FSObserver: A Performance Measurement and Monitoring Tool for Distributed Storage Systems

Xiao Zhang[1,2](✉), Lanxin Kong[1], Shunyi Zhu[1], Zhanhuai Li[1,2], and Xiaonan Zhao[1,2]

[1] School of Computer Science, Northwestern Polytechnical University, Xi'an, China
zhangxiao@nwpu.edu.cn
[2] MIIT Key Laboratory of Big Data Storage and Management, Xi'an, China

Abstract. It is a big challenge to measure and monitor the performance of a large-scale distributed storage system accurately. We present a flexible approach based on the message analysis, named FSObserver, which can accurately and fine-grained trace individual request or response by observing network traffic. Experiments results show that our approach can get accurate performance with slight performance degradation.

1 Introduction

Over the past few years, there are tremendous efforts to evaluate and debug the performance problems of large-scale distributed storage systems. Some practitioners concentrate on monitoring individual devices and machines independently. Some researchers focus on detailed analysis of all messages by inserting some unique IDs into messages during instrumenting the system. Some others are immersed in the study of the storage system log [1]. System evaluation based on inner messages analysis has been intensively studied in [2,3]. When designing a monitoring and evaluation system, we should consider the independence, accuracy, high performance, and broad-applicability.

In this paper, we propose FSObserver, an out-of-band approach to capture performance related messages between clients and servers. It extracts performance characteristics from the messages. The core idea of FSObserver is to capture the request and reply messages between clients and servers. It extracts the time, size and operations information from messages. The size of this information is very small compared with the size of the messages. By analyzing each individual message, we can accurately evaluate the performance characteristics such as IOPS, throughput, and latency.

In the following sections, we discuss how to monitor and evaluate the Ceph distributed file system, and show the experiment results.

https://doi.org/10.1007/978-3-030-05677-3_14

The remainder of the paper is organized as follows. In Sect. 2 we present the related work that evaluates Ceph and other distributed storage systems. Section 3 describes the architecture of FSObserver, and explains how it works on the Ceph distributed file system. Section 4 illustrates the performance evaluation results of FSObserver and other widely used tools fio[1] respectively. Finally, we conclude this paper and present the future work of this study.

2 Related Work

Performance of a distributed storage system is very important in data centers. Past studies proposed various methods to debug and diagnose the systems. They concentrated on the in-band and out-of-band monitoring systems, black box and white box, intrusive, and log analysis and so on. There are several tools developed to monitor the performance of Ceph clusters[2]. Many black-box diagnosis techniques have been devised for performance evaluation in distributed systems. Dianna *et al.* used 5 tools to evaluate the performance and scalability of the Ceph distributed storage system [4]. Wang *et al.* evaluated the file and block I/O performance and scalability of Ceph, using a commercial high-end storage system [5]. Computer-system logs provide a glimpse into the states of a running system, and system diagnostics research around logs never stopped [6].

3 Design and Implement

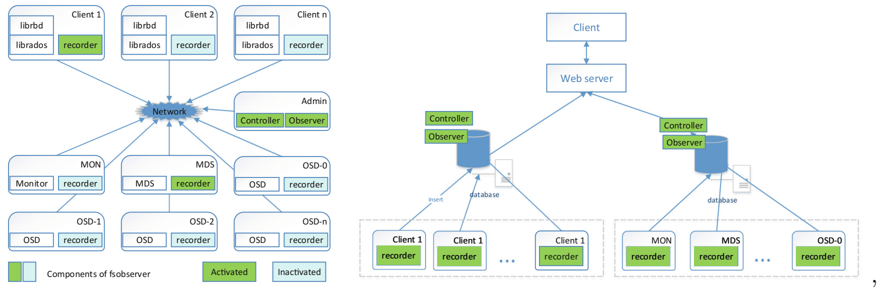
3.1 Architecture of FSObserver

The FSObserver are designed to measure different aspects of performance in a large-scale distributed file system by packets analysis. We designed a flexible packet analyzer, which can capture related packets and save a little information from the payload. The analyzers can be turned on/off by a controller. These designs can get performance data without too much impact on the system. Figure 1(a) shows the architecture of FSObserver. There are 3 kinds of components in the FSObserver. The recorder processes can be turned on/off dynamically.

- Recorder
  It captures related packets using libpcap[3], which is a portable C++ library for network traffic capture. When a recorder process captures a read request packet, it analyzes the header of payload and outputs key information, such as time, transaction id, data length. We measure the impact of the recorder in several read/write scenarios. The performance impact is less than 5%.
- Controller
  It can start/stop some observers according to the administrator's input. The FSObserver can be used flexibly for various purposes.

[1] http://freshmeat.net/projects/fio/.
[2] http://www.ceph.com/performance/.
[3] http://www.tcpdump.org/.

– Observer
  This is a python program for analyzing the output of observers. It gets IOPS by counting how many finished IO requests in a given period in the results. In our prototype, we output the performance information into a text file inside the nodes under test. Meanwhile, we run the observer on the same nodes. The performance characteristic of the nodes including IOPS, throughput, and latency can be calculated through one sequential scan of the text file. In a large-scale system, we can use the mechanism similar to ganglia. As shown in Fig. 1(b), we divide nodes into different monitor group. The nodes in one group save the raw performance data in a database like MySQL or RRDtool.



(a) Components of FSObserver (b) FSObserver in large-scale systems

Fig. 1. Architecture of FSObserver

3.2 Implement of FSObserver for Ceph

After capturing a related packet using libpcap, it extracts the necessary information for performance measurement. Our main purpose is to get performance data, so we only need to analyze the messages with tag equals to 0x07. Further, we can only capture and analyze messages from a specific client.

The observer program analyzes the results from recorders. It can get IOPS, throughput and latency data from the results. For example, IOPS is calculated by counting the number of transactions finished in a given period. To analyze the performance of a certain client, we only need to deploy a recorder on the client. We can also get the same metric from records from all related OSD nodes with the client. We put the implementation on the GitHub[4].

4 Evaluation

We evaluate the accuracy and application of our tools. First, we compare the test results of FSObserver and widely used benchmarks to show the accuracy

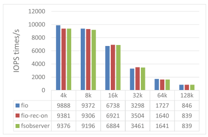
[4] https://github.com/zhangxiao2000/fsobserver.

of our tools. Then, we measure the performance of a real user application to demonstrate how to use FSObserver in a real environment. The test environment comprised 10 commodity servers. 6 nodes work as Ceph servers, 4 nodes act as Ceph clients. The release version of Ceph is 12.2.4 Luminous.

4.1 Block Storage Interface

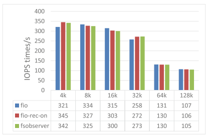
We use fio to test the performance of block storage interface. In our environments, we first test the performance use fio, then we lunch FSObserver and test the performance with fio again. We get two performance data from fio, and one performance data from FSObserver.

There are 6 different workloads used in our test, including sequence read, write, and mixed workloads and random read, write, and mixed workloads. For each workload, we test performance with different block sizes from 4k to 128k. Due to the page limitations, we only show the results of read and write. From these figures, the results of fio are almost the same, while one is taken without a recorder, the other was taken with FSObserver is working. The CPU and memory used by FSObserver are also very small. According to our experiments, it only used less than 0.3% CPU during the whole test (Figs. 2, 3 and 4).



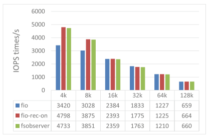
	4k	8k	16k	32k	64k	128k
🟦 fio	9888	9372	6738	3298	1727	846
🟥 fio-rec-on	9381	9306	6921	3504	1640	839
🟩 fsobserver	9376	9196	6884	3461	1641	839

(a) Sequence read



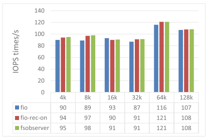
	4k	8k	16k	32k	64k	128k
🟦 fio	321	334	315	258	131	107
🟥 fio-rec-on	345	327	303	272	130	106
🟩 fsobserver	342	325	300	273	130	105

(b) Sequence write



	4k	8k	16k	32k	64k	128k
🟦 fio	3420	3028	2384	1833	1227	659
🟥 fio-rec-on	4798	3875	2393	1775	1225	664
🟩 fsobserver	4733	3851	2359	1763	1210	660

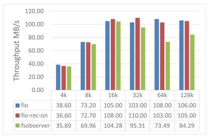
(c) Random read



	4k	8k	16k	32k	64k	128k
🟦 fio	90	89	93	87	116	107
🟥 fio-rec-on	94	97	90	91	121	108
🟩 fsobserver	95	98	91	91	121	108

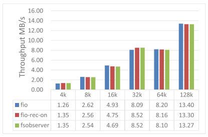
(d) Random write

Fig. 2. The IOPS measured by fio and FSObserver



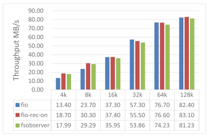
	4k	8k	16k	32k	64k	128k
🟦 fio	38.60	73.20	105.00	103.00	108.00	106.00
🟥 fio-rec-on	36.60	72.70	108.00	110.00	103.00	105.00
🟩 fsobserver	35.89	69.96	104.28	95.31	73.49	84.29

(a) Sequence read



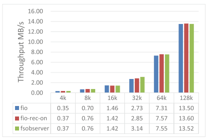
	4k	8k	16k	32k	64k	128k
🟦 fio	1.26	2.62	4.93	8.09	8.20	13.40
🟥 fio-rec-on	1.35	2.56	4.75	8.52	8.16	13.30
🟩 fsobserver	1.35	2.54	4.69	8.52	8.10	13.27

(b) Sequence write



	4k	8k	16k	32k	64k	128k
🟦 fio	13.40	23.70	37.30	57.30	76.70	82.40
🟥 fio-rec-on	18.70	30.30	37.40	55.50	76.60	83.10
🟩 fsobserver	17.99	29.29	35.95	53.86	74.23	81.23

(c) Random read



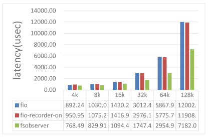
	4k	8k	16k	32k	64k	128k
🟦 fio	0.35	0.70	1.46	2.73	7.31	13.50
🟥 fio-rec-on	0.37	0.76	1.42	2.85	7.57	13.60
🟩 fsobserver	0.37	0.76	1.42	3.14	7.55	13.52

(d) Random write

Fig. 3. The throughput measured by fio and FSObserver

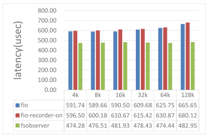
4.2 Capture Real Workloads

In this part, we demonstrate how to get the I/O sequence of a real application. The process of compiling a Linux kernel is a complex task. There are 67 thousand



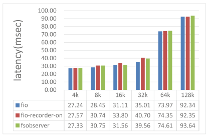
	4k	8k	16k	32k	64k	128k
🟦 fio	892.24	1030.0	1430.2	3012.4	5867.9	12002.
🟥 fio-recorder-on	950.95	1075.2	1416.9	2976.1	5775.7	11908.
🟩 fsobserver	768.49	829.91	1094.4	1747.4	2954.9	7182.0

(a) Sequence read



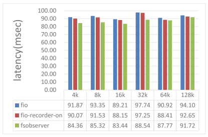
	4k	8k	16k	32k	64k	128k
🟦 fio	591.74	589.66	590.50	609.68	625.75	665.65
🟥 fio-recorder-on	596.50	600.18	610.67	615.42	630.87	680.12
🟩 fsobserver	474.28	476.51	481.93	478.43	474.44	482.95

(b) Random read



	4k	8k	16k	32k	64k	128k
🟦 fio	27.24	28.45	31.11	35.01	73.97	92.34
🟥 fio-recorder-on	27.57	30.74	33.80	40.70	74.35	92.35
🟩 fsobserver	27.33	30.75	31.56	39.56	74.61	93.64

(c) Sequence write



	4k	8k	16k	32k	64k	128k
🟦 fio	91.87	93.35	89.21	97.74	90.92	94.10
🟥 fio-recorder-on	90.07	91.53	88.15	97.25	88.41	92.65
🟩 fsobserver	84.36	85.32	83.44	88.54	87.77	91.72

(d) Random write

Fig. 4. The latency measured by fio and FSObserver

files in the Linux kernel 4.16.4. During the compiling process, several compilers read thousands of files and generate about 71 thousand new files. Figure 5 shows the IO throughput per minutes during the compiling process. We can find that during the compiling process, the write throughput is kept at a high level.

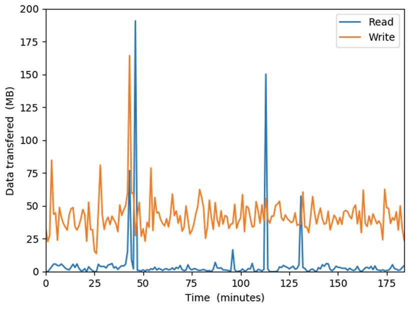


Fig. 5. Real workloads of compiling a Linux kernel

5 Conclusion

In this paper, we present a flexible performance monitoring tool for large-scale distributed storage systems. We have implemented it for Ceph. The experiments show that it can get coincident performance data with other widely used tools. We compared the accuracy with wide adapted benchmarks and measure a performance for a real application.

Acknowledgment. This work is supported by the NFS of China under Grant No. 61472323 and No. 61502392 and the Ministry of Science and Technology of China, National Key Research and Development Program (No. 2018YFB1004401).

References

1. Yuan, D., Zheng, J., Park, S., Zhou, Y., Savage, S.: Improving software diagnosability via log enhancement. ACM Trans. Comput. Syst. (TOCS) **30**(1), 4 (2012)
2. Zhao, Y., Cao, Y., Chen, Y., Zhang, M., Goyal, A.: Rake: semantics assisted network-based tracing framework. IEEE Trans. Netw. Serv. Manage. **10**(1), 3–14 (2013)
3. Määttä, M., Räty, T.: Automatic model creation to support network monitoring. IEEE Access **2**, 142–152 (2014)
4. Gudu, D., Hardt, M., Streit, A.: Evaluating the performance and scalability of the Ceph distributed storage system. In: 2014 IEEE International Conference on Big Data (Big Data), pp. 177–182. IEEE (2014)
5. Wang, F., Nelson, M., Oral, S., Atchley, S.: Performance and scalability evaluation of the Ceph parallel file system. In: Proceedings of the 8th Parallel Data Storage Workshop, pp. 14–19. ACM (2013)
6. Oliner, A., Ganapathi, A., Xu, W.: Advances and challenges in log analysis. Commun. ACM **55**(2), 55–61 (2012)