Journal of Computers Vol. 28, No. 4, 2017, pp. 189-196
doi:10.3966/199115592017082804020

A Distributed Resource Access Algorithm Based on DHT

Xiaotao Wei[1*], Taifeng Cao[1], Wenwei Wu[1], and Lei Su[2]

[1] School of Software Engineering, Beijing Jiaotong University,
Beijing, P. R. China
{weixt, 16121728, 16126221}@bjtu.edu.cn

[2] Leshi Internet Information and Technology Corp., Beijing,
Yao Jiayuan Road 105, Leshi building, Beijing, P. R. China
sulei1@le.com

Received 18 October 2016; Revised 13 June 2017; Accepted 26 June 2017

Abstract. Distributed storage system plays an important role in cloud computing. Its performance directly affects the efficiency of cloud computing system. To maintain load balancing among all the data servers is the key to achieve high performance of distributed storage system. In this paper, a novel algorithm named dynamic greedy algorithm is proposed to improve the data access efficiency of distributed storage system by keeping data servers load balancing. The algorithm can generate an optimized reading scheme when downloading data from distributed storage system. It takes into account not only the speed of data servers but also the current load of data servers. This algorithm is evaluated in a distributed storage system model using the dynamic hashing table mechanism. Experimental results show that compared with greedy algorithm, the proposed algorithm can generate better scheme in data reading operations and significantly improve the efficiency of distributed storage system.

Keywords: distributed storage system, dynamic greedy algorithm, load balancing

1 Introduction

Distributed storage system is one of the most important part in cloud computing. With the explosive growth of information to be handled, more efforts should be taken to improve the performance of distributed storage system. Load balancing plays a significant role in this field. Traditionally, there are two methods to implement the load balancing in distributed storage system. One is to provide new distributed storage architectures or make improvement on existed architectures. The other is to improve the load balancing algorithm for certain architectures.

The architecture is the foundation of the distributed storage system. So, improvements on architecture can make optimization from strategy level. Florence and Shanthi proposed an improved architecture and employed the firefly algorithm for load balancing [1]. They associate the node and the server, so that there is sufficient information for the access plan optimization. Nikita tried to make partition in the cloud environment, and assigned a master server to manage the load balancing in the cluster [2]. In addition, they improved the efficiency of data access by introducing the Bloom Filter algorithm. Belkhouraf et al. [3] developed a semi-centralized and multi cluster architecture, which can handle huge number of clients, requests, and deal with big data.

Besides providing new distributed storage architectures, researchers also try to optimize the existed architectures by improve the load balancing algorithms. Alam and Varshney used the Folded Crossed Cube Network in homogeneous multiprocessor system [4]. The algorithm achieves a global load balancing in the whole storage system by reducing the number of the maximum parallel tasks. Wu, Song and Gong proposed a time-based load balancing algorithm. The algorithm optimizes storage job scheduling by generating a load balancing table [5]. Wang, Liu, Chen, Xu and Dai proposed a scheduling

* Corresponding Author

algorithm which is based on adaptive time span and genetics algorithm [6]. They used variance to construct a load-dense, multi-weight fitness function between different nodes. And the task sequence can be optimized and the load balancing of the storage system is achieved. Shahapure and Jayarekha proposed an optimal cost based scheduling algorithm by constructing a service list [7]. Xu, Zhao, Hu and Hu proposed a Job scheduling algorithm based on Berger model [8].

In this paper, a dynamic greedy algorithm is proposed to improve the data reading performance of distributed storage system. It can make an optimized data reading schema which fully takes advantage of faster data servers while still makes these data servers load balance. Experiments show that this algorithm outperforms the greedy methods.

The rest of the paper is organized as follows. In the next section, we briefly introduced the related research. Section 3 describes the dynamic greedy algorithm. Section 4 shows the experiments and results. Then we make the conclusion in section 5.

2 Related Works

In the traditional distributed storage system, the resource access algorithm comes with some of their existing challenges, especially in the load balancing aspect. The first step in accessing the data is to obtain the list of blocks of data, the list of replicas of each block and the data node location using management node. The second step is to select the data node which has the shortest access time, and then read the data [9]. In other words, the reading algorithm of traditional distributed storage system is based on the greedy algorithm. The use of greedy algorithms may cause load imbalance on the data server. In order to solve the problem, the domain experts put forward various solutions.

Ma, Yang and Mi proposed two algorithms which are based on the Flow Table distributed memory architecture [10]. They are the static threshold algorithm and the dynamic threshold algorithm. In the distributed storage system based on master-slave model, static threshold algorithm is a way that determines the load status by the number of flows in N switches and the critical load of the node. In contrast, dynamic load algorithm is a way that determines the load state by the number of flows in the N switches. The authors improve the hash algorithm by adding the two load states. In this way, the authors can make the distributed storage system load balance.

Mallikarjuna and Venkata Krishna proposed an infiltration method to solve the load balancing problem of virtual nodes in cloud storage [11]. This method is used to allocate storage tasks of virtual nodes reasonably. This load balancing method is based on three agents. They are authentication agents, infiltration agents and redistribution agents. The role of the authentication agent is to determine the load status of each virtual node. To complete this work, the authors proposed an identification algorithm. The role of the infiltration agent is to schedule tasks from high-load nodes to low-load nodes. In order to complete this task, the authors proposed a penetration algorithm. The role of the redistribution agent is to reallocate tasks to a suitable virtual node. Different from the infiltration agent, redistribution agent is based on the routing algorithm. The cooperation of the three agents can achieve the load balancing of the distributed storage system and improve the storage efficiency.

Liu, Xu and Chen proposed a load balancing algorithm based on the virtual machine migration architecture [12]. This architecture includes the acquisition model, monitoring model, forecasting model and selection model. The load balancing algorithm can be divided into three parts. The first part is to determine whether the storage node is overloaded. The judgment is based on the CPU utilization, memory usage and other data. These data are collected by the collection model. The second part is to predict the possibility of overload node at the next time. This part is based on exponential smoothing algorithm. The third part is the selection of the migrated data node and the target node. This part calculates the appropriate data migration strategy by the results from the first two sections. This approach makes the distributed storage systems load balancing.

Tao proposed a cloud load balancing algorithm which is based on copy factors [13]. The copy factor is the selection criteria for the data server. The data access queues in the meta server are optimized by computing the unique checksum and theoretical storage time. In this way, an automatic dynamic load balancing strategy can be implemented and the storage efficiency of the distributed storage system can be improved.

Shao, Jibiki, Teranishi and Nishinage proposed a load balancing strategy which is based on virtual node. In the authors' model, each physical node corresponds to a virtual node [14]. And these virtual

nodes can be queried in P2P network. Load balancing is conducted in both overlay level (between neighboring virtual nodes) without global knowledge and physical level (among physical nodes) with limited global knowledge. In addition, the authors proposed a load balancing exchange algorithm which can make each virtual machine in a reasonable load range.

In summary, improvements are made from both architecture level and load balancing algorithm. But seldom is done on how to generate a better data reading schema while downloading files from distributed storage systems. This paper focuses on how to improve the overall downloading speed of the distributed storage system and proposes a novel algorithm to solve this problem. The algorithm is given below and its performance is proved by experiments.

3 The Dynamic Greedy Algorithm

The algorithm we propose to improve the efficiency of data reading operations in distributed storage system is named dynamic greedy algorithm (DGA). It is implemented in a model that employs the dynamic hashing table (DHT).

3.1 DHT Storage Architecture

In distributed storage system, large files are divided into smaller data blocks with equal size. These blocks are duplicated to multiple copies and stored in different data servers. In the DHT storage model, multiple copies are distributed to date servers by a consistent hashing algorithm. As shown in Fig. 1, suppose there is a ring of address space, each point on the ring is an integer value between 0 and 2^23-1 which corresponding to 2^23 virtual nodes. If there are m physical data servers in the cluster, we map these data servers to *m* virtual nodes named management nodes. When data is to be stored, a data block is firstly mapped to a virtual node by the consistent hashing algorithm. Then the virtual node is linked to the nearest management node clockwise around the ring. Thus, a mapping from the data block to the data server is created. We use a mapping table to keep the relationships between data blocks and data servers. By querying the mapping table, we can get the locations of every replica of a data block in the cluster.

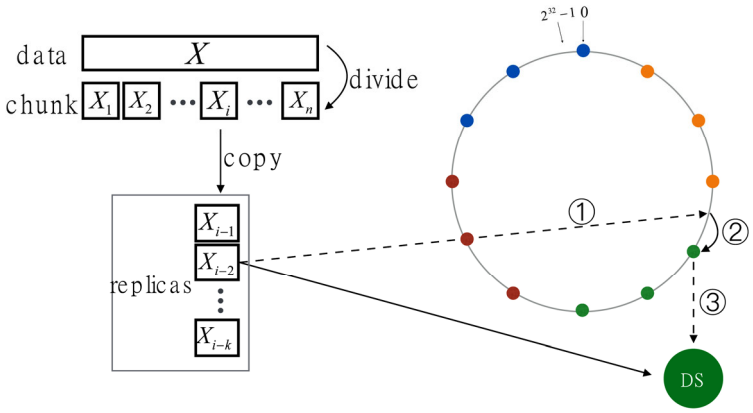

Fig. 1. DHT storage model

For example, data X is split into *n* data blocks, $X=\{X_1,X_2,...,X_n\}$, each of X_i is duplicated to *k* replicas $(X_{i-1},X_{i-2},...,X_{i-k})$. According to the mapping table, the distribution of data replicas in the data servers can be represented as a matrix P.

$$P=\begin{pmatrix} P_1,1 & P_1,2 & \cdots & P_{1,m} \\ P_2,1 & P_2,1 & \cdots & P_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n,1} & P_{n,2} & \cdots & P_{n,m} \end{pmatrix} \quad (1)$$

The row vector $P_i=(P_{i,1},P_{i,2},...,P_{i,m})$ represents the distribution of replica of the *i*th data block in *m*

data servers. If there is a copy of the *i*th data block stored in the *j*th data server, then $P_{i,j}$ is 1. Otherwise $P_{i,j}$ is 0. The summation of the *i*th row $\sum_{j=1}^{m} P_{i,j}$ represents the number of copies that data block X_i is duplicated, which is *k*, the copy factor. The summation of the *j*th column $\sum_{i=1}^{n} P_{i,j}$ indicates the total number of data blocks that are stored in the *j*th data server.

3.2 Description of DGA

When a client requests a data X from the distributed storage system, the master server should generate a reading schema by some algorithm with the mapping table. This schema tells the client the addresses of data servers where to read every data blocks. If this algorithm makes the workloads of data servers well balanced, then the high efficiency of data reading operation can be achieved. In fact, since the response time of each data server is different, the time cost to read the whole data depends on the time it takes for the slowest node returning its data blocks, as shown in formula 2.

$$T = \max_j \{\sum_{i=1}^{n} \min_j (T_j \cdot P_{i,j})\}. \quad \textbf{(2)}$$

Where T_j is the response time for the *j*th data server to return a requested data block. This information can be collected periodically from the data servers.

Normally the load balancing strategy for data reading is different from writing. While writing data, the distributing algorithm puts emphasis on balancing and spreading, without considering the response times of data servers. But when reading data, the algorithm should focus on I/O efficiency. So a commonly used strategy is the greedy algorithm. That means we always read data blocks from the faster data servers if there is a requested data block replica available in this node. This method will obviously cause overloading of the faster data servers. In fact, it may even worse than we just randomly chose a data server containing the required replica. Random choosing strategy may also balance the burden of data servers when reading data. But it does not take full advantage of the faster data servers to improve the I/O efficiency. So we propose a new algorithm named dynamic greedy algorithm to generate the reading schema. It is an improved version of greedy algorithm. It maintains a dynamic priority queue of data servers based on both the response time and current work load of data servers.

Suppose there exists an optimization reading schema J=(j_1, j_2, j_3,⋯, j_n), where j_i is a number meaning that data block X_i should be read from data server j_i. And this solution should satisfy the following two limitations.

$$\text{minimize}_j \sum K_j \cdot T_j. \quad \textbf{(3)}$$

$$\text{subjecct to} \sum_{i=1}^{m} K_m^{(n)} = n. \quad \textbf{(4)}$$

$$K_1^{(n)} \cdot T_1 = K_2^{(n)} \cdot T_2 = \cdots = K_m^{(n)} \cdot T_m. \quad \textbf{(5)}$$

Where $\overrightarrow{K^{(n)}} = (K_1^{(n)}, K_2^{(n)}, \cdots, K_m^{(n)})$ represents the number of data blocks read from each data server. $K_j^{(n)} \cdot T_j$ is the total time for the jth data server to return all the requested data blocks.

Now we give the procedure to generate an approximately optimized reading schema. Initially, $\vec{J} = \vec{0}$, $\overrightarrow{K^{(0)}} = \vec{0}$. Vector $\vec{T} = (T_1, T_2, \cdots, T_m)$ represents the response time of m data servers. We scan each row of matrix P from the first row down to the last row. For the *i*th row, from vector $P_i = (P_{i,1}, P_{i,2}, ..., P_{i,m})$ we can find the currently fastest data server holding data block X_i. It has the property of $\min_j \{T_j \cdot (K_j^{(i)} + 1)\}$, $P_{i,j} = 1$. We find the *j* and save it to j_i and add 1 to $K_j^{(i)}$. Then continue to the next row. This procedure repeats until the last row is processed.

After the iteration completed, we get the vector J=(j_1, j_2, j_3, ⋯, j_n) that tells the client for each data block X_i we can get it from data server j_i. And the total time cost for reading data X is

$$\max imize_j K_j^{(n)} \cdot T_j. \tag{6}$$

$$\text{subjecct to } \vec{T} = (T_1, T_2, \cdots, T_m). \tag{7}$$

$$\overrightarrow{K^{(n)}} = (K_1^{(n)}, K_2^{(n)}, \cdots, K_m^{(n)}). \tag{8}$$

The pseudo code of dynamic greedy algorithm is described as follows.

```
Algorithm 1. DGA

  Input:
    T[m], the up-to-date response time of m data servers.
    P[n][m], the replicas vs. data servers mapping matrix.
Output:
    J[n], the data reading schema.
    J[n]={0};
    K[m]={0};
    i=0;
    while i < n do
       min time=0;
       min_pos=-1;
       for j=1 to m
          if P[i][j]  then
             if !min_time  then
               min_time=T[j]*(K[j]+1);
                min pos=j;
             elseif  min time > T[j]*(K[j]+1)  then
                min_time= T[j]*(K[j]+1);
                min pos=j;
             end if
          end if
       end for
       J[i]=min pos; K[min pos]++;
       i++;
    end while
```

4 Experiments and Analysis of Results

In order to evaluate the performance of our proposed algorithm we set up a small cloud environment with 8 servers. The configuration of servers is shown in Table 1.

Table 1. The configuration of servers

	Master server	Sub server
Number	2	6
CPU	Intel Xeon E5-2670×4	Intel Xeon E5-2620×2
Memory	64.00GB RAM	32.00GB RAM
Hard disk	256GB SSD	256GB SSD
Operating system	Linux Ubuntu v14.10	Linux Ubuntu v14.10

We also develop a webpage interface to access the distributed storage system of our small cluster. With this interface, clients can upload and download data through a local area network. And the bandwidth of the network is set to 4M/s. At first, we upload a bunch of data files to the DSS. Then we download them and test the data transfer rate. The files are of different size and type, including PPT, JPEG, BMP, PNG, AVI, WMV, MKV and RAR zip files. The upper limitation of one download request is 3GB.

To compare the performance of proposed dynamic greedy algorithm with greedy algorithm, we upload 1GB data to the cluster with the copy factor k=3. And then download them. This test is repeated for ten

times and the downloading speeds are shown as Fig. 2.

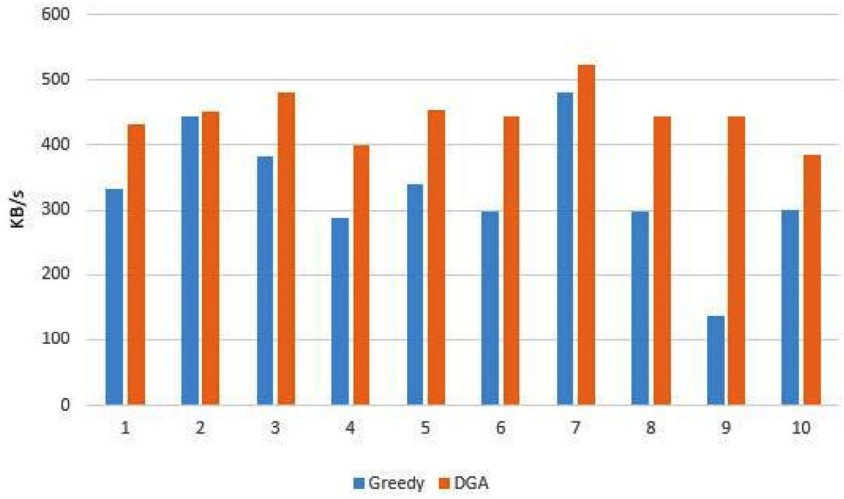

Fig. 2. The downloading speeds

As shown in Fig. 2, each time the DGA outperforms the greedy algorithm. The data transfer rate of DGA is averagely 10% better than the greedy method. The reason is that DGA can distribute data reading tasks evenly to the related data servers. While the greedy algorithm may cause the high-performance servers overload and reduce the overall system performance.

Then we double the data size to 2GB to test the performance of these two algorithms under higher load burden. The experimental results are shown in Fig. 3.

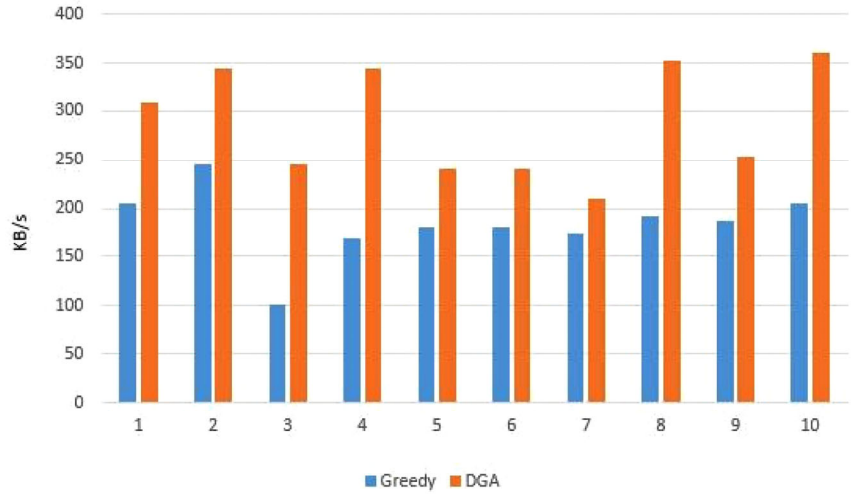

Fig. 3. The downloading speed under higher workload

The results show that under high workload environment, the DGA exhibits even better performance in downloading speed than the greedy algorithm. The data transfer rates are improved almost 40% in average. The reason is that the advantage of high-performance servers will be more prominent under the high workload environment. The use of greedy algorithm will aggravate the overload phenomenon of these servers and cause the reduction of system performance. But the DGA can reduce the load of high-performance servers and allocate the load to the lower burden servers, thereby make the data servers load balancing.

The next experiment is designed to evaluate the performance of the two algorithms when the copy factor is different.

Fig. 4 and Fig. 5 show the data transfer rates on greedy algorithm and dynamic greedy algorithm when the copy factor k is set to 3, 4 and 5. Compared with k=3 and k=4, the DGA has better performance when the copy factor k=5. But it is not true for the greedy method. This is because the dynamic greedy

algorithm can make full use of replicas. When there are more available replicas to choose, it can select the suitable candidate data servers and distribute the workload as evenly as possible, thus generate a more reasonable reading schema.

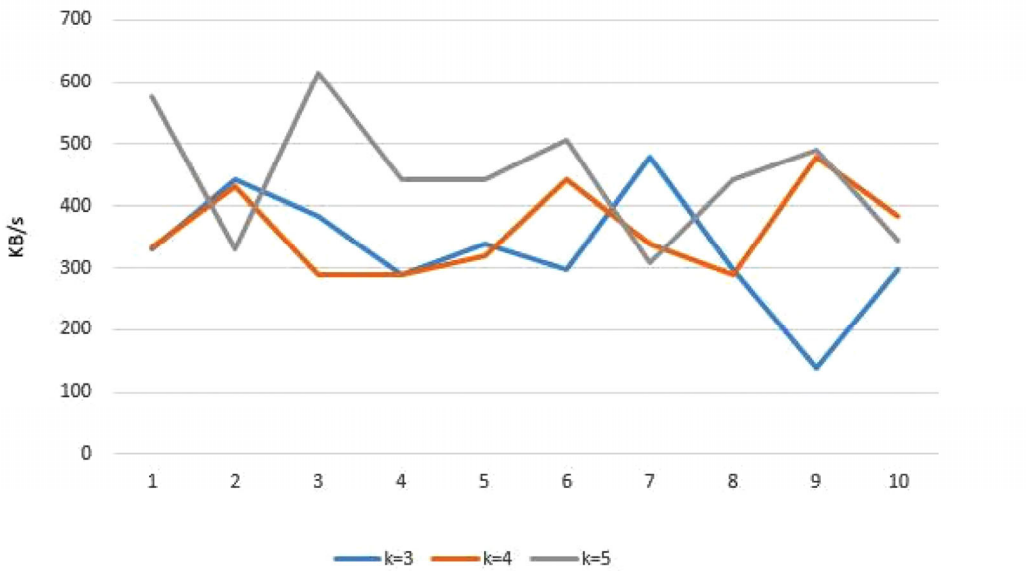

Fig. 4. The downloading speed of the greedy method

Fig. 5. The downloading speed of DGA

5 Conclusion

In this paper, we propose a dynamic greedy algorithm to improve the performance of data access in distributed storage systems. It increases the data transfer efficiency by generating a load balanced data reading schema. This schema takes into account not only the response time of data servers but also the current loads of data servers. We describe the algorithm in detail and implement it in a DHT storage model. The experimental results show that the DGA has better performance than greedy algorithm in file downloading transfer rate, and it can also take full advantage of multiple replicas to generate better data access strategies.

References

[1] A.P. Florence, V.A. Shanthi, A load balancing model using firefly algorithm in cloud computing, Journal of Computer Science 10 (7)(2014) 1156-1165.

[2] R. Nikita, Load balancing model in cloud computing environment, International Journal of Applied Evolutionary Computation 3(3)(2015) 1182-1185.

[3] M. Belkhouraf, A. Kartit, H. Ouahmane, H.K. Idrissi, Z. Kartit, M. El Marraki, A secured load balancing architecture for cloud computing based on multiple clusters, in: Proc. International Conference on Cloud Technologies and Applications, 2015.

[4] M. Alam, A.K. Varshney, A new approach of dynamic load balancing scheduling algorithm for homogeneous multiprocessor System, International Journal of Applied Evolutionary Computation 7(2)(2016) 61-75.

[5] Y. Wu, X. Song, G. Gong, Real-time load balancing scheduling algorithm for periodic simulation models, Simulation Modelling Practice & Theory 52(1)(2015) 123-134.

[6] T. Wang, Z. Liu, Y. Chen, Y. Xu, X. Dai, Load balancing task scheduling based on genetic algorithm in cloud computing, IEEE, International Conference on Dependable, Autonomic and Secure Computing. IEEE, 2014.

[7] N.H. Shahapure, P. Jayarekha, Load balancing with optimal cost scheduling algorithm, in: Proc. International Conference on Computation of Power, Energy, Information and Communication, 2014.

[8] B. Xu, C. Zhao, E. Hu and B. Hu, Job scheduling algorithm based on Berger model in cloud environment, Advances in Engineering Software 42(7)(2011) 419-425.

[9] Y.J. Wang, W.D. Sun, S. Zhouet, X.Q. Pei, X.Y. Li, Key technologies of distributed storage for cloud computing, Journal of Software 23(4)(2012) 962-986.

[10] H.Ma, Y. Yang, Z. Mi, A distributed storage framework of FlowTable in software defined network, Computers & Electrical Engineering 43(2015) 155-168.

[11] B. Mallikarjuna, P. Venkata Krishna, OLB: a nature Inspired approach for load balancing in cloud computing, Cybernetics & Information Technologies 15(4)(2015) 138-148.

[12] K. Liu, G. Xu, J. Chen, Research on cloud computing load balancing based on virtual machine Migration, Open Cybernetics & Systemics Journal 9 (1) (2015) 1334-1340.

[13] R.D. Tao, C. E. Department, Replication factor based cloud storage load balancing technology, in: Proc. Information Technology, 2015.

[14] X. Shao, M. Jibiki, Y. Teranishi, N. Nishinage, effective load balancing mechanism for heterogeneous range Queriable cloud storage, in: Proc. International Conference on Cloud Computing Technology and Science, 2016.