International Journal of Information Technology and Management

ISSN online: 1741-5179 - ISSN print: 1461-4111
https://www.inderscience.com/ijitm

A reliability and security enhanced framework for cloud-based storage systems

Peng Xiao

DOI: 10.1504/IJITM.2023.10055157

Article History:

Received:	19 April 2018
Last revised:	09 September 2019
Accepted:	03 December 2019
Published online:	05 April 2023

A reliability and security enhanced framework for cloud-based storage systems

Peng Xiao

Department of Computer Science,
Hunan Institute of Engineering,
No. 88 Fu-Xing Road,
Xiangtan City 411104, China
Email: zghu1963@126.com

Abstract: In cloud computing environments, reliable and secure data storage service plays a more and more important role in many data-intensive applications. However, existing storage systems either fail to provide them or provide them in a cost-ineffective manner. To provide better storage service in nowadays cloud environments, we propose a novel framework called reliability and security enhanced cloud storage (RSCS), which consists of several well-designed components to improve low-level data reliability as well as guarantee up-level data accessing security. In the RSCS framework, a simple yet effective file system scheme is proposed, which can duplicate stripped data in different storage nodes so as to increase the data reliability and aggregated throughput. We also introduce a centralised leasing mechanism, which allows accessing different portions of a file based on the multiple-reader-single-writer principle. Finally, we provide a secure data accessing tunnel technology, which allows the RSCS to establish secure communication channels between users and storage nodes without introducing too many extra costs. In a real-world cloud platform, we conduct sets of experiments and the results show that the proposed RSCS framework is able to meet the requirements for most of cloud-based storage platforms.

Keywords: cloud storage; data security; replication service; file system.

Reference to this paper should be made as follows: Xiao, P. (2023) 'A reliability and security enhanced framework for cloud-based storage systems', *Int. J. Information Technology and Management*, Vol. 22, Nos. 1/2, pp.160–174.

Biographical notes: Peng Xiao is received his master's degree in Xiamen University and doctoral degree in Central South University. Currently, he works in Hunan Institute of Engineering as an Associate Professor. His research interests include cloud computing, parallel and distributed systems, green network and software engineering. He is currently a member of IEEE and ACM societies.

1 Introduction

A major goal of cloud computing is to offer flexible and secure resource sharing mechanisms for diversity of users with different requirements (Subashini and Kavitha, 2011; Marmol and Kuhnen, 2015). To do this, cloud middleware should provide a

flexible and customisable resource provisioning service, by which un-trusted programs and applications from users can be executed in a configurable manner (Bohli et al., 2013; Sindhu and Mushtaque, 2014). Such a resource management service is often implemented by combining cloud middleware and resource virtualisation technology, which is responsible for managing virtual machine (VM) instances based on per-user fashion (Lagar-Cavilla et al., 2011; Canali and Lancellotti, 2014). As VM instances are often deployed across distributed resources, which means that a lot of large VM state data (e.g., disk, RAM, vCPU) needs to be transferred (Di et al., 2015). Besides, as more and more data-intensive applications are deployed in cloud environments, a reliable and secure storage platform plays a key role to execute these applications (Shamsi et al., 2013; Song et al., 2013; Xiao and Han, 2014).

Currently, some cloud middleware have provide data management service from the perspective of VM instance, such as VM-based data transferring, I/O scheduling, data backup, etc. (Wan et al., 2013; Khan et al., 2014; Long et al., 2014). Based on the experiences of using these VM-based data services, more and more researchers have noticed that transferring VM states (especially the RAM layout) is likely causing serious performance problems. In addition, it also may result in tight-coupled logics between resource infrastructure layer and the user application layer (Guan and Choi, 2014). On the other hand, by offering an effective data management service, the cloud middleware is capable of distributing the overheads of data management across different logical entities, such as state servers, data severs, monitoring servers, etc., each kind of logical entity being responsible for managing certain kind of VM-related information (Mao et al., 2014; Anglano et al., 2015). Unfortunately, in current available cloud-based storage systems, data security and data reliability are typically separated provided in different middleware. As a result, I/O performance bottleneck becomes more and more significantly in many real-world cloud environments since providing reliability and security enhanced storage service typically involves heavy overheads on cloud resources as cloud users (Zhang et al., 2013; Goncalves et al., 2016).

To provide better storage service in nowadays cloud environments, we design a novel framework called reliability and security enhanced cloud storage (RSCS), which consists of several well-designed components to improve low-level data reliability as well as guarantee up-level data accessing security. In the RSCS framework, a simple yet effective file system scheme is proposed, which can duplicate stripped data in different storage nodes so as to increase the data reliability and aggregated throughput. We also introduce a centralised leasing mechanism which allows to accessing different portions of a file based on the multiple-reader-single-writer principle. Finally, we provide a secure data accessing tunnel technology which allows the RSCS to establish secure communication channels between users and storage nodes without introducing too many extra costs.

The rest of this study is organised as follows: In Section 2, we present the related studies on the addressed topic in our study; In Section 3, we describe the RSCS framework and the corresponding implementation in details; In Section 4, we conduct sets of experiments to investigate the effectiveness of the proposed framework. Finally, we conclude out study with a brief discussion of the future work in Section 5.

2 Related work

In early years, many cloud-based storage systems developed and deployed in practical cloud environments mainly focused on block-level optimisation techniques. For example, ORTHRUS (Wan et al., 2013) is a light-weighted block-level storage system, in which a listen-detect-switch mechanism is designed deal with contingent volume servers' failure, and a strategy that dynamically balances load between multiple volume servers based on black-box model is also incorporated. NCCloud (Chen et al., 2014) is cloud storage platform, which can provide cost-effective fault-tolerant service in multi-cloud environments. In the NCCloud platform, erasure code technology is only used for maintaining data fault-tolerance and acceptable redundancy, while using a functional minimum-storage regenerating code technology to managing data transferring and movement. In this way, NCCloud can obtain desirable data fault-tolerance and management efficiency at the same time. BSS (Khan et al., 2014) is an energy-efficient block-based sharing scheme that provides confidentiality and integrity data accessing services for mobile users in the cloud environment. In Liang and Kozat (2014), the authors proposed a set of solutions which focus on reducing the data transferring delay in cloud storage platforms. These solutions are based on the performance measurements obtained by observing the Amazon S3 for long time durations. For example, they found that the operations of reading and writing small data files follow certain kind of random patterns, which can be used to design optimal erasure code scheme or I/O scheduling policy. MORM (Long et al., 2014) is an effective data management framework, which tries to find out the optimal data replication factor by solving a multi-objective optimisation problem through the immune algorithm. To meet the requirements from daily users who frequently modify the same dataset, Duan et al. (2015) proposed the CSTORE system, which uses a three-level hash mapping mechanism to realise the data de-duplication at the block level. In this way, the CSTORE system can effectively adjusting the trade-offs between data reliability and storage utilisation. Similarly, the RAMCloud (Ousterhout et al., 2015) is also an optimal cloud storage system, which always tries to use the massive RAM capacities for buffering the hashing tables of data files. For example, it allows to aggregating thousands of hashing table in a single key-value table with aiming at reducing the overheads of virtual file systems. WaFS (Wang et al., 2015) is cloud-oriented file system, which is especially designed for detecting the data-dependencies for workflow applications. By using the WaFS system, a task scheduler is able to suitable decisions on job scheduling, such as obtaining desirable trade-offs between storage utilisation and data movement overhead.

In the above cloud-based storage systems, I/O performance is often the primary designing goal, instead of reliability and security. In recent years, as reliability and security have become important QoS metrics from the perspective of cloud users, more and more systems are incorporated with corresponding services or mechanisms. For example, depot (Mahajan et al., 2011) is a simple but effective two-layer storage framework that is able to detect malicious behaviours on data accessing operations. In depot, the data consistency service is implemented based on the Fork-Join-Causal principle, which is more cost-effective than the traditional consistency maintaining mechanisms. Zhu et al. (2015) presented a data security technology called ABE-AH with aiming at offering an easy-to-manage access controlling service. The key feature of ABE-AH is that it allows to defining per-user access controlling rules, which makes it very suitable for opening cloud environments. Celesti et al. (2016) proposed a system with

aiming at offering reliable and secure storage service for long-term cloud applications. This storage system is able to log and analyse the data accessing patterns and then make intelligent adjustments on system's runtime parameters, such as block layout scheme, duplication factor, and accessing controlling thresholds. Cui et al. (2016) proposed a key-aggregate searchable encryption scheme, which tries to solve the problem of securely distributing to users a large number of keys for both encryption and search. The key features of this scheme is that the owner of a data file only needs to sent a single key when multiple files are accessed by another user.

3 System design and implementation

3.1 Framework overview

In Figure 1, we present the framework overview of the proposed RSCS system and the designing and implementation details will be described in the next sections. In this framework, cloud users interact with the RSCS system through an independent service portal which can be implemented as a web service or separated networking application. The key components in RSCS framework include data security service, data access service, data recovery service, replication service, mirror file system, and I/O monitoring service. It is noteworthy that although these components are independently designed and implemented, they are in fact interacting with others to achieve two goals:

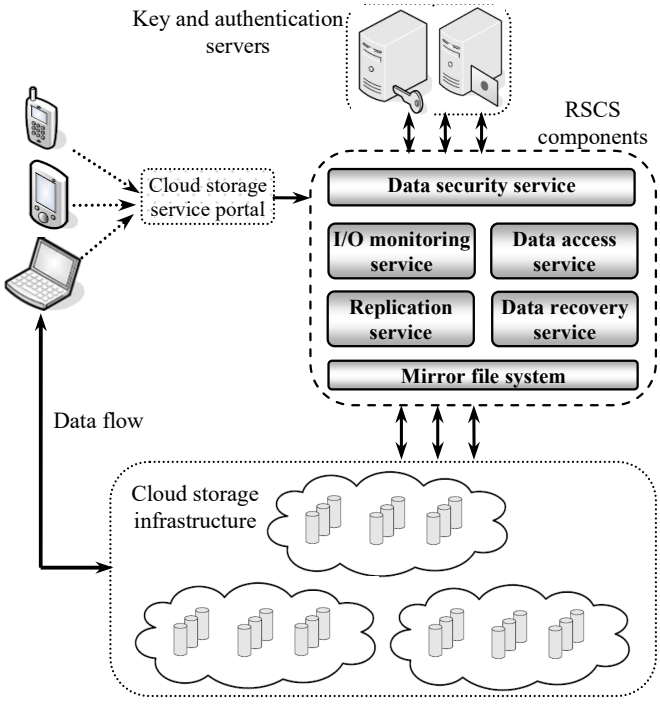
1 improving low-level data reliability

2 guaranteeing up-level data security.

Specifically, data recovery service, replication service and mirror file system are working together to achieve the goal of improving low-level data reliability, while data security service and data access service are responsible for guaranteeing up-level data security. As to the I/O monitoring service, it is designed for measuring the storage-related metrics, including I/O delays, throughput, responsive time, networking traffic, storage resource utilization, etc. As to the underlying cloud-based storage infrastructure, they can be organised by different manners as long as the interfaces of file system are compatible with the interfaces in the mirror file system component.

As we can see in Figure 1, the interactions between RSCS and cloud storage infrastructure are limited to the tasks of data management that is saying, our RSCS does not involve data transferring between users and underlying storage nodes. This is an important designing principle to avoid performance degradation in distributed storage platforms. For example, when a secure data transferring connection is successfully established by the RSCS system, the user request will directly communicate with the underlying storage nodes. In our RSCS framework, all the components are designed in a plug-in manner, which means they can be flexibly replaced by other similar services. Currently, the replication service and I/O monitoring service are directly based on the existing services in our cloud storage middleware since they are working properly. So, in the following sections, we mainly take efforts on describing the key mechanisms implemented in the RSCS framework, including mirror file system, data consistency maintenance and recovery, and secure data accessing tunnel.

Figure 1 Overview of the RSCS framework



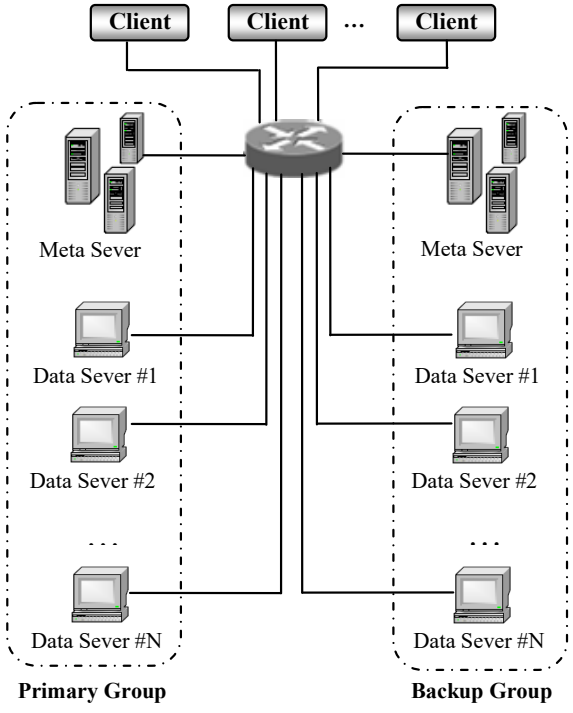
3.2 File system based on mirroring stripped data

In recent years, many approaches have been proposed to improve the fault-tolerance in distribute file systems. Among these approaches, stripping data on multi-RAID nodes is the most-mentioned one due to its simplicity and efficiency. Unfortunately, it can only offer low-level reliability because when multiple storage nodes are crashed at the time, it cannot work properly. The other alternative approach is using parity-based redundancy technology for providing data fault-tolerance, which is very effective in small-scale storage platforms but not suitable for large-scale environments, because multiple storage nodes failing may cause temporary or permanent data inaccessibility which is often negligible in thousands of storage nodes. More importantly, the parity-based redundancy technology is likely to degrade the I/O performance when frequently performing writing on small files. For instance, the writing operation in RAID-5 requires four I/O operations: two for old data parity and two for new data parity. If the I/O system is not optimised, the four I/O operations may lead to significantly I/O latency. Finally, in a distributed file system, the parity calculation should be performed in a distributive manner to avoid performance bottleneck instead of by any single node. As to the erasure-coding based methods, they may be suitable in P2P systems but not for GB/s scale file systems.

In RSCS, we design a simple but effective file system scheme which duplicate stripped data in different storage nodes so as to increase the data reliability and aggregated throughput. As the price of commodity hard disks has dropped rapidly in recent years, we believe that sacrificing some storage capacities for improving the storage performance and reliability is a more sensible option. Unlike the traditional fault-

tolerance approaches (such as parity and erasure coding technique), our solution introduce the extra overheads as less as possible, and the costs of data recovery process are kept in a very low-level. Meanwhile, the proposed solution is adaptive to various kinds of workloads and introduces less complexity in the current storage systems. For example, the data mirroring mechanism in our RSCS can double the performance of I/O operation while only introducing very small overheads on data management. In Figure 2, we demonstrate the framework of mirror-stripped-data file system.

Figure 2 Framework of mirror-stripped-data file system in RSCS



In the above framework, the underlying file systems are responsible for mirroring the stripped data across server nodes. We categorise the server nodes into two groups: primary and backup. The user requests are firstly dispatched to the primary server group for reducing the overheads of synchronisation operation. In the case of the failure of primary metadata server, all metadata requests will be re-dispatched to the backup servers until the primary sever is repaired and rejoins the storage platform. When the system receives a write request, the primary server is firstly used to perform data transferring operation, while the backup servers are used for storing duplicated data. As a result, our RSCS always maintains two sets of metadata structures: system-related metadata and individual file metadata. The former is for managing the life-cycle of storage nodes, since our RSCS allows the storage nodes to join/leave the platform during the runtime. If some failures occur in a storage node, the I/O requests sent to this node will be redirected to the corresponding mirror server. To detect the failures of a storage node in time, we use the heartbeat detecting approach due to its simplicity and cost-effectiveness. Specifically, if a

storage node does not send the periodic heartbeat message to the metadata server within a pre-defined time period, it is considered to be failed regardless of its failing reason.

As to the metadata server, it stores the stripping block layout, the data mirroring state, and other kinds of data information. It is clear that the metadata servers hold the data managing information in the mirror-stripped-data file system. So, it is important to incorporate a backup mechanism in the metadata service, since its failing may lead to the crashing of the whole storage system. In many conventional solutions, backup metadata file server should be implemented in remote storage nodes due to the limitation of namespace in local nodes. More specifically, the file names should be uniqueness which often is the same as its i-node number in the local storage node. To deal with problem, we use the MD-5 sum of the file name as the actual file name in metadata server, which can ensure that files in the metadata servers are always unique and therefore can backup them in the local storage nodes. Here, the only overhead is the cost of MD-5 calculation. According to our experiment, the calculation of a file name requires about 25~100 ns which can almost be ignored in distributed storage platforms.

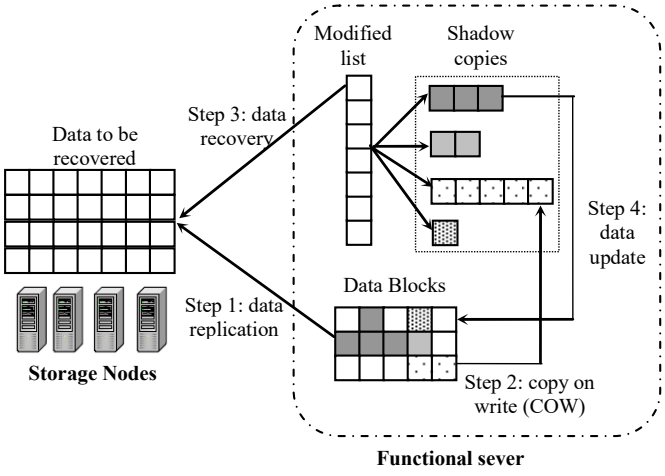
3.3 Data consistency maintenance and recovery

The key issue in a distributed storage system is to meet the concurrent accessing to same files from multiple clients. So, data consistency maintenance and recovery mechanisms play important roles to achieve these goals. In RSCS, we employ a centralised leasing mechanism which allows to accessing different portions of a file based on the multiple-reader-single-writer principle. First, we define the lease as a timed lock which holds certain rights on certain resources during limited time duration. To providing more flexible and fine-grained controlling, we also define that the leases are only based on the logical addresses of file's header and ending. The metadata server will issue exclusive lease of writing to a user as long as there is no existing leases on this data file. On the other side, multiple leases of reading can be assigned to different users if there is no conflicting writing lease exists. Such a leasing mechanism significantly reduces the overheads of maintaining data consistency.

As long as a failed server is detected and rebooted, all data on this storage node need be recovered. The recovery procedure in RSCS is simple and efficient in that all the data can be directly obtained from the mirrored server without any extra costs on communications. Meanwhile, if a storage node is in the process of recovering and receives a writing request, we apply the copy-on-write (COW) mechanism to ensure data consistency as well as uninterrupted storage service. To explain this mechanism, we demonstrate an example in Figure 3 to show the data recovery procedure in the RSCS system.

As shown in Figure 3, the first step is to replicate data located on the mirroring server. If a writing request is overlapped with the recover time, a shadow data copy is dynamically generated which will record the updated content and let the old content unchanged. As long as the old data has been recovered, the shadow copy will bee replicated to the repaired server in a sequential order. In this way, we can avoid data inconsistency. It is noteworthy that there may be several shadow copies for a single recovering data. We store them in a list based on its generated time and write back to the recovered data block. In Figure 3, we can see that the COW mechanism only works on the replicated data which can significantly reduce the probability of data inconsistency if the recovery process is interrupted by some unknown reasons.

Figure 3 Data recovery procedure in RSCS



3.4 Secure data accessing tunnel

In wide-area data sharing environments, data security is always a key concern in the view points of both system administrator and clients. In most of applications, remote procedure calling (RPC) is the most used technology for data accessing. For RPC-based applications, communication security is often supported by other mechanisms instead of the RPC protocol itself, since this offers more flexibility for programmers. For example, secure RPC-based communication can use TCP/IP tunnelling technology, which allows application programmers to separate the logics of data encapsulation from data encryption. In this way, private communication channels can be established in an application-transparent manner. Tunnelling of RPC connections can be obtained by secure sockets layer (SSL) and secure shell (SSH). In our RSCS framework, the underlying file system relies on the interfaces in SSH to establish encrypted and authenticated communication connections between users and storage nodes. Comparing with the existing solutions of section management, the key advantage of our RSCS framework is that it allows to establishing I/O sections based on per I/O requests. Meanwhile, the RSCS framework has incorporated a per-user identity-mapping mechanism which allows to performing ID searching and matching across different management domains. By establishing per-user communication tunnels as well as the per-user ID mapping table, users coming from different administrative domains can use a uniform model of file system section.

To deploy the SSH tunnels and secure file system channels, RSCS only needs the conventional kernel-level routines, such as file system mounting, I/O exporting, pipe commands, etc. In the RSCS framework, the data security model assumes that cloud middleware has incorporated the accounting service for users, which implies that both server-side and client-side can be authenticated through the accounting service. As a result, the cloud middleware is required to permit to mount logic file directory tree to the file system in the proxy. Otherwise, the cloud middleware may not be able to delegate their access controlling service to RSCS's server administrator. In fact, we can easily implement an independent accounting service in the RSCS framework. However, doing

this will introduce unpredictable overheads on the whole system. Considering the case that the kernel server only exports to the local-host and other users cannot mount file systems, the root directory of users will be exported through RSCS, while authentication to this file system will be managed by the kernel account service.

It is noteworthy that the authentication service in RSCS is based on the proxy model, which means that the server-side proxy is responsible for authenticate the I/O requests by establishing secure tunnels. From the perspective users, they are able to present suitable credentials to the RSCS system to ask for a secure tunnel. Unlike the previous authentication mechanisms which only allow trusted IP/Port to perform data accessing operations, our RSCS introduce an inter-proxy authentication mechanism, which uses the local-host to intermediate requests from non-privileged IP/Port. Specifically, we generate a random session key for the inter-proxy when I/O operation is required, and the actual execution of data accessing may be delayed until the user has complete the authentication. In this way, our RSCS can significantly reduce the I/O latency when plenty of users issues massive requests from different locations.

4 Performance evaluation

4.1 Experimental configuration

We deploy the implementation of the RSCS framework in a campus-based cloud platform, where the underlying storage infrastructure is consisted of 12 massive storage clusters connected through Myrinet network, and above 50 distributed storage severs each with 250 GB~1 TB hard disk. To generate various kinds of I/O traffics, we use a modified MPI program as the experimental benchmark, it enables us to make fair performance comparisons on different data accessing and management methods. Besides the simulative I/O traffics, we also use the I/O traffics generated in our experimental test-bed. So, the experimental results are categorised into two classes: real world performance, simulative performance.

4.2 Performance evaluation under real-world traffics

As shown in Figure 1, the I/O monitoring service is designed to evaluate the I/O-related metrics. For the convenience of our experiments, the prototype implementation of RSCS framework has incorporated several extra functions. According to our logs, the total number of registered users is 1,885 and about 21% of these users use the RSCS system over 10 times per day. So, we define them as the active users and the following experimental data are all obtained from this kind of users. In the first set of experiments, we take efforts on examining the performance and efficiency of storage nodes. In Table 1, we demonstrate the results obtained from 2017-4-1 to 2017-4-30.

According to the results shown in Table 1, we can conclude several valuable results on the proposed RSCS system. Firstly, we can see that the speed of downloading is significantly higher than the speed of uploading by about 91%. This is because that the reading speed in user's equipment is generally higher that the writing speed. Also, we notice that the throughput metric is significantly lower than the designing expectation of the RSCS system. Such a result can be contribute to the small number of active users, which means when more active users use the RSCS, the throughput metric will be

increased as well. It is noteworthy, if we only take into account the working time (from 8:00 to 17:00), the throughput metric will be higher than the current result. According to the experimental results, the average data recovering time is 3.25 seconds and the average time of secure tunnelling is only 0.08 seconds. Such a result clearly indicates that the costs of security mechanisms in RSCS system are very acceptable in real-world scenarios.

Table 1 Performance statistics on data operations

Performance metrics	*Max value*	*Min value*	*Mean value*
Data upload speed (MB/sec.)	450	191	335
Data download speed (MB/sec.)	785	112	613
Data throughput (GB/day)	1555	16	256
Response time of file accessing (sec)	3.7	0.4	1.54
Time of data integrity check (sec/GB)	0.6	0.07	0.274
Total storage capability (TB)	33.31	7.42	18.11
Average time of data recovering (sec)	11.2	0.88	3.25
Average time of secure tunnelling	0.31	0.03	0.08
Online active users (/min)	368	0	35
Data size of users (GB)	69	0	13.65

To evaluate the file system I/O performance of the RSCS implementation, we demonstrate the measurements of the I/O related operations in Table 2.

Table 2 File system I/O Performance in RSCS

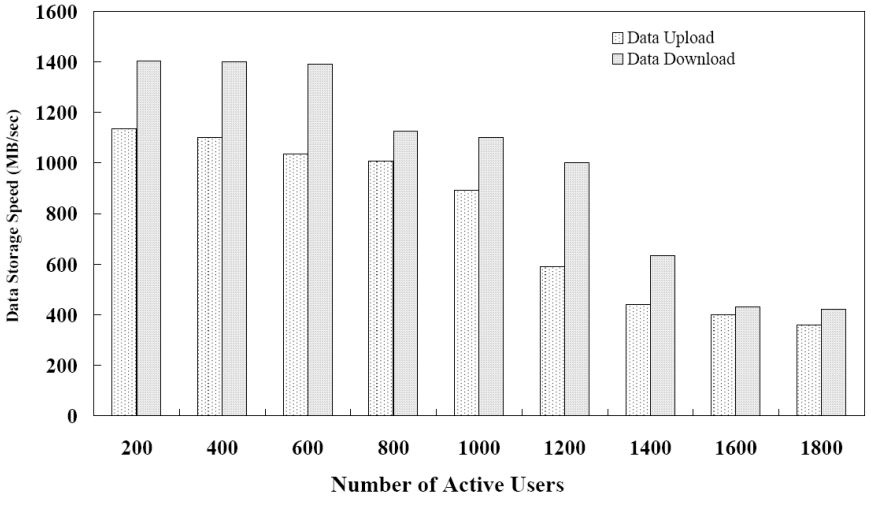
Operation	*Time (s)*	*I/O Vol.*	*Percentage in I/O time*	*Percentage in exec time*
Open	1.33	N/A	1.5%	0.08%
Read	14.4	20MB	32.5%	36.2%
Seek	1.25	N/A	1.06%	0.06%
Write	25.6	50MB	61.15%	66.3%
Close	0.21	N/A	0.13%	0.01%

According to the results in Table 2, we first notice that read and write operations almost occupy 99% I/O processing time due to the I/O feature of our tested benchmarks. Specifically, the tested benchmarks will repeatedly read data in different iterations and then write the processed data consequently. According to the results in Table 2, we can guess that the reading speed and writing speed is very close in file system, which can be explained by the I/O buffer mechanism in local operation system. As to the other operations (e.g., open, close, seek, and flush), their total costs are less than 2% of the overall I/O costs. Based on the abovementioned experimental data, it can be concluded that the RSCS prototype system is able to cope with the scenarios where there are plenty of active users asking for retrieving small data files. Limited by our experimental environment, if the number of potential active users has exceeded over a certain level, we believe that performance bottleneck will occur at the link layer instead of the RSCS system.

4.3 Performance evaluation under simulative traffics

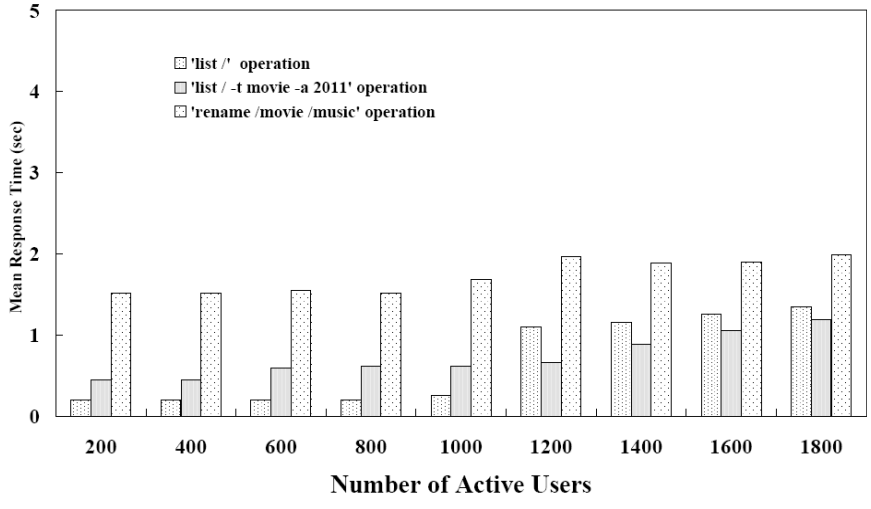
In this set of experiments, we select 55 PCs to simulate the behaviours of users with aiming at simulating the real-world scenarios as possible as we can. The selected PCs are located at different geographical positions, and are executing a specially designed software agent, which create continuous user requests to the RSCS system based on certain random distribution model. In Figure 4, we demonstrate the upload/download speeds of the RSCS system under the scenarios with different number of active users.

Figure 4 Data storage speed with various numbers of active users



In our experiments, we gradually increase the active user number from 200 to 1,800. It is clear that the speed of upload/download will be reduced with the increasing of active users. Even so, we still notice that such a performance decreasing is not very significant at the beginning (when the number of active users is less than 800). So, we conclude that the number of 800 active users seems to be a threshold in our test-bed platform. For other system, this number may be less or more. As long as the number exceeds over this threshold (800), we can see that upload/download speed will be decreased rapidly. When the number of active users has reached 1,800, we notice that upload/download speed is only 25% of the maximal speed. Such a performance level can be maintained for a while if more active users join the system. Based on these experimental data, we conclude that the proposed RSCS system can accommodate at least 1,800 active users concurrently. It is noteworthy that the number of active users in RSCS is generally about 10% of the total users in our cloud platform. Therefore, we believe that the current RSCS system can provide data storage service for those mobile communities where the total number of users is less than 20,000.

Figure 5 Response time of file system navigation operations



In order to examine the performance when interacting with users, we investigate the response time when users ask for file navigating operations. In Figure 5, we demonstrate the experimental results on three file navigating operations:

1 'list /' is to ask for the list of directories in user's root file system

2 'list / -t move –a 2011' is to query all the movie files s that stored in 2011

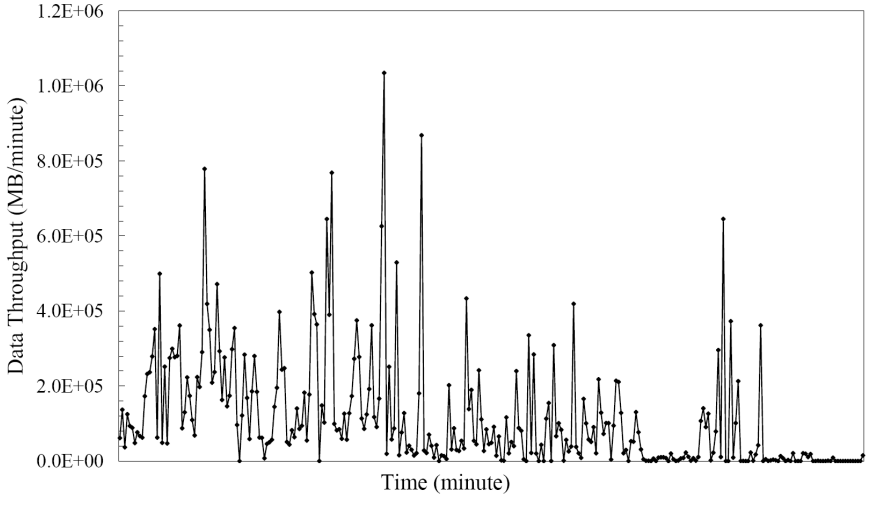
3 'rename /movie /music' is to rename a user directory.

The results in Figure 5 have shown several interesting observations. At first, we notice that file name changing operation seems to have more delays than other file navigating operations, while its performance seems to be not affected by the number of active users. By analysing the logs of file system callings, we find that it is because that the filename changing operation often requires re-mapping the namespace in distributed file system, which is a more cost-consuming operations comparing with other navigating operations. Secondly, we notice that the response time when performing 'list /' operation is closely related with the number of active users as long as this number is larger than 1000. Such a result can be attributed to the I/O accessing patterns of users. For example, we find that the frequency of 'list /' operation is nearly six times of the 'list / -t move –a 2011' operation in our experiments. As a result, when more and more users join the RSCS system, the response time of this operation will be significantly decreased.

Finally, we demonstrate the throughput metric of the proposed RSCS when the test-bed platform is facing very intensive workloads (the number of active number is 1,800). To reduce the size of measuring data, we set the sampling frequency is one minute. According to data shown in Figure 6, we can see the throughput metric fluctuates dramatically during the execution time. Such a performance mainly affected by the user's I/O accessing patterns. To deal with the problem of bursty workloads, some auto-scaling mechanism may be useful, which may be our future work. Currently, our designing objective of RSCS is that it can provide about 50 TB throughput per day. According to our experimental results, it is clear that current throughput metric is nearly 13% of the

designing limitation, which means it can be applied to those cloud systems with more intensive data-accessing requirements.

Figure 6 Real-time throughput in simulative experiments with 1,800 active users



5 Summary and future work

To provide better storage service in nowadays cloud environments, in this study we propose a novel framework called RSCS, which consists of several well-designed components to improve low-level data reliability as well as guarantee up-level data accessing security. In the RSCS framework, a simple yet effective file system scheme is proposed, which can duplicate stripped data in different storage nodes so as to increase the data reliability and aggregated throughput. We also introduce a centralised leasing mechanism which allows to accessing different portions of a file based on the multiple-reader-single-writer principle. Finally, we provide a secure data accessing tunnel technology which allows the RSCS to establish secure communication channels between users and storage nodes without introducing too many extra costs. In a real-world cloud platform, we conduct sets of experiments and the results show that the proposed RSCS framework is able to meet the requirements for most of cloud-based storage platforms. According to our experiments, we have shown that when the number of active users is less than 20000, the RSCS platform perform well in different experimental scenarios. Meanwhile, the experimental results also reveal some limitations in the proposed RSCS system. For example, the interacting performance of RSCS is affected by the number of users, and the data throughput metric changes dramatically when facing intensive workloads. So, we are planning to conduct more performance evaluations in more scenarios with aiming to find more characteristics of the RSCS. Also, we are planning to incorporate some auto-scaling mechanism in the data transferring service in the RSCS. Finally, we are planning to device some load-balancing mechanism in the I/O layer of RSCS with aiming at obtaining more stable performance.

Acknowledgements

This work is supported by a grant from the National Natural Science Foundation of China (No. 61402163) the Research Foundation of Education Bureau of Hunan Province, China(Grant No. 18A343).

References

Anglano, C., Gaeta, R. et al. (2015) 'Exploiting rateless codes in cloud storage systems', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, No. 5, pp.1313–1322.

Bohli, J-M., Gruschka, N. et al. (2013) 'Security and privacy-enhancing multicloud architectures', *IEEE Transactions on Dependable and Secure Computing*, Vol. 10, No. 4, pp.212–224.

Canali, C. and Lancellotti, R. (2014) 'Improving scalability of cloud monitoring through PCA-based clustering of virtual machines', *Journal of Computer Science and Technology*, Vol. 29, No. 1, pp.38–52.

Celesti, A., Fazio, M. et al. (2016) 'Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems', *Journal of Network and Computer Applications*, Vol. 59, No. 2, pp.208–218.

Chen, H.C.H., Hu, Y. et al. (2014) 'NCCloud: a network-coding-based storage system in a cloud-of-clouds', *IEEE Transactions on Computers*, Vol. 63, No. 1, pp.31–44.

Cui, B., Liu, Z. et al. (2016) 'Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage', *IEEE Transactions on Computers*, Vol. 65, No. 8, pp.2374–2385.

Di, S., Kondo, D. et al. (2015) 'Optimization of composite cloud service processing with virtual machines', *IEEE Transactions on Computers*, Vol. 64, No. 6, pp.1755–1768.

Duan, H., Yu, S. et al. (2015) 'CSTORE: a desktop-oriented distributed public cloud storage system', *Computers and Electrical Engineering*, Vol. 42, No. 1, pp.60–73.

Goncalves, G.D., Drago, I. et al. (2016) 'Workload models and performance evaluation of cloud storage services', *Computer Networks*, Vol. 109, No. 1, pp.183–199.

Guan, X. and Choi, B-Y. (2014) 'Push or pull? Toward optimal content delivery using cloud storage', *Journal of Network and Computer Applications*, Vol. 40, No. 1, pp.234–243.

Khan, A.N., Kiah, M.L.M. et al. (2014) 'BSS: block-based sharing scheme for secure data storage services in mobile cloud environment', *Journal of Supercomputing*, Vol. 70, No. 2, pp.946–976.

Lagar-Cavilla, H.A., Whitney, J.A. et al. (2011) 'SnowFlock: virtual machine cloning as a first-class cloud primitive', *ACM Transactions on Computer Systems*, Vol. 29, No. 1, pp.1–44.

Liang, G. and Kozat, U.C. (2014) 'FAST CLOUD: pushing the envelope on delay performance of cloud storage with coding', *IEEE/ACM Transactions on Networking*, Vol. 22, No. 6, pp.2012–2025.

Long, S-Q., Zhao, Y-L. et al. (2014) 'MORM: a multi-objective optimized replication management strategy for cloud storage cluster', *Journal of Systems Architecture*, Vol. 60, No. 2, pp.234–244.

Mahajan, P., Setty, S. et al. (2011) 'Depot: cloud storage with minimal trust', *ACM Transactions on Computer Systems*, Vol. 29, No. 4, pp.1–45.

Mao, B., Jiang, H. et al. (2014) 'Read-performance optimization for deduplication-based storage systems in the cloud', *ACM Transactions on Storage*, Vol. 10, No. 2, pp.1–38.

Marmol, F.G. and Kuhnen, M.Q. (2015) 'Reputation-based web service orchestration in cloud computing: a survey', *Concurrency Computation*, Vol. 27, No. 9, pp.2390–2412.

Ousterhout, J., Gopalan, A. et al. (2015) 'The RAMCloud storage system', *ACM Transactions on Computer Systems*, Vol. 33, No. 3, pp.1–44.

Shamsi, J., Khojaye, M.A. et al. (2013) 'Data-intensive cloud computing: requirements, expectations, challenges, and solutions', *Journal of Grid Computing*, Vol. 11, No. 2, pp.281–310.

Sindhu, R. and Mushtaque, A. (2014) 'A new innovation on user's level security for storage data in cloud computing', *International Journal of Grid and Distributed Computing*, Vol. 7, No. 3, pp.213–220.

Song, S., Khil, K-J. et al. (2013) 'Software RAID for data intensive applications in cloud computing', *Journal of Internet Technology*, Vol. 14, No. 3, pp.529–534.

Subashini, S. and Kavitha, V. (2011) 'A survey on security issues in service delivery models of cloud computing', *Journal of Network and Computer Applications*, Vol. 34, No. 1, pp.1–11.

Wan, J., Zhang, J. et al. (2013) 'ORTHRUS: a light weighted block-level cloud storage system', *Cluster Computing*, Vol. 16, No. 4, pp.625–638.

Wang, Y., Lu, P. et al. (2015) 'WaFS: a workflow-aware file system for effective storage utilization in the cloud', *IEEE Transactions on Computers*, Vol. 64, No. 9, pp.2716–2729.

Xiao, P. and Han, N. (2014) 'A novel power-conscious scheduling algorithm for data-intensive precedence-constrained applications in cloud environments', *International Journal of High Performance Computing and Networking*, Vol. 7, No. 4, pp.299–306.

Zhang, D., Coddington, P. et al. (2013) 'Improving data transfer performance of web service workflows in the cloud environment', *International Journal of Computational Science and Engineering*, Vol. 8, No. 3, pp.198–209.

Zhu, Y., Huang, D. et al. (2015) 'From RBAC to ABAC: constructing flexible data access control for cloud storage services', *IEEE Transactions on Services Computing*, Vol. 8, No. 4, pp.601–616.