Effects of Replica Placement Algorithms on Performance of structured Overlay Networks

Bassam A. Alqaralleh, Chen Wang, Bing Bing Zhou, and Albert Y. Zomaya
School of Information Technologies
University of Sydney, NSW 2006, Australia
{bassam, cwang, bbz, zomaya}@it.usyd.edu.au

Abstract

In DHT-based P2P systems, Replication-based content distribution and load balancing strategies consists of such decisions as which files should be replicated, how many replicas should be created and where to replicate them in order increase the system performance in the presence of non-uniform data and access distribution. There are many works on replica placement policies; however, the impact of system workload on different replica placement strategies is not well studied. We investigate this problem under the context of content addressable overlay networks. We compare a trace based replica placement algorithm with two of its variations, namely random placement and priority based placement under different workloads. Our experimental results show that the effect of replica placement policy is highly affected by the workload of the system, which indicates that an adaptive replica placement strategy is desirable for content distribution in an overlay network.

1. Introduction

In recent years, decentralized search with the support of Peer-to-Peer (P2P) technologies draw lots of interests as it potentially enables us to organize distributed systems for information diffusion and storage in a very flexible manner. A structured P2P overlay network normally maps data and computer nodes into the same ID space. With a Distributed Hash Table (DHT) [5][6][7] like routing mechanism, it effectively reduces the cost of data search. In a structured overlay, data are mapped to nodes based on the relationships between their IDs. The ID of a data item is normally calculated from the attributes [12] of the data item or the digest of the data itself. However, this causes data items unevenly distributed among content nodes due to the difference in popularity of these data items. It also causes the workload of nodes that serve queries to these data items unevenly distributed. The former may lead to *data skew* where many data items corresponding to the same ID are mapped to one content node, the latter leads to *access skew* where queries to a popular data ID may overwhelm the hosting node. Solving the skew problem is crucial to the scalability of systems built on P2P technologies. We discuss the different approaches for solving access skew problem in this paper.

Constructing a content distribution network through replicating popularly accessed data is a common way for solving access skew problem. However, replication-based content distribution and load balancing strategies consist of such decisions as which data items should be replicated, how many replicas should be created and where to replicate them in order to enhance the system performance in the presence of non-uniform data distribution and dynamic data access pattern. In this paper, we first describe three replica placement algorithms for content distribution network construction on top of a structured overlay. We then investigate the effect of replica placement algorithms on the performance of content distribution network. We implement the following three replica placement algorithms on top of prefix routing based DHT overlay.

The first algorithm, *CDN-QueryStat*, is a query route based replica placement algorithm. In a DHT routing mechanism, the routes of messages converge to nodes surrounding the destination node hop by hop exponentially in speed. This can be explained by that the destination node is only included in the routing tables of a limited number of nodes and messages have to be routed to the destination through one of these nodes. Putting number of hops a query has to travel.

The second algorithm, *CDN-Rand*, is a random node selection algorithm for replica placement. The hosting node randomly generates an ID from the ID space and the content node that is responsible for the ID is selected to store the replica. The third algorithm, *CDN-PR*, is priority based. A content node is likely responsible for multiple data IDs, and each data ID may have its own content distribution network which consists of all nodes holding data items with the same data ID, therefore the content node can be a member of multiple content distribution networks. The *CDN-PR* gives priority to nodes that already store replicas. This algorithm clusters replicas of different IDs to a few content nodes. Only when the existing nodes with replicas are saturated, a new node is selected for replicas. This approach tries to minimize the number of nodes that store replicas, which may be useful for the maintenance of content distribution networks.

The rest of the paper is organized as follows. Section 2 discusses related works. Section 3 describes the system model. In Section 4, we detail the three replica placement algorithms mentioned above. Section 5 presents experiment results and Section 6 concludes the paper.

2. Related work

Recently, there are many efforts attempting to balance the load using replication-based strategies in the context of both unstructured [10][16][17] and structured [5][6][7] overlay networks. In the context of unstructured P2P networks, three replication strategies proposed in [2] rely on different replica placement strategies: *owner replication* replicates a data items to the peer that has successfully received the service through a query. In other words, the receiver node becomes a service provider. *Path replication* replicates the data on all peers along the query forwarding route between the peer requesting the data and the peer having the data. The service-providing peer receives the query which contains information about the sequence of peers that forwarding the query, then the service provider peer sends a reply and replica in the reverse direction of the query forwarding route. Path replication method has a good search performance and it is easy to implement [3][4]. *Random replication* replicates data objects randomly amongst other peers. However, the authors claimed that for unstructured p2p networks, random replication is superior to owner and path replication, and it is the most effective approach for achieving both smaller search delays and smaller deviations in search.

Paper [4] proposed two replica placement methods derived from *path replication*: *Path random Replication* is straightforward extension of path replication. This method is a combination of path replication method coupled with a replication ratio. Based on the probability of the pre-determined replication ratio, each intermediate peer randomly determines whether or not a replica is created and placed there. *Path adaptive Replication* determines the probability of the replication in each peer based on a predetermined replication ratio and its resource status. Paper [9] proposed *uniform* and *proportional* replication strategies. In the uniform strategy, replicas are uniformly distributed over the network, while the proportional strategy replicates popular files more frequently to improve its availability. A proportional strategy improves the availability of the popular items.

In the context of structured p2p networks, paper [8][14] proposed the “power of two choices” load balancing strategy for Chord. This strategy relies on different algorithm for replica placement. In order to provide load-balancing, multiple hash functions are used instead of only one, each object is hashed to multiple IDs, and placed on the least loaded node of the nodes responsible for those IDs, and the other nodes are given a redirection pointer to hosting node. Also, [15] proposed another replica placement strategy relies on utilizing multiple hash functions. When the demand for popular file exceeds the overall capacity of the current serving nodes, a previously unused hash function is used to obtain a new node ID where the file will be replicated. A set of distributed algorithms proposed to choose an unused hash function when replicating a file and used function when requesting a file. Another replication-based load balancing strategy proposed in [11] relies on a fully distributed mechanism to maintain the file access history in order to predict the future file access frequencies. This strategy replicates the files on the peers adjacent to a group of peers which have high probability to access these files in the future.

Comparing to existing replica placement algorithms, the algorithms given in this paper is based on decentralized decision making and can construct content distribution networks in a self-organizing way.

3. System Model

A node in a structured overlay stores a number of data items mapped to it based on their IDs. Each content node in the overlay network runs a process for query processing. Query processing is *FCFS* based. The maximum queue size defines the query processing capacity of a node. When the queue is full, the queries coming subsequently are either dropped or forwarded to a node that might be able to solve the query. When the data hosted by a node is popular, it is likely that the queue is full most of the time.

Each node in the system is able to create replicas of its local data items onto other nodes selected by replica placement algorithm.

The system contains the following components:

- Query routes and access history collection mechanism to capture the temporal locality of incoming queries, in order to collect the most popular routes where queries come from.

- A replica placement algorithm.
- A mechanism that constructs and maintains a content distribution network for data items of popular IDs.
- A load balancing mechanism that dispatches queries in the content distribution network in order to make efficient use of content nodes and to efficiently cope with dynamic query streams.

3.1 Query Routes and Access History Collection Mechanism

Algorithms *CDN-QueryStat* and *CDN-PR* rely on a simple and efficient mechanism to maintain data access history information and to further capture the temporal locality of incoming queries to make content distribution and load balancing decisions. This mechanism collects the pattern of paths from recent queries as follows: Each individual peer manages its own data access history in order to cope efficiently with the distributed nature of p2p networks, and to reduce the effects of the disappearance of peers on the overall functionality of the network. Each peer uses *QueryStat* table to record the last-hop nodes incoming queries travel through, as well as the count of queries coming from these last-hop nodes in the latest time frame. The time frame is defined to reflect recent query arriving patterns, all the access history before the latest time frame has no much usage and should be removed to limit the size of the *QueryStat* table. In this *QueryStat* table, we only need to keep records for a certain period which is enough to estimate the highly demanded data IDs and help selecting nodes for data replication.

3.2 CDN Construction and Load Balancing

We denote a set of content nodes which hold the data items with ID *k* as cdn_k. A content node N is likely to be responsible for multiple data IDs. We denote the set of content nodes that store data items of IDs held in node N as cdn_A. Node *N*'s capacity of processing queries is described by a query queue bounded by the maximum number of queries it can process within a time frame, denoted by $\boldsymbol{C_N}$. Node *N* also has a watermark $\boldsymbol{W_N}$, which is a certain percentage of its capacity. A content node is considered as overloaded if the workload assigned to it is above the watermark. When the number of queries in the queue is above the watermark, the node will forward the subsequent incoming queries to selected nodes in cdn_k to prevent message from being dropped from the queue.

There is a pre-defined value called *acceptance margin*, denoted by *m* to help determining whether a query should be forwarded to another content node. A procedure for query forwarding and replicas creation is described in Procedure 1. This procedure is activated only when access frequency exceeds the watermark $\boldsymbol{W_N}$. Nodes in cdn_k form a complete graph where each node maintains the direct address of other nodes that store the same data. Workload is exchanged among these nodes periodically. The load of a content node is defined as the number of queries in its queue. A node that hosts multiple data IDs maintains a set of content nodes. The maximum times that a query can be forwarded inside a CDN is bounded by a constant *max_fwds*.

The *New_Content_Node()* function in Procedure 1 uses *ContentNet protocol* to negotiate with a candidate content node for replica creation of a selected key. *ContentNet protocol* defines 8 types of messages, four of which are exchanged between a node requesting new content nodes and the candidate new nodes, one is used for query forwarding among content node, one is used for exchanging load information among content nodes and the rest two used for removing rarely used content nodes. These messages and the *ContentNet protocol* are described in [1].

Procedure 1: Query forwarding and content node creation

```
if the query's forwarding count has reached max_fwds
    insert the query into the query queue
    return
endif
find the least loaded node n from $cdn_k \setminus \{t\}$
if $load_n < W_n - m$
    forward the query to n and increase its
        forwarding count by 1
    return
endif
insert the query into the query queue
// create a new content node
create new content node via function
New_Content_Node( )
return
```

4. Replica Placement Algorithms

In this section, we give three different replica placement algorithms *CDN-Rand*, *CDN-PR* and *CDN-QueryStat* algorithms which can be described as aggressive, conservative and moderated algorithms, respectively, in terms of the number of replica nodes created by these algorithms. Because *CDN-Rand*, *CDN-PR* and *CDN-QueryStat* replica placement algorithms create relatively large, small and medium number of replica nodes, respectively. The first algorithm CDN-QueryStat, as shown in Algorithm 1, places replicas on nodes where queries frequently come from. Algorithm 2 and 3 are *CDN-Rand* and *CDN-PR*, respectively. The *CDN-*

QueryStat relies on simple and efficient mechanism to maintain data access history information and further capture the nodes from which the queries come from. Replicas are therefore created along the query incoming paths. *CDN-QueryStat* most likely places replicas on nodes one hop a way from the overloaded node.

Algorithm 1: Content node creation - CDN-QueryStat

```
function New_Content_Node( )
if new content node creating flag is not set
    set new content node creating flag

    find the node n from Query_Stat table
        with maximum query rate, assume n
        passes most queries to key k and
        $n \notin cdn_k$
    if not found
        randomly select node n from the DHT
            ID space, $n \notin cdn_k$.
    assign n the most requested Key in
    Query_Stat table
    endif

    send cdn_request message to n
endif
```

In *CDN-Rand*, the new content node is selected randomly from the ID space of the overlay. Therefore, this algorithm shows a tendency of distributing the replicas uniformly across the network. Furthermore, some replicas may be placed far away from the overloaded node. Therefore, the negotiations and communications between the node requesting new content nodes and the candidate nodes may consume longer time. On the other hand, that may increase the query travel time between the source nodes and the destination nodes as well.

Algorithm 2: Content node creation - CDN-Rand

```
function New_Content_Node( )
if new content node creating flag is not set
    set new content node creating flag

    randomly select node n from the DHT ID space,
        $n \notin cdn_k$.
    assign n the most requested Key in Query_Stat
    table
    send cdn_request message to n
endif
```

Algorithm *CDN-PR* selects the new content nodes using *CDN-QueryStat* algorithm, and then keeps replicating different data index keys on the same node until that node is saturated, then another new content node is selected using the *CDN-QueryStat* algorithm again. Assume that a new replica of data ID k is to be created; *new* content nodes will not be selected for storing the replica until there is a clear evidence that the current content replica nodes cdn_A of all data index keys hosted by the same node N_{host} are saturated or included in the content delivery network of data ID k, cdn_k. *CDN-PR* differs from *CDN-QueryStat* by showing a tendency of replicating data index keys on less number of content nodes in order to reduce the overhead of load updating between the nodes in content distribution networks.

Algorithm 3: Content node creation - CDN-PR

```
function New_Content_Node( )
if new content node creating flag is not set
    set new content node creating flag

    find the node n from $cdn_A$, where $n \in cdn_A$,
        and $\mathbf{load}_n < W_n - m$ and assign n the next
        most requested Key in Query_Stat table
    if not found
        randomly select node n from the DHT
            ID space , $n \notin cdn_k$
    assign n the most requested Key in Query_Stat
    table
    endif

    send cdn_request message to n
endif
```

In order to explain in details the differences between these algorithms, we give a simple example to describe the behaviour of each algorithm. In this example we assume that content node N0 responsible for hosting a certain range of data index keys {K0, K1, K2, K3, K4, K5}, was overloaded, and replicated the following data index keys {K0, K1, K2, K3} in order to relieve itself. Node N0 created the following number of replicas: 2 replicas of K0, 2 replica of K1, 2 replicas of K2 and 1 replica of K3, and in the following order: K0, K1, K2, K3, K2, K0 then K1.

Algorithm *CDN-QueryStat* replicates the data index keys based on the information from the QueryStat table. Table 1 shows the current QueryStat table of the original hosting node N0, this *QueryStat* table records top-3 frequently query routing nodes. As mentioned before, only top-x nodes are kept in each list. In this table, K0 has been queried 65 times recently, while 25 of them are routed via node N1, 20 of them are routed via N2, 10 are routed via node N4 and the rest 10 are routed via other nodes. As shown in Fig. 1, *QueryStat* algorithm replicated each key on the top frequently routing nodes of that key; K0 was replicated on N1 and N2 because those nodes are the top two frequently routing nodes of K0. K1 was replicated on N2 and N3, K2 on N3 and N4, and K3 on N2.

Table 1: QueryStat table of the original node

Key (msg rate)	**Node-Id** (msg rate)		
K0(65)	N1(25)	N2(20)	N4(10)
K1(60)	N2(24)	N3(22)	N6(14)
K2 (50)	N3(20)	N4(18)	N9(12)
K3(30)	N2(15)	N5(10)	N8(5)

CDN-PR relies on *CDN-QueryStat* algorithm to select only the new replica nodes. Therefore, *CDN-QueryStat* was used to select the first replica node to host replica of K0 based on the information from the *QueryStat* table. As shown in Fig. 2, Node N1 was selected because it's the first top frequently routing node of K0. Then *CDN-PR* algorithm replicated K0, K1, K2, and K3 on the same node N1. Then *CDN-QueryStat* algorithm was used again to select another replica node to host a replica of K2 because the current content node N1 was already hosting a replica of K2. Therefore, N2 was selected because it's the second top frequently routing node of K2. Then *CDN-PR* algorithm replicated K0 and K1 on the same node N2. If there is more than one content node meet the required criteria to host a new replica to choose from, the least loaded node will be selected to host the new replica.

Algorithm *CDN-Rand* most likely replicates every data index key on different randomly selected node using uniform random replica node selection mechanism. Fig. 3 shows one of the possible replicas distribution scenarios.

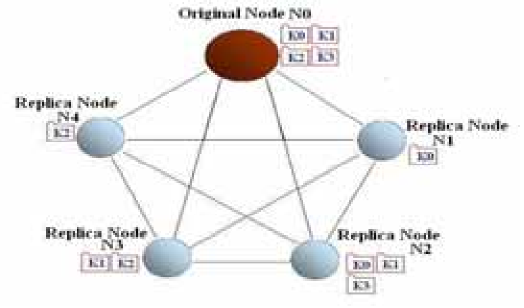

Fig. 1 Construction of content delivery network using CDN-QueryStat algorithm

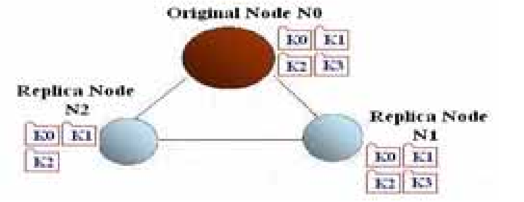

Fig. 2 Construction of content delivery network using CDN-PR algorithm

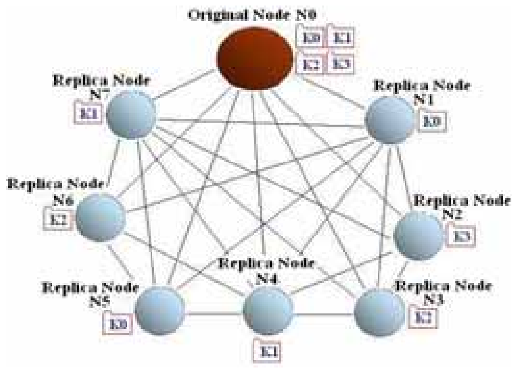

Fig. 3 Construction of content delivery network using CDN-Rand algorithm

5. Experiment Results

We measure the performance of the three replica placement algorithms with a simulator built on top of FreePastry 1.4.1, in which the routing base is set to 2. Our simulation consists of 1000 nodes. The node IDs are randomly generated based on uniform distribution. Data items of 10,000 IDs published on the overlay are generated based on *zipf* distribution with α =0.5. Queries select these keys as targets based on *zipf* distribution with $\alpha = 1.0$. In this scenario, some data nodes are likely to host many hot keys each. Queries are submitted to the whole overlay in a Poisson process with varying average query arrival rate. The source nodes of these queries are randomly selected from the overlay based on uniform distribution. The hop-to-hop latency is set to 9ms. The query processing time is 20ms/query. In the experiment, we assume homogeneous node query processing capacity and set the maximum queue size to 50. The watermark of the load in a content node is set to 80% of its capacity, i.e., 40 queries in its queue. A load-updating message is sent out when there is a workload change of 20%. Different query arrival rate defines different workload of the system. Query arrival rates vary from 1000 to 20000 queries per second in our experiment.

Query Dropping Rate: In Fig. 4, *CDN-QueryStat, CDN-Rand* and *CDN-PR* show comparable performance in terms of reducing query dropping rate. The experiment results show that *CDN-QueryStat* outperforms *CDN-Rand* under all workload circumstances; it is about 2-30% better than *CDN-Rand.* This is due to two main factors; firstly, requests to create new content nodes may be affected by the distance between the overloaded content node and the selected candidate content nodes. *CDN*-QueryStat selects the new candidate content nodes from a group of nodes one hop away from the overloaded content node. These nodes are most likely closer to the overloaded content nodes than the candidate content nodes selected randomly by CDN-Rand from the ID space. Therefore, not surprisingly, *CDN-QueryStat* was able to select a new content node and serve the coming queries faster than CDN-Rand. Secondly, Using CDN-Rand, most likely the queries will be forwarded to the key node which is responsible for dispatching them to other content nodes within the content distribution network, if it is overloaded. The dispatching incurs additional cost which reduces the query processing capability of the content network. On the other hand, CDN-QueryStat pushes the content closer to the requestor. Therefore, queries most likely will arrive to some content nodes which hold the required data items before they arrive in the key node. Those content nodes will be responsible for dispatching queries to other content nodes if they are overloaded and not able to serve them. That most likely will distribute the query dispatching load among many content nodes and reduce the query forwarding delay. As a result, that increased the capability of the content distribution network to process those queries faster and reduced the number of dropped queries.

Under heavy workload, *CDN-PR* performed the worst in terms of query dropping rate. *CDN-QueryStat* is about 11-71% better than *CDN-PR* when the load was above 5000 queries/second, and CDN-*Rand* is about 50-60% better than *CDN-PR* when the load was above 10000 queries per second. This can be explained as follows: Firstly, as mentioned in section 3, as *CDN-PR* algorithm based on concentrating replicas on a minimum number of content nodes, and as the nodes in the network are heavily loaded, these nodes are most likely saturated faster and are not able to accept hosting more replicas as required by this algorithm. Fig. 8 shows that *CDN-PR* algorithm has the maximum number of rejected *CDN-Requests* which increased the delay of new content node creation and affected the performance of this algorithm. Secondly, as shown in Fig. 5, under heavy workload, CDN-PR performed the worst in terms of Queuing delay, *CDN-QueryStat* and *CDN-Rand* are about (16-34%), (6-10%), respectively, better than CDN-PR in terms of reducing Queuing delay. Therefore, as we mentioned before, less queuing delay results from short waiting queues which decreases the probability of dropping queries.

Under light workload, *CDN-PR* achieved the best performance in terms of reducing the query dropping rate when the load was below 10000 queries per second. *CDN-PR* performed similar to *CDN-QueryStat* when the load was below 5000 queries/second, and is about (17-93%) better than *CDN-Rand* when the load was below 10000 queries per second. The performance of *CDN-PR* algorithm was improved because the number of rejected CDN-Requests of CDN-*PR* was reduced as shown in Fig. 8, due to the low query incoming rate which improved the ability of the candidate content nodes to accept more CDN-Requests.

Average query traveling distance: as shown in Fig. 6, the experiments results show that *CDN*-PR and *CDN-QueryStat* algorithms outperform CDN-Rand algorithm in terms of reducing query travel distance (number of hops) between the source node of the query and the destination content node. The results show that CDN-QueryStat is about (3-12.5%) better than CDN-Rand. Also, *CDN-PR* is about (1-7%) better than CDN-Rand as well, because both of *CDN-QueryStat* and *CDN-PR* algorithms most likely select candidate content nodes from a group of nodes one hop away from the overloaded content node. Therefore, replicating the content to these nodes will not only relieve the overloaded content but also will push the content closer to the requestor to reduce the travel time of many queries. Therefore, many queries will be able to get the required data items before they arrive in the key node.

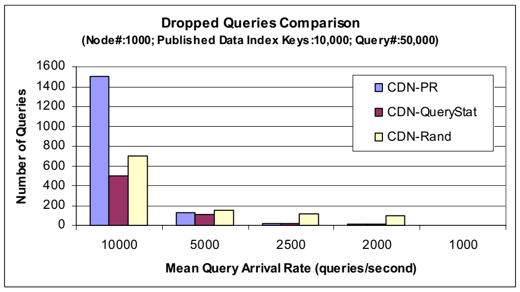

Fig. 4 Number of dropped messages under different workloads, network size: 1000 node, # keys: 10,000

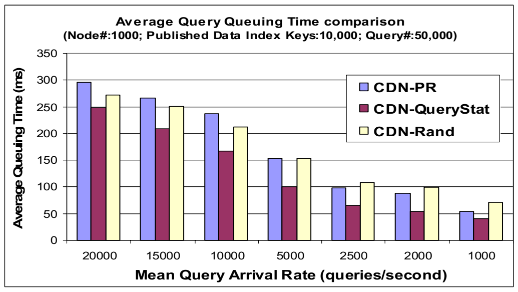

Fig. 5 Average queuing time, network size: 1000 node, # keys: 10,000

Furthermore, The reason which makes *CDN-QueryStat* outperforms (1-7%) better than *CDN-PR* is, as we mentioned before, CDN-QueryStat algorithm relies completely on QueryStat table to select candidate content nodes, Therefore, CDN-QueryStat algorithm was able to select the best location to place the replicas and was able to push more replicas to be closer to the requestor than *CDN-PR*.

Query forwarding hops: As shown in Fig. 7,

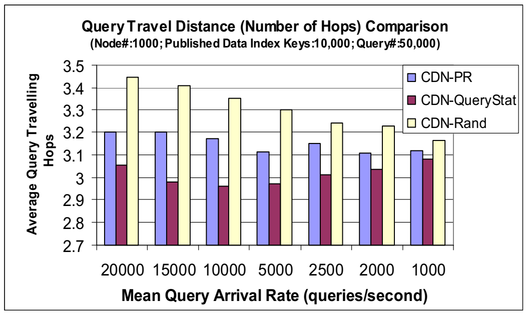

Fig. 6 Average of query travel distance in terms of number of hops, network size: 1000 node, #keys: 10,000

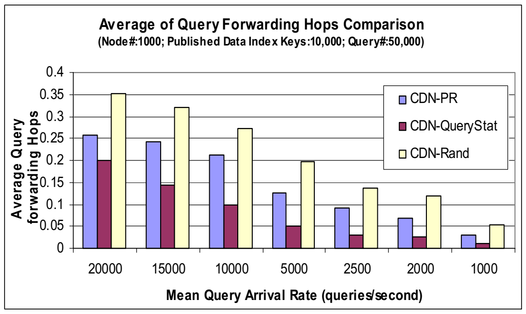

Fig. 7 Average of query forwarding hops among the CDN-Nodes, network size: 1000 node, #keys: 10,000

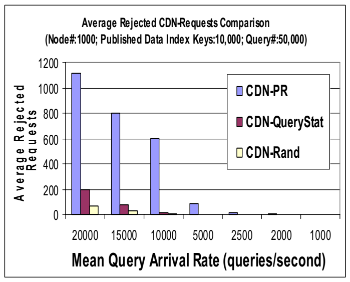

Fig. 8 Average number of rejected CDN-Requests, network Size: 1000 node

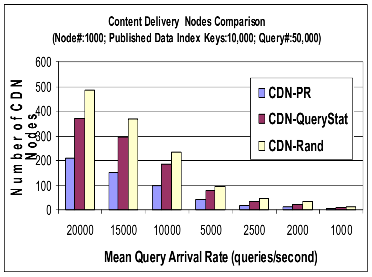

Fig. 9 Number of CDN-Nodes under different workloads

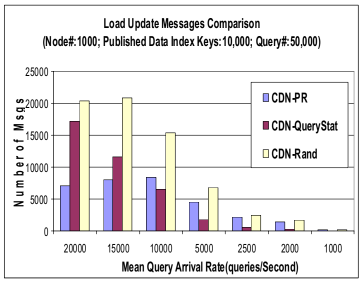

Fig. 10 Number of load update messages, network size: 1000 node

CDN-QueryStat algorithm performed the best in terms of reducing *query forwarding hops* among the content nodes which hold data items with the same data ID. On the other hand, *CDN-PR* outperforms *CDN-Rand* as well, because both of *CDN-QueryStat* and *CDN-PR* select the candidate content nodes to be closer to the overloaded content node than the candidate content nodes selected by *CDN-Rand* to be uniformly distributed over the network. Also, *CDN-QueryStat* outperforms *CDN-PR* because, using *CDN-PR,* the content nodes most likely will be hosting more replicas, and may become saturated and heavily loaded faster, and the load update messaging mechanism will not be able to provide instant accurate load information to the other content nodes due to communication time. Therefore, those queries will be forwarded based on inaccurate information to some overloaded content nodes which may cause re-forwarding those queries again

Rejected CDN-Requests: We compare the number of rejected CDN-Requests under different workloads in Fig. 8. The results of our experiments show that *CDN-PR* performed the worst in terms of rejected CDN-Requests because *CDN-PR* algorithm based on concentrating replicas on a minimum number of content nodes, these nodes most likely will be saturated soon and will not be able to accept hosting more replicas as required by this algorithm. As a result, as shown in Fig. 8, *CDN-PR* algorithm has more rejected *CDN-Requests* than *CDN-QueryStat* and *CDN-Rand.* Furthermore, the experiment results show that *CDN-QueryStat* algorithm has more rejected CDN-requests than *CDN-Rand* as well, because this algorithm also shows a tendency of replicating many data index keys on the same node. Therefore, under heavy workload, some nodes were saturated faster and rejected more CDN-Requests.

Content nodes: Also, we compare the number of content nodes in Fig. 9 under different workloads. *CDN-PR* algorithm created the minimum number of content nodes, about 56-65% less content nodes than *CDN-Rand* and 32-34% less than *CDN-QueryStat*, because *CDN-PR* based on concentrating hosting replicas on less number of content. On the other hand, *CDN-QueryStat* shows about 16-36% less content nodes than *CDN-Rand* as well because *CDN-QueryStat* also shows a tendency of replicating many data items on the same node while *CDN-Rand* shows a tendency of distributing the replicas uniformly over the network.

Load update messages: Decentralized content distribution and load balancing systems rely on periodic load updates from individual content nodes to balance the load and expand the content networks. However, more frequent load updates increases freshness of the load information at the cost of higher overhead. Fig. 10 shows that decreasing the number of content nodes does not necessarily lead to a drop in the number of load updating messages. Our experiments results show that *CDN-QueryStat* outperforms the other two algorithms in terms of reducing the overhead by achieving the minimum number of load updating messages when the query arrival rate was less than 10000 queries per second. When the load exceeded 10000 queries per second, Fig. 10 shows that *CDN-PR* had less number of load update messages, but it does not necessarily mean that *CDN-PR* outperforms the other two algorithms in terms of reducing load update messages; Because *CDN-PR* dropped high number of queries under heavy workload which leaded to less number of queued and served queries by the network. As a result, the number of load update messages was reduced.

6. Conclusions

The replica placement problem has drawn lots of attentions in unstructured overlay networks. However, it is not well studied in a structured overlay. In this paper, we gave three replica placement algorithms proposed for solving the access skew problem in a data index DHT overlay.

We compared the performance of three algorithms. We detailed the advantages and disadvantages of each algorithm through simulation. Our simulation results revealed that system workload has great impact to the performance of replica placement algorithms. The results indicated that an adaptive mechanism that combines the three algorithms together is likely to improve the performance of content distribution and load balancing. Our results also showed that reducing the number of content nodes does not necessarily lead to reduced overhead of exchanging load update messages between the content nodes.

References

[1] C. Wang, B. Alqaralleh, B. Zhou, F. Brites and A. Zomaya, Self-Organizing Content Distribution in a Data Indexed DHT Network, Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06) pp. 241-248.

[2] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, Search and Replication in unstructured peer-to-peer networks, In Proceedings of 16th ACM International Conference on Supercomputing Systems (ICS'02), June 2002.

[3] S. Ata, Y. Gotoh, and M. Murata, Replication Strategies in Peer-to-Peer Services over Power-law Overlay Networks, The 7th Asia-Pacific Network Operations and Management Symposium (APNOMS 2003), Fukuoka, JAPAN October 2003.

[4] H. Yamamoto, D. Maruta and Y. Oie, Replication Methods for Load Balancing on Distributed Storages in P2P Networks, In Proceedings of the 2005 Symposium on Applications and the Internet (SAINT'05).

[5] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications, In *Proceedings of the ACM SIGCOMM Conference*, San Diego, California, Aug. 2001.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, A scalable content-addressable network, In *Proceedings of the ACM SIGCOMM Conference*, San Diego, California, August 2001.

[7] A. Rowstron and P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, In *Proceedings of IFIP/ACM Middleware*, Heidelberg, Germany, November 2001.

[8] J. Byers, J. Considine, and M. Mitzenmacher, Simple load balancing for distributed hash tables, In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[9] E. Cohen and S. Shenker, Replication strategies in unstructured peer-to-peer networks, In *Proc. ACM SIGCOMM'02,* August 19-23, 2002, Pittsburgh, Pennsylvania, USA.

[10] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, Freenet: A distributed anonymous information storage and retrieval system, Proc. ICSI Workshop on Design Issue in Anonymity and Unobservability, LNCS 2009, pp.46–66, Springer, July 2000.

[11] Z. Xu and L. Bhuyan, Effective Load Balancing in P2P Systems, Cluster Computing and the Grid, 2006. CCGRID, Sixth IEEE International Symposium on Volume 1, 16-19 May 2006 Page(s): 81–88, 2006.

[12] N. Harvey, M. Jones, S. Saroiu, M.Theimer, and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties," *Fourth USENIX Symposium on Internet Technologies and Systems (USITS03)*, Seattle, WA, March 2003.

[13] L. Garces-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, K. W. Ross, Data Indexing in Peer-to-Peer DHT Networks, In *Proceedings of 2004 ICDCS Conference*, 2004.

[14] J. Byers, J. Considine, M. Mitzenmacher, Geometric generalizations of the power of two choices, in: Proceedings of the SPAA, 2004.

[15] Y. Xia, S. Chen and V. Korgaonkar, Load Balancing with Multiple Hash Functions in Peer-to-Peer Networks, Proceeding of the 12th International Conference on parallel and Distributed Systems (ICPADS'06), 2006.

[16] Napster Website: http://www.napster.com

[17] Kazaa Website: http://www.kazaa.com