A High Throughput Distributed Log Stream Processing System for Network Security Analysis

Jingfen Zhao, Peng Zhang, Yong Sun,
Qingyun Liu, Guolin Tan
Institute of Information Engineering, Chinese Academy of Science
National Engineering Laboratory for Information Security Technologies
Beijing China
E-mail: {zhaojingfen, tanguolin}@iie.ac.cn

Zhengmin Li
National Computer Network Emergency Response and Coordination Center
Institute of Information Engineering, Chinese Academy of Science
Beijing, China
E-mail: lizhengmin@cert.org.cn

***Abstract*—Computer-system logs often contain high volumes of interesting, useful information, and are an important data source for network security analysis. In this paper, we propose a distributed log stream processing system consisting of three main parts: log collection module, log transmission module and log statistics module. The system uses several open source technologies, not only supports multi-source heterogeneous log collection, but also provides near-real-time online statistics for log stream and offline statistics for massive log. In addition, we adopt a layered architecture in the log collection module, and accomplish a reliable Kafka consumer to get higher scalability as well as reliability. Using log entries generated by the network security platform as data source to do experiment, demonstrates that the proposed system is an effective and practical log stream processing system.**

Keywords-Log Stream; Security Analysis; Big Data; Scalability

I. INTRODUCTION

Log comes from Internet surfing, application system such as cloud computing services, network monitor system, sensor networks, etc. and network devices, with the characteristics of huge volume, heterogeneity and low value density. There are two forms of log: log records from the log file and log stream. Log stream is continuously, time-varying sequence of tuples generated by application system that records events occurring during its use.

Logs are an important data source for academic research and business analysis. They are always used for security applications, such as detecting breaches or misbehavior, and for performing postmortem inspection of security incidents [1]. Depending on the system and the threat model, logs of nearly any kind might be amenable to security analysis [19]: logs related to firewalls, login sessions, resource utilization, system calls, network flows, etc.

Traditionally these logs are collected and stored in some sort of a data store just as relational databases and then are processed to find any interesting and useful information. Unfortunately, the method mentioned above isn't always adequate to support long-term, large-scale analytics for several reasons [2]: firstly, when the volume and velocity of the data grow at unprecedented rates, traditional relational databases either do not support such volumes or face performance issues (see [3] for a comparison of the database size limits). Secondly, performing analytics and complex queries on large, unstructured datasets with incomplete and noisy features was inefficient. Finally, the management of large data warehouses has traditionally been expensive, and their deployment usually requires strong business cases. As a result, when traditional data storage and analysis approaches fail to meet the expectations of log analysis in the era of big data, it becomes necessary to adapt new technologies, namely, big data technologies, to be able to cope with these problems.

This paper outlines the architecture and implementation of a novel, distributed, and scalable log data collection, storage and analysis system, based on modern big data technologies. The distributed log stream processing system we present below uses open source technologies: Flume, Kafka, Storm, HDFS, Hive, supports multi-source log collection, near-real-time online statistics and offline statistics for massive log.

The remainder of the paper is structured as follows: in the next section, we make a brief introduction of the open source technologies involved in this paper. The third section presents the distributed log stream processing system by explaining its architecture and components. Then we give a use case for it. The last section contains a conclusion.

II. BACKGROUND, RELATED CONCEPTS,AND TECHNOLOGIES

A. Flume

Flume [4] is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms.

A Flume agent is a JVM process which has three components: source, channel and sink. A Flume source

consumes events delivered to it by an external source like a web server. The external source sends events to Flume in a format that is recognized by the target Flume source. When a Flume source receives an event, it stores it into one or more channels. The channel is a passive store that keeps the event until it's consumed by a Flume sink. The two most common types of channel are file channel and memory channel. The sink removes the event from the channel and puts it into an external repository like HDFS or forwards it to the Flume source of the next Flume agent in the flow.

B. *Kafka*

Kafka [5] is a distributed, scalable streaming platform, used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.

In Kafka, a stream of messages of a particular type is defined by a topic. A producer can publish messages to a topic. The published messages are then stored at a set of servers called brokers. A consumer can subscribe to one or more topics from the brokers, and consume the subscribed messages by pulling data from the brokers. Since Kafka is distributed in nature, a Kafka cluster typically consists of multiple brokers. To balance load, a topic is divided into multiple partitions and each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time.

C. *Storm*

Apache Storm [6] is a free and open source distributed real-time computation system. It makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing.

The logic for a real-time application is packaged into a Storm topology. A Storm topology is represented by directed acyclic graphs (DAG), where vertices, called processing elements (Spout or Bolt), represent operators, and edges, called streams, represent the data flow from one PE to the next. For scalability, streams are partitioned into sub-streams and processed in parallel on a replica of the PE called processing element instance.

D. *Hadoop*

The Apache Hadoop [7] software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is built on lots of cheap hardware devices, but has the feature of high reliability and good extensibility. The key technologies of Hadoop are Hadoop Distributed File System (HDFS) which is the open source version of Google File System [8], MapReduce program model and HBase distributed database.

HDFS is to store huge amounts of information, scale up incrementally and survive the failure of significant parts of the storage infrastructure without losing data. Hadoop is ideal for storing large amounts of data, like terabytes and petabytes, using HDFS as its storage system.

Apache Hive [9] is an open-source data warehouse system for querying and analyzing large datasets stored in HDFS files. HiveQL is the Hive query language, similar to the other SQL dialects in widespread use.

III. SYSTEM ARCHITECTURE

In this paper, we have developed a distributed log stream processing system using the aforementioned technologies. The system uses open source software and supports multi-source log collection, near-real-time online statistics and offline statistics for massive log.

The overview of the proposed system is illustrated in Figure 1. The system architecture consists of five main parts: log data collection module, log transmission module, log statistics module, log storage module and display module.

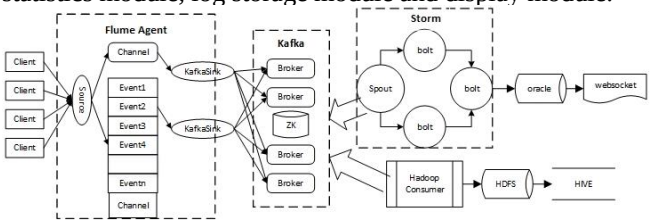

Figure 1. System architecture.

The system has the following characteristics:

- Availability: by adopting distributed architecture, partial nodes downtime do not affect the availability of the whole system.
- Reliability: it uses message queuing Kafka to ensure reliable transmission of messages. Beyond that, we design a reliable consumer which makes sure that the message is transmitted exactly once without lost nor re-transmission.
- Scalability: using Kafka as the transmission module makes the coupling between log collection module and statistics module lower and easy to scale out; it is convenient to add and remove log collection nodes to accommodate changes of front-end servers that generate logs; the log transmission module as well as statistics module can scale out easily by increasing the number of server nodes to improve the processing capability of the entire system.
- High performance: the system lets you produce, consume logs and make statistics with high throughput, low latency.

A. *Log Collection Module*

Nowadays, systems become increasingly composed of many, often distributed, components, using a single log file to monitor events from different parts of the system is difficult. In some scenarios, logs from entirely different systems must be cross-correlated for analysis. On the other hand, the size of logs has become bigger and bigger that the issues of collecting and storing them instantly is a big challenge. As a result, any of the existing centralized architectures is not competent. In the proposed system, we use Flume which is an open source distributed service for efficiently collecting large amounts of log data as the collection module.

In the project, we need to deploy the Flume Agent on each front-end server that generates log data, responsible for collecting log in real time, and sending it to the Kafka cluster as a producer.

If not adopting layered architecture, there exists the following problems: supposing that the number of front-end node is large, they are distributed in different network segments, belong to different business and should send log data to different topic in Kafka cluster, all the Flume agent deployed in the same layer will make it difficult for group management; when the Kafka cluster is shut down for restart or upgrade, it is necessary to inform every business side to do a good job in response, resulting in poor scalability. Fortunately, as shown below, we adopt a layered architecture.

The first layer is the log acquisition layer. Flume Agent is deployed on each front-end server, responsible for collecting log data in real time, and sending it to the second layer. For the Flume agent, which source is used depends on the specific business, the channel type is memory channel and the sink type is Avro sink in order to connect to the Avro source on the second tier.

The second layer is the log convergence layer. Flume agent is deployed on each aggregation node whose number is less than the front-end server and receives the log data from the acquisition layer then sends it to Kafka. The source type is Avro source. In order to avoid data loss caused by Kafka broker failure, it is necessary to cache log data and improve the stability as much as possible. Clearly, file channel is the best choice. The sink type is Kafka sink.

In addition, the L1 layer can be configured with sink group, using the failover mechanism to get higher reliability and stability.

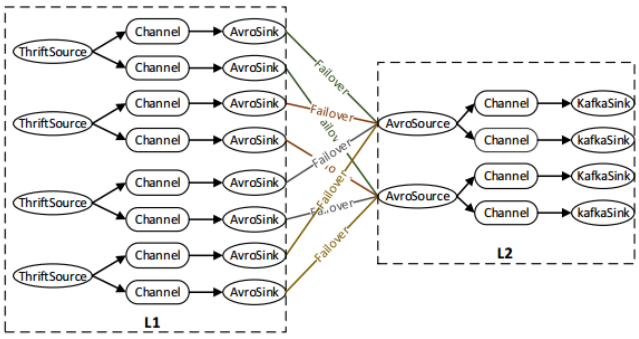

Figure 2. Flume's layered architecture diagram.

B. *Log Transmission Module*

The log transmission module is located between the collection module and the statistics module, lowering the coupling. If a module fails, the other module can provide services normally to ensure the high availability of the system. Besides, it is responsible for caching the log data receiving from Flume agent, thus supporting processing data asynchronously as well as reducing the peak value of the traffic. Here, we use Kafka as the transmission module. Flume, as a producer, pushes log into the appropriate topic, and Hadoop as well as Storm as consumers pull data from Kafka to perform offline analysis or online real-time statistics.

A message queuing system should be highly available and support high concurrency, and what's more, it has to ensure the reliable transmission of data. However, the consumer API provided by Kafka stores the log data separately from offset, so cannot guarantee that a message is delivered once and only once, which is what people actually want. Therefore, this paper presents a reliable consumer design, the HadoopConsumer function module in the system. The solution is based on Kafka's low-level API, and makes the log data consuming from Kafka brokers as well as offset stored in HDFS.

There are three possible message delivery guarantees that could be provided between producer and consumer[5]: at most once--messages may be lost but are never redelivered, at least once--messages are never lost but may be redelivered, exactly once--this is what people actually want, each message is delivered once and only once, which is not guaranteed by Kafka at present. In Kafka, offset is controlled by the consumer itself. A consumer reads some messages, then processes them, finally updates the offset value storing in ZooKeeper or specific topic. It has several options for processing the messages and updating the offset, and of course, different option leads to different reliability. You can view the official website [10] to learn more.

The HadoopConsumer module is designed and implemented based on Kafka low-level API as well as HDFS API. It reads the configuration file to obtain the topic list to be consumed and IP address of Kafka brokers, then creates a thread for each partition as a consumer to read messages, finally appends messages to specific file in HDFS, while writing the most up-to-date offset value to file. The module consists of two Java JAR files: DataGenerator.jar is invoked only for the first time when running HadoopConsumer to complete a series of initialization work; KafkatoHDFS.jar is responsible for consuming and appending messages to HDFS, finally rewriting the offset.dat file.

DataGenerator creates the following files for each partition of each topic:

- A file named offset.dat: it contains the following items: topic, partition id, IP address and port of the leader broker, as well as the offset. The initial value of offset is -1, indicating that the consumer reads messages from the very beginning.
- A file named message.txt: it stores messages reading from Kafka. Because HDFS does not support appending messages one by one, we write the data to a local file, then append the entire file to message.txt.

The following figure shows the output of HadoopConsumer module: it describes that the consumer started consuming from the 2783th message of partition 0 of the topic named bigdata; the IP address of leader broker was 192.168.11.179; the offset range was 2783-2783, which means that no message was read and consumption was faster than production; then HadoopConsumer wrote messages to a local file and appended the entire file to HDFS, finally set offset to the most up-to-date value for continuing consumption next time.

```
<terminated> KafkatoHDFSMain (2) [Java Application] /opt/jdk/jdk1.8.0_60/bin/java (2015-11-18 上午1
KafkatoHDFSMain be called!
threadCount1
Setting hadoop jar file for class:class kafka.etl.impl.SimpleKafkaETLJob  to null
*********************************************************************
          Running on Real Hadoop Cluster(local)
*********************************************************************
Init request from tcp://192.168.11.179:9797     bigdata 0       2783
Update starting offset with 2783
Using offset range [2783, 2783]
the_last_offset=2783
Totally consume messages _readBytes = 0
Write messages into local file Success!
Append local file to hdfs Success!
Updata offset.dat Success!
```

Figure 3. The Output of HadoopConsumer Module.

C. *Log Statistics Module*

Major function of this module is reading messages from the messaging system Kafka as a consumer, then processing each log record, and writing statistics result to the storage module. It includes two parts: near-real-time online statistics and offline batch processing. We use Storm for real-time streaming computation and HDFS as well as Hive for offline statistics.

In Storm, we write code to package the logic of an application into a Storm topology. A topology is a graph of spouts and bolts that are connected with stream groupings. A spout is a source of streams, here, KafkaSpout is used to pull log data from brokers. Bolt handles tuples receiving from spout or other bolts ahead of it. It can do anything from filtering, functions, aggregations, joins, talking to databases, and more. In the following example, the bolts perform counting over sliding windows and outputting results to Oracle database. In the last step, we use WebSocket for interface display.

In addition to real-time online statistics, there is also a need for offline statistics for massive log in the project. These application scenarios are less stringent on timeliness, but need to aggregate large amounts of historical data for statistics, thus generate reports. In the proposed system, a module named HadoopConsumer is implemented based on Kafka low-level API as well as HDFS API, which acts as a data channel between Kafka and HDFS to pull log messages from Kafka and then write them to files in HDFS. All we need to do is writing correct HQL statement for the queries.

IV. Experiment Results and Analysis

In this experiment, we use the proposed system to collect and analyze log entries formatted as table 1 generated by the network security platform. The rate at which logs are generated is about 10 million every 5 minutes. Next, we mainly present the performance of the online statistics module.

TABLE I. Log Entry

<IP address of the host,Configuration ID, StartTime, Interval>
<192.168.11.34,1019491,157892,300>

The experiment was performed on a cluster consisting of 12 machines each with 16GB of RAM and Intel Xeon E5-2620v2 CPU. Flume and Oracle database are deployed on the same three servers. And Kafka as well as Storm's cluster both consist of three machines. The other three computers form Hadoop cluster, on which ZooKeeper is also deployed. One computer of Hadoop cluster plays the role of NameNode, and the other two are DataNode. One computer in Storm cluster is Nimbus node and the remaining servers are supervisor node. All the servers are managed and monitored in Cloudra.

The operating state of the Storm cluster is as shown in Figure 4 and Figure 5. Figure 4 presents that CPU utilization of the Storm cluster maintains a relatively stable state. It means the deployed cluster can meet operation requirements of the log statistics program. Particularly, the inflection point in the graph is because of garbage collection of JVM. Figure 5 shows the throughput of Storm changes over time, which is about 32701 log records per second.

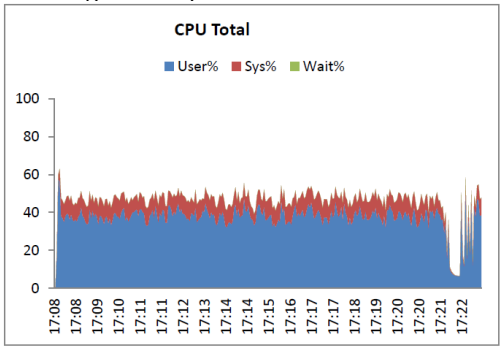

Figure 4. CPU Status.

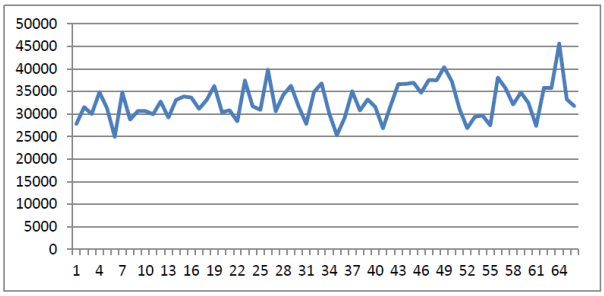

Figure 5. The Throughput of Storm.

V. Conclusion

In this paper, we demonstrated the architecture for a distributed log stream processing system, that supports multi-source heterogeneous log collection, near-real-time online statistics for log stream and offline statistics for massive log. The architecture can be scaled to support a large number of front-end servers that generate logs and huge data size. We deploy the system on a cluster consisting of 12 servers and use it to collect and analyze log entries generated by the network security platform in our laboratory. The test results present that the proposed system does a good job. More importantly, we prove that open source technologies as well as big data frameworks can be utilized for large-scale log data analysis.

Acknowledgment

Project supported by National Key R&D Program 2016 (2016YFB0801300) and National Natural Science Foundation of China (61402464, 61402474).

REFERENCES

[1] Oliner A, Ganapathi A, Xu W. Advances and challenges in log analysis [J]. Communications of the Acm, 2012, 55(2):55-61.

[2] Cardenas A A, Manadhata P K, Rajan S P. Big Data Analytics for Security [J]. IEEE Security & Privacy, 2013, 11(6):74-76.

[3] Comparison of Relational Database Systems, http://en.wikipedia.org/wiki/Comparison of relational database management System.

[4] Apache Flume http://flume.apache.org/ Retrieved: Jul, 2016.

[5] Kreps J, Narkhede N, Rao J. Kafka: A distributed messaging system for log processing[C]//Proceedings of the NetDB. 2011: 1-7.

[6] Storm wiki. 2016. http://en.wikipedia.org/wiki/Storm.

[7] Official Hadoop WebSite, 2016, http://hadoop.apache.org/.

[8] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03), pp. 29–43, October 2003.

[9] Official Hive WebSite, 2016, http://hive.apache.org/.

[10] Official Kafka WebSite, 2016, http://kafka.apache.org/.

[11] Logothetis D,Trezzo C,Webb KC, et al. In-situ MapReduce for Log Processing[C]//Proceedings of USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, 2011:115–129H.

[12] M.Cherniack, H.Balakrishnan,M.Balazinska,D.Carney,U. Cetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In CIDR, volume 3, pp. 257 – 268, 2003.

[13] Patel D, Khasib F, Sadooghi I, et al. Towards In-Order and Exactly-Once Delivery Using Hierarchical Distributed Message Queues[C]// Ieee/acm International Symposium on Cluster, Cloud and Grid Computing. 2014:883-892.

[14] Katsipoulakis N R, Thoma C, Gratta E A, et al. CE-Storm: Confidential Elastic Processing of Data Streams[C]// ACM SIGMOD International Conference on Management of Data. ACM, 2015:66-7.

[15] Aydin G, Hallac I R, Karakus B. Architecture and Implementation of a Scalable Sensor Data Storage and Analysis System Using Cloud Computing and Big Data Technologies [J]. Journal of Sensors, 2015.

[16] Yen T F, Oprea A, Onarlioglu K, et al. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks[C]// Computer Security Applications Conference. 2013:199-208.

[17] Yu Fu, Hongcheng Li, Xiaoping Wu, Jiasheng Wang. Detecting APT Attacks:a Survey from the Perspective of Big Data Analysis[J]. Journal on Communications, 2015, (11):1-14.

[18] Chuanyong Zhang. Analysis of Network Security Problem Based on Large Data Age [J]. Network Security Technology & Application, 2015, (01):101+104.

[19] Kazuyoshi Furukawa, Satoru Shimizu, Masahiko Takenaka, and Satoru Toriil, "On Detection for Scarcely Collided Super-Slow Port Scannings in IDSs' Log-Data," Journal of Communications, vol. 8, no. 11, pp. 788-794, 2013. doi: 10.12720/jcm.8.11.788-794.

AUTHORS' BACKGROUND

Your Name	Title*	Research Field	Personal website
Jingfen Zhao	master student	Network Security, Big Data	
Zhengmin Li	Phd candidate	Network Security, Big Data	
Peng Zhang	associate professor	Large-scale data stream real-time processing	http://www.mesalab.cn/f/PersonnelTraining/toPersonDetail?id=6
Yong Sun	associate professor	Network Security	http://www.mesalab.cn/f/PersonnelTraining/toPersonDetail?id=47
Qingyun Liu	associate professor	Network Security	http://www.mesalab.cn/f/PersonnelTraining/toPersonDetail?id=3
Guolin Tan	Phd candidate	Large-scale data stream real-time processing	

*This form helps us to understand your paper better, the form itself will not be published.

*Title can be chosen from: master student, Phd candidate, assistant professor, lecture, senior lecture, associate professor, full professor