IOSIG+: on the Role of I/O Tracing and Analysis for Hadoop Systems

Bo Feng, Xi Yang, Kun Feng, Yanlong Yin and Xian-He Sun
Department of Computer Science
Illinois Institute of Technology, Chicago, IL, USA
Email: {bfeng5, xyang34, kfeng1, yyin2}@hawk.iit.edu, sun@iit.edu

***Abstract*—Hadoop, as one of the most widely accepted MapReduce frameworks, is naturally data-intensive. Its several dependent projects, such as Mahout and Hive, inherent this characteristic. Meanwhile I/O optimization becomes a daunting work, since applications' source code is not always available. I/O traces for Hadoop and its dependents are increasingly important, because it can faithfully reveal intrinsic I/O behaviors without knowing the source code. This method can not only help to diagnose system bottlenecks but also further optimize performance. To achieve this goal, we propose a transparent tracing and analysis tool suite, namely IOSIG+, which can be plugged into Hadoop system. We make several contributions: 1) we describe our approach of tracing; 2) we release the tracer, which can trace I/O operations without modifying targets' source code; 3) this work adopts several techniques to mitigate the introduced execution overhead at runtime; 4) we create an analyzer, which helps to discover new approaches to address I/O problems according to access patterns. The experimental results and analysis confirm its effectiveness and the observed overhead can be as low as 1.97%.**

Keywords*—*MapReduce; Hadoop; I/O; Tracing; I/O analysis

I. Introduction

MapReduce paradigm and its open source implementation Hadoop [1], [2], are widely applied for data-intensive jobs due to its capability of scaling out and fault tolerance. With the help of Hadoop, the data processing ecosystem is advancing continuously, such as HBase [3], Pig Latin [4], Hive [5], and Mahout [6]. Recently, YARN, the next generation of Hadoop, enables HDFS to support various workloads and evolves into a larger scale [7].

However, due to performance gap between computing and storage devices, Hadoop ecosystem is generally becoming more data-intensive. Thus, the overall performance of Hadoop applications is greatly determined or affected by their I/O performance. In order to improve their performance, tracing I/O behaviors is one of simple yet powerful ways. Tracing and analysis has been studied for many years in HPC environment as an optimization prerequisite, such as mpiP [8], IOSIG [9] and Darshan [10]. These tools or interfaces can trace for MPI-based applications, but neither are they suitable for Java based applications nor Hadoop frameworks.

Even though Hadoop's system log is another convenient source of logging system behaviors, it is often problematic to analyze Hadoop logs. Because loggers in Hadoop are added by random independent developers, they are incomplete, less reliable, and even misleading especially for I/O behavior analysis [11], [12].

Our solution is to inject byte codes into Hadoop systems and get related I/O traces. There are further reasons to trace and analyze Hadoop systems: First, many existing works chose to use data mining, black-box, and gray-box methods [12], [13], [14] to make guidelines to improve I/O performance but missed potential fine-grain optimization opportunities. Second, there are many configuration parameters in Hadoop systems, some of which can make a huge impact on the performance but neither users nor system can be aware of the optimal ones in advance. Third, tuning parameters for I/O throughput is not enough because MapReduce employs lots of threads that are in charge of functions of a task and working corporately. Their behaviors' impact to the performance is also important but remains unknown. Finally, since Hadoop widely adopts buffering and asynchronous I/O, I/O traces reveal the data accesses across multiple I/O stacks in order to reflect the causality between data accesses and I/O performance.

This paper introduces our solution for Hadoop systems. Hadoop applications can be naturally decoupled into two layers: Application(MapReduce) and HDFS. However, acquiring the I/O behaviors for two layers without API interference is not an easy task. To address this problem, our technique contribution in this paper, is to trace I/O calls in Hadoop systems without API interference, avoiding high performance overhead. Our tool suite can record Hadoop applications and underlying HDFS I/O requests without changing any code for targeted applications and Hadoop itself. To reveal I/O behaviors and further optimize Hadoop systems, we also build comprehensive tools for trace analysis, and pattern visualization, especially for Hadoop systems. The contributions are:

- We build a Java-based I/O signature agent to automatically collect Hadoop traces without introducing interferences to applications. Trace files are stored in compression at runtime.
- We run several Hadoop benchmarks to evaluate its performance and overhead.
- We analyze the trace information and show the potential usage of how to optimize Hadoop I/O stack.
- The built-in analyzer in IOSIG+ can visualize I/O access patterns. We preview some patterns in this paper.

The rest of this paper is organized as follows: Section II illustrates the design and implementation of IOSIG+. In Section III, we show our experimental results by evaluating overhead of IOSIG+. In this section, we also conduct trace processing and analysis, showing many I/O patterns and optimization opportunities. Section IV reviews the related work of Hadoop tracing and performance analysis, and discusses the differences from this work. Finally, we conclude the paper in Section V.

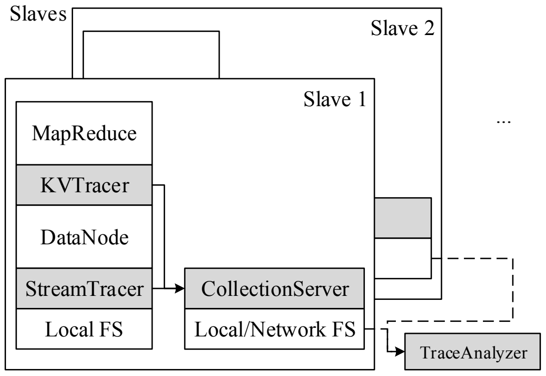

Fig. 1. Overview of IOSIG+: KVTracer and StreamTracer are injected into MapReduce and DataNode respectively on all slave nodes and traces from them will be sent to CollectionServer. Finally all trace data are fed to TraceAnalyzer to do analysis.

II. Design and Implementation

A. Design Overview

IOSIG+ aims at introducing tracing capability with low overhead in the first place. Our design addresses three challenges successfully. First, as a tracing tool, IOSIG+ should yield a low I/O overhead in target applications. We use ramdisk as temporary trace storage and compress traces on ramdisk in a non-blocking way. Second, IOSIG+ traces both HDFS and MapReduce layers. Therefore two-layer tracing is used in Hadoop systems: one layer for MapReduce framework, which reads/writes intermediate key-value pairs; the other layer for HDFS, which actually executes I/O operations. Third, IOSIG+ is implemented according to Java agent specification supported by JVM, which makes it transparent to all Hadoop applications.

Figure 1 shows the overall architecture of IOSIG+, which consists of four major components: 1) KVTracer, 2) StreamTracer, 3) CollectionServer and 4) TraceAnalyzer. KVTracer is responsible for tracing intermediate key-value pairs in map and reduce tasks. KVTracer monitors some Hadoop classes such as RecordReader, RecordWriter and their inheritances. StreamTracer resides inside of each DataNode. It can capture general Java I/O function calls in some common classes, such as FileInputStream, FileOutputStream and etc. CollectionServer is a non-blocking trace storage server, which saves all I/O signatures in ramdisk, compresses trace files and transfers compact trace files into local or network file systems. TraceAnalyzer is an offline tool to extract and analyze I/O behaviors based on trace files.

B. Trace Information Levels

Fine-grained tracing is time-consuming, but coarse-grained tracing may lead to incomplete information or misleading conclusion. IOSIG+ balances the overhead with tracing granularity by grading information into levels. Since achieving both fine-granularity and low-overhead is paradoxical, IOSIG+ provides runtime tunable parameters indicating the expected tracing granularity so that the users can specify on which level it traces, thus the tracing overhead can be under control. Similar to Herodotou's Profiling [11], IOSIG+ also traces the information of job-level and task-level. Moreover, we have more fine-grained options in order to not just show the statistics of jobs and tasks but also I/O accesses to data blocks. The information levels which IOSIG+ can trace are shown in Table I.

TABLE I. The Matrix of Tracing Information on Levels

Trace	Logical	Thread	Phase	Task	Job
record	x				
properties	x	x			
xml	x	x			
classes	x	x			
jars	x	x			
thread	x	x	x		
Hadoop log	x	x	x	x	x
pipe	x	x	x	x	x
blk	x	x	x	x	x
blk.meta	x	x	x	x	x

C. Implementations

IOSIG+ is built in compliance with Java agent specification. Java Agent Instrument mechanism allows Java application to work with agents to run agent codes before running programs on the JVM. We use an agent to generate and replace the bytecode of some classes in memory. IOSIG+ can be deployed as a JAR file, in which an agent class is specified and the permissions to instrument classes are granted. Before the target program getting started, IOSIG+ works as an agent to instrument some bytecodes into target applications. Since all the writing functions are traced, it incurs recursive calls if the logging functions are implemented in Java. To avoid this problem, we also use Java Native Interface (JNI) to implement the trace file saving mechanism in C language.

III. Experimental Results

A. Experimental Setup

Craysun is a cluster in our lab, consisting of 17 nodes, of which one node is master and the other 16 nodes are salves. The master node is a Dell PowerEdge 2850 Server, which has a dual-core Intel Xeon CPU 2.80GHz with 6GB of memory. All slave nodes are Dell PowerEdge SC 1425 Server, each of which has an Intel Xeon CPU 3.40GHz processor, 1GB of memory, and a 36GB (15,000 rpm) SCSI hard drive. All nodes are connected within a local area network switch, running Ubuntu 14.04 with latest available Linux kernel.

B. Tracing Overhead Analysis

The overhead of IOSIG+ is mainly brought by capturing I/O operations and writing trace files. To better evaluate the pure overhead of IOSIG+, we compared the performance of TestDFSIO with and without tracing. TestDFSIO benchmark is delicately writing 1GB-file on each node in parallel. Figure 2 shows the performance comparison between the stock system and that with activated IOSIG+ in throughput and average I/O rate. In these experiments, the throughputs of write and read for the stock system are 28.76MB/sec and 74.66MB/sec respectively; while the average I/O rate are 31.21MB/sec and 75.94MB/sec. By contrast, when IOSIG+ is active, the throughputs of write and read are 25.99MB/sec and

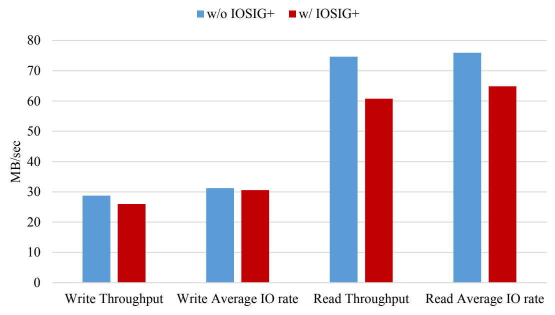

Fig. 2. Throughput comparison between with and without IOSIG+ for TestDFSIO

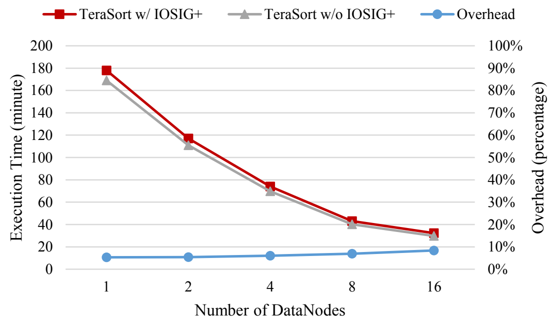

Fig. 3. IOSIG+ overhead evaluation in TeraSort

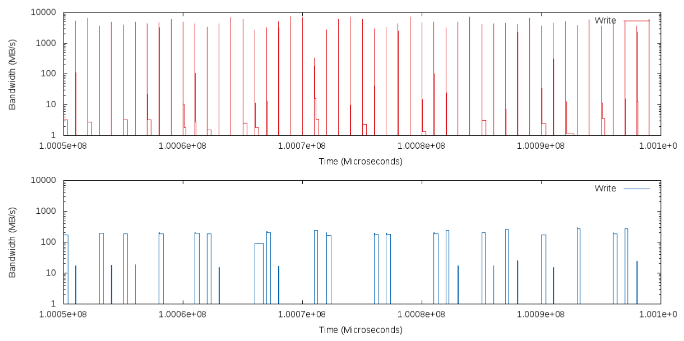

Fig. 4. Two-layer bandwidth comparison from RandomTextWriter in a time window. The above subfigure shows the requests in MapReduce layer and the bottom one shows the requests from DataNode.

60.75MB/sec; while the average I/O rate are 30.60MB/sec and 64.87MB/sec. The least performance degradation with IOSIG+ is in write tests, while the most performance degradation with IOSIG+ is in read tests. This is because the throughput of read is more than double of that of write. Since write operations are more expensive and saving tracing files are all write accesses, the time spent for writing is longer than that for reading. Therefore, the overhead for write tests is not as obvious as that for read tests. In these experiments, the highest overhead in throughput and average I/O rate are 18.62% and 14.58%; while the lowest are 9.63% and 1.97%.

C. Scalability

This experiment is to show the overhead of IOSIG+ when the Hadoop system scales out as well as its capability of capturing traces from every DataNode and stores traces through CollectionServers. In these experiments, 4GB data generated from TeraGen are sorted by TeraSort. We vary the number of slaves in a Hadoop cluster from 1 to 16. In this set of experiments, as the number of DataNodes increases, we can observe that the overhead of IOSIG+ is all less than 10% (Figure 3). Because I/O capturing in tracers and storing traces in collection servers for HDFS are distributed on each DataNode, each node has almost the same overhead so that the overall overhead is constant.

D. Request Patterns

During executions of Hadoop applications, periodic I/O waves can be easily observed. Figure 4 presents the comparison of two-layer trace diagrams in a short period in RandomTextWriter. RandomTextWriter is another benchmark in Hadoop, which generates random unsorted sequence of words. We trace the I/O calls from two layers for RandomTextWriter and plot the requests in Figure 4. The above subfigure shows the requests in MapReduce layer and the bottom one shows the requests from DataNode. We observe that the number of requests from upper layer (MapReduce) is larger than that from lower layer (mainly from DataNode). This reveals that the request pattern varies between layers.

IV. Related Work

A. I/O Tracing for System Diagnosis and Debugging

mpiP [8], IOSIG [9] and Darshan [10] are tracing tools for MPI-based applications. However, they cannot be used for Java based programs. For example, our previous work IOSIG [9] supports the tracing for MPI-IO. In addition, plethora of existing works are designed for the I/O stack on a single node, and cannot trace applications on a large-scale cluster [15], [16].

Hadoop provides its internal logging support. The earlier work SALSA and Mochi [17], [18] intelligently process the raw system logs but these works were based on the assumption that similarity of MapReduce tasks and homogeneous hardware. However, for other non-MapReduce applications, it is not suitable for non-fixed I/O patterns on HDFS. For example, it is inevitable for HBase to send random requests to HDFS in some cases. Pan et al. proposed Ganesha [13] to transparently diagnose MapReduce performance, by collecting OS-level metrics and using data mining clustering method to identify outliers. The web-page based Hadoop monitoring tool [19] from Cloudera company provides detailed descriptions on resource utilization and task duration with visualization, especially on CPU utilization rate through the execution duration. McDougall et al. categorized I/O on Hadoop system and took TeraSort benchmark as an example to quantitatively measure these I/O patterns, especially the behaviors from the client-side and the DataNode I/O stacks [20]. While IOSIG+ neither assume homogeneous hardware nor rely on high-level statistics, it can provide detailed view of I/O operations. Moreover, IOSIG+ supports various course-grained options, like phase-level and thread-level granularity matrices, especially concentrates on I/O sensitive data in order to bring lower overhead than fine-grained ones.

B. Tracing for Hadoop System Characterization and Performance Analysis

Hadoop parameter tuning is another popular topic in this field, since it has many built-in parameters which impact performance. Application traces are essential input for performance tuning. For example, Herodotou et al. proposed a tuning system [11] to find optimal configurable parameters for MapReduce applications in Hadoop. The input for a cost-based optimizer traces detailed statistics of both job or task level including resource consumptions, and timing information for thread worker or functions in task, such as spill, merge, etc. Kwon et al. proposed SkewTune [21] to tune MapReduce application skew in runtime. An assigned split is no longer an atomic unit for task scheduling. An straggler task can be partitioned into multiple subtasks. The subtasks are assigned to idle workers and completed in parallel according to both collected statics and the status of task processing. Li et al. proposed mrOnline [14], an online tuning system based on YARN that exploits YARN's enhanced resource allocation. The fundamental of optimal parameters search is the runtime statistics, including task and node resource utilization information that collected by system periodically. Across I/O stacks tracing and analysis is a promising methodology for performance optimization. Harter et al. analyzed the tracing logs of HBase and HDFS respectively in Facebook's shadow cluster [22], and observed the data access patterns among different I/O stacks have huge differences that read intensive applications issue read requests to memory but write requests to I/O devices.

V. Conclusions and Future Work

We present our solution to reveal I/O behaviors of Hadoop applications from building a tracing and analysis tool suite, namely IOSIG+, which helps to capture, compress, save and visualize I/O traces under Hadoop systems. Compared with the related work, its advantages are three-fold: 1) it uses white-box tracing, which accurately record application behaviors; 2) it presents the pattern comparison between different component layers in the whole I/O stack of Hadoop; and 3) to our best knowledge, it has the most comprehensive information without using a shadow cluster. In addition, we provide all these features with low overhead. The experiments and analysis show that IOSIG+ is capable of capturing comprehensive details of I/O behaviors of Hadoop applications, with a low execution overhead at runtime. From our analysis, as non-API-interference tool, IOSIG+ is capable of providing I/O optimization opportunities, even for applications that users are not familiar with or applications whose source codes are not available. In future, we would like to provide more comprehensive views of traces and hints for I/O optimization.

Acknowledgment

The authors would like to thank all the anonymous reviewers from Cluster 2015. This work is supported in part by a research grant from Huawei Technologies Co, Ltd., the US National Science Foundation under Grant No. CNS-0751200, CCF-0937877, CNS-1162540, and CNS-1338078.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[3] A. Foundation, "Apache HBase," 2008.

[4] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-so-Foreign Language for Data Processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1099–1110.

[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive:a Petabyte Scale Data Warehouse Using Hadoop," in *IEEE 26th International Conference on Data Engineering (ICDE)*, 2010, pp. 996–1005.

[6] A. Mahout, "Scalable Machine-Learning and Data-Mining Library," 2008.

[7] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16.

[8] J. Vetter and C. Chambreau, "mpiP: Lightweight, Scalable Mpi Profiling," *URL: http://www. llnl. gov/CASC/mpiP*, 2005.

[9] Y. Yin, S. Byna, H. Song, X.-H. Sun, and R. Thakur, "Boosting Application-Specific Parallel I/O Optimization Using IOSIG," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, May 2012, pp. 196–203.

[10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, Apr. 2010.

[11] H. Herodotou and S. Babu, "Profiling, What-If Analysis, and Cost-Based Optimization of MapReduce Programs," *Proc. of the VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

[12] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting Large-scale System Problems by Mining Console Logs," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 117–132.

[13] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Ganesha: BlackBox Diagnosis of MapReduce Systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 8–13, Jan. 2010.

[14] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller, "MRONLINE: MapReduce Online Performance Tuning," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: ACM, 2014, pp. 165–176.

[15] P. Kranenburg, B. Lankester, and R. Sladkey, "strace," 1992.

[16] B. Gregg and J. Mauro, "DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X, and FreeBSD," 2011.

[17] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "SALSA: Analyzing Logs as StAte Machines." *WASL*, vol. 8, pp. 6–6, 2008.

[18] ——, "Mochi: Visual Log-analysis Based Tools for Debugging Hadoop," in *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), San Diego, CA*, vol. 6, 2009.

[19] J. Zuanich, "A Profile of Apache Hadoop Mapreduce Computing Efficiency," Jul. 2010.

[20] R. McDougall, "Analyzing Hadoops internals with Analytics," 2012.

[21] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: mitigating skew in mapreduce applications," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 25–36.

[22] T. Harter, D. Borthakur, S. Dong, A. Aiyer, L. Tang, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis of HDFS Under HBase: A Facebook Messages Case Study," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies*. Berkeley, CA: USENIX, 2014, pp. 199–212.