Indian Journal of Communication Engineering and Systems (IJCES)
Vol.3.No.1 2015 pp 107-117.
Available at: www.goniv.com
Paper Received: 04-04-2015
Paper Accepted: 14-04-2015
Paper Reviewed by: 1. R. Venkatakrishnan 2. R. Marimuthu
Editor: Prof. P.Muthukumar

AVOIDING PERFORMANCE FLUCTUATION IN CLOUD STORAGE

M. Pradhaban Raja
Information Technology
AMS Engineering College
Namakkal

ABSTRACT

File distribution and storage in a cloud storage environment is usually handled by storage device providers or physical storage devices rented from third parties. Files can be integrated into useful resources that users are then able to access via centralized management and virtualization. Nevertheless, when the number of files continues to increase, the condition of every storage node cannot be guaranteed by the manager. High volumes of files will result in wasted hardware resources, increased control complexity of the data center, and a less efficient cloud storage system. Therefore, in order to reduce workloads due to duplicate files, we propose the index name servers (INS) to manage not only file storage, data de-duplication, optimized node selection, but also file compression, chunk matching, real-time feedback control, IP information. To manage and optimize the storage nodes based on the client-side transmission status by our proposed INS, all nodes must elicit optimal performance and offer suitable resources to clients. In this way, not only can the performance of the storage system be improved, but the files can also be reasonably distributed, decreasing the workload of the storage nodes.

I. Introduction

DATA IN a cloud storage environment is usually stored in the space offered by third-party companies. Instead of being provided by a single host, the storage space is integrated and distributed through centralized management. Generally speaking, the commonly seen storage protocols are NAS and SAN. Nevertheless, due to the great number of users and devices in the cloud network, the managers often cannot effectively manage the efficiency of various storage nodes. As a result, the complexity of controlling the hardware and the network traffic is increased and the performance of the cloud network is decreased bandwidth as well as the server workload, and also degrade efficiency.

In addition, the cloud network covers a great scope and domain and the data written on storage devices by different users might be similar or identical. In addition, because of user habits and available resources, most users access similar data, operate the same functions, or repeat similar behaviours. Consequently, the system manager can no longer guarantee the optimal status of each storage node in the cloud system. With the enlargement of the network, data integration bottleneck and waste of resources may occur as the system processes duplicate and redundant data, despite the flexibility and rapidity of the cloud storage system.

Cloud computing services can be classified as either com-

puting or storage. As far as data storage is concerned, al-though numerous schemes have been presented to improve chunking and data compression, the waste of resources caused by revisions or changes is often overlooked. For instance, a file that is reuploaded to the server may seriously affect the network.

This study uses the index name server (INS) to process cloud storage functions, including file compression, chunk matching, data de-duplication, real-time feedback control, IP information, and busy level index monitoring. Therefore, our proposed INS can manage and optimize the storage nodes according to the client-side transmission conditions so that every storage node can maintain its optimal status and provide suitable resources to clients.

The rest of this paper is organized as follows. Section II introduces related works and some background information. Section III presents our proposed INS. Section IV illustrates the cloud-based file chunking and management scheme. Section V describes real-time feedback controls. We offer our conclusions in Section VI.

II. Background And Related Works

In addition to the basic background techniques, such as run-length encoding (RLE), dictionary coding, calculation for the
digital fingerprinting of data chunks, distributed hash table (DHT), and bloom filter

A. Run-Length Encoding (RLE)

RLE is a data compression method that converts repeated characters into a single character for the length of the run. Notably, the RLE technique is often used for compressing black and white images into strings of black and white pixels. Because the length of the run does not distribute equiprobably, a statistical method is usually adopted for encoding, i.e., Huffman coding

B. Dictionary Coding

Dictionary coding is another kind of data compression algorithm that encodes data by compressing repeated char-acters and strings. Using some codes to substitute for these characters and strings, we can compress a document because of the correlation of the symbols. A dictionary is just a synopsis of the strings and codes. Practical dictionary coding algorithms aim to encode data dynamically and choose a simple notation to reduce

redundant characters. Dictionary coding algorithms can be divided into two types. The first, which includes LZ77 and Lempel–Ziv–Storer–Szymanski (LZSS), compares the characters; these check whether the characters have appeared previously and then replace the characters by strings that have occurred earlier in the text. The second type includes LZ78 and Lempel–Ziv–Welch (LZW), and uses an index instead of a point to represent the input strings.

C. Calculation for Digital Fingerprint of Data Chunks

Through hash algorithms, hash functions can generate an exclusive fixed-sized digital fingerprint for each data chunk. In order to transform the variable-length data into fixed-length data, hash functions scatter and remix the data through mathematical functions to produce a fixed-sized value shorter than the raw data. This calculated hash value, the fingerprint or the signature of the raw data, is usually expressed by strings of random characters and numbers.

A digital fingerprint is the essential feature of a data chunk. The optimal state is such that each data chunk has its unique fingerprint, and different chunks have different fingerprints. As long as the data with the same primary structures have the same hash values, we can say that data with the same hash values must have the same original data, and that data with different hash values must have different original input data. Nevertheless, the input of a hash function does not thoroughly correspond with the output. Supposing two hash values are the same, this simply implies that the original inputs might be the same. But, different data inputs with the same hash values indicate that different data chunks might generate the same fingerprints. We call this situation hash collision. Compared with secure hash algorithms (SHA), the MD5 hash function presents a lower possibility of hash collisions making it a good candidate for operations, such as fingerprint calculation and recognition.

D. Distributed Hash Table (DHT)

As one of the most commonly used data retrieval methods in the distributed computing system, the DHT aims to efficiently distribute data to different nodes in the system to guarantee that the message reaches the peer with a specific given key value. Using DHTs, we can develop more complex distributed network architectures, such as distributed file systems, peer-to-peer file sharing, and web caching. Instead of being managed by the central node, this kind of service allows different nodes to take charge of parts of the data to construct all the information in the DHT network. Moreover, a DHT node does not maintain and possess all the information in the network, but stores only its own data and those of its neighbouring nodes. This greatly reduces hardware and bandwidth consumption. Essentially, DHTs highlight the following features.

1) Decentralization: there is no central coordination mech-anism in the system.

2) Scalability: the system can maintain efficiency even the number of nodes becomes increasingly larger.

3) Fault tolerance: the system can be reliable (to a certain extent) even when the number of nodes keeps changing.

To ensure the distribution, querying efficiency, and accuracy of data, most DHTs use consistent hashing; this alters only the key/value of the neighboring nodes when the number of nodes changes, but nodes outside of the region will be unaffected. Compared to traditional hashing tables that have to remap the key/value when any change in the key/value occurs, consistent hashing can avoid the enormous change of network information when the number of nodes changes. To remap the key/value, the data in one of the DHT nodes might be moved to another node, which would waste bandwidth resources. Therefore, to efficiently support large numbers of nodes joining or leaving, the reconfiguration must be reduced as much as

possible.

E. Bloom Filter

Structurally, the bloom filter is composed of a long binary vector and a series of random mapping functions. The bloom filter is presented to test whether an element is included in the set. Generally speaking, to test whether an element is a member of a set or not, collecting all the elements for further comparison is the most common method, e.g., linked lists and tree structures. However, with the increase of the elements in the set, more storage space will be needed and the retrieval speed will be slowed down. Due to the presentation of hash functions, the bloom filter can map an element to a point in the bit array via a hash function, compare whether the point in the array is equal to 1, and determine whether the element exists in the set. In addition, to improve the accuracy, more than one hash function will be adopted to increase different mapping points.

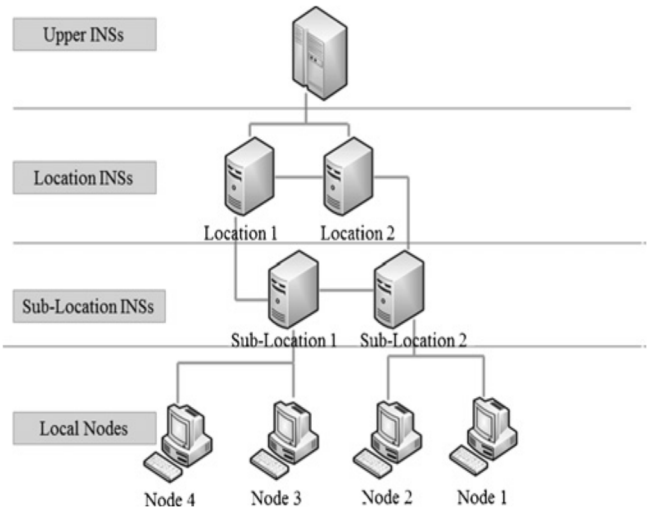

Fig. 1. Hierarchical INS architecture.

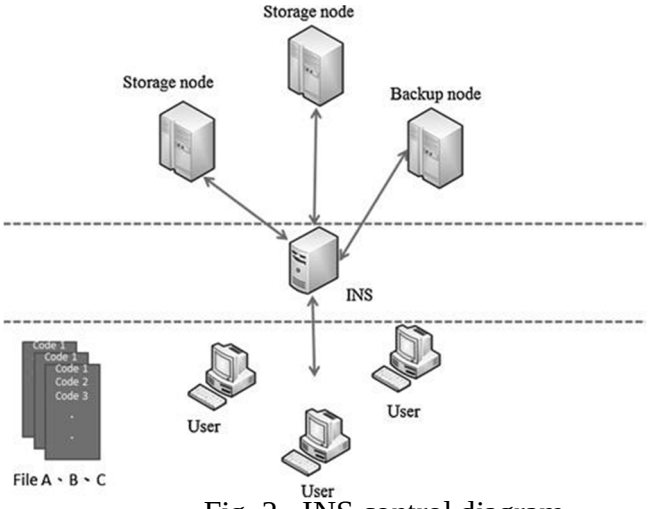

Fig. 2. INS control diagram.

III. Index Name Server (INS)

As an index server resembling domain name system (DNS), the INS uses a complex P2P-like structure to manage the cloud data.Although similar to DNS in architecture and function, the INS principally handles the one-to-many matches between the storage nodes' IP addresses and hash codes. Generally speaking, three main functions of INS include:

1) switching the fingerprints to their corresponding storage nodes;

2) confirming and balancing the load of the storage nodes;

3) fulfilling user requirements for transmission as possible.

Each INS has its own specific database in the domain that stores the fingerprints and their corresponding storage nodes to optimize the file transmissions. Nevertheless, if we use few INSs to monitor the file system in a WAN cloud network environment, a large portion of the workload will be allocated to the INSs. Thus, according to the current DNS structure, we propose to separate the INSs based on their domains and loading capacity, and use the hierarchical management structure to

mitigate the burden of the INSs.

A. INS Architecture

Based on the database, the INSs adopt the stack structure of DNS, manage the storage nodes in their domain, and process users' file-access requirements. Although the INSs are similar to DNS in structure and functions, the INSs mainly query and control the data between fingerprints and storage nodes, and coordinate the transmissions by the feedback control between storage nodes and clients The hierarchical INS architecture is shown in Fig. 1.

As displayed in Fig. 2, the INSs can be regarded as the central managers of the nodes and have server–client relation-ships with one another in a hierarchical architecture to record the fingerprints and the storage nodes of all data chunks.

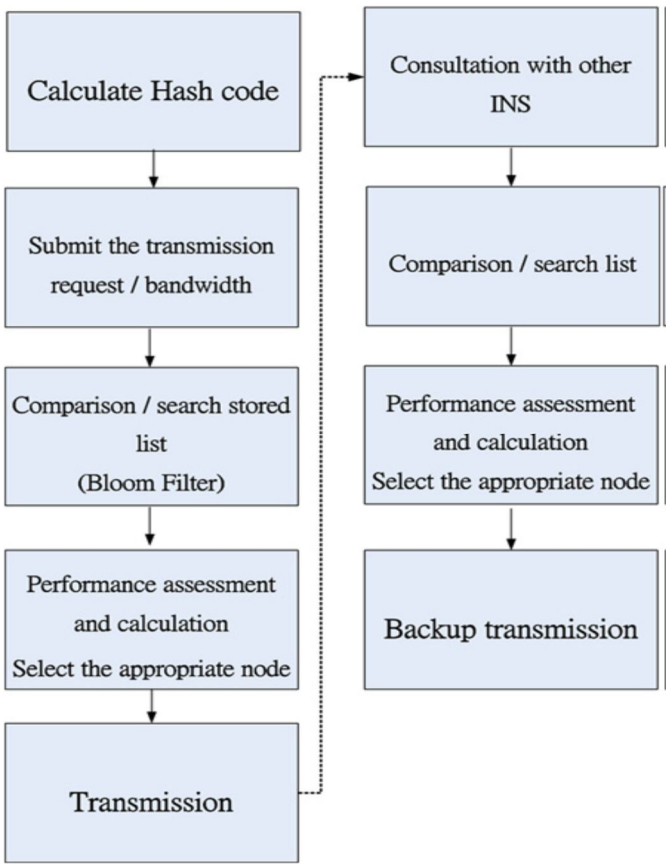

Fig. 3. INS transmission flowchart.

Instead of taking down the information of chunk fragments, the INSs record only the locations of the fingerprints and manage the storage nodes. Every storage node offers its condition and data for INSs to record and users demand the INSs for correlative information during the transmissions. When a novel INS is built up, this INS will choose the storage node with the maximum throughput in its domain as the backup node. Because the INSs focus on computing and transmitting data, we do not concentrate on the storage space, but on the efficiency of the databases and the throughput of data transmission.

B. INS Querying Process

Every domain-based INS has databases of fingerprints and storage nodes. The database of fingerprints records the finger-prints of different files and their corresponding storage nodes. When a user looks for specific fingerprints, the INS queries and confirms if the file already exists in the storage node within the domain before taking the next step. While the clients want to access data, they can use the fingerprints obtained as the index and query the INS of the upper layer, which searches for the best access node based on the content in the database in case the inefficiency of the access node or data loss. The INS transmission flowchart is shown in Fig. 3.

Different requirements will lead to different query results. If the file that the client wants to access does not exist in the storage nodes in the local domain, the INS queries the INS of the upper layer. With the help of the Bloom Filter, the INS can find out the domain of the INS with that file chunk and also the accurate storage node through the destination INS for transmission.

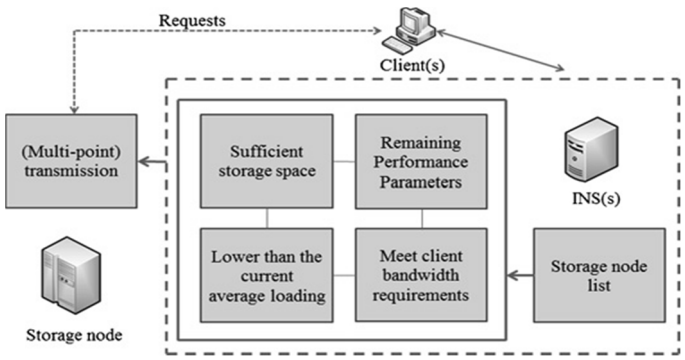

Fig. 4. INS flowchart.

Because we consider the workload of the INSs in different layers, the INSs in this article are divided into several layers and have the server–client relationships with one another in a hierarchical architecture, that is, the INS of the upper layer provides service to the INS of the lower layer only. The burden of each INS thus can be distributed efficiently.

C. De-Duplication

De-duplication is a technique for eliminating duplicate copies of data through a de-duplication scanning process, which improves the system performance and decreases the bandwidth occupied by data transmission. This technique divides a file into chunks and calculates a unique 128-bit hash code of each chunk by MD5, i.e., the only signature of the chunk.

Because of its uniqueness, every fingerprint is regarded as the identification and fingerprint of a data chunk. After check-ing a requested fingerprint, the INSs will confirm whether the file chunk of the same fingerprint exists in the storage space. If not, the system continues the following uploading procedure and assigns tasks to the storage node. Fig. 4 displays the INS flowchart.

Therefore, in hash algorithms, hash functions can convert the variable length data into a unique fixed-sized digital fingerprint. In other words, the significant feature of hash functions is to map the keys to the same value and the values calculated by hash functions are thus called hash values, the signature or fingerprint of the raw data. Hash values are usually expressed by strings of random characters and numbers.

Current de-duplication-related techniques and research have all aimed at deleting duplicate data at the server side, but none has been proposed to discuss data de-duplication and redundant data elimination at the client side. When a file is sent to the cloud storage device, no matter modified or not, the file must be divided into chunks and compressed before sending out, which results in the waste of the processing time

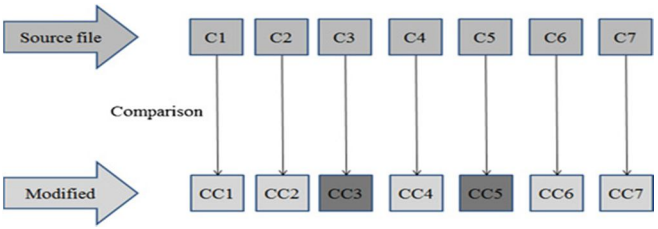

Fig. 5. Client-side chunk matching and differences in chunk comparison.

IV. Cloud Storage File Chunking And Compression

Structurally, the INS architecture consists of INSs, IPs, and clients, in which the INSs are responsible for controlling the whole network and handling the upload, download, and storage of data.

1) Synchronization: The nodes that store IPs keep reporting related information to the INSs. This includes the stor-age space, the memory space, the network bandwidth, the current array number, and the surplus hardware resources. Through the information, the INSs can find the best storage nodes for clients to store data.

2) Match and lookup: Before uploading files, the clients first send the INSs a table, which records the fingerprints of the file chunks. According to the fingerprints, the INSs can match and lookup the fingerprints already

stored in the INSs.

3) Assignment: Without determining the same fingerprints, the INSs will arrange specific IP addresses for the clients to upload files. Matching the fingerprints can accelerate data matching and delete duplicate data.

4) Transmission: The clients transmit the files to the storage nodes assigned by the INSs and the storage nodes will report the resource spent on the task (such as CPU capacity, memory space, bandwidth, and storage space) back to the INSs for regulation and record.

A. Chunk

The chunking method in this paper divides a file into fixed-sized chunks and assigns numbers to each chunk according to the data match. Before the file chunking, the chunks are defined as (*C*): $C = C_1, C_2, C_3, \ldots, C_n$. After partitioning a new file, we give new serial numbers to the chunks. To restore the chunks to a complete file, all we need to do is to arrange the chunks according to the serial numbers and decompress the chunks to get the original file.

We propose to assign numbers to the chunks so that after the file is downloaded from the server and modified by the user, our method can compare the differences between the original chunks and the modified ones. As shown in Fig. 5, the modified chunks are defined as (CC) $CC = CC_1, CC_2, CC_3, \ldots, CC_n$. Once any differences between the original chunks and the modified chunks are found, we redeploy and reupload the modified chunks. The chunk size after user modification is unfixed.Therefore, client-side and INS-side chunk matching are different.

1) *INS-Side Chunk Matching:* After the file is compressed and chunked by the client, a unique 128-bit hash value generated by MD5 is sent to the INSs for server-side chunk matching. Once determining the duplicate chunks, the INSs inform the client to send the nonduplicate chunks to the IPs designated by the INSs. Thus, the INS-side chunk matching is based on the uniqueness of MD5.

2) *Client-Side Chunk Matching:* After the client down-loads the chunks from the server and restores the file, the chunks might be different from the original chunks due to user modification or alteration. As shown in Fig. 5, the client-side chunk matching compares the chunks of the original file and the modified chunks.

B. Client-Side Chunk Comparison

This paper defines the chunk sizes based on user update ratios. Once the file is altered or modified by the user, our proposed method will compare the original file with the altered version. Supposing the altered data chunks differ from the original ones, such as C_3 versus CC_3 and C_5 versus CC_5 as shown in Fig. 5, our method compresses, partitions, and uploads the file again. Equation (1) gives the definition for chunk comparison; when the contrast value between the original chunk and the altered one is not equal to 1, the chunk will be compressed and partitioned again. The number of the repartitioned chunks, *K*, is defined in (2). Equation (3) defines the total number of the original chunks, TC or TCC, while (4) defines RC, the rate of change.

1) When the contrast value between the original chunk and the altered one is not equal to 1, the chunk will be compressed and partitioned again

$$\frac{C_n}{CC_n} \neq 1 \qquad (1)$$

2) The number of the repartitioned chunks (*K*)

$$K = \sum_{n=1}^{C_n} \begin{cases} \frac{C_n}{CC_n} & \text{if } \frac{C}{C} > \text{or} < 1, K = K + 1 \end{cases} \qquad (2)$$

3) The total number of the original chunks (TCC)

$$TCC = CC_n \quad (3)$$

4) The rate of change (RC)

$$RC = \frac{K}{\overline{TC}} \times 100\% \quad (4)$$

We use (1) to (3) to calculate the rate of change. Equation

(1) determines whether the original chunks and the modified ones are the same. If the contrast value is larger or smaller than 1, it means that the chunks have been modified. Next, (2) is used to calculate the number of the repartitioned chunks, *K*. Then, *K*, the number of the repartitioned chunks, divided by TCC, the number of the original chunks, equals the rate of change

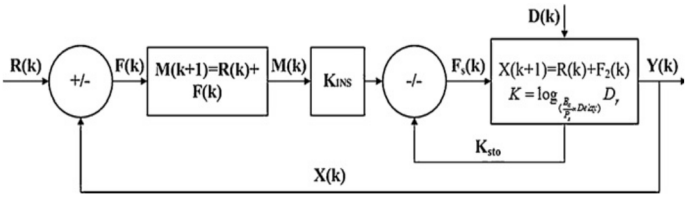

Fig. 6. INS controlling process.

V. Real-Time Feedback Control

A. Performance Parameters of Storage Nodes and Multipoint

Transmission

The performance parameters of storage nodes greatly in-fluence the whole network. To bring the storage nodes into full play based on their efficiency, we define a parameter metric for files, which refers to the number of files that a storage node can actually process. When a storage node starts for the first time, the node examines its own performance. However, every storage node has different hardware (such as CPU, RAM, and hard disk), and the actual efficiency of the storage node cannot be determined according only to hardware specifications. Therefore, modifying the measuring method is necessary. We propose to test the maximum write/read speed before the system achieves 90% of the load and to get the access efficiency of the storage node based on its available maximum bandwidth. Since the chunk size is fixed, our proposed scheme is able to figure out the performance metric of all storage nodes in the INS environment

$$B_s = \frac{B_c}{[N_{download} + (N_{upload} - F_u)] \times (1 - F_d)} \quad (5)$$

In (5), B_s is the bandwidth provided by the storage node, n is the number of storage nodes for the following transmissions, $N_{download}$ and N_{upload} are the number of files that the client will download and upload, respectively, and B_c is the bandwidth that the client will use for transmission. Moreover, F_d is the network delay time after several transmissions. To include F_d, we can ensure that the INSs assign the most suitable storage node with the most appropriate bandwidth to the client so that the utilization efficiency of the storage nodes can be en-hanced. F_u signifies the number of duplicate files determined by the INS databases. The duplicate file chunks will not be reuploaded again to the storage node.

Moreover, the intervals between packet transmissions usu-ally result in extra network delay. When transmitting file chunks, the storage node might face unnecessary waiting time due to some protocol packets (e.g., ACK packets). Since the current network bandwidth reaches a certain level, only the waiting time caused by 10-byte protocol packets is re-garded as the network delay. Consequently, as for the trans-mission completed every second, the delay time caused by protocol packets is

where K is the stream number, B_c, the client-side bandwidth, P_s, the packet size, and D_r , the incidence of delay. With this equation, we can control the incidence of delay caused by the waiting time and limit the stream number to attain the optimal performance of the storage node.

B. Feedback Control System Procedures

Because of external interferences, such as network delay, the transmission value in fact is not equal to the bandwidth that users can use. Thus, while choosing the storage node, the INSs might overestimate users' capacity and result in waste of efficiency. So, we propose to improve this problem by feedback control. Regarded as an automatic control system, the INS keeps receiving the feedback of the former transmissions and adjusting the parameters to reach the optimal performance of storage nodes. Fig. 6 displays the INS controlling process:

1) $R(k)$: The initial expected value;

2) $F(k)$: The output feedback;

3) $M(k)$: The modified feedback;

4) $Fs(k)$: The modified internal function of the storage node;

5) $D(k)$: The external interference factor (random variable);

6) $X(k)$: The result within the storage node;

7) $Y(k)$: The actual result;
8) K_{INS}: The optimal node determined by the INS based on the feedback.

At the initial stage, the INS uses the client-side parameters to compute the bandwidth that the client will use for the storage node as $R(k)$. Next, the system adjusts the parameters according to the results of the former transmission, $M(k)$, with the aim of adjusting the client-side parameters and allocating the suitable storage node to the client proposed method, other backup mechanisms only search for the neighboring nodes with better performance, without con-sidering whether this node is idle or sufficient for data backup.

After the new temporary backup is generated, the INS that was initially requested to appoint nodes will rearrange the deployment. Once $B(i)$, the busy level of the requested node increases, the INS will alter the connection target of $L(i)$, and lead the later demanders to the nodes for temporary backup. If the demand increases, the backup continues. If $B(i)$ is reduced to $B(c)$, the INS changes $L(i)$ and leads the demanders back to the original requested node. In order to save the network resources, the data for temporary backup will be de-duplicated.

Fig. 8 shows that when the target node, A, reveals $B(i) = B(b)$, the requested INS first analyzes the source of the demands. If 60% of the demands come from remote sites and $Q(i)$is too high, it means that the path from the node to the demander is too far or not good. At this time, the requested INS uses $R(i)$ to calculate the suitable area for temporary backup and analyzes $Q(i)$ and $B(i)$ to choose the optimal node as B for remote backup. Without wasting too many resources in crossing domains, this method can limit the distance between the requested end and the backup node. Once the new temporary backup is created, the requested INS immediately leads the demanders to the new nodes for tem-porary backup. For the time being, the INS changes $L(i)$ and leads the demanders X and Y to their adjacent requested nodes. The data for temporary backup also will be de-duplicated to save

VII. Conclusion

This paper proposed the INS to process not only file compression, chunk matching, data de-duplication, real-time feedback control, IP information, and busy level index mon-itoring,. Major contributions of this paper include the following.

1) By compressing and partitioning the files according to the chunk size of the cloud file system, we can reduce the data duplication rate. The processed files are encoded into the signature by MD5 fingerprint for the INSs to match, file, designate to the storage servers, and provide necessary uploading information for the clients. After downloading and modifying the files, the clients compress and partition the modified chunks only, encode these chunks by MD5 fingerprint and reupload the chunks.
2) According to the transmission states of storage nodes and clients, the INSs received the feedback of the previous transmissions and adjusted the transmission parameters to attain the optimal performance for the storage nodes.

References

[1] Chen. C.-Y, Chang K.-D, and Chao H.-C (Mar. 2011), "Transaction pattern based anomaly detection algorithm for IP multimedia subsystem, IEEE Trans.Inform. Forensics Security, vol. 6, no. 1, pp. 152–161

[2] Connell .J.B (Jul. 1973), "A Huffman–Shannon–Fano code," Proc. IEEE, vol. 61, no. 7, pp. 1046–1047

[3] Costa L. B and Ripeanu. M (Oct. 2010), "Towards automating the configuration of a distributed storage system," in Proc. 11th IEEE/ACM Int. Conf. Grid Comput., pp. 201–208.

[4] Dinerstein.J,Dinerstein.S, Egbert.P.K, andClyde.C.W(Dec. 2008), "Learning based fusion for data deduplication," in Proc. 7th Int. Conf. Mach. Learning Appl. , pp. 66–71.

[5] Sun.X, Li.K, and Liu.Y(Jun. 2009), "An efficient replica location method in hierarchical P2P networks," in Proc. 8th IEEE/ACIS Int. Conf. Comput. Inform. Sci. , pp. 769–774.

[6] Tin-Yu Wu, Member, IEEE, Jeng-Shyang Pan, Member, IEEE, and Chia-Fan Lin(March 2014)," Improving Accessing Efficiency of Cloud Storage Using De-Duplication and Feedback Schemes", ieee Systems Journal, Vol.8, No. 1,

[7] Urdaneta.G,Pierre.G, and Van Steen.M(Jan. 2011), "A survey of DHT security techniques," ACM Computing.

[8] Wu. T.-Y,Lee W.-T, and Lin(C. F)(Apr. 2012), "Cloud storage performance enhancement by real-time feedback control and de-duplication," in Proc. Wireless pp.1–5.