

A Proactive Method for Content Distribution in a Data Indexed DHT Overlay

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 a data-indexed DHT overlay network, published data annotations form distributed databases. Queries are distributed to these databases in a non-uniform way. Constructing content distribution networks for popular databases is effective for addressing the skew problem. However, various crucial replication decisions such as which data object should be replicated, how many replicas should be created and what replica placement policy should be used, may affect the performance of replication based content distribution and load balancing mechanisms in the presence of non-uniform data and access distribution. Particularly, the impact of the propagation speed of replicas on the performance of such type of overlay networks is not well studied. In this paper, a proactive method is given to tackle this problem. The method can adaptively adjust the number of replicas to create based on the changing demand. It works in a fully distributed manner. Our experiments show that the approach is able to adapt quickly to flash query crowds and improve the query service quality of the systems.

1 Introduction

Peer-to-Peer (P2P) technologies are effective for flexible information sharing among a variety of applications. This is mainly because autonomous data sources can flexibly annotate their data (called data indexing in this paper) and the data can be located through decentralized search. A structured P2P overlay network achieves this through mapping data, data index and computers (content nodes) that store the data into the same ID space [1]. The ID of a data item is normally calculated from the attributes [2] of the data object or the hash key of the data itself. However, this mapping can cause *data skew* problem as data objects are unevenly distributed among content nodes. It can also cause *access skew* problem where the workload of nodes that serve queries to these data objects are unevenly distributed due to different data popularity. For a file-sharing P2P system, the access skew problem is not as severe as that observed in Web traffic due to the “fetch-at-most-once” behavior of P2P file-sharing users [3], however, for P2P overlays mainly used for data indexing, the problem is hard to ignore. In a data indexed overlay, data annotations are grouped and stored in nodes selected by underlying DHT mapping algorithms. These stored data annotations (or indexes) form a number of distributed databases. We consider a group of annotations

mapped into the same ID as a data object. Data objects in these nodes are frequently changed; therefore “fetch-at-most-once” behavior is not a common phenomenon in this scenario. Due to the database search cost, the non-uniform access distribution is likely to overwhelm the nodes that are responsible for popular data IDs. Solving the skew problem is crucial to the scalability of systems built on P2P technologies.

Constructing content distribution networks (CDN) through replicating popularly accessed data objects is a common technique to address the *access skew* problem. However, fundamental decisions must be made on issues such as which objects should be replicated, how many replicas should be created whenever the new replicas creation conditions apply, what replica placement mechanism should be used, and how fast these replica should be created and distributed.

In this paper, we give a self-organized content distribution system as well as a proactive content distribution algorithm in order to address the access skew problem. Determining the number of replicas is important for a content distribution system to efficiently use resources and reduce the cost for maintaining consistency among replicas. However it is a non-trivial issue to decide the number of replicas in a dynamic environment. Existing replication strategies for structured P2P networks often leave this problem un-tackled. They either create one new replica at a time when the load exceeds the capacity of the a content node [4,5], or create a fixed number of replicas at a time for caching and fault-tolerant purpose[6], or create replicas along query coming path in order to distribute content as fast as possible and serve flash crowds [7]. As shown in [4], too many replicas can make a system perform poorly due to high overhead. On the other hand, creating replicas slowly will result in poor query serving quality and long query queuing delays. In this paper, we propose a proactive CDN expansion mechanism to make replica creation adapt to the incoming query demands. Meanwhile, we give mechanisms to optimize the CDN performance by reducing the queuing delay and improving the content node utilizations.

The rest of the paper is organized as follows: Section 2 is related work; Section 3 is our CDN construction mechanism and system model; in Section 4 we describe our proactive content distribution mechanism; Section 5 presents experiment results, and Section 6 concludes the paper.

2 Related Work

Replication methods have long been used in distributed systems to exploit the data locality and shorten the data retrieval time. Content distribution networks such as Akamai[8] demonstrate the use of replication methods in the Internet environment. However, Internet content distribution networks often require global load information and manual deployment.

Recently, there are many replication strategies in the context of both unstructured [9, 10] and structured overlay networks [4,6,7]. They can be classified into three different types based on the number of replicas created whenever new replica creation conditions apply.

The aggressive strategy does not control the replica number. Some unstructured overlays [7] replicate data on all peers along the query forwarding route between the peer requesting the data and the peer having the data. This method, called path

replication is not efficient and may result in a large amount of replicas with low utilization. The overhead may also drag down the performance of the whole system. On the other hand, the cost of maintaining consistency among replicas is high and this strategy is not suitable for a data index overlay where the replicated data objects are dynamic database tables. Another replication method called *owner replication* [10,11] replicates a data items to the peer that has successfully received the service through a query. The effect of this method is limited by the bandwidth and system performance of the receiving node.

Another strategy creates replicas based on a fixed replication ratio at a time. For example, paper [12] proposed two replication-based methods derived from *path replication* and replicate data on selected nodes on query path. *Path adaptive replication* determines the probability of the replication in each peer based on a predetermined replication ratio and its resource status. *Path random replication* 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 to be created and placed there. Apparently, a fixed replication ratio can not adapt to the change in incoming queries.

Most replication strategies create one replica at a time when new replica creation conditions apply [4, 5]. A new replica is created only when exiting replicas can not catch up with the increase in incoming queries. However, creating a new replica takes time depends on the network condition and replica size. This strategy may cause large amount of queries loss due to that the increasing speed of the query serving capacity can not catch up with that of the query incoming rate.

The algorithm given in this paper differs from these algorithms mainly in that it can determine the number of replicas for proactive content distribution to adapt to the incoming query rate. Determining the number of replicas is normally discussed in the context of reducing search cost [1] and system recoverability. However, it is important for addressing access skew problem. Recently, the work in [13] proposes to use historical data to predicate whether a data key is hot or not and make multiple replicas for hot data, however, it lacks proper metric to determine how many replicas are sufficient.

3 System Description

In a structured overlay, an ID is associated with a data object or a group of data objects who share the same data index. A data object here can be a data file or a data entry annotating a data file. Replicas of data objects that have the same ID form the content distribution network of the data ID.

As shown in Fig. 1, each data ID may have its own content distribution network. Each content node in the overlay network runs a process for handling queries to the data objects it stores. The query processing is FCFS based. We assume a lossless queue with a *capacity* defined as the number of queries the node can process in a certain time frame. When the capacity is reached, the node is overloaded and the queries coming subsequently may suffer long delay. They may be forwarded to a node that might be able to solve the query. In its simplest form, the content distribution network of a data ID is fully connected and load information is exchanged

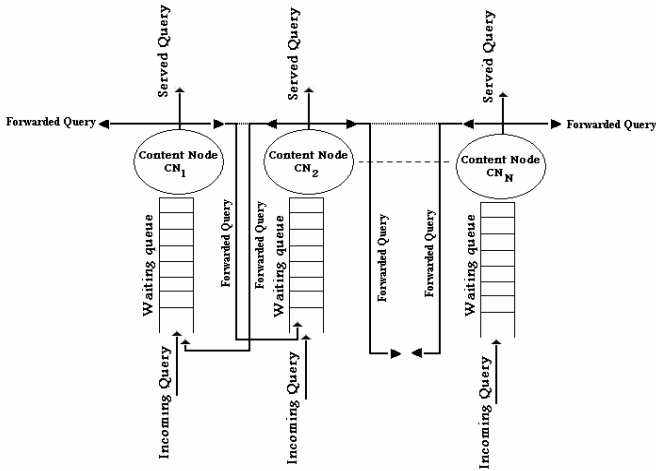


Fig. 1. Content distribution model

periodically along these links so that the forwarding destination can be easily obtained. When the data hosted by a node is popular, it is likely that the queue size is close to or over the capacity most of the time. When all the nodes in a content distribution network are overloaded, the network will be expanded by creating a new replica. The new replica node is selected from available nodes in the underlying network. Each node in the system can autonomously create replicas of its local data objects onto nodes selected by replica placement mechanism.

The system contains the following components for achieving the above:

- Query routes and access history collection: this component captures the temporal locality of incoming queries, and to measure the request arrival rate for every data ID over some measurement time interval.
- Content distribution network construction: this component uses a proactive content distribution algorithm to create new replicas.
- Load balancing: this mechanism dispatches queries in the content distribution network in order to make efficient use of the content nodes.

A. Query Routes and Access History Collection

The analyses of traces collected from Gnutella (<http://www.gnutella.com>) revealed that access locality exist in P2P systems. A popular file that has been accessed recently is also likely to be requested in the near future. We give a mechanism to collect useful information for exploiting query locality, in which each node manages its own data access history. This mechanism maintains the access history for every data ID hosted by a node as follows: Firstly, it obtains the search paths from recent queries in order to determine the pattern of paths recent queries come from. Each node maintains a *QueryStat* table to record the last-hop nodes queries travel through before they arrive in the destination node, as well as the count of queries coming from these last-hop nodes in the latest time frame. A sample *QueryStat* table is shown in Table 1. This table records top-3 frequently query routing nodes. Only top-K nodes

are kept in each list to limit the size of *QueryStat* table. 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. Secondly, this mechanism measures request arrival rate for every data ID hosted by a node over some measurement time interval, in order to estimate the number of replicas needed for an ID. In the simplest form, a *KeyStat* table records the number of requests node n has received for data ID k in the latest time frame.

Table 1. QueryStat table of an overloaded Node

ID (req#)	Node-Id (#Requests)		
K0(65)	N1(25)	N2(20)	N4(10)
K1(30)	N2(18)	N3(7)	N6(5)
K2 (5)	N3(5)	-	-
K3(3)	N2(2)	N5(1)	-

B. CDN Construction

Our replica placement mechanism places replicas on nodes where queries frequently come from. *QueryStat* table is used to select nodes to host replicas. Node n which does not have a copy of the data ID k to replicate and passes most queries to ID k will be selected to replicate data objects of k . Replicas are therefore created along the query incoming paths. Our algorithm most likely places replicas on adjacent nodes which are one hop a way from the overloaded node. However, if there was no candidate nodes available in the *QueryStat* table to host new replicas; new content node will be selected randomly from the ID space of the overlay.

Fig. 2 shows the protocol used by an overloaded content node to recruit new node to join the CDN. An overloaded node can send CDN requests to multiple candidate content nodes simultaneously. However, the data distribution is done sequentially to each candidate nodes that accept the CDN request as different node may replicate different data IDs. We do not assume multicast support in the underlying networks.

C. Load Balancing in the CDN

Nodes storing the same data ID form the CDN of the ID. We denote a set of nodes that hold the data objects with ID k as cdn_k . As mentioned above, a content node's capacity of processing queries is described by the number of queries the node can process within a certain time frame. A content node is considered as overloaded if the number of queries in its queue reaches the capacity value. Before initiating CDN expansion, the overloaded node will forward the queries it can not process in time to lightly loaded nodes in cdn_k to ensure a certain query processing quality. A content node will update other nodes in the same CDN its load when the load change is above a pre-defined threshold. This cost can be high when the size of CDN becomes large. We gave a method in [5] to reduce the load information updating cost. cdn_k needs expansion when there is no lightly loaded node available to share the load.

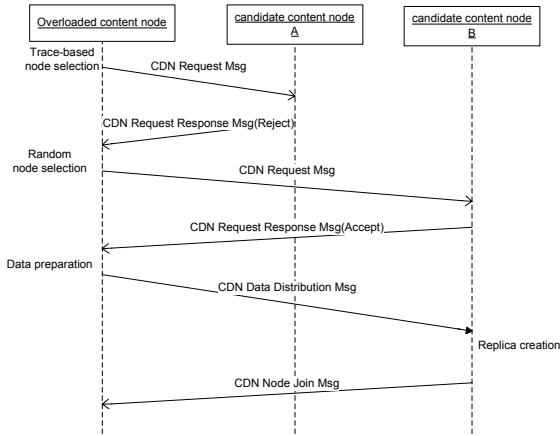


Fig. 2. CDN expansion example

A CDN node can withdraw from the CDN it belongs to if the CDN contains at least two nodes. This happens when the node keeps a low utilization on the stored content for a long time. A hand-off process is performed during the departure of a CDN node.

4 Proactive Content Distribution

In this section, we describe a proactive mechanism for CDN expansion in order to improve the query serving quality, especially for flash query crowds.

Normally, queries are unevenly distributed among data IDs. Similar phenomena are often modelled using Zipf distribution, which indicates that the number of queries to the *i*th most popular data ID, denoted by *n*, is inversely proportional to *i*, i.e.,

$$n \sim i^{-\alpha}, \text{ in which } \alpha \text{ is the shape parameter.}$$

In another words, assuming the expected number of queries to the *i*th popular data ID is \bar{n} , there will be *i* data IDs that have \bar{n} or more queries. According to [14], the probability that a data ID has \bar{n} or more queries is as below:

$$\Pr\{X > \bar{n}\} \sim \bar{n}^{-1/\alpha} \tag{1}$$

in which X is the number of queries to the data ID.

This is a Pareto distribution. We further have the following conditional probability when we observe there are *a* queries to the data ID:

$$\Pr\{X > b \mid \text{queries} = a\} \sim \left(\frac{a}{b}\right)^{1/\alpha} \tag{2}$$

in which a is the observed number of queries to the data ID and b is the total number of queries to this ID. Equation (2) indicates that the more queries to a data ID we observe, the more likely that the total number of queries is greater than b .

When $\alpha = 1.0$, we have the following:

$$\Pr\{X > N\} \sim \frac{1}{N} \tag{3}$$

$$\Pr\{X > b \mid n = a\} \sim \left(\frac{a}{b}\right) \tag{4}$$

Equation (2) and (4) gives a way to predicate the popularity of a data ID based on the number of observed queries to the ID. Early research on the number of visits to web sites further revealed that the difference in the number of visits to a site in two successive time frames is proportional to the total visits to the site [15]. This can be applied to our case: the difference in the number of queries to a data ID in two successive time frames is proportional to the total number of queries to the ID. With this, we can estimate the changing popularity of among data IDs as follows:

We denote the number of queries data ID k receives at time t in current time frame as n_k^t . Assume n_k^{ti} is the number of queries k received in previous time frame ending at ti . $\Delta n_k = n_k^t - n_k^{ti}$ can therefore be used to estimate the number of queries to k by the end of the current time frame, which is

$$\Pr\{n_k^{t(i+1)} > b \mid \Delta n_{ii} = a\} \sim \left(\frac{a}{b}\right) \tag{5}$$

When the capacity of a content node reaches, the number of new content nodes need to create is calculated using algorithm 1 based on equation (5). In algorithm 1, each data ID k hosted in the node is associated with the number of queries to it in previous time frames, denoted as n_k^{tp} and the average number of queries to it in the current time frame n_k^{tc} . The two values are obtained through data ID access history collection mechanism. The number of replicas to create contains two parts. The first part is the number of replica needed by the end of current time frame, which is calculated using n_k^{tp} and n_k^{tc} . The second part is the number of replicas needed for reducing the current workload in the node, which is calculated using the current queries waiting in the queue.

Algorithm 1 assumes homogeneous nodes in the overlay. It is not difficult for us to extend the algorithm to heterogeneous environment.

Algorithm 1: Data ID Selection and Replica Number Calculation

Input: $S = \{(ki, n_k^{tp}, n_k^{tc}) \mid ki \text{ is stored in the local node}\}$

c – the query processing capacity of the local node

q – the current request queue length, normalized by the query processing capacity

thr – the probability threshold

Output: (k, r) : k is the data ID to replicate, r is the number of replicas to create.

for each ki in S do

$$\Delta n_{ki} = n_k^{tc} - n_k^{tp}$$

$$r_{ki} = \lfloor \Delta n_{ki} / c \rfloor; m_{ki} = \Delta n_{ki} - r_{ki} \cdot c$$

$$P_{ki} = m_{ki} / c$$

if $P_{ki} > thr$

$$r_{ki} = r_{ki} + 1$$

end if

$$R = R \cup \{r_{ki}\}$$

end for

sort R in descending order

get r_{ki}^0 from R

$$r_{ki} = r_{ki}^0 + (q - c) / c$$

return (k, r_{ki})

5 Evaluation

To study the performance of the proactive replica creation mechanism on structured overlays, we implement a simulator using FreePastry 1.4.1. The overlay has the following parameters unless stated otherwise. It consists of 3000 nodes with ID randomly generated in 128 bit ID space. There are 200 data IDs published in the overlay and the total number of published data objects is 5000. Replicating a data ID incurs replica node negotiating cost, data dissemination cost and data processing cost. The average latency between two overlay nodes is set to 5 milliseconds. The average bandwidth between two overlay nodes is 10Mbps. The average data processing time including constructing a data table and populating its contents is 500 ms in average. Queries in an overlay network follow Poisson distribution. These queries can be sent from any node to published data IDs. Query sources are randomly selected. Queries select data IDs based on Zipf distribution with $\alpha = 1.0$. The query processing time is randomized between 10 to 100 ms. The time frame for counting queries is set to

100ms. The node capacity is set to 30 queries in the queue and workload change threshold is set to 30% of the capacity value. The probability threshold in algorithm 1 is set to 0.8.

A. Effect of the proactive CDN expansion mechanism

Our CDN expansion algorithm is adaptive to the changes of query arrival rate. In our experiment, queries arrive in three batches. The first batch contains 2000 queries with mean generating interval 2ms; the second batch contains 4000 queries with mean generating interval 1.4ms and the third batch contains 6000 queries with mean generating interval 0.8ms. These three batches of queries have an increasing demand to the CDN. Fig. 3. shows the overall query processing capacity change of the CDN that hosts the most popular data ID. The overall query processing capacity and the demands from incoming queries are normalized using the average query processing speed in a node. Fig. 3 shows that the CDN expansion mechanism is capable of identifying the popular data ID. The CDN expansion mechanism is also sensitive to the changes in query arrival patterns and it adapts to the demand of incoming queries well.

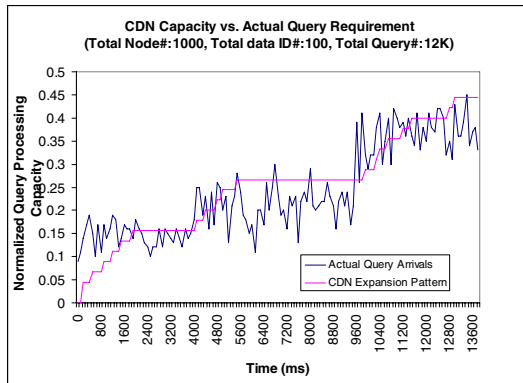


Fig. 3. Proactive CDN expansion vs. actual query arrivals: mean arrival interval- first 2000 queries: 2ms; next 4000 queries: 1.4ms; last 6000 queries: 0.8ms

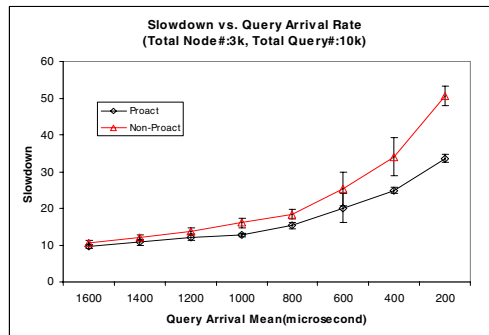


Fig. 4. Average query slowdown vs. query arrival rate

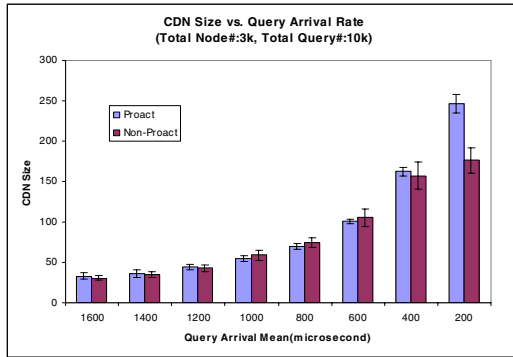


Fig. 5. CDN size vs. query arrivals

B. CDN performance comparison

To further quantify the effect of the data ID selection and replica number calculation mechanism, we compare the CDN performance under the proactive mechanism (*Proact*) to that under conservative CDN expansion mechanism (*Non-Proact*). A conservative mechanism adds one replica at a time when the CDN is overloaded. It is commonly used as an adaptive replication method. Fig. 4. compares the average query slowdown changes with the query arrival rates under the two mechanisms. The slowdown of a query is defined as below:

$$slowdown = \frac{query_processing_time + query_queuing_delay}{query_proc_time}$$

It is clear that *Proact* outperforms *Non-Proact* in terms that queries suffer shorter queuing delay. As the increase of query arrival rates, the slowdown gap also increases. *Proact* apparently creates replicas faster than *Non-Proact* does when the query arrival rate is high, as shown in Fig. 5. Note that under *Proact*, the average

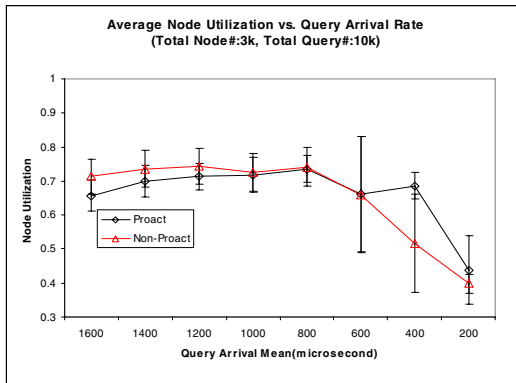


Fig. 6. Average node utilization vs. query arrival rate

query slowdown increases along with the query arrival rate. This is mainly due to that the cost of replica creation makes the CDN expansion speed can not keep up with the increasing query arrival rate. Even though *Proact* tends to create more replicas than *Non-Proact* does, their average CDN node utilizations are comparable as Fig. 6. shows, which means the proactive replica creation mechanism can efficiently control the CDN expansion speed.

C. Query forwarding hops

To study the load-balancing cost of our proactive mechanism, we compare the average query forwarding hops within a CDN under different replica creation mechanisms, namely *NonProact-Route*, *Proact-Route* and *NonProact-Random*. *NonProact-Route* uses conservative CDN expansion mechanism and new content node is selected based on *QueryStat* table discussed in section 3. *Proact-Route* uses proactive CDN expansion mechanism and new content node is selected based on *QueryStat* table. *NonProact-Random* mechanism uses conservative CDN expansion mechanism and new content node is randomly selected from the overlay ID space. Table 2 shows the average query forwarding hops in the CDN. It is clear that the proactive mechanism does not incur extra load-balancing cost.

Table 2. Query forwarding hops comparison

Mean Query Arrival	NonProact-Routes	Proact-Routes	NonProact-Random
1600 (microsecond)	0.183141	0.186367	0.2552
1200 (microsecond)	0.2392	0.194386	0.3304
800 (microsecond)	0.315944	0.276759	0.396439
400 (microsecond)	0.465948	0.416457	0.504721

6 Conclusion

We have presented a proactive content distribution method for data indexed DHT networks, which effectively addressed the access skew problem in such type of overlay networks. The method is capable of estimating the number of replicas needed for unevenly distributed demands to data IDs. Our simulation evaluation shows that the mechanism can significantly reduce the query slowdown when the query arrival rate is very high. Meanwhile, it uses resources efficiently by keeping the content node utilization reasonably high and load-balancing cost low.

References

1. Balakrishnan, H., Kaashoek, M.F., Karger, D.R., Morris, R., Stoica, I.: Looking up data in P2P systems. *Communications of the ACM* 46(2), 43–48 (2003)
2. Garces-Erice, L., Felber, P.A., Biersack, E.W., Urvoy-Keller, G., Ross, K.W.: Data indexing in Peer-to-Peer DHT networks. In: ICDCS 2004. Proc. of the 24th International Conference on Distributed Computing Systems, pp. 200–208 (2004)

3. Gummadi, K.P., Dunn, R.J., et al.: Measurement, modeling, and analysis of a Peer-to-Peer file-sharing workload. In: SOSP 2003. Proc. of the 19th ACM Symposium on Operating System Principles, pp. 314–329. ACM Press, New York (2003)
4. Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., Keleher, P.: Adaptive replication in Peer-to-Peer systems. In: ICDCS 2004. Proc. of the 24th International Conference on Distributed Computing Systems, pp. 360–369 (2004)
5. Wang, C., Alqaralleh, B., Zhou, B., Brites, F., Zomaya, A.Y.: Self-organizing content distribution in a data indexed DHT network. In: P2P 2006. Proc. of the 6th IEEE International Conference on Peer-to-Peer Computing, pp. 241–248. IEEE Computer Society Press, Los Alamitos (2006)
6. Rowstron, A., Druschel, P.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: SOSP 2001. Proc. of the 18th ACM Symposium on Operating System Principles, pp. 188–201. ACM Press, New York (2001)
7. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: SOSP 2001. Proc. of the 18th ACM Symposium on Operating System Principles, pp. 202–215. ACM Press, New York (2001)
8. Akamai (2006), <http://www.akamai.com>
9. Cohen, E., Shenker, S.: Replication strategies in unstructured peer-to-peer networks. In: Proc. of 2002 SIGCOMM conference, pp. 177–190 (2002)
10. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proc. of the 16th ACM Conf. on Supercomputing Systems, pp. 84–95. ACM Press, New York (2002)
11. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: A decentralized peer-to-peer web cache. In: PODC 2002. the 21st Annual ACM Symposium on Principles of Distributed Computing, pp. 213–222. ACM Press, New York (2002)
12. Yamamoto, H., Maruta, D., Oie, Y.: Replication methods for load balancing on distributed storages in P2P networks. In: SAINT 2005. Proc. 2005 Symposium on Applications and the Internet, pp. 264–271 (2005)
13. Xu, Z., Bhuyan, L.: Effective load balancing in P2P systems. In: CCGrid 2006. Proc. of 6th IEEE International Symposium on Cluster Computing and the Grid, pp. 81–88. IEEE Computer Society Press, Los Alamitos (2006)
14. Adamic L. A.: Zipf, Power-laws, and Pareto - a ranking tutorial. Available online, <http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html>
15. Adamic, L.A., Huberman, B.A.: The nature of markets in the World Wide Web. Quarterly Journal of Electronic Commerce 1(1), 5–12 (2000)