File Placement Mechanisms for Improving Write Throughputs of Cloud Storage Services Based on Ceph and HDFS

Chun-Feng Wu[1], Tse-Chuan Hsu[2], Hongji Yang[2], and Yeh-Ching Chung[3]
[1] Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
Email: tom.cfwu@gmail.com
[2] Centre for Creative Computing, Bath Spa University, England
Email: davidhsu@hcu.edu.tw, h.yang@bathspa.ac.uk
[3] Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China
Email: yehching.chung@gmail.com

Abstract

Cloud storage services are pervasive nowadays. Many cloud storage services use distributed file systems as their backend storage systems. Some research results on file-size distribution of file systems show that file systems contain lots of small files. Therefore, this paper proposed a hybrid distributed file system based on Ceph and HDFS that can deliver satisfactory write throughputs for a cloud storage system with 80-90% small files and 10-20% large files. The experimental results show that the file allocation mechanism without RAM disk caching can improve the write throughputs of Ceph and HDFS by approximately 10% to 50%. While the one with RAM disk caching can have up to 200% write throughputs improvement.

Key words: Hybrid Distributed File System, Cloud Storage, Ceph, HDFS

1. Introduction

Cloud storage services, such as Dropbox, Google Drive, iCloud, etc., are pervasive nowadays. Many cloud storage services use distributed file systems as their backend storage systems [1][11][12]. Some research results on file-size distribution of file systems show that file systems contain lots of small files, such as document files, image files, music files, etc., which sizes in general are less than 10MB. But most capacity of the cloud storage system is occupied by few large files, such as operating system image files (Ubuntu for example), video files (movies for example), etc., which sizes in general are greater than 1GB. In addition, the distribution of file sizes obeys Heavy-Tailed Distribution according to past research results. As a result, we could assume that 80-90% files in a cloud storage system are small files while 10-20% are large files. In another word, file systems contain lots of small files, but most capacity is occupied by few large files [14].

Different distributed file systems may have different design goals. For example, some of them are designed to have good performance for small file operations, such as Ceph, while some of them are designed for large file operations, such as Hadoop distributed file system. With the divergence of applications, a distributed file system may provide good performance for some applications but fails for some other applications, that is, there has no universal distributed file system that can produce good performance for all applications.

In [7], the research results indicate that Ceph and HDFS are more suitable for write operation with small files and large files, respectively. In order to deliver satisfactory write throughputs for a cloud storage system with 80-90% small files and 10-20% large files, in this paper, we proposed a hybrid distributed file system based on Ceph and HDFS. According to the characteristics of Ceph and HDFS, we proposed two file placement mechanisms based on memory resources. One is to store files with limited memory resource. In this mechanism, a K-Nearest Neighbors algorithm (KNN) [11] is used to decide where to store a file (Ceph or HDFS) based on the input file size. The other is to store files with sufficient memory resource. In this mechanism, we use Ramdisk with caching and parallel programming technique to improve write throughputs in the proposed hybrid distributed file system.

We have implemented the proposed hybrid distributed file system for SSBox, a Dropbox like cloud storage service. The experimental results show that the file placement mechanism without RAM disk caching can improve the write throughputs of Ceph and HDFS by approximately 10% to 50%. While the one with RAM disk caching can have up to 200% write throughputs improvement.

The rest of the paper is organized as follows. Section 2 discusses the background and related work. Section 3 presents hybrid mechanisms. The experiments designed are described in Section 4. After that we show evaluation results and discuss the improvement in Section 5.

2. Background and Related Work

Distributions of files based on file sizes in computer system and in World Wide Web obey Heavy-Tailed Distribution [15][16]. Some research results in analyzing storage system show that there are around 90% or even more higher ratios of small files in storage systems [17][18]. Moreover, to elevate write throughputs in cloud storage, making good use of Heavy-Tailed Distribution is crucial.

Ceph [2] is a reliable and scalable distributed file system which provides some advantages such as avoiding single point of failures, using replicas to achieve fault tolerance, POSIX compliant, etc. The architecture of Ceph shows that all interfaces in Ceph are built on RADOS [5]. In this paper, we focus on the file level interface and three components, monitors (MON), object storage devices (OSD) and metadata servers (MDS). MON is used to issue heartbeats messages to ensure all components are healthy. OSD is used to store files and replicas by using CRUSH algorithm [4]. MDS is used to store metadata of files and replicas.

HDFS [3], derived from Google File System (GFS) [13], is a distributed file system that aims to tackle large data sets reliably and to stream large data sets at high bandwidth to clients. There are two important components in HDFS, NameNode and DataNode. The NameNode maintains namespace tree and the locations for each file blocks on DataNodes. The DataNodes are used to store physical files in HDFS.

SSBox [9], designed and implemented by SSLAB, National Tsing-Hua University, is a Dropbox-like system and provides services from SaaS to PaaS. SSBox is mainly composed by Nginx, Memcached, PostgreSQL, Server-core and Ceph. The system architecture is shown in Fig 1. In Fig. 1, incoming queries will be redirected to Server-core by Nginx, an inverse proxy service. Server-core responses to manage APIs and communicates with PostgreSQL. When managing APIs such as download files, Server-core may need to get files from Ceph. Memcached is an in-memory database which could improve the performance in download scenario.

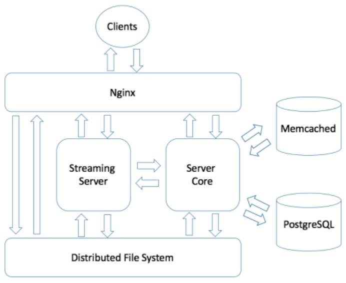

Fig 1. SSBox Architecture

3. The File Placement Mechanism

A. Without RAM disk caching

Algorithm 1 shows the file placement mechanism without RAM disk caching mechanism. In Algorithm 1, parameter **α** represents a critical point in which the write performance of Ceph is better than that of HDFS for a given file with size ≤ **α**. Parameter **β** represents a critical point in which the write performance of HDFS is better than that of Ceph for a given file with size > **β**. For a given file with size in between **α** and **β**, the write performance of Ceph and HDFS depends on the current status of Ceph and HDFS.

To determine the values of **α** and **β**, we conduct a file size prediction test with different file sizes on Ceph and HDFS on SSBox. The detailed specifications of SSBox will be given in Section 4. Fig. 2 shows the predicting results. From Fig 2, we can see that the write performance of Ceph is better than that of HDFS when the file size is less than or equal to 500MB (the value of **α**). When the file size is greater than 800MB (the value of **β**), the write performance of HDFS is better than that of Ceph. Therefore, the values of **α** and **β** are set to 500MB and 800MB in Algorithm 1, respectively. Note that for different systems, the values of **α** and **β** may be different.

With the values of **α** and **β**, Algorithm 1 stores files to Ceph when the file size is smaller than or equal to 500MB and stores files to HDFS when the file size is greater than 800MB. For files with sizes in between **α** and **β**, Algorithm 1 uses KNN algorithm to decide where to store them. KNN is a supervised machine learning algorithm. In this paper, in the classification phase, an unlabeled vector is classified by the most frequent label among the K (a user-defined constant), and training samples which are measured by Euclidean distances.

Algorithm 1 Hybrid Mechanism For Limited Resources

```
1: Definition: Get α and β by file size preprocessing. KNN represents K-
   Nearest Neighbors algorithm.
2: procedure HYBRID_MECHANISM_FOR_SUFFICIENT_RESOURCES(data, size)
3:     if size > β then
4:         HDFS ← data
5:     else if size <= α then
6:         Ceph ← data
7:     else
8:         if KNN.Predict(size) = 0 then
9:             HDFS ← data
10:        else
11:            Ceph ← data
12:        end if
13:    end if
14: end procedure
```

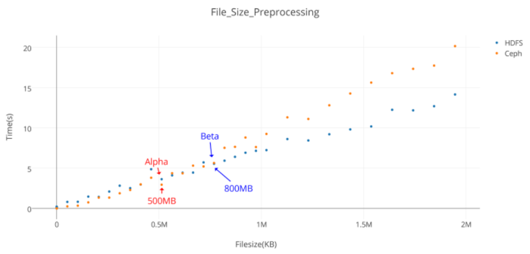

Fig 2. Results of file size prediction

B. With RAM disk caching

Algorithm 2 shows the file placement mechanism with RAM disk caching mechanism. In Algorithm 2, the values of **α** and **β** and KNN prediction function in are different from those in Algorithm 1. They need to take the file writing time of Ceph and RAM disk into consideration. Based on observations, the system write thoughts may be increased when the file writing time of Ceph is smaller than twice of the file writing time of RAM Disk. Therefore, **α** and **β** is set to 40MB and 120MB in Algorithm 2.

All coming queries may be scheduled sequentially when multiple files uploaded to the cloud storage system, SSBox. Therefore, if we have sufficient RAM capacity as RAM disk, say 100GB on each node in our system, the proposed mechanism could cache large files in the RAM Disk. The files cached in RAM disk could be written to Ceph or HDFS by using parallel programming technique to speed up the writing time. There are two reasons to cache large files in the RAM Disk instead of caching small files. One is that writing large files to the RAM Disk is dramatically faster than writing them into Ceph or HDFS. The other is that the time to write small files (less than 40MB) to Ceph is faster than that to write them into the RAM Disk and then writing them to Ceph or HDFS in parallel.

Algorithm 2 Hybrid Mechanism For Sufficient Resources

```
1: Definition: Get $\alpha$ and $\beta$ by file size preprocessing. KNN represents K-
   Nearest Neighbors algorithm. RAMDisk represents the RAM disk folder.
2: procedure HYBRID_MECHANISM_FOR_SUFFICIENT_RESOURCES($data, size$)
3:     if $size > \beta$ then
4:         $RAMDisk \leftarrow data$                          ▷ Cache files in RAM Disk
5:     else if $size <= \alpha$ then
6:         $Ceph \leftarrow data$
7:     else
8:         if KNN.Predict(size) = 0 then
9:             $RAMDisk \leftarrow data$
10:        else
11:            $Ceph \leftarrow data$
12:        end if
13:    end if
14:    if $RAMDisk.full() = True$ then
15:        $HDFS.Parallel\_Write(RAMDisk.getall())$
16:        $RAMDisk.delete(RAMDisk.getall())$
17:    end if
18: end procedure
```

4. System Configuration and Experiments Design

To validate the proposed mechanisms, we set up a cluster with five nodes. Each node is a server that is equiped with 20 Intel CPU cores and a QDR InfiniBand (40Gbps) card. The file system used is Btrfs. Whole system is set up with Ubuntu 14.04.4 and Linux kernel version is 4.4. The version of Ceph in the experiments is Infernails 9.2 and the version of HDFS is 2.4.1.

To equalize disk resources between Ceph, HDFS and the proposed hybrid system, all disks are shared by Ceph and HDFS. In this setting, we could avoid hybrid system to take any benefits from the system setting. There are 10 HDDs in our system. Each HDD is divided into 2 equal-size partitions, one for Ceph and the other for HDFS. For the Ceph used in the experiments, it contains 3 MONS, 10 OSDs and 1 MDS. Block sizes in Ceph are set to the default size, 4MB. Hadoop is installed on the same 5 nodes where Ceph has installed. Block sizes in HDFS are set to default size, 64MB. Each file will have three replicas no matter whether it is in Ceph or in HDFS.

For experiments, we create ten file sets and run the experiments on two environments, local and SSBox. The local and SSBox environments are used to test the proposed mechanisms without/with cloud storage services, respectively. Two data groups are used for the evaluations. Each data group contains five file sets and each file set has 500 files. The main difference of the two data groups is the ratio of small and large files. Table 2 shows the composition of two data groups.

The SSBox cloud storage service will be deployed in one of the 5 nodes. In SSBox, data is written to HDFS with the HDFS API that is Python package for Hadoop and is written to Ceph with POSIX write APIs. Data will be renamed with the hash of its contents and the location will be recorded in PostgreSQL database. The execution flow of uploading files into SSBox is shown in Fig. 3.

In the local environment, files read or write are processed sequentially inside a local cluster by using file operation APIs provided in Ceph, HDFS, hybrid mechanism without RAM disk caching (HLR), and hybrid mechanism with RAM disk caching (HSR). In the SSBox environment, a client side in different clusters connected together with an Ethernet will sequentially uploads files from 6 file sets, file set 1, file set 2 and file set 3 in the first and the second data group shown in Table 2, to SSBox with RESTful APIs.

Table 2. Composition of 10 File Sets

	File set 1	File set 2	File set 3	File set 4	File set 5
Data group1	50_400M 450_1M	50_800M 450_1M	50_1500M 450_1M	50_2500M 450_1M	50_4000M 450_1M
Data group2	100_400M 400_1M	100_800M 400_1M	100_1500M 400_1M	100_2500M 400_1M	100_4000M 400_1M

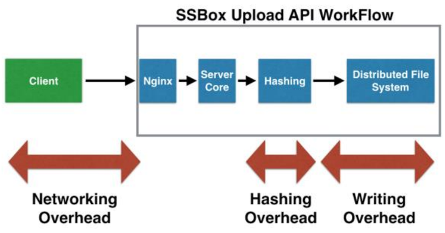

Fig 3. The execution flow of SSBox upload API

5. Evaluations

A. Local Environment

Fig. 4 and Fig. 5 show the write performance of Ceph, HDFS, and the proposed hybrid system with file sets in the first and the second data groups, respectively. From Fig.4 and Fig. 5, we observe that the file placement mechanism without RAM disk caching can improve the write throughputs of Ceph and HDFS by approximately 10% to 50%. While the one with RAM disk caching can have up to 200% write throughputs improvement. From Fig. 4 and Fig. 5, we have the following observations:

Observation 1: The write throughputs of Ceph and HDFS are sensitive to file sizes. However, the proposed hybrid system avoids the fluctuated phenomenon and shows stable write throughputs compared with Ceph and HDFS.

Observation 2: HSR has very high throughputs in the first file set of two data groups. Depending on the design of HSR, files managed by Ceph should be finished earlier than twice the time that files are written to RAM disk. Fig. 4 and Fig. 5 show that the throughput of HSR is around 900MB for the first file set which is approximately half the RAM disk's writing time. This indicates that most files larger than 1MB in the first file sets are accessed by Ceph. Therefore, when the maximum file size grows, there may be more files managed by HDFS and the overhead of writing files from RAM Disk to HDFS may show.

B. SSBox environment

In SSBox environment, Fig. 6 shows that the networking overheads and hashing overheads may be nearly the same for Ceph, HDFS, and the proposed hybrid system. The reason is that all files are managed sequentially and there has no pipeline acceleration for networking, hashing and writing steps. According to the result, we could infer that the more portions of networking overheads and hashing overheads are, the more overall overheads are saved. The result shows that the HSR could save 12% to 15% overheads for SSBox.

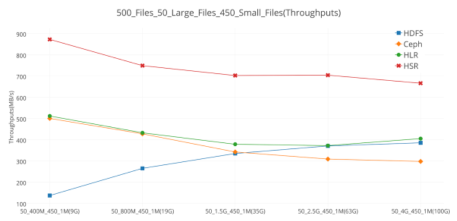

Fig 4. Result after Running the First Data Group

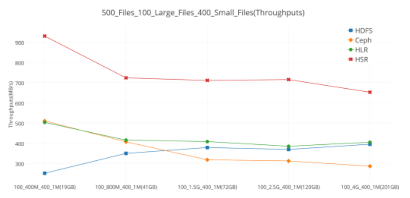

Fig 5. Result after Running the Second Data Group

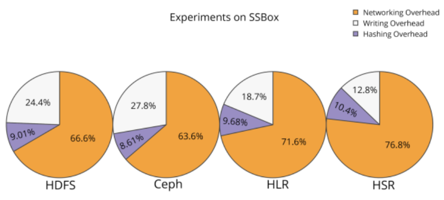

Fig 6. Results of Experiments on SSBox

6. Conclusions

As cloud storage becomes widespread nowadays, write throughputs may be a bottleneck for cloud servers. To improve write throughputs, this paper proposed two mechanisms to hybrid two distributed file systems, Ceph and HDFS, according to the amount of memory resources. These hybrid mechanisms mainly composed of three parts, the file size prediction, KNN clustering, and RAM disk caching, to enhance the write throughputs of proposed hybrid system. The experimental results show that the proposed hybrid system can have 20% to 200% write throughput gain compare with Ceph or HDFS. In addition, the proposed hybrid mechanisms could eliminate 12% to 15% overheads caused by upload operation in SSBox.

References

[1] Jun, S., and Sha-sha, Y. (2011, May). The application of cloud storage technology in SMEs. In E-Business and E-Government (ICEE), 2011 International Conference on (pp. 1-5). IEEE.

[2] Grossman, R. L., Gu, Y., Sabala, M., and Zhang, W. (2009). Compute and storage clouds using wide area high performance networks. Future Generation Computer Systems, 25(2), 179-183.

[3] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., and Maltzahn, C. (2006, November). Ceph: A scalable, high-performance distributed file system. In Proceedings of the 7th symposium on Operating systems design and implementation (pp. 307-320). USENIX Association.

[4] Weil, S. A., Brandt, S. A., Miller, E. L., and Maltzahn, C. (2006, November). CRUSH: Controlled, scalable, decentralized placement of replicated data. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (p. 122). ACM.

[5] Weil, S. A., Leung, A. W., Brandt, S. A., and Maltzahn, C. (2007, November). Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing '07 (pp. 35-44). ACM.

[6] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010, May). The hadoop distributed file system. In Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on (pp. 1-10). IEEE.

[7] Depardon, B., Séguin, C., and Mahec, G. L. (2013). Analysis of six distributed file systems Tech. Rep. hal-00789086 Université de Picardie Jules Verne.

[8] Donvito, G., Marzulli, G., and Diacono, D. (2014). Testing of several distributed file-systems (HDFS, Ceph and GlusterFS) for supporting the HEP experiments analysis. In Journal of Physics: Conference Series (Vol. 513, No. 4, p. 042014). IOP Publishing.

[9] Hsing-Chang Chou, Che-Rung Lee, and Yeh-Ching Chung. (2015) Container-Based Scale-Out Architecture for Cloud Storage Service. Master Thesis in National Tsing Hua University.

[10] Cover, T. M., and Hart, P. E. (1967). Nearest neighbor pattern classification. Information Theory, IEEE Transactions on, 13(1), 21-27.

[11] Zeng, W., Zhao, Y., Ou, K., and Song, W. (2009, November). Research on cloud storage architecture and key technologies. In Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (pp. 1044-1048). ACM.

[12] Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., ... and Haridas, J. (2011, October). Windows Azure Storage: a highly available cloud storage service with strong consistency. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (pp. 143-157). ACM.

[13] Ghemawat, S., Gobioff, H., and Leung, S. T. (2003, October). The Google file system. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 29-43). ACM.

[14] Downey, A. B. (2001). The structural cause of file size distributions. In Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on (pp. 361-370). IEEE.

[15] Barford, P., and Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. ACM SIGMETRICS Performance Evaluation Review, 26(1), 151-160.

[16] Crovella, M. E., Taqqu, M. S., and Bestavros, A. (1998). Heavy-tailed probability distributions in the World Wide Web. A practical guide to heavy tails, 1, 3-26.

[17] Welch, B., and Noer, G. (2013, May). Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions. In Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on (pp. 1-12). IEEE.

[18] Agrawal, N., Bolosky, W. J., Douceur, J. R., and Lorch, J. R. (2007). A five-year study of file-system metadata. ACM Transactions on Storage (TOS), 3(3), 9.