Towards An Open Data Set for Trace-Oriented Monitoring

Jingwen Zhou*†, Zhenbang Chen*†, Ji Wang*†, Zibin Zheng‡§, and Michael R. Lyu†‡§
*Science and Technology on Parallel and Distributed Processing Laboratory
National University of Defense Technology, Changsha, China
†College of Computer, National University of Defense Technology, Changsha, China
‡Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
§Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China
Email: {jwzhou, zbchen}@nudt.edu.cn

***Abstract*—Trace-oriented monitoring is one of the main methods for monitoring cloud systems. However, there is no free trace data set available, which hinders the development of trace-oriented monitoring. Therefore, we want to collect a trace data set in a real environment and make it free. During collection, many aspects are considered, including cluster sizes, user requests, workload speeds, injected faults, *etc*., to simulate different situations. The structure of this data set is well-designed. We believe that this data set will be helpful for the research of trace-oriented monitoring.**

***Keywords*-tracing; data set; workload generation; fault injection; cloud computing**

I. Introduction

As the increasing of cluster scale and system complexity, problems happen more and more often in cloud systems and usually cause enormous loss. For example, on 16 August 2013, many of Google websites suffered a meltdown, in which Google lost $550,000$ dollars in less than 5 minutes, and the global internet traffic dropped 40% [1]. Therefore, how to improve the reliability of cloud systems is a very important problem.

Trace-oriented monitoring, or called tracing, is one of the methods to improve system reliability at runtime. With the data recorded by tracing systems, namely trace, many activities can be carried out, such as fault detection, fault diagnosis, and even remediation. However, to our best knowledge, there is no freely available trace set existing in academia and industry. As a result, before starting a certain trace-based research, plenty of work need to be done first, including choosing or even implementing a tracing system, instrumenting a target system, deploying the instrumented system, collecting different kinds of traces, and so on, which is a tedious and time-consuming process. Actually, the lack of trace data hinders the development of trace-based research, which also motivates our work.

We are collecting a trace data set for supporting trace-based research. Traces in this data set are collected from a Hadoop Distributed File System (HDFS) [4] deployed in a real environment, and record the processes of handling user requests in HDFS. During collection, we consider different cluster sizes, user requests, workload speeds, injected faults, *etc*., to simulate real situations. In our plan, this data set consists of three classes, where the ***Normal*** class and the ***Abnormal*** class are collected during the HDFS running normally and abnormally, respectively, while the ***Combination*** class is collected by randomly injecting faults and later recovering the system. Since the structure is well-designed and the scenarios are close to reality, we believe this data set will be helpful for trace-based research. For example, in fault detection, the *Normal* class and *Abnormal* class can be used to learn the features of normal behaviors and abnormal behaviors, while the *Combination* class can be used to test the effectiveness of fault detection algorithms.

II. Structure of Data Set

Each element in our data set is a trace, which records the execution path of a user request. A trace consists of the events and the relationships, where each event records the context of a request step, such as function name and latency, and a relationship records the casual relation between two events, like local and remote function calls. According to the events and the relationships, a trace can be constructed to a trace tree, in which the nodes correspond to the events and the edges correspond to the relationships. More details can be referred to [2].

In our plan, this data set consists of three classes: *Normal* , *Abnormal* and *Combination*, and each class includes several types. When collecting the traces of each type, we consider different requests, workloads, faults, *etc*., to simulate various scenarios. Thus, each type contains many trace files, each of which contains the traces collected in a scenario.

The traces in the *Normal* class are collected when the monitored HDFS system is running normally. There will be two types in this class: *Workload* type and *Datanode* type. Different trace files in the *Workload* type are collected under different workload speeds, while the *Datanode* type mainly considers various cluster sizes decided by the number of the datanodes in the HDFS system. Besides, different user requests, including read, write and RPC (Remote Process Call), are also introduced in both types. Therefore, the *Normal* class records the behaviors of the HDFS when

Table I
INJECTED FAULTS

Type	*Fault*
Process	killDN, suspendDN
Network	disconnectDN, slowHDFS, slowDN
Data	corruptBlk, corruptMeta, lossBlk,lossMeta, cutBlk, cutMeta
System	panicDN, deadDN, readOnlyDN

handling different user requests under different speeds with different cluster sizes.

When collecting a trace file in the *Abnormal* class, a fault is injected into HDFS and holds during the whole collection. The *Abnormal* class consists of four types according to the fault types, shown in Table I. The *Process* faults affect the processes on HDFS nodes, while the *Data* faults introduce errors in the data on datanodes, and the faults in the *Network* type and the *System* type bring anarchies to the network in the cluster and the OSs of the HDFS nodes, respectively.

During collecting a trace file in the *Combination* class, faults are randomly chosen and randomly injected into HDFS, and the sysetm is recovered automatically after a while. According to whether faults are picked from a single fault type or all the four types, the *Combination* class can be divided into the *Single* type and the *All* type. In addition, together with a trace file in this class, a description file is also given, describing the details of injected faults, including the fault name, the related parameters, the injection and recovery time stamps, *etc*.

III. Details in Collection

We employ an effective tracing system, called MTracer [2], developed by us to collect the traces in the HDFS, deployed on a real environment composed by virtual machines.

Figure 1 illustrates the architecture of trace collection. The HDFS system, containing one namenode and many datanodes, handles user requests. Clients, simulating real users, generate different requests to HDFS. The MTracer server tracks the process of handling requests and collects the traces. The controller controls the collection process and injects faults. The Ganglia server [3] monitors all nodes and the whole process. By the way, in our collection, we have 50 datanodes and 50 clients in maximum.

When starting a trace collection, we first start the MTracer server and the HDFS system with a certain cluster size, and then launch *bash* scripts concurrently on some clients to generate workload. After finishing the collection, workload terminates first, and the HDFS system and the MTracer server stop after finishing all requests. When collecting a trace file in the *Abnormal* class, faults are injected before starting workload, and the system is recovered after finishing all requests, which makes sure the HDFS is running abnormally during collection. While in the *Combination* class, faults are randomly injected after starting workload, then the system

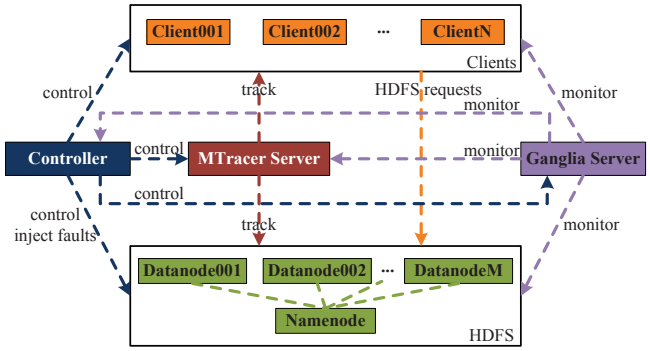

Figure 1. The architecture of trace collection.

is recovered after an interval, and the next fault is carried out in a same way, which simulates occasionally happened faults.

IV. Threads to Invalidity

We collect traces only on a HDFS system, because HDFS is a widely used system in academia and industry, and many mechanisms and procedures in dealing user requests in HDFS are shared by other systems. Thus, the traces from HDFS are representative.

During collection, the HDFS system maximumly contains 50 datanodes, which is smaller than production systems. However, the traces are collected in different scenarios, which is enough for exhibiting various features of HDFS.

Besides the faults we introduce, many others exist. Nevertheless, the faults we inject contain different fault types, which cover the most frequent and representative faults in real systems.

V. Conclusion

We plan to collect an open trace data set in a real environment, considering many aspects during collection. The data set has a well-designed structure and is deemed to be helpful for trace-oriented monitoring.

Acknowledgment

This work is supported by the National 973 Program of China under the Grant No.2011CB302603, the NSFC under the Grant No. 61161160565 and No. 61303064, and the SRFDP under the Grant No. 20114307120015.

References

[1] J. Garside, "Nasdaq crash triggers fear of data meltdown," http://www.theguardian.com/technology/2013/aug/23/nasdaq-crash-data, 2013.

[2] J. Zhou, Z. Chen, H. Mi, and J. Wang, "MTracer: a trace-oriented monitoring framework for medium-scale distributed systems," In *Proc. of SOSE 2014*, 2014, pp. 266–271.

[3] M. L. Massie, B. N. Chun., and D. E. Culler, "The Ganglia distributed monitoring system: design, implementation, and experience," *Elsevier PARCO*, vol. 30, no. 7, pp. 817–840, 2004.

[4] Apache, "Hadoop," http://hadoop.apache.org/, 2014.