2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems

An Integrated Distributed Log Management System with Metadata for Network Operation

Minoru Ikebe
Department of Computer Science and Intelligence Systems,
Faculty of Engineering, Oita University
Oita, Japan
minoru@oita-u.ac.jp

Kazuyuki Yoshida
Center for Academic Information and Library Services,
Information Technology Center, Oita University
Oita, Japan
yoshida@oita-u.ac.jp

***Abstract*—An enormous amount of log data is generated by servers and other devices on the network, and server/network administrators analyze the logs to investigate anomalous communications or troubleshoot. However, server/network management tasks increase in volume and complexity, resulting in greater burden on the administrator. In this paper, we propose an integrated management system for a sensor network where log data is output from many different kinds of sensors. We consider a server or network device as one of the sensors. We also propose a cross-processing system for several kinds of log data. In particular, we describe the management and collection of logs in our campus networks.**

***Keywords*-Server Log; Network Operation; Distributed System; Metadata;**

I. Introduction

University and corporation networks typically contain many servers and devices for information processing. These servers and network devices generate an enormous amount of log data; therefore, they require high reliability. In addition, virtualized servers run on cloud computing environments.

Administrators manage many servers in the local area network, and automatically managing the network operations can be difficult when server systems become large and complex. They must monitor the network operations by analyzing several logs (e.g., server logs, hypervisor logs, network device logs, and traffic data) to detect anomalies and troubleshoot.

Because various devices output many types of log data, administrators could also have difficulty finding the correct log for network operations (Figure 1).

In this research, we propose an integrated management system for log data from many different kinds of sensors, as well as servers and network devices, which we consider as sensors in this study. We also propose a cross-processing system for several kinds of log data. We aim to provide a flexible method to access the distributed log data using the log attributes and values. In particular, we describe the management and collection of logs in our campus networks.

In the remainder of this paper, we describe related work in Section 2. Section 3 describes the design of our proposed

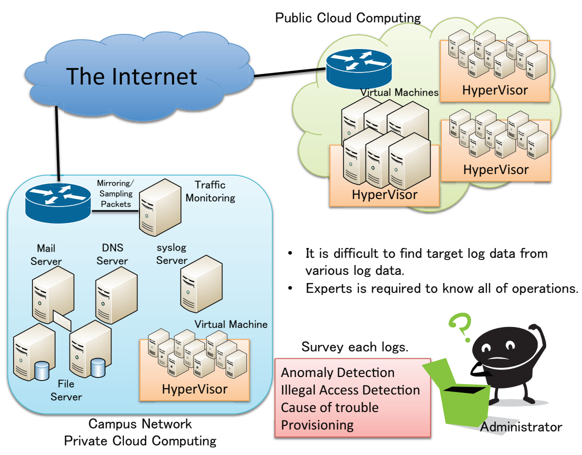

Figure 1. Various devices output many types of log data

system and log data collection and management components; Section 4 describes the implementation of these components. Section 5 summarizes this paper and describes future work.

II. Related Work

Searching for a specific error across hundreds of log files on hundreds of servers is difficult without a good method. A common approach to this problem is to set up a centralized logging process so that multiple logs can be aggregated in a central server.

A simple approach is to copy several logs to a central server on a cron schedule, i.e., a method using rsync or scp. These methods can perform simple work; however, they do not provide timely access to log data.

If each server mounts a file server using the NFS[1] protocol, then administrators can get timely access to log data. However, with a central NFS server, the administrator must determine how to scale the service and make it highly available.

The syslog[2] daemon allows processes to send log messages to another server. The syslog has two implementations (rsyslog and syslog-ng), and it is already installed on several systems. However, with a central syslog server as well, the administrator must determine how to scale the server and make it highly available.

III. Proposal System Design

Our system collects log data and offers search function of the log data. Furthermore, the cross-sectional search is possible. The purpose of our research is to support the server or network administrators. We propose an integrated management system for log data from many different types of sensors, as well as servers and network devices. Our system collects logs and provides the capability to perform a search of the log data, including cross-sectional search. We also built a cross-processing system for several kinds of log data.

A. Design

Figure 2 shows our log management system architecture to manage the log data and the metadata in a client—server model. In figure 2, 4 log and metadata management system has the same functions and roles. The log collector obtains the log data from each process on each server. Log data management systems can manage the log data and metadata in each server node using a distributed hash table[3].

As a policy, we avoid log management, except when running users' application programs, to use computing resources efficiently on the server nodes. We designed the central log data management server to collect log data and metadata from each server and then copy the collected data to secondary log data management servers, so that administrators can get the same response from any log data management server.

Our proposed system does not manage schemas of log data and metadata. Instead, administrators have to register and manage the log data schemas, and the log management system manages the log data table for every log data schema. However, the system must distinguish requests from the administrator for every application when using this method, and some log data may not be in the same format. When one log data is in a different format, managing the schema can be difficult; therefore, log data management systems tend to become complex systems. In addition, the log data management system must keep all the log data schemas of the entire network. If administrators later change the log data format, then they modify all elements in the log data database. Thus, managing log data schemas on networks is difficult.

We propose that administrators continue using a simple system architecture for managing log data. A schema-less log data management method, such as our proposed method, can manage various log data schemas in one system.

However, users cannot tell which server the same type of log data were output by. The log data output the result of the event. The information for identification is not included

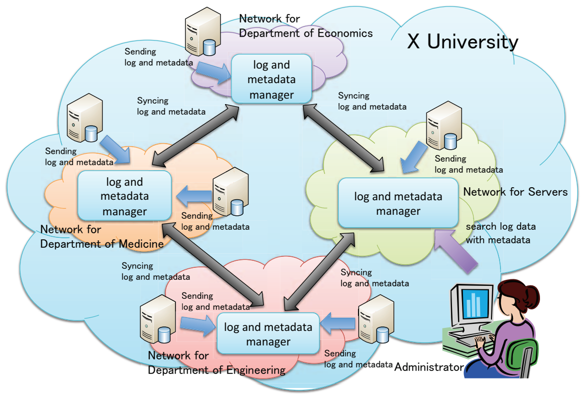

Figure 2. Log management system architecture

in log data. Therefore, we use metadata to distinguish log data by our system. We perform integrated management of different types of log data by including IP addresses and the types of the data including the service name (e.g. httpd's log, sshd's log, pcap, and so on) in metadata when users refer for log data.

Our system stores log data and metadata in multiple log data management servers, so that administrators can obtain the same results from any server, and our log management system offers this feature to administrators.

B. Usage

We have been developing an anomaly detection system using TCP connection states to detect malicious attacks. By mirroring packets from a layer-3 border switch, our anomaly system detects packets that are different from the TCP three-way handshake procedure. At same the time, we use tcpdump to store the mirrored packets into a pcap file, which is used to confirm an attack. After we determine the attacker's IP address through our anomaly detection system, we compare the IP address with the pcap data to determine whether an attack has occurred.

The anomaly detection system can be used with our log management system as follows:

1) Our anomaly detection system determines the IP address of the attacker.
2) The administrator queries the IP address of the attacker from our log management system.
3) The administrator obtains the pcap data about the specified IP address.
4) The administrator reads the obtained data and confirms the attack.

IV. Prototype Implementation

A. Prototype system overview

Our proposed system collects log data from each server and manages the log data/metadata on the local area network (Figure 3).

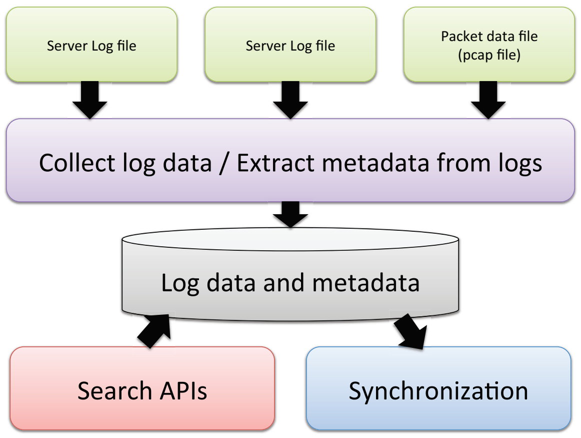

Figure 3. Log collector/management system overview

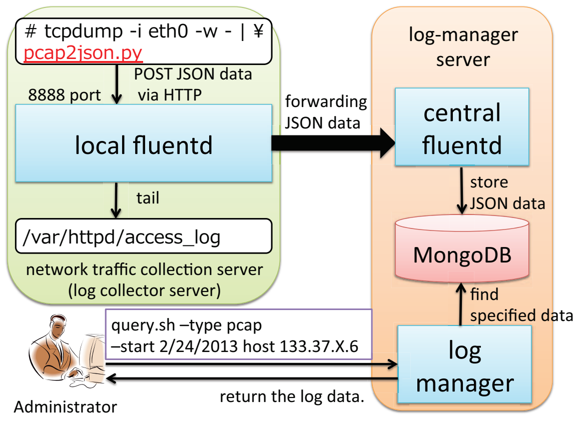

Figure 4. Prototype implementation

B. Log collector

Our prototype system uses Fluentd[4], which is a lightweight, flexible open-source log collector. Fluentd receives logs as JSON streams, and the Fluentd nodes in application servers forward the local logs to the Fluentd node in the central server. For example, the log collector can handle an Apache combined log, bind9 queries log, and the pcap file. The Apache log plugin is included in Fluentd by default, and we have written the scripts for the bind9 queries log and the pcap file.

Figure 4 shows the log-manager system and the log-collector system. The *pcap2json.py* script converts the pcap data (Figure 5) to JSON data (Figure 6), and *pcap2json.py* posts the JSON data to localhost:8888 via the HTTP protocol. The Fluentd daemon then forwards the JSON data to the central Fluentd server.

A printf-style format string is the de facto standard of logging for almost all software, because this log format style is optimized for human readability. However, the log data must also have a well-defined structure that can be parsed easily by programs. JSON is human-readable and its structure is easy to parse. We consider that log data have an affinity for the JSON.

C. Log manager

Our prototype system also uses MongoDB[5], which is a scalable, high performance, open source NoSQL database. MongoDB can store JSON-style documents with dynamic schemas. Fluentd receives the JSON data from other Fluentd servers through port 24224 and stores the JSON data into the MongoDB. The log manager extracts the metadata from the JSON data, and our system extracts the metadata from the pcap file. We extract the log format type (in this case, pcap), the log source (IP address), the start time, the end time, and the filename (if available) from the pcap file. The log manager copies the log data and metadata to other log managers. The replication function is currently being implemented. We develop the replication function using one master node and two slave nodes. Because, the MongoDB requires least three nodes in replica set function.

V. Conclusion

In this paper, we described an integrated log management system that collects log data from each server and manages them in one system. We also implemented part of a prototype system. In the future, we will evaluate the effectiveness of our system and implement functions to handle several log formats (i.e., CloudStack, OpenStack, Eucalyptus, Sendmail, bind9, and the system logs of the OS).

In addition, we are going to implement the search function and a query interface, which will be available as a shell script language or an interpreter language.

References

[1] B. Nowicki, "NFS: Network File System Protocol specification," RFC 1094 (Informational), Internet Engineering Task Force, Mar. 1989. [Online]. Available: http://www.ietf.org/rfc/rfc1094.txt

[2] R. Gerhards, "The Syslog Protocol," RFC 5424 (Proposed Standard), Internet Engineering Task Force, Mar. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5424.txt

[3] J. M. J.Xing, J.Xiong and N. Sun, "Memory based metadata server for cluster file systems," in *7th International Conference on Grid and Cooperative Computing*, Oct 2008, pp. 287–291.

[4] "Fluentd: Log everything in json," http://fluentd.org/.

[5] "MongoDB," http://www.mongodb.org/.

```
03:22:02.098772 IP 186.193.xxx.131.4973 > 133.37.X.0.telnet: Flags [S], seq 557142724, win 5760
03:22:02.100265 IP 210.6.xxx.4.netplay-port1 > 133.37.Y.0.telnet: Flags [S], seq 2324441067, win 5840
03:22:02.121585 IP 182.150.xxx.6.35532 > 133.37.Z.0.telnet: Flags [S], seq 1506311687, win 5440
03:22:02.135671 IP 220.62.xxx.90.50716 > 133.37.X.6.pop3: Flags [P.], seq 238510740:238510755, ack 1265051637,
win 256
03:22:02.142085 IP 182.147.xxx.40.60669 > 133.37.Y.34.websm: Flags [S], seq 2849085523, win 5840
```

Figure 5. Pcap file format

```
{”psh”: 0, ”dstip”: ”133.37.X.0”, ”tcpflag”: 2, ”proto”: 6, ”ack”: 0, ”srcport”: 4973, ”syn”: 1, ”rst”: 0, ”tcpwin”: 5760,
”time”: [1354299722, 98772], ”srcip”: ”186.193.xxx.131”, ”plen”: 74, ”dstport”: 23, ”fin”: 0, ”urg”: 0}
{”psh”: 0, ”dstip”: ”133.37.Y.0”, ”tcpflag”: 2, ”proto”: 6, ”ack”: 0, ”srcport”: 3640, ”syn”: 1, ”rst”: 0, ”tcpwin”: 5840,
”time”: [1354299722, 100265], ”srcip”: ”210.6.xxx.4”, ”plen”: 74, ”dstport”: 23, ”fin”: 0, ”urg”: 0}
{”psh”: 0, ”dstip”: ”133.37.Z.0”, ”tcpflag”: 2, ”proto”: 6, ”ack”: 0, ”srcport”: 35532, ”syn”: 1, ”rst”: 0, ”tcpwin”: 5440,
”time”: [1354299722, 121585], ”srcip”: ”182.150.xxx.6”, ”plen”: 74, ”dstport”: 23, ”fin”: 0, ”urg”: 0}
{”psh”: 1, ”dstip”: ”133.37.X.6”, ”tcpflag”: 24, ”proto”: 6, ”ack”: 1, ”srcport”: 50716, ”syn”: 0, ”rst”: 0, ”tcpwin”: 256,
”time”: [1354299722, 135671], ”srcip”: ”220.62.xxx.90”, ”plen”: 69, ”dstport”: 110, ”fin”: 0, ”urg”: 0}
{”psh”: 0, ”dstip”: ”133.37.Y.34”, ”tcpflag”: 2, ”proto”: 6, ”ack”: 0, ”srcport”: 60669, ”syn”: 1, ”rst”: 0, ”tcpwin”:
5840, ”time”: [1354299722, 142085], ”srcip”: ”182.147.xxx.40”, ”plen”: 74, ”dstport”: 9090, ”fin”: 0, ”urg”: 0}
```

Figure 6. JSON format