Popularity-Based Hierarchical Caching for Next Generation Content Delivery Networks

Nima Najaflou[1(✉)], Selin Sezer[3], Zeynep Gürkaş Aydın[1], and Berk Canberk[2]

[1] Istanbul University - Cerrahpasa, Istanbul, Turkey
nima.najaflou@ogr.iu.edu.tr
[2] Istanbul Technical University, Istanbul, Turkey
[3] Medianova CDN, Istanbul, Turkey

Abstract. More than half of the content over the Internet is carried by content delivery networks (CDNs). CDNs cache popular and most requested contents on the edges of the network. Thus helping to increase Quality of Experience (QoE), e.g., by decreasing time to first byte (TTFB) for different contents. In the present paper, we focus on developing a hierarchical caching structure for CDNs to improve their QoE. We focus on unpopular content here, since it accounts for a big portion of content over the Internet. Our novel data-driven method forms caching clusters or hierarchies to deal with unpopular contents. In order to form our clusters and assign edge servers into these clusters, we consider the pattern in which contents have been requested including the total number of requests, similar objects between two edge servers, and requests for those objects. Using $tf - idf$ method, which is widely used in information retrieval, we find the similarities between requests landed on each of our edge servers and use these similarities to form clusters using the Markov Clustering algorithm. We evaluate our approach using different hierarchical models, and with real-world requests from a large-scale global CDN. We demonstrate that our hierarchical caching approach improves cache hit ratio by 9.05%. Additionally, a 7.39% decrease in TTFB is observed.

Keywords: Hierarchical caching · User generated content · Long tail · Content delivery network · Clustered caching

1 Introduction

With the increasing number of Internet users and devices connected to the Internet, network data consumption has seen a tremendous growth [10]. Content providers heavily rely on Content Delivery Networks (CDNs) to reduce the latency by placing the content at the network's edges, closer to the end-users, and benefit from in-network content caching. Thus content benefits from improved availability and boosted performance.

N.-S. Vo et al. (Eds.): INISCOM 2021, LNICST 379, pp. 73–87, 2021.
https://doi.org/10.1007/978-3-030-77424-0_7

Numerous studies have focused on content caching in the literature. Web caching has extensively been studied [25,32]. Some studies have focused on caching in the Base Stations(BSs) [3] as well as caching in Heterogeneous Networks (HeNets) [33]. To cope with some of the challenges in content caching, several studies have suggested a hierarchical caching architecture to improve the amount of content that can be served from the cache clusters. These hierarchical caching architectures also reduce latency and bandwidth consumption.

Though there have been many studies in various areas of content caching, most of which have focused on delivering popular content since it is economically more feasible for CDN companies to deal with popular content instead of dealing with unpopular contents which will be consumed by a small fraction of the users. But with the rise of Online Social Networks (OSNs) the amount of User-Generated Contents(UGCs) produced proliferated. A big portion of the content produced by OSNs is having the characteristics of unpopular content. Moreover, today we see a transformation from a limited number of content producers, to where everybody is a content producer and publisher. Consider for example e-commerce (online classified advertisement) platforms such as Facebook Marketplace, eBay, or Alibaba.com where users can publish their contents. Here again, the characteristic of the content and its' popularity is different from a news website such as The New York Times. The long tail of the content in OSNs or UGC providers such as classified advertisement platforms make it harder to keep these contents in the cache pool of a CDN without the content getting evicted by cache replacement algorithms. The result is a lower hit ratio for unpopular content, which in some cases may even add to overall Time to First Byte (TTFB) by adding another hop. This low cache hit ratio will consequently affect QoE and QoS provided by CDNs.

Despite the explosive growth of UGCs and OSNs and the importance caching these content play for CDNs and the challenges they may face, there have only been few works on caching unpopular long-tailed content such as the work in [1,7,28]. But most of these studies have focused on video and explicitly YouTube platform and less has been done for images or other formats of contents which are different in many aspects from the video. In this paper, we will focus on unpopular content and we will propose a new approach for efficiently caching this type of content. Our data-driven approach will provide this flexibility to react to the changes in content access patterns over time and adapt accordingly. Our proposed method is a dynamic hierarchical caching topology which may change over time if the content access pattern changes. Moreover, instead of defining a fixed topology for all content providers and serving all content using this structure, we try to group content providers with similar access patterns and construct unique topologies for each of the groups. We will enforce similar patterns into these groups and will increase the chance of a cache hit.

1.1 Our Contribution

Firstly we will introduce a new data-driven two-level hierarchical caching topology to deal with unpopular content. To form our clusters we use info about the

characteristic of the requests landed on surrogate servers. This method in contrast to other arbitrary methods tries to cluster edges with similar behavior into the same cluster. Secondly, this design, in contrast to arbitrary two-level hierarchical caching topology suggested in previous works, and those used by major CDN companies are dynamic and adaptive and have the capability to change and reshape based on the changes in content access patterns. Thirdly we will use a novel low-complexity approach to model this access pattern and group similar patterns at the same clusters. The approach we use is a combination of the well-known tf-idf method used in natural language processing and the Markov clustering algorithm. Finally, We benchmark our proposed method against the static origin shield method used by many CDNs. To do so we use data from a large-scale global CDN which consists of more than a million requests for various contents.

2 Related Work

There have been several studies focusing on content caching in the literature. Web caching has extensively been studied, see for example [32] and [25] for background information about web caching and its application. Several studies have focused on caching at the base stations (BSs) in mobile networks and the so-called edge caching. Authors in [3,16,31] and [34] deal with the issue of caching contents, mostly popular contents, at the BSs, user devices or intermediate servers, routers or gateways to bring the content closer to the client. In addition to caching popular contents at BSs, the concept of the heterogeneous network (HeNets) and techniques to redirect the user into less busy networks with higher capacity have studied widely [13]. In [33] the authors suggest a scheme where popular contents are pushed and pre-cached into BSs during off-peak hours of the network to be ready.

Many of the challenges in content caching have been discussed in [12,17,22]. Between the challenges mentioned in those articles cache replacement strategies have gained lots of attention. Though several other variations of LRU such as LRFU [14], CFLRU [21], xLRU [19] and ARC [18] have been proposed, but LRU is still the cache replacement algorithm of choice in many systems due to its simplicity and effectiveness.

To tackle the problems with traditional architectures, a tree-like cooperative and uncooperative hierarchical caching was suggested in [8] and they demonstrated that cooperative hierarchical caching architecture outperforms the traditional uncooperative one. The work in [20] suggests a cooperative hierarchical caching scheme that considers link latency while fetching the contents from origin servers. Authors in [4] suggest a content replication method which aims at reducing the handover latency and packet loss. Authors in [5] formulate and design a cooperative cache management algorithm aiming at minimizing bandwidth cost and maximizing cache hit ratio in CDNs. [11] considers a joint cache content placement and request routing problem and seeks for an efficient approximate solution. A content placement algorithm based on the popularity is suggested

in [9]. Their algorithm decides whether to cache or not to cache the content. It caches a number of chunks of the content determined in chunk marking window (CMW) and updates CMW based on the number of requests received for the file. [2] and [24] also deals with the problem of content placement across hierarchical caching networks. In [2] authors focus on content placement across heterogeneous distributed servers. They formulate disk space, bandwidth, and other constraints and their algorithm suggest a number of copies needed and the location they should be placed (Fig. 1).

Reference	Objective	Topology	Content Type	Technique	Solution	Content Type
[4]	Maximize traffic volume served from cache, Minimize Bandwidth cost	Hierarchical, with leaf and parent node	IPTV, VoD content	Linear Programming	Optimal	Popular
[10]	Maximize amount of supported traffic with respect to link utilization	Collaborative Hierarchical Caching	IPTV	Lagrangian Relaxation	Optimal	Popular
[8]	Minimizing Cache management overhead, Maximizing cache usage efficiency	Arbitrary	Chunk-based caching in Content Centric Networks	Greedy	Sub-Optimal	Popular
[2]	Minimize System resource usage (Disk, Bandwidth)	Arbitrary	IPTV, VoD content	Mixed Integer Programming combined with Lagrangian Relaxation	Near Optimal Solution	Popular
[22]	Minimize server load	*m*-level Hierarchy, *m* >2	IPTV, VoD content	Linear Programming, Submodular maximization	Optimal	Popular
This Paper	Maximize Long-tail content delivery from cache, Minimize System Load	2-level Dynamic Hierarchical	Object delivery over CDN	Unsupervised Clustering Algorithm	Optimal	Popular and Unpopular

Fig. 1. Comparison of different works done on hierarchical caching

3 Problem Description

Content Delivery Networks deploy several geographically distributed heterogeneous cache servers, namely *surrogate servers* or *edge servers*, to serve the contents to the clients from the nearest location. This architecture which only consists of surrogate servers and origin servers is shown in Fig. 2a. There are several drawbacks to this architecture such as limited capacity of surrogate servers, flow of numerous requests from surrogate servers to the origin server, and the spike of traffic and load these requests might cause in origin servers.

Some CDN companies such as Akamai and Fastly use a secondary layer of caching called *Origin Shields* to rule out the problems mentioned earlier. They assign a certain and perpetual number of surrogate servers to these origin shield servers. This architecture is also shown in Fig. 2b. Origin shields in this architecture have higher caching capacity (space) which provides redundancy and a bigger pool of cached contents. This will result in a higher HIT ratio for CDN companies because those contents which can't normally be cached on edges due to their unpopularity and lower frequency of use, now can be cached in this

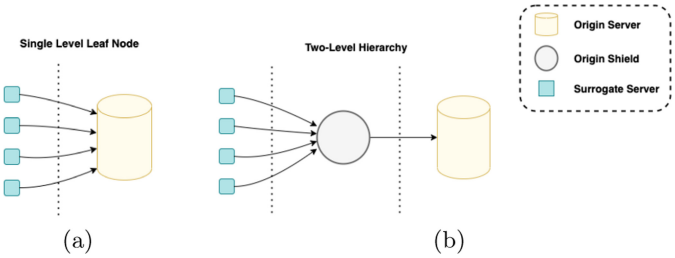

Fig. 2. Cache clusters for a network of edge servers (a) before forming a cluster each edge acts independently and in cache MISS situation they request content from origin server, while in (b) by forming clusters, in cache MISS situation they form a shield for origin server.

redundant area. But there are few problems with these architectures which CDNs have to deal with: Firstly these architectures assign a certain number of edge servers to an origin shield. The number is regardless of content providers' (CP) content types and behavior of their clients. While some CPs may have a limited number of contents with a high volume of requests for these contents, others may have numerous contents with few requests for each of these contents. This will lead to a problem which we call it *battle for survival*. Those contents with fewer requests will be in a battle for survival with more popular content. This situation is shown in Fig. 3. This figure shows the requests made for three different CPs' objects over a certain period of time. Objects belonging to content provider A have lower popularity (lower number of requests per object) compared to content provider B and C.Therefore if we serve all these contents from the same cache pool, cache replacement algorithms such as LRU or others [23] will evict contents with lower popularity (more likely contents belonging to content provider A.) to open up space for upcoming traffic. The behavior of content provider A is very similar to OSN, with a large number of contents and a few requests per content. Hereby the problem with caching UGCs arises. This is what we called earlier a battle for survival, and it makes it hard to cache these contents.

Second problem with this architecture is that when you assign an origin shield to a few edge servers perpetually, in fact, you restrain this capacity. Due to the static nature of these architectures, inefficiencies in handling requests and sub-optimal usage of capacity will be inevitable.

Thirdly, since they don't have a data-driven architecture, they can't react autonomously to any changes such as changes in the number of requests, edge servers load, content popularity, and other metrics, and adapt themselves accordingly.

To deal with long-tailed contents and use our distributed and heterogeneous infrastructure optimally and deliver UGCs efficiently, we propose to form cache clusters dynamically and assign shielding responsibility to one of the surrogate servers within the cluster.

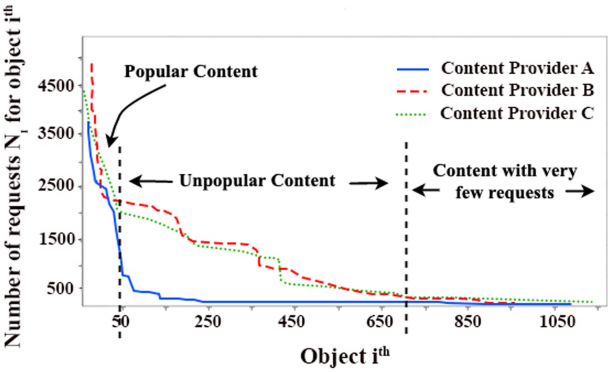

Fig. 3. Problem with the current architecture.

4 Proposed Architecture

As discussed earlier, instead of assigning a fixed number of surrogate servers to their corresponding origin shields which are typically used by many CDN companies, our architecture constructs these building blocks dynamically considering the volume of the requests and different parameters in the surrogate server and underlying network. Additionally, we calculate and form these clusters separately for each customer with unique behavior, so for example customer A's cluster is different from customer B's cluster. These calculations are done in certain time intervals. These clusters may vary from time i to time j ($i \neq j$). Figure 4 depicts the overall system architecture and it gives a detailed representation of how our system works.

In each cluster C_i we have two types of nodes: A Selected Surrogate Server (SSS) and one or more Distribution Surrogate Servers (DSS). Requests from clients first land on DSS, in case of a cache MISS in DSS, the request will be handed to that cluster's SSS. If SSS has that content, it will pass it to DSS, otherwise, SSS will ask the origin server of the content provider for the content, gets it, and delivers it to DSS that had asked for it. A copy of the content will be cached on SSS. SSS here will also act as a shield for the origin server by reducing the number of requests for the same content from the same cluster to only one. These role assignments are done after doing the measurements and calculations depicted in Fig. 5. The final output of these two modules is the roles for edge servers which will consequently form clusters of edge servers with DSS and SSS. Figure 5 shows these two modules and their components. We will explain these modules after giving preliminary information about the graph we constructed for our network and related topics, then Edge Side and Calculation modules are explained.

We model our geo-distributed CDN as a set of heterogeneous nodes with different capacities and resources. These nodes are connected through heterogeneous network connections or links. Each of these links has different inbound

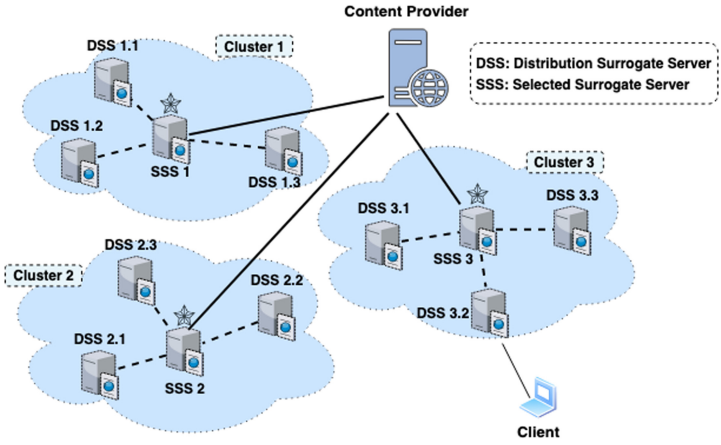

Fig. 4. Proposed Architecture and corresponding components

and outbound bandwidth and latency. We assume that all these nodes and the links between them have a similar reliability rate. Additionally, we assume that between each pair of nodes there is a routing path, which emphasizes that the graph forming our network is strongly connected. These links are weighted and undirected.

4.1 Edge Side Module

In order to collect data from our edge servers, we designed a module called the Edge Side module which is located on each edge server (Fig. 5). For all of our customers, the TailStat unit measures the number of requests received for each of the contents. It will give us the total number of received requests (TLR) and the total number of requested objects (TRO). This info can give us an insight into frequently requested content and less frequent ones. Edge Resources unit measures available resources of the edge server such as disk usage, bandwidth, and other load-related metrics of the server (D, B, LA, P). All these data are collected and sent to the Calculation module in some time intervals.

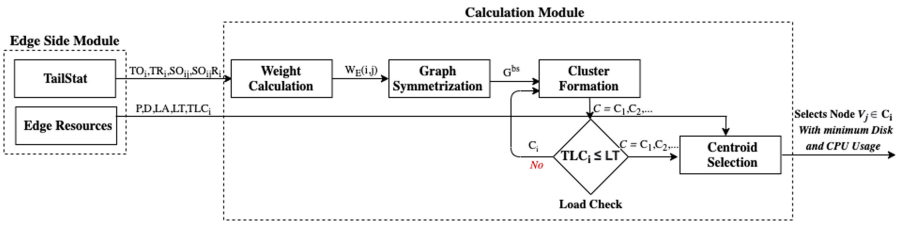

Fig. 5. A scheme showing our modules and their corresponding sub-modules.

Table 1. Notations

Symbols	Definitions
G	A graph represented as a matrix with elements being weights on corresponding edges
i, j	Node indices, $1 \leq i, j \leq$ n
n	Number of nodes in G
P	CPU Usage
D	Disk I/O Usage
LA	Load Average
LT	Load Threshold
TO_i	Total number of objects at node i
TR_i	Total number of requests received at node i
SO_{ij}	Similar objects between node i and node j
$SO_{ij}R_i$	Number of requests at node i for Similar objects between node i and node j
TLC_i	Total load average of cluster C_i
NVC_i	Number of vertices in cluster C_i

4.2 Calculation Module

The calculation module is made of four units each of which does a certain job to finally obtain the clusters and their corresponding centroids. These units are shown in Fig. 5.

Initially, a directed weighted graph is created. Each of the nodes V of this graph are our surrogate servers and we have an edge between all the nodes of this graph, in another word it is a complete digraph. For each edge E_{ij} of the digraph $\mathcal{G}$ which is connecting vertices v_i to v_j a weight W_E is calculated using our weight formula.

We use the Weight Calculation unit (1) to calculate the weight $W_E(i, j)$ for each directional edge E_{ij} from node v_i to node v_j. For doing so we adapt *tf-idf* [26] which is mainly used as a weighting factor in information retrieval or text mining. In order to prevent confusion we simply call it $W_E(i, j)$ and it can be obtained by:

$$W_E(i,j) = \frac{SO_{ij}}{TO_i} \times \log(\frac{TR_i}{SO_{ij}R_i + 1})$$

In this phase of our research we will continue with limited attributes, but adding further attributes to our weighting factor would't change the main concept (Table 1).

The graph $\mathcal{G}$ which is obtained from the Weight Calculation unit is a directed graph. Since many of the works done in community detection or graph partitioning are applicable on an undirected graph and only a few of them are

concerned with a directed graph, we would transform our directed graph into an undirected one in the Graph Symmetrization unit (2). The work done in [27] suggests two methods for graph symmetrization; namely *bibliometric symmetrization* and *degree-discounted symmetrization*. Since our graph is a complete digraph we are going to use bibliometric symmetrization. So for original directed graph $\mathcal{G}$ with associated adjacency matrix A, bibliometric symmetrized graph G^{bs} can be obtained as follow:

$$G^{bs} = AA^T + A^TA$$

All the steps and operations below are applied on Graph G^{bs}. We use Cluster Formation unit (3) to find clusters in our network. Inside this unit, we use the Markov Clustering algorithm (MCL) which was proposed by Stijn van Dongen [30] and is based on stochastic flow through a network. MCL computes stochastic flow through a network by iteratively applying two operators called *inflation* and *expansion* on the initial stochastic matrix till it reaches convergence. MCL has shown to be very effective in problems such as bacterial molecular network clustering [29] as well as biological networks [6,15]. Using MCL you can't define the number of clusters but you can control the reinforcement step by *inflation*. By playing with *inflation rate* you can obtain clusters with different levels of granularity.

Algorithm 1. ClusterFormation

Require: Graph $\mathcal{G}^{bs} = (\mathcal{V}, \mathcal{E}), Adjacency matrix A$
Ensure: Clusters $\mathcal{C} = C_1, C_2, ..., C_k$ such that $\bigcup C_i = \mathcal{V}$

```
A := A + I                          #Adding Self-loops to the graph
M := AD^-1                          #Calculate Initial Stochastic matrix M
while M hasn't converged do
    M := M_exp                      #Expand M by taking power e of M
    M := M_inf                      #Inflate M with parameter r
    M := M_Pru                      #Prune M
end while
Interpret resulting Matrix M to detect Clusters
return C
```

Though our first goal is to find communities inside our network of edge servers, we also would like each cluster to be capable of handling the load inside the cluster without getting overwhelmed. If any of our clusters get over-loaded it will result in it the requests being dropped which will affect that cluster's availability. Availability is an important factor for CDN companies since their customers highly rely on the availability provided by CDNs.

In this unit, we provide our bibliometric symmetrized graph G^{bs} with node set $\mathcal{V}$ and edge set $\mathcal{E}$ respectively as the input to our *ClusterFormation* algorithm. A is adjacency matrix where $A(i, j)$ denotes the weight between node v_i and node v_j and M is the initial stochastic matrix and represents the transition probability

Algorithm 2. LoadCheck

Require: Clusters $\mathcal{C} = C_1, C_2, ..., C_k$ with their corresponding load data
Ensure: Load of each cluster C_i is below a threshold LT
1: **for all** Cluster $C_i \in \mathcal{C}$ **do**
2: **if** $TLC_i \leq LT$ **then**
3: $CentroidSelection(C_i)$
4: **else**
5: $LoadCheck(ClusterFormation(C_i))$
6: **end if**
7: **end for**
8: **return** $\mathcal{C}$

of a random walk or in other words Markov chain on the graph. Additionally, we make sure that the load of each cluster is below a threshold through *LoadCheck* algorithm. If any of the clusters has load above the threshold LT we recalculate clusters for the nodes inside that cluster separately. We calculate the total load average for each cluster C_i denoted by TLC_i as follow:

$$TLC_i = \frac{\sum_{v_j \in C_i} LA(v_j)}{NVC_i}$$

Algorithm 3. CentroidSelection

Require: Set v_j of vertices belonging to cluster C_i
Ensure: Load of each cluster C_i is below a threshold LT
1: **for all** Vertices $v_j \in C_i$ **do**
2: **if** $LA(v_j) \leq TLC_i$ **then**
3: Select Node with Min Disk I/O D and CPU P
4: **end if**
5: **end for**
6: **return** Centorid CN_i for Cluster C_i

In order to create a hierarchical topology and reduce the number of requests flowing to the origin server, we need to select a centroid for each of our clusters. We do so using the Centroid Selection unit (4). The centroid of each cluster which we call the Selected Surrogate Server (SSS), depicted in Fig. 4, will act as an origin shield, providing a higher hit ratio and lower latency.

5 Numerical Evaluation

Here in this section, we will present the implementation results of our proposed architecture in a realistic testbed. Throughout the tests we assume, there are $NON = 7$ edge nodes each assigned a cache size CS which is less than what is needed to store all the objects in the cache. The reason for this decision is

to create a cache competition situation explained earlier. All these seven nodes have similar hardware such as CPU, RAM, etc., and have the same caching configurations. These nodes are located at the same location, all in a data center in Germany which means the latency value from the origin server to these edges and vice versa are almost the same.

We use a collection of $N = 100$ objects with identical size of OS. For simplicity we assume all the objects belong to one origin server. Therefore the number of objects which can be stored on each edge node can be calculated as $NOO = CS/OS$. It's good to know that in real deployments there are more than one origin servers.

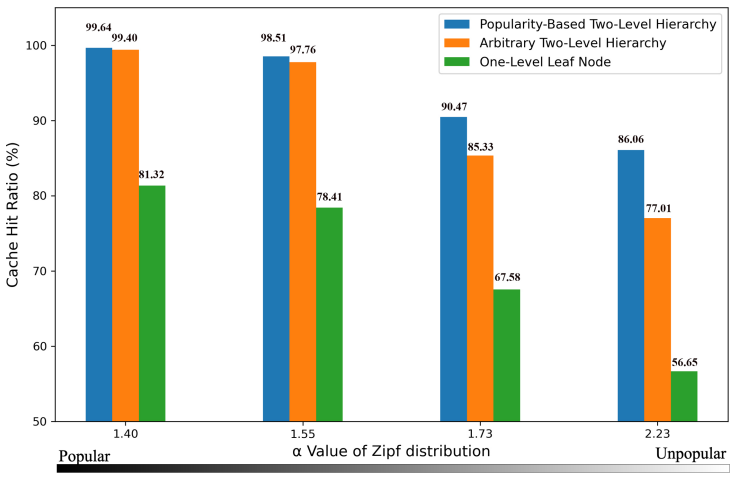

Fig. 6. Total Hit Ratio of all edge servers over different shape parameter α

We assume that the popularity of the contents follows Zipf Distribution with shape parameter α. We use five different values for parameter α as $\alpha = \{1.35, 1.45, 1.85, 2.00, 2.15\}$ and use $NumPy$ library to generate numbers with following Zipf distribution. In each test, we generate seven sets of numbers using a single α value. We repeat this for all the α values given above. In all of our edge servers, we run LRU for cache replacement. We set the load threshold TL needed for *LoadCheck* algorithm to be 75%.

The main objectives of our tests are to observe the effect of our proposed architecture on Cache Hit ratio and Latency. We can describe Cache Hit ratio as the ratio between the number of requested contents served from the cache to the total number of contents requested. Similarly, we measure the latency here through a metric called Total download time. Since all the objects in the cache are similar and of the same size, this will give us the time in which the requested object has been retrieved.

We conducted three sets of experiments, one using one-level leaf node, another with arbitrary hierarchy structure and the third one was formed as

the output of our *ClusterFormation* algorithm. Also in our second set of experiments we randomly chose a node as the origin shield, but in the third set we used *CentroidSelection* algorithm for this purpose. The number of requests and other data in the first set were given as input to form the clusters and choose centroids in the third set.

Finally We performed 10 tests and calculated the Cache Hit ratio and latency with 95% confidence interval.

5.1 Test Results

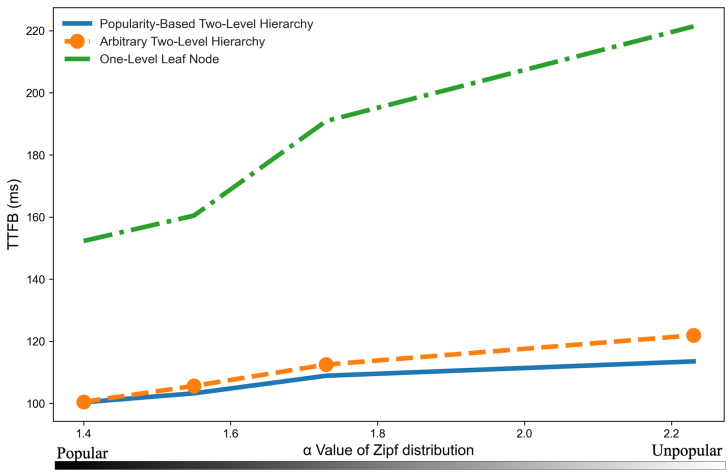

Fig. 7. Average content access time latency (ms) for different shape parameter α

The results of our tests are shown in Figs. 6 and 7. Figure 6 shows Hit ratio measurements of three different topologies used. As you can see here, though our proposed popularity-based two-level hierarchical topology outperforms other topologies in popular contents (contents with low α value) this difference gets more significant when the value of α grows and it deals with unpopular content. Our proposed method shows 9.05% improvement compared to the arbitrary two-level topology. We believe this is due to the pattern-sensitivity of our algorithm and the way it clusters edges into different groups.

Additionally, looking at Fig. 7 we can see similar effect. As you can see here, our proposed algorithm has lead to 7.39% improvement in TTFB. It's good to know that this result may vary in similar tests. Since TTFB value between Client-Edge, Edge-OriginShield and OriginShield-Origin may vary due to network situation and the distance of the servers to each other. Again what is obvious here is that our proposed method has taken away arbitrary two-level hierarchy in reducing TTFB for the contents.

6 Conclusion

In this paper, we have proposed a novel approach to form a hierarchical caching topology in content delivery networks. In particular, we focused on user-generated content which tends to have a long tail, and impose a challenge for CDNs. We have analyzed the performance of our approach with different Zipf distributions. We also found that our proposed scheme can significantly improve cache hit ratio for higher shape parameter α in Zipf distribution which is equivalent to long-tail contents. Numerical results showed this approach's performance benefits over other common cache hierarchies used by CDNs. To the best of our knowledge, this work is the first to address the problem of user-generated content caching. We hope this work will facilitate future research in this area.

References

1. Ager, B., Schneider, F., Kim, J., Feldmann, A.: Revisiting cacheability in times of user generated content. In: 2010 INFOCOM IEEE Conference on Computer Communications Workshops, pp. 1–6. IEEE (2010)
2. Applegate, D., Archer, A., Gopalakrishnan, V., Lee, S., Ramakrishnan, K.K.: Optimal content placement for a large-scale VoD system. IEEE/ACM Trans. Netw. **24**(4), 2114–2127 (2015)
3. Bastug, E., Bennis, M., Debbah, M.: Living on the edge: the role of proactive caching in 5G wireless networks. IEEE Commun. Mag. **52**(8), 82–89 (2014)
4. Bilen, T., Canberk, B.: Handover-aware content replication for mobile-CDN. IEEE Netw. Lett. **1**(1), 10–13 (2018)
5. Borst, S., Gupta, V., Walid, A.: Distributed caching algorithms for content distribution networks. In: 2010 Proceedings IEEE INFOCOM, pp. 1–9. IEEE (2010)
6. Brohee, S., Van Helden, J.: Evaluation of clustering algorithms for protein-protein interaction networks. BMC Bioinform. **7**(1), 488 (2006). https://doi.org/10.1186/1471-2105-7-488
7. Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.Y., Moon, S.: I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp. 1–14 (2007)
8. Che, H., Tung, Y., Wang, Z.: Hierarchical web caching systems: modeling, design and experimental results. IEEE J. Sel. Areas Commun. **20**(7), 1305–1314 (2002)
9. Cho, K., Lee, M., Park, K., Kwon, T.T., Choi, Y., Pack, S.: Wave: popularity-based and collaborative in-network caching for content-oriented networks. In: 2012 Proceedings IEEE INFOCOM Workshops, pp. 316–321. IEEE (2012)
10. Cisco: Cisco annual internet report (2018–2023) white paper (2020). https://bit.ly/3e8MYuk
11. Dai, J., Hu, Z., Li, B., Liu, J., Li, B.: Collaborative hierarchical caching with dynamic request routing for massive content distribution. In: 2012 Proceedings IEEE INFOCOM, pp. 2444–2452. IEEE (2012)
12. Dehghan, M., et al.: On the complexity of optimal routing and content caching in heterogeneous networks. In: 2015 IEEE Conference on Computer Communications (INFOCOM), pp. 936–944. IEEE (2015)

13. ElSawy, H., Hossain, E., Haenggi, M.: Stochastic geometry for modeling, analysis, and design of multi-tier and cognitive cellular wireless networks: a survey. IEEE Commun. Surv. Tutor. **15**(3), 996–1019 (2013)
14. Lee, D., et al.: LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Comput. **12**, 1352–1361 (2001)
15. Li, L., Stoeckert, C.J., Roos, D.S.: OrthoMCL: identification of ortholog groups for eukaryotic genomes. Genome Res. **13**(9), 2178–2189 (2003)
16. Liu, D., Chen, B., Yang, C., Molisch, A.F.: Caching at the wireless edge: design aspects, challenges, and future directions. IEEE Commun. Mag. **54**(9), 22–28 (2016)
17. Maddah-Ali, M.A., Niesen, U.: Fundamental limits of caching. IEEE Trans. Inf. Theory **60**(5), 2856–2867 (2014)
18. Megiddo, N., Modha, D.S.: ARC: a self-tuning, low overhead replacement cache. Fast **3**, 115–130 (2003)
19. Mokhtarian, K., Jacobsen, H.A.: Caching in video CDNs: building strong lines of defense. In: Proceedings of the Ninth European Conference on Computer Systems, pp. 1–13 (2014)
20. Najaflou, N., Arış, A., Canberk, B., Aydın, Z.G.: The nearest origin-shield (NOS): a jitter-free overlay routing framework for content delivery networks. In: 2019 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–6. IEEE (2019)
21. Park, S.Y., Jung, D., Kang, J.U., Kim, J.S., Lee, J.: CFLRU: a replacement algorithm for flash memory. In: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp. 234–241 (2006)
22. Paschos, G.S., Iosifidis, G., Tao, M., Towsley, D., Caire, G.: The role of caching in future communication systems and networks. IEEE J. Sel. Areas Commun. **36**(6), 1111–1125 (2018)
23. Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Comput. Surv. (CSUR) **35**(4), 374–398 (2003)
24. Poularakis, K., Tassiulas, L.: On the complexity of optimal content placement in hierarchical caching networks. IEEE Trans. Commun. **64**(5), 2092–2103 (2016)
25. Rabinovich, M., Spatscheck, O.: Web Caching and Replication, vol. 67. Addison-Wesley, Boston (2002)
26. Ramos, J., et al.: Using TF-IDF to determine word relevance in document queries. In: Proceedings of the First Instructional Conference on Machine Learning, New Jersey, USA, vol. 242, pp. 133–142 (2003)
27. Satuluri, V., Parthasarathy, S.: Symmetrizations for clustering directed graphs. In: Proceedings of the 14th International Conference on Extending Database Technology, pp. 343–354 (2011)
28. Traverso, S., Huguenin, K., Trestian, I., Erramilli, V., Laoutaris, N., Papagiannaki, K.: Tailgate: handling long-tail content with a little help from friends. In: Proceedings of the 21st International Conference on World Wide Web, pp. 151–160 (2012)
29. Van Dongen, S., Abreu-Goodger, C.: Using mcl to extract clusters from networks. In: van Helden, J., Toussaint, A., Thieffry, D. (eds.) Bacterial Molecular Networks, vol. 804, pp. 281–295. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-61779-361-5_15
30. Van Dongen, S.M.: Graph clustering by flow simulation. Ph.D. thesis (2000)
31. Wang, X., Chen, M., Taleb, T., Ksentini, A., Leung, V.C.: Cache in the air: exploiting content caching and delivery techniques for 5G systems. IEEE Commun. Mag. **52**(2), 131–139 (2014)

32. Wessels, D.: Web Caching. O'Reilly Media, Inc., Sebastopol (2001)
33. Yang, C., Yao, Y., Chen, Z., Xia, B.: Analysis on cache-enabled wireless heterogeneous networks. IEEE Trans. Wirel. Commun. **15**(1), 131–145 (2015)
34. Zeydan, E., et al.: Big data caching for networking: moving from cloud to edge. IEEE Commun. Mag. **54**(9), 36–42 (2016)