\chi(t)$ to evaluate performance. The Calculated formula as follows:

$$\alpha(t) = \frac{\sum_{r=1}^S \omega_r(t)}{S}, \quad \beta(t) = \frac{\sum_{r=1}^S v_r(t)}{S}, \quad \chi(t) = \frac{\sum_{r=1}^S h_r(t)}{S} \quad (2)$$

To simulate the real network scenarios, we have done three experiments. In simulation 1, 2000 contents are produced by the server with the size of 1024 bit, where the time delay is set as $200 \mu s$. We compare the performance of each strategy across different cache sizes. The cache size ranges from 5% to 50% (100Mb to 1000Mb). We set the request arrival frequency as 2req/s, Zipf parameter s and q are 0.7. The results shown in figure 3.

From figure 3, we can see as the size of the cache increases, the performance of every policy gets better and better. When a node's cache space becomes larger, it can store more contents to respond the same interest. Therefore, RHCCS has the best network performance. RHCCS can store more data through the nodes' cooperation than the other strategy which stores data on-path. The figure (a) indicates that RHCCS has lower average request delay because it makes sure the content is kept in the cluster as much as possible by the cooperation and responds the request at the current area without being forwarded to the peripheral links. Figure (b) shows RHCCS has a higher cache hit rate where collaborative caching can store more content than single-point caching. In figure (c), we can see RHCCS has a lower average number of routing hops since most of the requests can be obtained in the corresponding collaborative clusters without requesting farther nodes or even server.

Fig. 3. Caching performance with the buffer size curve.

Fig. 4. Caching performance with Poisson distribution parameters curve.

Fig. 5. Caching performance with Zipf parameter curve.

The parameters of simulation 2 are the same as previous experiment except that the buffer capacity and Poisson distribution. The caching capacity sets to a fixed value of 500Mb (25% of the total content) and increments the request arrival frequency from 1 to 5. The results are shown in figure 4.

Figure 4 shows while the parameters of the Poisson distribution gets larger, the average delay of all strategies becomes stable, and the RHCCS demonstrates the best performance.

In simulation 3, the cache capacity is 500Mb, and the exponent s of Zipf-Mandebrot is changed from 0.2 to 1.2 while the other parameters remain the same as simulation 1. The results are shown in figure 5. We can see with the increasing of s , the popular content requests will be more and more concentrated from figure 5. Therefore, every strategy can cache more contents which will be re-requested later, and that makes the nodes can respond to more interest packages. Through comparing with various strategies, we can conclude that RHCCS performs better in the same situation.

4 Conclusion

In-network caching is a very important feature of CCN, this paper proposed a Regional Hashing Collaborative Caching Strategy(RHCCS) for it. The strategy of RHCCS calculates the importance of each node and divides them into core and edge nodes, clusters core nodes and caches fine-grained data blocks by the hash algorithm. Besides, a routing model is proposed by add a new table in original data structure for routing in clusters. Simulation

results confirm that RHCCS can reduce the request delay and the transmission distance than LCE and probability methods. In future, we will combine the popularity of data to classify the all content into different level and accommodate networks dynamically.

Acknowledgment

This work is supported by National Natural Science Foundation of China under grant No. 61602462 and Shenzhen Science and Technology Foundation under grant No. JCYJ20150630114942277.

References

1. N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From psirp to pursuit," in International Conference on Broadband Communications, Networks and Systems. Springer, 2010, pp. 1–13.
2. B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," IEEE Communications Magazine, vol. **50**, (2012)
3. C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (netinf)—an information-centric net-working architecture," Computer Communications, vol. **36**, no. 7, pp. 721–735, (2013)
4. Z. Li and G. Simon, "Cooperative caching in a content centric network for video stream delivery," Journal of network and systems management, vol. **23**, no. 3, pp. 445–473, (2015)
5. S. Wang, J. Bi, Z. Li, X. Yang, and J. Wu, "Caching in information-centric networking," in AsiaFI Future Internet Technology Workshop, (2011)
6. A. Jiang and J. Bruck, "Optimal content placement for en-route web caching," in Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on. IEEE, 2003, pp. 9–16.
7. N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," in Performance, Computing, and Communications, 2004 IEEE International Conference on. IEEE, 2004, pp. 445–452.
8. I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in Proceedings of the second edition of the ICN workshop on Information-centric networking. ACM, 2012, pp. 55–60.
9. D. Kim, S.-W. Lee, Y.-B. Ko, and J.-H. Kim, "Cache capacity-aware content centric networking under flash crowds," Journal of Network and Computer Applications, vol. **50**, pp. 101–113, (2015)
10. W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache less for more in information-centric networks (extended version)," Computer Communications, vol. **36**, no. 7, pp. 758–770, (2013)
11. Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networking," in Computer Communication and Networks (ICCCN), 2014 23rd International Conference on. IEEE, 2014, pp. 1–8.
12. S. Wang, J. Bi, and J. Wu, "Collaborative caching based on hash-routing for information-centric networking," in ACM SIGCOMM Computer Communication Review, vol. **43**, no. 4. ACM, 2013, pp. 535–536.
13. S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnsim 2.0: A new version of the ndn simulator for ns-3," NDN, Technical Report NDN-0028, (2015)