SEERDIS —— A DHT-Based Resource Indexing and Discovery Scheme for the Data Center

Zhilin Zhang[1] Junying Zhang[1]
[1] School of Computer Science & Technology,
Xidian University, Xi'an, China
E-MAIL: zhilin-zhang@hotmail.com

Qiuyan Huo[2] Jingyu Chen[2] Xuezhou Xu[2]
[2] School of Software Engineering
Xidian University, Xi'an, China
E-MAIL: xxz@mail.xidian.edu.cn

Keywords: resource indexing and discovery, DHT, Erasure Code, replication, data center

Abstract

A fundamental challenge in a data center is how to achieve high data availability and query efficiency at an appropriate storage cost. This paper presents a Distributed Hash Table (DHT) indexing and discovery scheme based on the single replication erasure code (Seerdis) for the Data Center. Seerdis gains high data availability with less storage space by using erasure code. In addition, the design of single replication in Seerdis can provide similar performance as replication schemes do for datacenter-based applications. Seerdis virtualizes each host into several nodes and each node corresponds to one data object with a high probability. Compared with existing resource indexing and discovery schemes, the benefits of Seerdis include increasing data availability and improving query efficiency.

1. INTRODUCTION

The emergence and evolution of commodity clusters and datacenters have enabled a new class of globally distributed high performance applications that coordinate over vast geographical distances. This type of application often requires the underlying storage system to keep pace with the explosive data growth and provide a quick and accurate data query. For example, Google's datacenters, which are distributed all over the world, need to cope with growing index data caused by growing data on the Internet and return expected results in an extremely short time.

Availability is the primary design goal of a storage system. High data availability and query efficiency are crucial for datacenter-based applications' success. It would be a commercial disaster for Facebook if its datacenters' failures resulted in loss of user data, or if it took hundreds of seconds to return expected data.

DHT-based storage systems support large scale data storage, fast data access, reliable data backup and replication and automatic data recovery with high availability, reliability and scalability. In the past ten years, DHT research has been preoccupied with peer-to-peer (P2P) file-sharing networks. Such networks are characterized by short node lifetimes and low key lookup rates per node. Recent research suggests that DHT is also fit for stable, high-capacity enterprise networks (Figure 1). Huang and Fox reported that DHTs are used for Yahoo user profiles and for Amazon catalog items [1].

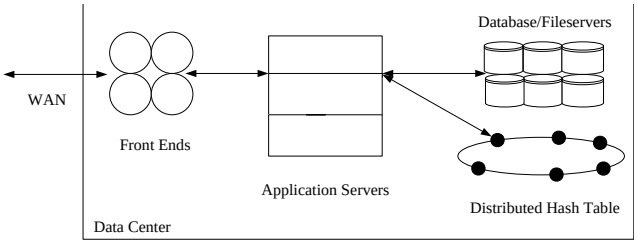

Figure 1. DHT in an enterprise datacenter [1]

In order to satisfy the requests for data storage and search for datacenters, this paper presents a DHT indexing and discovery scheme based on the single replication erasure code (Seerdis) approach. The main features of Seerdis are (i) each administrative domain [2] owns only one piece of data with a great probability; (ii) it supports lookups for resources that span multiple administrative domains without specifying domain names.

The underlying storage architecture of Seerdis is based on DHT. For each key, there is one replication and some fragments from Erasure Code's encoding in DHT. The single replication is used to provide good performance and fragments to improve data availability.

The design of Seerdis and its underlying storage architecture is based on the following two assumptions: (i) all the key-value pairs in the storage system need to be maintained with the same data availability; (ii) nodes in the system have high node availability, and these nodes will not join or leave as frequently as they will in P2P. Through observations of datacenters for banks and telecommunications industries, the two assumptions above are considered to be reasonable.

In summary, we have proposed a new type of resource indexing and discovery scheme that possesses two desirable features for the data center. This is the main contribution of this work, and has been confirmed by simulations and

experiments. A potential downside of our solution is that the number of indexing nodes of Seerdis is more than what it is in the traditional Chord. However, we believe this cost is well justified by its scaling, searching and fault tolerance features.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 presents the design of Seerdis. Section 4 presents an experimental evaluation and discussion, and Section 5 concludes this paper.

2. RELATED WORK

Based on the metadata storage scheme, we classify distributed storage systems for the datacenter into two kinds: (i) metadata are put on a centralized index server (metadata server), (ii) metadata are distributed to the entire network.

In the first case, the index server is in charge of indexing, searching and all metadata storage of the whole system. There are some implementations of this kind of distributed storage, GFS/BigTable [3], Ceph [4] and SRB [5], to name a few. Their typical query process is like this: the client sends a query request to the index server and receives some information on the location of data in the system, and then the client makes use of this information to get data. Since the metadata server has large data records, a high-loading network bandwidth, and disk I/O, it is easy to become the bottleneck of security and performance [6, 7]. Compared with such systems, Seerdis can get around all these problems because it doesn't use a centralized index server.

In the second case, the metadata's storage is decentralized in systems such as Amazon [8, 9] and Cassandra [10]. In this case, the routing is performed by using multiple index segments, each belonging to one node, collaboratively. Most of these systems are based on DHT, and therefore their indexing and searching schemes closely resemble those of the traditional Chord [11, 12]. Basically, the data redundancy schemes of all these systems are based on replication strategy. For instance, the number of copies in Dynamo [9] is 3. The redundancy scheme will cause rapid consumption of storage space for the datacenter-based applications in which data are stored in persistent storage. Seerdis can de-escalate the problem to a certain extent by using a hybrid redundancy scheme of single replication and erasure coding.

In addition, some recent work tries to apply one-hop DHT to data storage of the datacenter. Rission introduced one-hop DHT into the datacenter for reducing the network load [13]. Verma also used one-hop DHT to manage metadata for cloud computing applications [14].

As an index scheme developed by March et al., Midas [15] has an indexing structure similar to that of Seerdis. Their main focus has been on supporting efficient multi-attribute range queries on read-only DHT. In Midas, each key-value pair has multiple replications. Moreover, it belongs to a fixed node, but Seerdis, by contrast, stores only one replication for each key-value pair with some fragments to provide appropriate data redundancy. The design objectives of Seerdis's storage, indexing and searching are not limited to supporting multi-attribute range queries. Actually, we aim to provide a common DHT Service.

3. SEERDIS DESIGN

3.1. Overview of the Single Replication Erasure Code for DHT

The underlying storage architecture of Seerdis is Seed (**s**ingle r**e**plication **e**rasure code for **D**HT). For each key, its replication is distributed to the right place in Chord with consistent hashing, and the fragments resulting from the Erasure Code are stored in some other hosts with random distribution of restriction. Here, to avoid a single point of failure, it is not allowed that a host stores multiple pieces of data with the same key. Therefore, each host stores less than one piece of data for a specified key (a replication or fragment of this key). Figure 2 shows how key d is inserted into Seed when the number of fragments from the Erasure Code is 2.

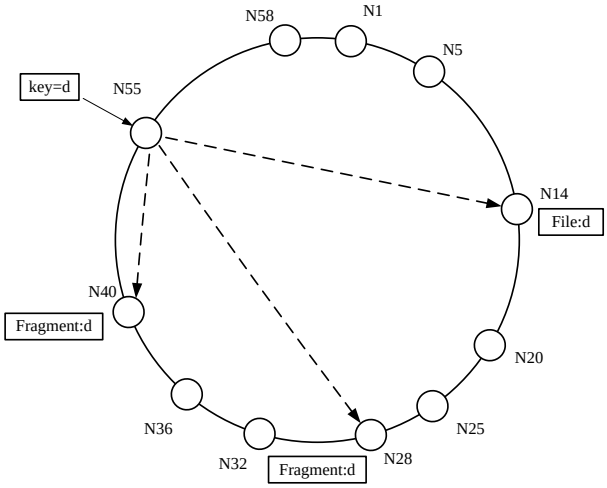

Figure 2. Insert key=d into Seed (M=2)

To achieve the aim that each administrative domain stores only one key identifier, Seed virtualizes each host into several nodes by associating node $n_{k,f,h}$ to each unique key identifier k owned by host h. Thus, Seed virtualizes h into $|T_h|$ nodes where $|T_h|$ denotes the set of unique key identifiers owned by h.

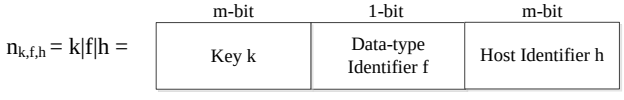

Figure 3. 2m+1-bit node identifier space

Seed uses two different identifier spaces: an m-bit identifier space for both keys and host identifiers, and a (2m+1)-bit identifier space for node identifiers. A (2m+1)-bit node identifier, denoted as k|f|h , is formed by concatenating an m-bit key identifier k, a 1-bit data type

identifier f and an m-bit host identifier h (Figure 3). Here, f=0 indicates that the host stores a replication and that f=1 stores a fragment. The (2m+1)-bit identifier space prevents collisions between node identifiers when several hosts share the same key identifier.

In Chord, the probability of two hosts or keys hashing to the same identifier can be negligible when the identifier length m is large enough [12]. It is because each node in Seed corresponds exactly to one administrative domain, while the probability that two key identifiers are the same is a very small number, with each node corresponding to one data object with a high probability. Thus, there are two kinds of nodes: replication nodes and fragment nodes, identified with f in the (2m+1)-bit node identifier.

3.2. Construction of Indexing

This paper introduces into Seerdis the concept of the segment [15]. Seerdis organizes nodes as a Chord ring and divides the ring into segments based on the key identifiers. Segment S_k , corresponding to the portions which are relevant to k in the Chord ring, consists of all nodes whose key identifier is k in Seerdis. By using the (2m+1)-bit node identifier, the replication node's always occupying the starting position of its segment is ensured, and the nodes belonging to the same segment are adjacent to each other. Figure 4 illustrates segment S_k in the Chord ring when the number of fragments for k is 2.

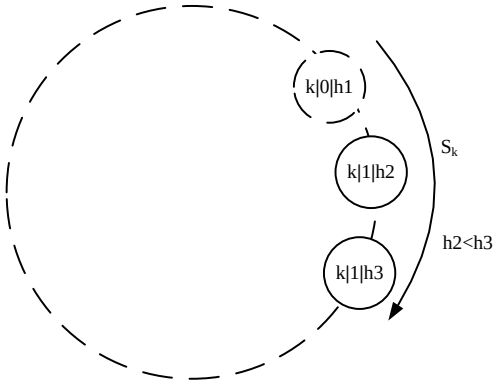

Figure 4. Segment S_k in Chord (M=2)

With segment addressing, Seed supports the flat-naming scheme. In addition, segment-based organization also lends its support to increase the query accuracy whereby key k can still be found even though some nodes in segment S_k fail. This is because S_k consists of nodes with the same prefix k but a different data type or suffix (belonging to different hosts).

The procedure of Seerdis's indexing construction can be generally divided into three steps: (i) resources are stored on the appropriate hosts according to consistent hashing and random distribution of restriction; (ii) hosts are virtualized into nodes based upon the key identifier, the data type and the host identifier of each key;(iii) Seerdis organizes all the nodes into a Chord ring according to the size of node identifiers.

Figure 6 shows the mapping scheme of Seed and the process of constructing a Chord ring by Seerdis eventually. In this example, host A owns two unique keys, i.e. $T_A = \{2,9\}$, and both are fragment data, and thus we virtualize host A into two nodes: node 2|1|A and node 9|1|A. Similarly, we virtualize host B, host C and host D into one node 5|1|B, three nodes 2|1|C, 5|1|C and 9|1|C and three nodes 2|0|D, 5|0|D and 9|0|D, respectively. So the ring overlay of Seerdis consists of three segments, with S2 consisting of node 2|0|D, 2|1|A and 2|1|C, segment S5 consisting of node 5|0|D, 5|1|B and 5|1|C, and segment S9 consisting of 9|0|D, 9|1|A and 9|1|C. These segments represent the three keys: 2, 5 and 9.

The number of indexing nodes in the mapping scheme of Seed is up to $1 + M$ times greater than what it is in the traditional Chord when without hashing more than one resource to the same identifier, for example, using SHA-1 as the hashing function. These additional indexing nodes result in extra cost of Seerdis's management and maintenance.

To improve query accuracy due to node failures, Seerdis nodes maintain backup fingers [12] for each entry in the finger table so that when a finger fails, i.e., it points to a disabled or non-existing node, we promote a backup finger to replace failed fingers (Figure 5). A possible implementation of backup fingers is to update the finger periodically [12]. When n corrects its finger f, in addition to successor(f), the correction process also returns the fingers that share the same prefix as f.

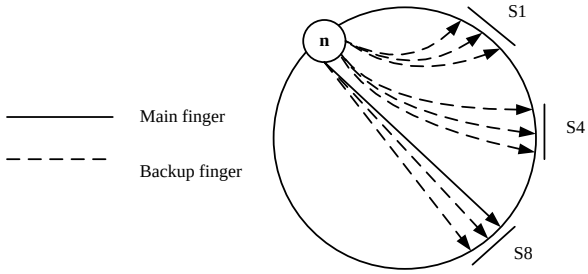

Figure 5. Backup fingers

3.3. Lookup Services

3.3.1. Routing

In a system with N hosts and K unique key identifiers, the lookup path length in Seed is $O(\min(\log K, \log N))$ hops. The lookup algorithm is based on Chord [11, 12] lookup in which (i) every node n maintains a finger table consisting of $O(2m)$ entries, and (ii) the i^{th} entry (finger) points to the successor of $(n + 2^{i-1})$. However, we improve the basic Chord lookup by exploiting the Seed key-to-node mapping scheme:

(1) Routing by segments: To locate key k, we route a lookup request from one segment to another. Each routing step halves, in terms of segments, the distance to the destination segment S_k. Thus, the lookup path length is $O(\log K)$.

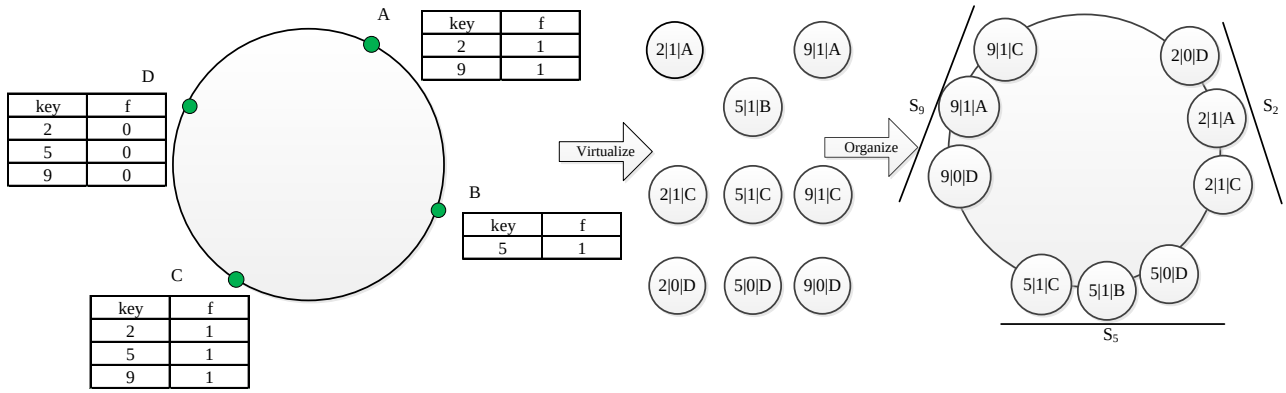

Figure 6. Seed virtualizes four hosts into nine nodes and organizes them into a Chord ring (M=2)

(2) Shared finger tables: To limit the lookup path length at $O(logN)$ hops even when K > N, each routing step utilizes all the $|T_h|$ finger tables maintained by host h. In other words, a node's finger table is shared by all nodes belonging to the same host. Thus, visiting one host is equivalent to visiting all the nodes corresponding to the host. Since the maximum distance between two segments is $O(N)$ hosts, it takes $O(logN)$ hops to locate a segment.

3.3.2. Searching Scheme

Seerdis routes each query (key=k) to any node in segment S_k. Then the nodes take appropriate actions. The main steps of the search algorithm are as follows:

(1) If there is a replication node in S_k, query is routed to the node, and then go to (2);
(2) If the service performance of the host on which the replication node locates is good, the host will return the expected result, else go to (3);
(3) If there is no replication node for improving service performance in S_k, or if the performance value of the host owning the replication node is below a threshold value while a replication node exists in Seerdis, the query is forwarded to the node which provides the best performance in S_k, then the node exploits the fragments to regain a complete data object for k and returns the data object to the initiator. When there exists no replication node in S_k, the obtained data object needs to be sent to an appropriate host which will store this replication.

The primary design goal of Seerdis's search algorithm is to get data at as low a cost as possible. In other words, when the cost that the host storing a replication node affords to return data is lower than that of recovering a complete data object from fragments, Seerdis chooses to return the replication directly. This strategy is based on the following two factors: (i) the transmission time in datacenters is quite short because of their high bandwidth; (ii) time for retrieving a data object by the Erasure Code is much longer than that for transmitting the data object.

From our observations, when parameters of the Erasure Code is set as K′=4 and M=8, the average decoding rate of the optimized Erasure Code [16] is about 20MBps in common PC. By contrast, the bandwidth in datacenters using VPN is up to 10Gbps. It is reported in [17] that 10Gbps network can achieve a 6.99Gbps throughput by using some better NICs and switches and setting an appropriate TCP window size and MTU. Consequently, Seerdis should avoid retrieving data by the Erasure Code to the greatest extent.

4. EVALUATION

In this section, we modify the Chord simulator [11] to evaluate our protocols and algorithms. Seerdis, compared with the traditional Chord, is implemented by using Seed as the underlying overlay topology. The simulation consists of queries for single-key and multi-attribute range queries which use incremental search [15]. Using simulation, we evaluate two metrics: lookup resiliency and query cost. Unless stated otherwise, our experiments use the following settings:

- K', the number of blocks of encoded source data equals the minimum number required to reconstruct the source data by the Erasure Code, is 4.
- M, the number of blocks of encoded data, is 8.
- m, the number of bits for keys and host identifiers, is 18-bit.
- K, the number of unique keys. We choose it to be 10,000 and 50,000 in simulation.
- N, the number of hosts, is 1000.
- For multi-attribute range queries, the number of attributes per resource is 4 and each range query covers 1% of the 4-dimensional attributes space. Such query selectivity has also been used in other works [18, 19].

To facilitate the following description, we give two definitions here:

1. **Redundancy factor** [20]: Redundancy factor is defined as the ratio of storage space required for reaching specified data availability to that of one replication.
2. **Lookup resiliency** [15]: Lookup is resilient when it is able to locate keys in the presence of node failures.

Thus, we can easily determine the redundancy factor in Seed to be is 3 ($= 1 + M/K'$). For fairness, there are three replications in Chord for each key-value pair, and Chord applies a backup and recovery strategy similar to that by Dynamo.

Lookup resiliency is primarily shaped by the two main factors:(i) data redundancy---the higher the redundancy, the higher the availability when there exists node failure in the system; (ii) index maintenance efficiency---the sooner an index reflects changes in the underlying overlay network, the higher accuracy for queries can be obtained.

4.1. Resiliency to Node Failures

4.1.1. Single-Key Lookup

For different percentages of node failures (F), we simulate 200,000 single-key queries. Table 1 shows the different lookup resiliencies in Seed and Chord.

Table 1. Lookup resiliency for single-key queries

Type	F / K	Keys Retrieved			
		5%	10%	20%	40%
Chord	10000	0.94289	0.88616	0.77062	0.50542
	50000	0.94252	0.88585	0.76989	0.50542
Seed	10000	1.00000	0.99502	0.98977	0.98013
	50000	1.00000	0.99498	0.98979	0.97972

Our results show that lookup resiliency is higher in Seed and that nearly all 200,000 single-key queries gain data. We put all this down to the design in Seed because each node is responsible only for its own keys. When a node fails, then only its own keys are affected. And by the routing design based on segments and the backup fingers design, Seed can locate a key as long as there is a replication of this key or enough fragments to retrieve a complete data object. On the other hand, due to the serialization in simulation, hosts would not fail in parallel. As a result, none of the data objects would be irrecoverably lost when hosts fail. Moreover, to speed the simulation up, some tricks in implementation are adopted, which improve the lookup resiliency. However, because of a fairly low node fail rate in the datacenter, the negative influence on the analysis in resilience created by those tricks is very limited. The same effect also appears in the analysis of multi-attribute range queries.

4.1.2. Multi-Attribute Range Queries

Table 2 shows the lookup resilience of 50,000 range queries for different percentages of node failures (F). We have obtained the results in Table 2 by averaging results for five different kinds of query attribute selectivity when each range query covers 1% of the 4-dimensional attribute space.

Table 2. Lookup resiliency for multi-attribute range queries

Type	F / K	Keys Retrieved			
		5%	10%	20%	40%
Chord	10000	80.55136	76.21083	67.13556	48.83678
	50000	80.55079	76.21083	67.13552	48.83672
Seed	10000	84.84573	84.83546	83.54628	81.64728
	50000	84.84574	84.83574	83.44532	80.34734

From Table 2 we can find out that the number of available keys retrieved in Seed is nearly constant as the percentage of node failures increases from 5% to 40%, in comparison with the fact that in Chord the number of the available keys retrieved is nearly halved. Thus, we can conclude that lookup resiliency in Seed is also much higher than that in Chord for multi-attribute range queries.

The reasons for this situation are the same as those in single-key lookup, which includes two main aspects: (i) the consequences of node failures are limited. In fact, each node can only affect its own keys; (ii) Seed can regain a complete data object by exploiting their fragments for those keys whose replication nodes have failed. It is because of the desirable data availability that Seed can retrieve more available keys than Chord in multi-attribute range queries.

4.2. Cost of Query Processing

4.2.1. Single-Key Lookup

For single-key lookup, the cost of query processing depends on index efficiency: the more quickly an index locates a key, the fewer nodes the query will visit. Table 3 shows the average number of nodes visited per query when the total number of single-key lookup is 20,000 in Seed and Chord.

Table 3. Average number of nodes visited per query for single-key lookup

Type	F / K	Node Visited			
		5%	10%	20%	40%
Chord	10000	5.9285	6.0327	6.2643	6.7193
	50000	5.9212	6.0365	6.2629	6.7192
Seed	10000	4.2030	4.2030	4.2030	4.2030
	50000	4.2059	4.2059	4.2059	4.2059

Table 3 shows that the average number of nodes visited per query in Chord increases as the percentage of node failures raises. This is because a higher percentage of node failures bring about more disabled fingers in the finger table, which has a negative effect on the index efficiency. However, Seed visits a constant number of nodes per query. We attribute this to localizing keys precisely in Seed by using backup fingers besides the effect of implementation in simulation as mentioned above. For a determined

percentage of node failures, the probability of all backup fingers for one finger becoming invalid is rather low, which leads to the fact that Seed processes a query with less hops.

4.2.2. Multi-Attribute Range Queries

For multi-attribute range queries, aside from index efficiency described earlier, the cost of processing is affected seriously by two other factors: the number of search keys per query, and the number of available keys per query.

March stated that the number of search keys per query is constant [15]. In our experiments, it is obvious from the results given above that the available keys in Seed are more than those in Chord when the percentage of node failures is the same, the reason being chiefly that when we use the Erasure Code as the redundancy scheme, the data availability is much higher than that by using replication with the same redundancy factor [21, 22].

Table 4 shows the average number of nodes visited per query for multi-attribute range queries in Seed and Chord under the experimental conditions identical to those in Table 2.

Table 4. Average number of nodes visited per query for multi-attribute range queries

Type	F / K	Nodes Visited			
		5%	10%	20%	40%
Chord	10000	37.9021	36.8211	34.2016	28.9509
	50000	37.9021	36.8211	34.2015	28.9509
Seed	10000	223.5635	223.5635	223.5635	223.5635
	50000	223.5635	223.5635	223.5635	223.5635

Table 4 indicates that the average number of nodes visited per query in Chord decreases as the percentage of node failures rises. We attribute this to the impact of available keys and search keys. With the number of available keys decreasing as the percentage of node failures increases when the number of search keys per query is constant, the quantity of visited nodes is reduced due to the fact that the incremental search algorithm [15] avoids initiating a DHT lookup for those unavailable keys in search keys.

Seed has a larger average number of nodes visited per query than that in Chord, which results mainly from the following two reasons: (i) more available keys cause Seed to initiate more DHT lookups while the number of search keys is constant; (ii) in Seed, a large number of range queries may cover the same key, which degrades the performance of the host storing a replication of this key and results that Seed needs to utilize some fragments to retrieve a complete data object for the query initiator. On the other hand, many of the hops happen between two different nodes from a same host, and these hops have a little routing cost in the actual routing. As a result, the actual average number of nodes visited per query is much smaller than that in Table 4. Table 5 presents the ratio of the number of hops between two different nodes from a same host to the average number of nodes visited for multi-attribute range queries in table 4.

Table 5. The ratio of hops between two nodes from the same host to the total hops

F / K	1	2	3	4	5
10000	0.280484	0.229824	0.24028	0.237762	0.228476
50000	0.280484	0.236464	0.24028	0.231086	0.228476

5. CONCLUSIONS

In this paper, we have presented Seerdis, a DHT indexing and discovery scheme based on the single replication erasure code for the Data Center. Seerdis improves query performance by directing lookups only to administrative domains that own the requested resources, and constructs a secure and efficient indexing architecture for the hybrid redundancy scheme of single replication and Erasure Code.

Through simulation, it is shown that Seed-based Seerdis compensates its higher overhead with a smaller average path length (hops) and a smaller number of failed lookups. Seerdis can perform the query task accurately and efficiently even though the percentage of node failures is rather high. In addition, Seerdis is more tolerant to the single point of failure. Thus, Seerdis can meet the datacenters' requirements of data storage reliability and searching efficiency well.

In recent years, many large data centers are being built to provide increasingly popular online application services, such as search, e-mails, IMs, web 2.0, and gaming, etc. Reliable data storage and efficient data retrieval is key to the success of the applications based on datacenters, which own stable networks and high bandwidths. Seerdis take the advantages of long lifetime and high availability of nodes in datacenters to meet data availability demands with Erasure Code, which can save storage space greatly. In addition, the design of single replication in Seerdis can provide the similar performance as replication schemes do for the applications.

Acknowledgment

We would like to thank V. March for his selfless help.

References

[1] A. C. Huang and A. Fox, 2004, "Cheap recovery: a key to self-managing state." ACM Trans. on Storage 1 (1) (2004): 38-70.
[2] Y. M. Teo, V. March, and X. Wang. 2005. "A DHT-based grid resource indexing and discovery scheme." In Proc. of Singapore-MIT Alliance Annual Symposium, Jan. 2005.

[3]Fay Chang , Jeffrey Dean , Sanjay Ghemawat , Wilson C. Hsieh , Deborah A. Wallach , Mike Burrows , Tushar Chandra , Andrew Fikes , Robert E. Gruber. 2006. “Bigtable: a distributed storage system for structured data.” Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation, p.15-15, November 06-08, 2006, Seattle, WA
[4] Sage A. Weil , Scott A. Brandt , Ethan L. Miller , Darrell D. E. Long , and Carlos Maltzahn. 2006. “Ceph: a scalable, high-performance distributed file system.” Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation, p.22-22, November 06-08, 2006, Seattle, WA
[5] C. Baru, R. Moore, A. Rajasekar and M. Wan Michael. 1998. “The SDSC Storage Resource Broker.” Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research(CASCON’98), Toronto, Canada, 1998, pp.5-17.
[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. “Bigtable: A Distributed Storage System for Structured Data.” ACM Transactions on Computer Systems, Vol. 26,No. 2, Article 4, June 2008.
[7] Chun-Ting Chen, Chun-Chen Hsu, Jan-Jan Wu, and Pangfeng Liu. 2009. “GFS: A Distributed File System with Multi-Source Data Access and Replication for Grid Computing.” GPC 2009, LNCS, vol. 5529, pp.119–130, N. Abdennadher and D. Petcu (Eds.), May 2009.
[8] http://s3.amazonaws.com
[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. 2007. “Dynamo: amazon’s highly available key-value store.” SOSP, 2007.
[10] A. Lakshman and P. Malik. 2010. “Cassandra: a decentralized structured storage system.” SIGOPS Oper. Syst. Rev., 2010.
[11] The Chord Project,
http://www.pdos.lcs.mit.edu/chord/#downloads
[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. 2001. “Chord: A scalable peer-to-peer lookup service for internet applications.” In Proc. of ACM SIGCOMM, Aug. 2001, pp. 149–160.
[13] John Risson, Aaron Harwood, and Tim Moors. 2006. “Stable high-capacity one-hop distributed hash tables.” In ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications, pages 687-694,Washington, DC, USA, 2006. IEEE Computer Society.
[14] https://www.ideals.illinois.edu/bitstream/handle/2142/14820/paper.pdf?sequence=2
[15] V. March and Y. M. Teo. 2006. “Multi-Attribute Range Queries on Read-Only DHT.” Proc. of the Intl. Conf. on Computer Communications and Networks, pp. 419-424, 2006.
[16] James S. Plank. 2005. “Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Storage Applications.” Technical Report CS-05-569 Department of Computer Science University of Tennessee. December, 2005.
[17] http://www.networkworld.com/community/node/58789
[18] P. Ganesan, B. Yang, and H. Garcia-Molina. 2004. “One torus to rule them all: Multi-dimensional queries in P2P systems.” In Proc. of the 7th Web DB, June 2004, pp. 19–24.
[19] C. Schmidt and M. Parashar. 2003. “Flexible information discovery in decentralized distributed systems.” In Proc. of the 12th HPDC, June 2003, pp.226–235.
[20] Wu, F. and Qiu, T. and Chen, Y. and Chen, G.. 2005. “Redundancy Schemes for High Availability in DHTs.” In Proceedings of the 3rd Intl. Symposiumon Parallel and Distributed Processing and Applications (ISPA), 2005.
[21] H. Weatherspoon and J. Kubiatowicz. 2002. “Erasure Coding vs. Replication: A Quantitative Comparison.” In IPTPS, 2002.
[22] R. Rodrigues and B. Liskov. 2005. “High Availability in DHTs: Erasure Coding vs. Replication.” IPTPS, 2005.

AUTHOR’S BIOGRAPHY

Zhilin Zhang was born in 1987. He attended Xidian University from 2005 to 2009, specializing in computer science and technology. In 2009, he received the bachelor degree in computer science from Department of Computer Science, Xidian University, China. In the same year, he was a recommended student for admission to Xidian University and Zhejiang University, and he received an offer to study for a PhD degree in HKUST. Now he is a MA candidate of Department of Computer Science, Xidian University. His research focuses on distributed system, reliable data storage and high performance computing.