Secure And Efficient Of Cloud Storage With Data De-Duplication

Geetha.G

Ms. Thresphine

Prist University Puducherry Campus, Deemed University, Pondicherry, India

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, and server load balancing, but also file compression, chunk matching, real-time feedback control, IP information, and busy level index monitoring. 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.

Index Terms: Cloud storage, de-duplication, hash code, load balancing.

IJIRAS

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 [2]–[4].

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 file 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 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 behaviors. 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 [5], [21].

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. Moreover, to balance the load in the system, we use INS to dynamically monitor IP information and busy level index to avoid network congestion or long wait-ing times during transmissions. The simulation results prove that the performance enhancement of the data transmission can reach up to 20%–50%, which not only improves the performance of the cloud storage system, but also distributes the files to reduce the load on the storage nodes [9].

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 illus-trates the cloud-based file chunking and management scheme. Section V describes real-time feedback controls and the load-balancing backup mechanism. A performance simulation and analysis is provided in Section VI. We offer our conclusions in Section VII.

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, there have been several investigations into load balancing in cloud computing systems.

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 [13].

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 cod-ing algorithms aim to encode data dynamically and choose a simple notation to reduce redundant characters. Dictio-nary 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 [6], [19].

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 neighboring nodes. This greatly reduces hardware and bandwidth consumption. Essentially, DHTs highlight the following features [7].

- ✓ Decentralization: there is no central coordination mech-anism in the system.
- ✓ Scalability: the system can maintain efficiency even the number of nodes becomes increasingly larger.
- ✓ 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

IJIRAS

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.

F. LOAD BALANCING IN CLOUD COMPUTING SYSTEMS

In order to improve the efficiency and maintain the load balancing of cloud systems, most research has aimed at utilizing different scheduling algorithms for better resource utilization and performance enhancement. However, different scheduling mechanisms have different features and suit different states [22].

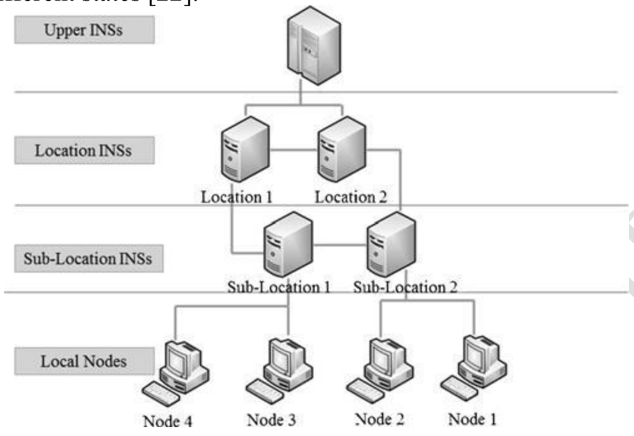

Figure 1: Hierarchical INS architecture

In [8], the opportunistic load balancing (OLB) algorithm keeps each node busy. Thus, OLB does not consider the current workload of each node, but distributes the unprocessed tasks randomly to available nodes. Although OLB is easy and direct, this scheduling algorithm does not consider the expected task execution time and therefore cannot achieve good execution time in make span. Although the minimum completion time (MCT) algorithm gathers statistics to determine the node with the MCT, some tasks still cannot be scheduled to attain the minimum execution time (MET). The MET algorithm, on the other hand, allots the unprocessed tasks to the node with the MET, but this may cause severe load imbalance and does not suit heterogeneous network systems. Based on the MCT, the min–min scheduling [10] algorithm considers both the MCT and the MET, and assigns the tasks to the node with the MCT.

Load balancing in the cloud-computing environment now has become a significant issue for the public. Taking place due to load imbalance, Gmail crashes in 2009 frustrated users around the world several times. Notably, in WAN architecture, some problems occur easily. For example, the demander has to wait for a long time or control the traffic when the net-work resources are insufficient. Consequently, to achieve load-balancing success, numerous schemes have been presented to reach the maximum throughput.

The above-mentioned load-balancing schemes can be di-vided into several types. Virtual servers [11], [12] present an integrated heterogeneous environment in which virtual nodes are classified into idle nodes and high-performance nodes to deal with different loading conditions. With the attempt to attain load balancing, [14] proposed monitoring environmental performance for regional resource integration and different workload allocation to the nodes. In [15] and [17], virtual IDs symbolize the performance of nodes. Different IDs accept different load demands and establish different processing times to figure out the idle servers. Furthermore, [18] classified the subnodes according to their performance together with dy-namic load status monitoring and report load balancing. While facing different load demands, [17] adjusted the hierarchical relationships to balance the load by integrating the light-loaded subnodes or separating the heavy-loaded subnodes.

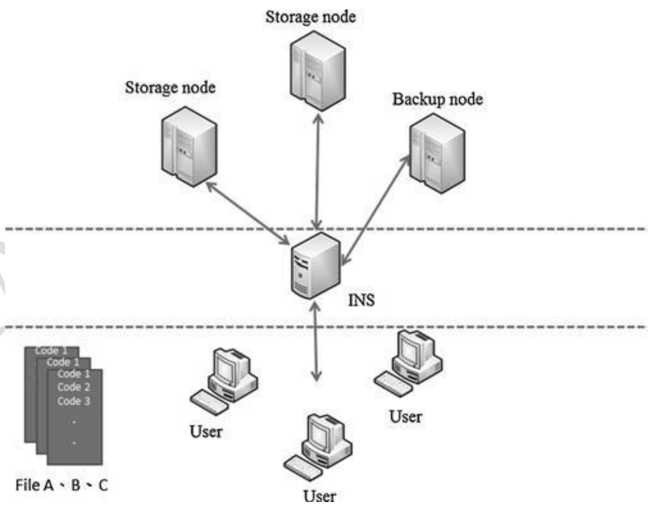

Figure 2: INS control diagram

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 [8]. 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:

- ✓ switching the fingerprints to their corresponding storage nodes;
- ✓ confirming and balancing the load of the storage nodes;
- ✓ 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 [1], [10], [16]. 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 [11] with one another in a hierarchical architecture to record the fingerprints and the storage nodes of all data chunks.

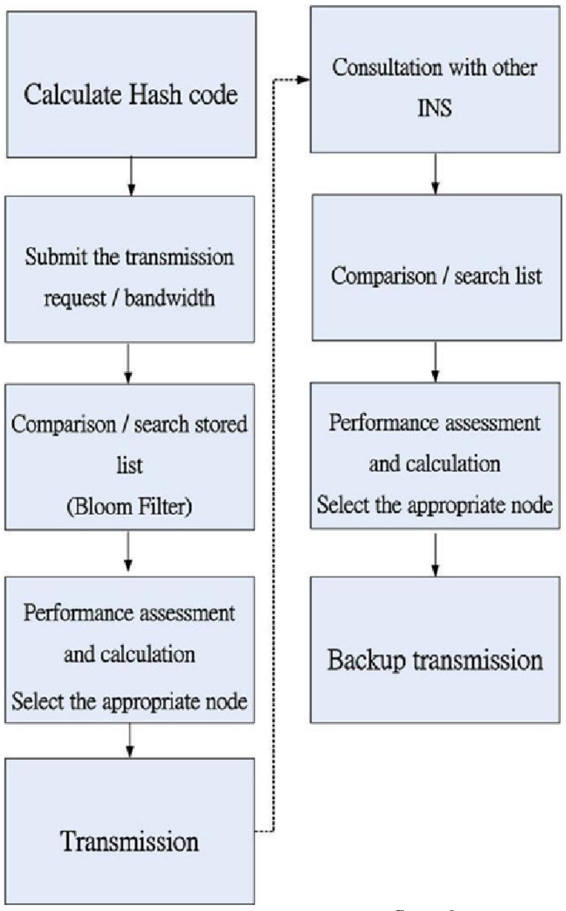
Calculate Hash code
Submit the transmission request / bandwidth
Comparison / search stored list (Bloom Filter)
Performance assessment and calculation Select the appropriate node
Transmission
Consultation with other INS
Comparison / search list
Performance assessment and calculation Select the appropriate node
Backup transmission

Figure 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 [12]. 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 [20].

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.

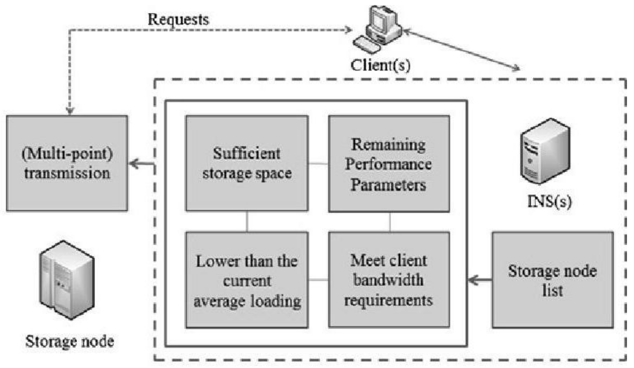
Requests
Client(s)
(Multi-point) transmission
Sufficient storage space
Remaining Performance Parameters
INS(s)
Lower than the current average loading
Meet client bandwidth requirements
Storage node list
Storage node

Figure 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 at the client side.

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.

- ✓ Synchronization: The nodes that store IPs keep reporting related information to the INSs. This includes the storage 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.
- ✓ 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.
- ✓ 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.
- ✓ 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.

- ✓ *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.
- ✓ *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.

- ✓ 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

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

- ✓ The number of the repartitioned chunks (K)

$$K = \sum_{n=1}^{C_n} \frac{C_n}{CC_n} \quad if \quad \frac{CC}{CC} > \text{or} < 1, K = K + 1 \qquad (2)$$

- ✓ The total number of the original chunks (TCC)

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

- ✓ The rate of change (RC)

$$RC = \frac{K}{TCC} \times 100\% \qquad (4)$$

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.

IJIRAS

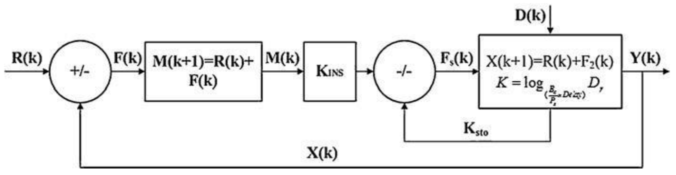

Figure 6: INS controlling process

V. REAL-TIME FEEDBACK CONTROL AND BALANCING BACKUP MECHANISM

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

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

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

$$D_t = \frac{B_c}{P_s} \qquad (6)$$

Owing to the waiting time, the delay probability per second is

$$P_{delay} = \frac{1}{D_t} \qquad (7)$$

According to this probability, we can figure out the rela-tionship between the stream number and the delay probability by

$$P_{delay} = \left(\frac{1}{D_t}\right)^k \qquad (8)$$

After regulating the positions of the data, we get

$$K = \log_{\frac{B_c}{P_s} \times Delay} D \qquad (9)$$

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:

- ✓ $R(k)$: The initial expected value;
- ✓ $F(k)$: The output feedback;
- ✓ $M(k)$: The modified feedback;
- ✓ $Fs(k)$: The modified internal function of the storage node;
- ✓ $D(k)$: The external interference factor (random variable);
- ✓ $X(k)$: The result within the storage node;
- ✓ $Y(k)$: The actual result;
- ✓ 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.

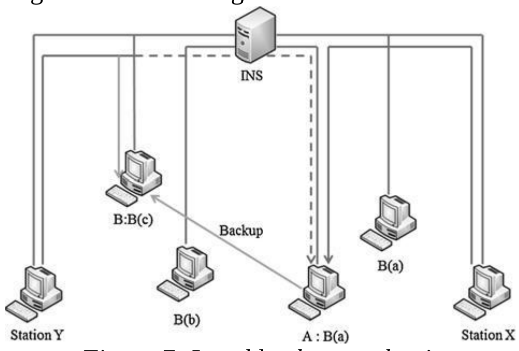

Figure 7: Local backup mechanism.

IJIRAS

C. LOCAL AND REMOTE BACKUP

According to the relativity between time and distance, the INSs not only can classify the paths and find the best path of a given weight in an ad hoc network, but also can analyze the transmission efficiency among the storage nodes and select the shortest path. It is worth noting that we include several transmission parameters in the environment and take down the records in the INSs in an attempt to achieve the flexible system performance and the optimal resource management. The content of the INS record is defined in

$[D(i)][L(i)][Q(i)][B(i)][R(i)]$ (10)

- ✓ $D(i)$: The hash code of a data chunk, also the identification and fingerprint of the chunk, which is expressed by a 128-bit string.
- ✓ $L(i)$: The location of the node that stores the data chunk. This column stores the IPs and geographical information of all storage nodes in accordance with the index. When users search for the needed hash code, this column lists all corresponding IPs.
- ✓ $Q(i)$: The transmission quality of all paths from the INS to the destination storage node. We can figure out the transmission quality of all paths according to the parameters returned by both terminals. To transmit a standard-sized packet (1/4 MB), the INS can ascertain the transmission quality of all nodes and other INSs in the domain.
- ✓ B(i): The present busy level of the storage node is a parameter based on the current loading status of the hardware. While confirming the existence of other nodes, the target node first broadcasts its own busy level for the INSs to compile. The INSs then make a judgment on whether or not to connect.
- ✓ $R(i)$: The IPs and completion time of visitors who connect to the storage node successfully through the INSs. To offer visitor records, this parameter allows the INSs to cancel data backup, control network resources, and manage backup data efficiently.

Fig. 7 shows that when the target node reveals $B(i)$ = $B(b)$, the INS first analyzes its neighboring nodes to find the suitable node for temporary backup; the neighboring node with $B(i)$ = $B(c)$ will be suitable for data backup. Compared with our proposed method, other backup mechanisms only search for the neighboring nodes with better performance, without considering 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 network resources. The remote backup is created after $B(i)$ turns to $B(b)$ for two reasons: 1) directly creating the remote backup after the request will waste resources; and 2) we create the most efficient remote backup only when enough remote requests are made.

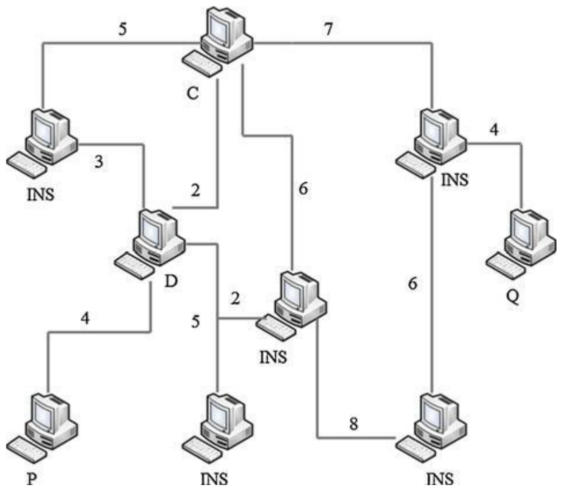

Figure 8: Remote backup mechanism

VI. PERFORMANCE SIMULATION AND ANALYSIS

Here, we present the simulation of the file chunks and INS parameters, in which the transmission efficiency and load balancing of the INSs is further analyzed.

Data length (MB)	CHUNK SIZE (MB)	Number of chunks
128	8	16
	4	32
	2	64
	1	128
	1/2	256
	1/4	512
	1/8	1024

Table 1: Simulation Parameters For Different Chunk Sizes

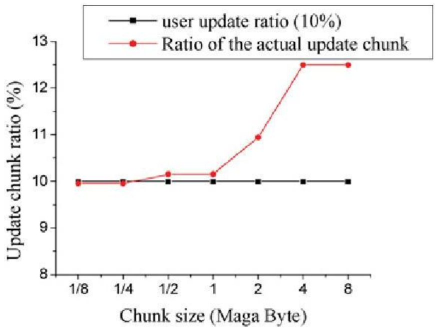

Figure 9: User update ratio (10%) vs. ratio of the actual update chunk

IJIRAS

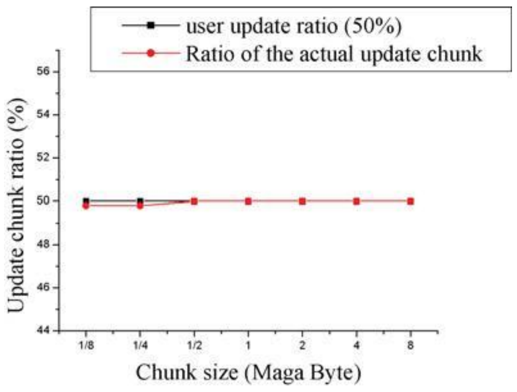

Figure 10: User update ratio (50%) versus ratio of the actual update chunk

A. UPDATING DATA CHUNKS

The purpose of this section is to determine the best chunk size for cloud file systems based on the update chunk ratio after users modify or alter the data. The simulation is run based on the parameters listed in Table I and the simulation results are given in Figs. 9–11. The simulation results show that when the user update ratio is 10%, 50%, and 90%, and the chunk size is under 1/2 MB, the update chunk ratio is about 1:1, and appears to be stable between 1/2 and 1/8 MB. Thus, the update chunk size above 1/2 MB will be insufficient. If the chunk size is 1/8 MB, the number of chunks will double that of 1/4 MB, which aggravates the burden on the system. Therefore, based on the chunk size of P2P, we choose 1/4 MB as the chunk size in the simulation [15].

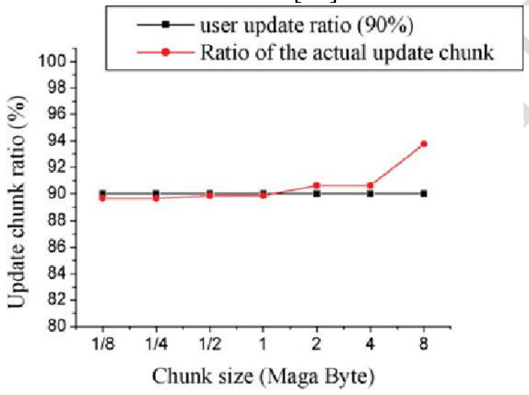

Figure 11: User update ratio (90%) versus ratio of the actual update chunk

Parameter	Content
Performance metric of storage node	200–500 (fps)
Space of storage node	1–10 (TB)
Bandwidth of storage node	10–100 (Mb/s)
Client-side write request	1–5 (fps)
Client-side read request	1–5 (fps)
Client-side bandwidth	2–10 (Mb/s)
Extra transmission delay	5–100 (ms)
Number of clients	1–2700+

Table 2: Parameters for INS Performance Simulation

B. INS Performance Simulation

An INS system includes the client, the storage node, and the INS. Table II lists the related simulation parameters.

Fig. 12 shows the average delay time of different transmission schemes in the same environment. After figuring out which storage nodes elicit better performance and bandwidth, the hybrid scheme selects the suitable nodes for transmission based on their current loading states. Although this scheme can ensure that the transmission load is first handled by the nodes with better performance, these storage nodes have heavier loads, comparatively. As opposed to other schemes that select the nodes based on various orientations, the random scheme allocates the load to all nodes equally without considering the capacity of every node. On the other hand, our proposed INS considers both the load and performance of the storage nodes. While choosing the nodes, the INS scheme selects the nodes with the best bandwidth and performance allowing the most suitable storage nodes for client-side transmissions to be chosen.

The effects of different schemes to the system's average loading rate are given in Fig. 13, in which the loading rate of the INS at 100% is regarded as the benchmark. The figure shows that the INS scheme is able to distribute the load to all storage nodes efficiently by taking the bandwidth, performance, and load balancing into account. On the other hand, because the load is distributed to some of the nodes only, the load of the hybrid scheme appears to be much higher. The hybrid scheme is a significant issue in the current literature. Notably, the random scheme was originally proposed to distribute the load equally. However, because different storage nodes have different bandwidths and present different performances, the random scheme cannot attain a fair load balancing.

IJIRAS

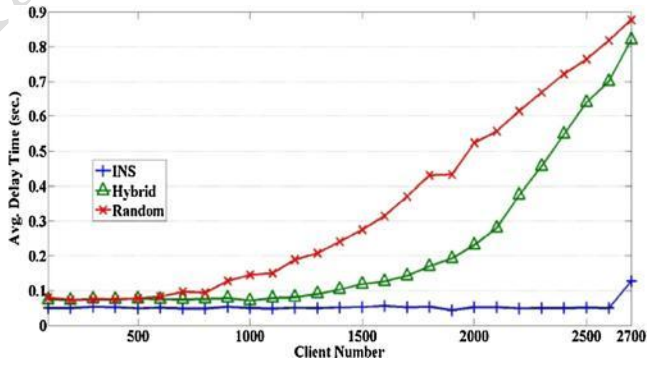

Figure 12: Average delay time

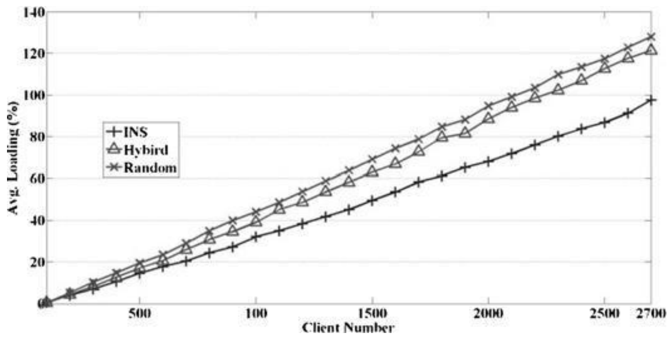

Figure 13: Average loading rate

Data duplication rate (%)	Maximum number of clients
0	2600
10	2800
20	2900
30	3000

40	3300
50	3700
60	3700
70	4000
80	4300
90	5200

Table 3: Data Duplication Rate and the Corresponding Maximum Number of Clients

After analyzing the above scenarios, we further add the mechanism of data duplication identification and management to decrease the load of the INS and storage nodes allowing the clients to have more resources available for performance enhancement. Table III lists different data duplication rates and the corresponding maximum number of clients when the loading rate of the INS scheme reaches 100%.

Fig. 14 displays the average loading rate of the INS scheme under different data duplication rates, and a 0% data duplication rate is taken as the maximal value for further comparison. We found that with the increase of duplication rate, the loading rate under the same node number can be approximately reduced by 50%. This result supports the notion that data de-duplication mechanism can efficiently reduce the load of the system.

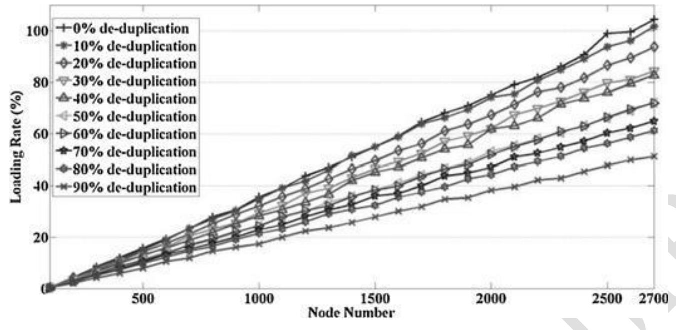

Figure 14: Average loading rate under different data de-duplication rates

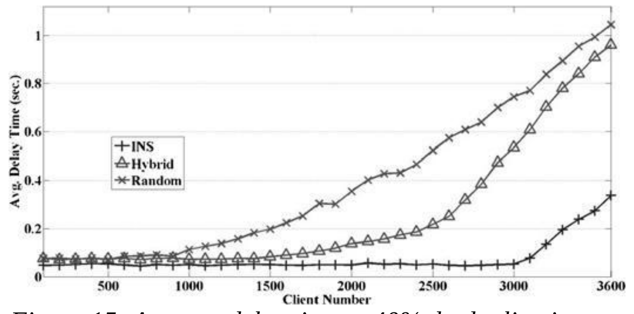

Figure 15: Average delay time at 40% de-duplication rate

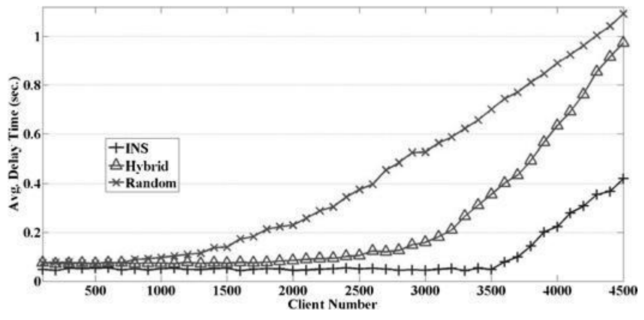

Figure 16: Average delay time at 70% de-duplication rate

B. LOAD BALANCING AND BACKUP EXAMINATION OF INS

This experiment compares three methods for selecting backup nodes: random selection, performance-based selection (INS Basic), and our proposed nonbusy and high-efficient selection (INS Adv.). Fig. 17 presents the simulation results of local backup and compares the backup efficiency of the three methods. In order to reduce the system load to 60%, the backup is executed the fewest times. The figure reveals that with the increase of node numbers in the domain, the backup must be continued to handle the increasing load. If choosing the backup nodes efficiently, we can support the system load using only a few backup nodes. Although INS Adv. is only slightly better than INS Basic, the stability of INS Adv. appears to be great because the high-efficient nodes are usually busy nodes.

Fig. 18 examines the performance of the backup nodes when the node number in the domain increases. The figure reveals that when the node number increases, INS Adv. performs as usual and presents certain stability.

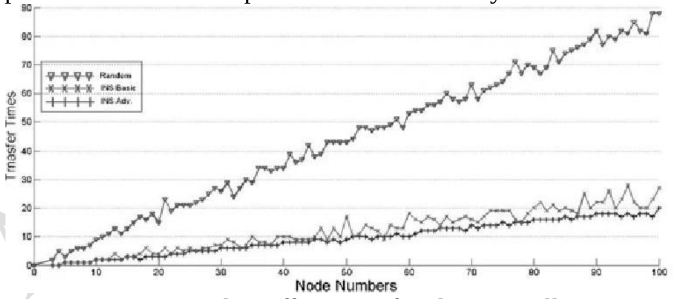

Figure 17: Backup efficiency of nodes in small areas

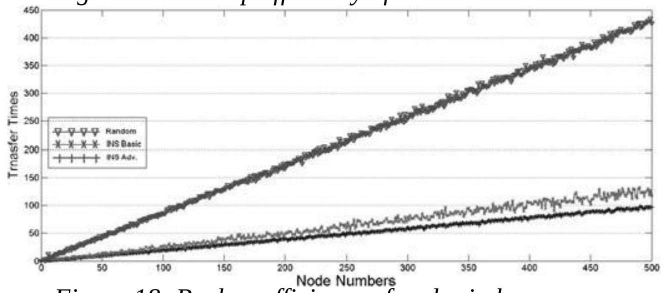

Figure 18: Backup efficiency of nodes in large areas

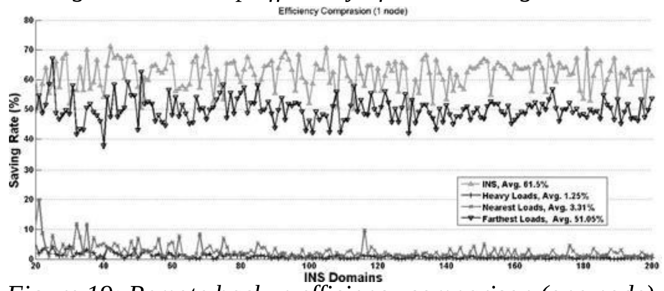

Figure 19: Remote backup efficiency comparison (one node)

C. LOAD-BALANCING BY INDEX NAME SERVER

As for cross-domain backup, [24], [25] proposed dynamic monitoring according to the loading performance, while [26] proposed high-efficient node groups to handle large-scale cross-domain load. To consider the bandwidth cost of each domain, we use the INSs to track the load sources by re-mote backup and select the idle nodes as the backup nodes preferentially. In the simulation, we deploy nodes of different performance randomly and set up the load to a certain extent. To compare our proposed scheme with heavy loads, nearest loads, and farthest loads, we examine the performance of the established backup nodes in each scheme.

The one-node remote backup efficiency comparison as given in Fig. 19 reveals that the performance of our proposed INS is not decreased with the increase of nodes. Nearest loads and heavy loads cannot set up efficient backup because: 1) they cannot reduce the cross-domain cost of heavily loaded nodes in handling remote demanding links; and 2) they waste the backup cost for the inability to backup at the best locations.

The performance of farthest loads and our INS is quite similar because both the two schemes manage the cross-domain cost well and achieve high-efficient backup that is similar to local backup. INS backup has better efficiency, which explains that the backup sites are established on the more efficient locations.

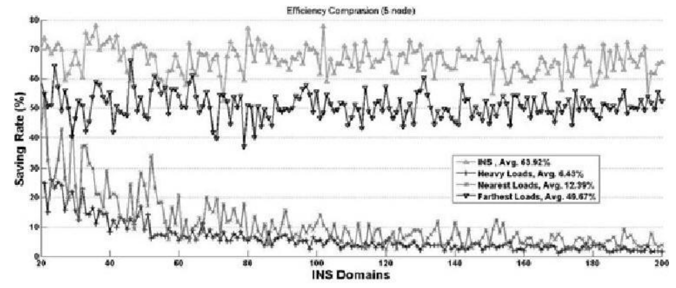

Figure 20: Remote backup efficiency comparison (five nodes)

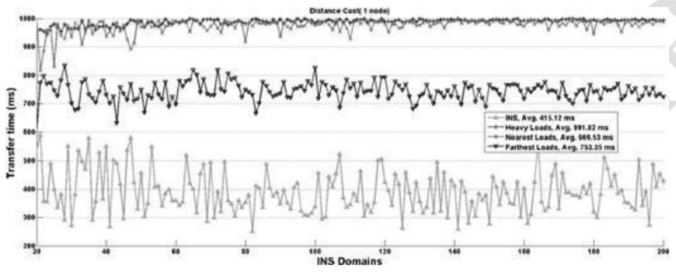

Figure 21: Remote backup distance cost comparison (one node)

Fig. 20 displays the five-node remote backup efficiency comparison. The figure shows that the efficiency of farthest loads and INS is as usual but the efficiency of nearest loads and heavy loads obviously improves for 10%, which proves the above-mentioned cross-domain cost and the inability to backup at the most efficient locations.

As shown in Fig. 21, our next simulation testifies the backup capability of each scheme according to distance cost. It is revealed that although reducing the load effectively previously, farthest loads does not perform well in transfer time. On the contrary, compared with other schemes, our proposed INS decreases the transfer time efficiently and advantageously.

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 monitoring, but also file storage, optimized node selection, and server load balancing. Three major contributions of this paper include the following.

- ✓ 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.
- ✓ 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.
- ✓ Based on several INS parameters that monitor IP information and the busy level index of each node, our proposed scheme can determine the location of maximum loading and trace back to the source of demands to determine the optimal backup node. Consequently, the backup efficiency can be improved and the load balancing among the nodes is considered.

ACKNOWLEDGMENT

Our gratitude goes to W.-M. Lu and Y.-S. Lin for discussion about some issues of this paper.

IJIRAS

REFERENCES

[1] J. B.Connell, “A Huffman–Shannon–Fano code,” Proc. IEEE, vol. 61, no. 7, pp. 1046–1047, Jul. 1973.
[2] J. Zha, J. Wang, R. Han, and M. Song, “Research on load balance of service capability interaction management,” in Proc. 3rd IEEE Int. Conf. Broadband Netw. Multimedia Technol., Oct. 2008, pp. 212–217.
[3] R. Tong and X. Zhu,. “A load balancing strategy based on the com-bination of static and dynamic,” in Proc. 2nd Int. Workshop Database Technol. Appl., Nov. 2010, pp. 1–4.
[4] T.-Y. Wu, W.-T. Lee, Y.-S. Lin, Y.-S. Lin, H.-L. Chan, and J.-S. Huang, “Dynamic load balancing mechanism based on cloud storage,” in Proc. Comput. Com. Appl. Conf., Jan. 2012, pp. 102–106.
[5] Y. Zhang, C. Zhang, Y. Ji, and W. Mi, “ A novel load balancing scheme for DHT-based server farm,” in Proc. 3rd IEEE Int. Conf. Comput. Broadband Netw. Multimedia Technol., Oct. 2010, pp. 980–984.
[6] I. Keslassy, C.-S. Chang, N. Mckeown, and D.-S. Lee, “Optimal load-balancing,” in Proc. IEEE Comput. Infocom, Mar. 2005, pp. 1712–1722.
[7] L. Zhou and H.-C. Chao. “Multimedia traffic security architecture for internet of things,” IEEE Netw., vol. 25, no. 3, pp. 29–34, May 2011.
[8] Y.-X. Lai, C.-F. Lai, C.-C. Hu, H.-C. Chao, and Y.-M. Huang, “A personalized mobile IPTV system with seamless video reconstruction algorithm in cloud

networks," Int. J. Commun. Syst., vol. 24, no. 10, pp. 1375–1387, Oct. 2011.

[9] T.-Y. Wu, C.-Y. Chen, L.-S. Kuo, W.-T. Lee, and H.-C. Chao, "Cloud-based image processing system with priority-based data distribution mechanism," Comp. Commun., vol. 35, no. 15, pp. 1809–1818, Sep. 2012.

[10] M. Chen, C. M. Leung, L. Shu, and H.-C. Chao, "On multipath balancing and expanding for wireless multimedia sensor networks," Int. J. Ad hoc Ubiquitous Comput., vol. 9, no. 2. pp. 95–103, Feb. 2012.

[11] Z. Feng, B. Bai, B. Zhao, and J. Su, "Redball: Throttling shrew attack in cloud data center networks," J. Internet Technol., vol. 13, no. 4, pp. 667–680, Jul. 2012.

[12] D. Han and F. Feng, "Research on self-adaptive distributed storage system," in Proc. Wireless Commun. Netw. Mobile Comput., Oct. 2008, pp. 1–4.

[13] J. Wang, P. Varman, and C. Xie, "Avoiding performance fluctuation in cloud storage," in Proc. Int. Conf. High Performance Comput., Dec. 2010, pp. 1–9.

IJIRAS