Hayabusa: Simple and Fast Full-Text Search Engine for Massive System Log Data

3.1 Background and Objectives

In the field of network and system operations, network administrators store logs to display statistical information and/or search the logs to identify the source of trouble, to ensure stable network operations and perform troubleshooting. When responding to security incidents, as with troubleshooting, there are also cases where logs are used to investigate the type of incident that occurred.

In large-scale networks, communication logs are output in bulk from many servers, networking and security devices on a daily basis. Network administrators use systems to accumulate these logs in storage systems so that they can search the logs quickly. Large-scale search and storage systems tend to require the use of clustering systems or dedicated management software, and in many cases, operators have to use their time to manage these systems which should be spent for analyzing logs.

If we can build a system to “store” and “search” logs simply without the use of complex clustering systems, network administrators would no longer need to devote as much time to managing the storage and search systems, allowing them to focus more on their core tasks of network troubleshooting and the analysis of security incidents.

In this report, we discuss the Hayabusa open-source software that implements a system capable of quickly storing and retrieving a large amount of logs output by a wide range of devices from multiple vendors. We also introduce a conceptual model for a distributed system using the same software that dramatically improves search speed by scaling out the system’s searching capability when the volume of logs handled by the system increases.

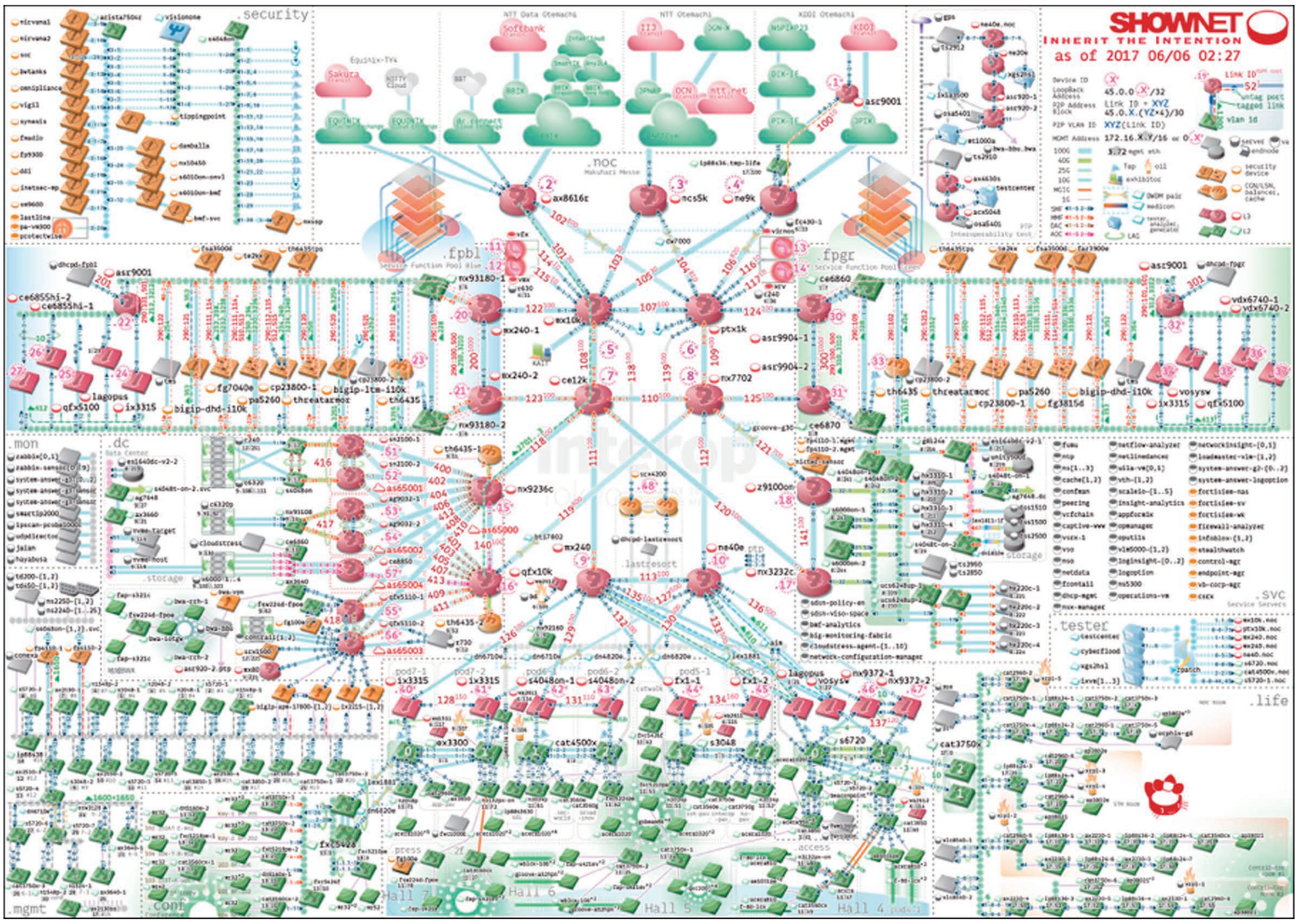


Figure 1: 2017 ShowNet Topology Diagram

To evaluate the search speed of Hayabusa using a large amount of log data output by various devices from multiple vendors, we performed tests based on actual ShowNet syslog data collected at Interop Tokyo*1, output from over 600 servers, networking and security devices.

3.2 ShowNet

ShowNet is a test network environment for verifying interoperability and performing demonstrations constructed at Interop Tokyo, which is held annually at Makuhari Messe. Because it also offers an Internet access service to exhibitors, it has two-fold aspects, one for a real business network service and the other for an experimental network service.

As shown in Figure 1, ShowNet is comprised of hundreds of pieces of equipment, including not only the products in the market but also many products that are connecting to a network on which services are running for the first time, with some devices and software still in the prototype stage.

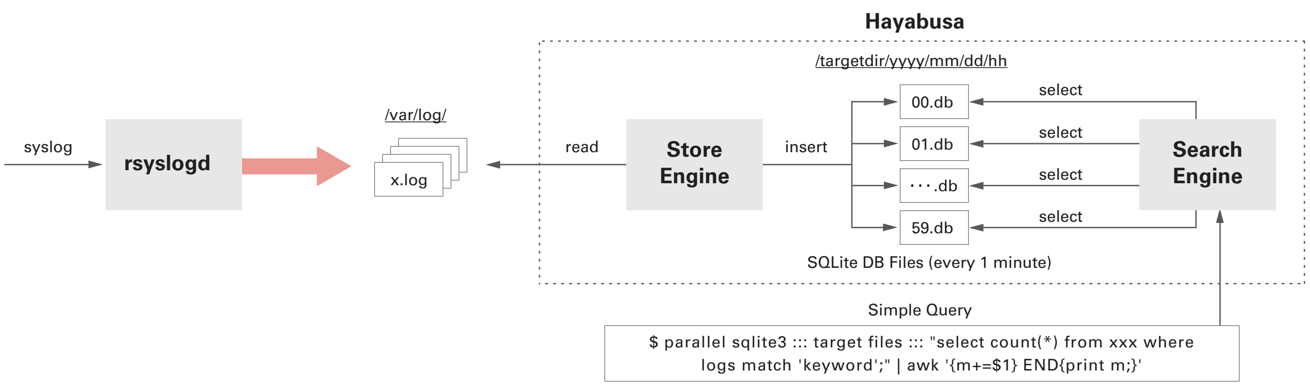
NOC (Network Operation Center) members operating ShowNet combine these devices and software to design, build, and operate networks to provide services. To construct a stable system, NOC members assess system bugs and errors along with the feasibility of configurations by collecting and analyzing syslog data output from the various equipment. Since ShowNet is comprised from various equipment incorporating a number of cutting-edge software and hardware equipment, so it is difficult for operational members managing the equipment and software to know what kind of logs that the various equipment will output in advance.

In ShowNet, we run a system to collect and analyze syslog data as one of the monitoring systems. Almost all physical and virtual devices send syslog data to the constructed system with which we can aggregate and search the data. In the past, there have been times when over 20,000 log entries were exchanged in a second, and over 200 million log entries were stored in a single day.

It is possible to infer the format of logs if they come from a small set of specific software or hardware devices, but there are very few cases where the same software or hardware is used in the construction of ShowNet. A large number of messages are also output as debug information for the newest firmware or software. This makes it very difficult to display statistical data and search in real time using standard log storage and analysis software.

3.3 Hayabusa

Hayabusa*2 was designed as a system for high-speed searching through a large amount of syslog data collected at ShowNet. Figure 2 shows the architecture of Hayabusa.



```
$ parallel sqlite3 ::: target files ::: "select count(*) from xxx where
logs match 'keyword';" | awk '{m+=$1} END{print m;}'
```

Figure 2: Hayabusa Architecture

*1 Interop Tokyo (https://www.interop.jp/).
*2 Hayabusa (https://github.com/hirolovesbeer/hayabusa).

Hayabusa runs on a stand-alone server and uses multiple CPU cores efficiently to achieve high-speed parallel search processing. It is comprised of two core parts: the StoreEngine, and the SearchEngine. The StoreEngine is launched every minute as a cron job. It opens the target log file and converts the log messages into SQLite3*3 files. Log data is split into SQLite3 files every minute, and multiple search processes use these files in parallel. The directory for storing logs is defined using a hierarchy that indicates the time as shown below.

/targetdir/yyyy/mm/dd/hh/SQLite3 files for each minute

By defining in this way, it is possible to match with directories without retaining time data in a database for searching, and time ranges do not need to be specified in a query when searching through the log messages, which is an operation that sometimes takes time. The SQLite3 files where logs are saved are created as FTS (Full-Text Search) tables dedicated to full-text searches, enabling high-speed log retrieval. The SearchEngine accesses the FTS formatted SQLite3 files split each minute to improve the performance of parallel searches. SQL search queries are executed in parallel for each SQLite3 file using GNU Parallel*4 and results are aggregated via the UNIX pipeline using the awk or count commands.

Hayabusa operates in a stand-alone environment, but it features higher full-text search performance than small-scale Apache Spark*5 clusters. As shown in Figure 3, a preceding study*6 indicated that search performance was 27 times faster than a 3-node Apache Spark cluster.

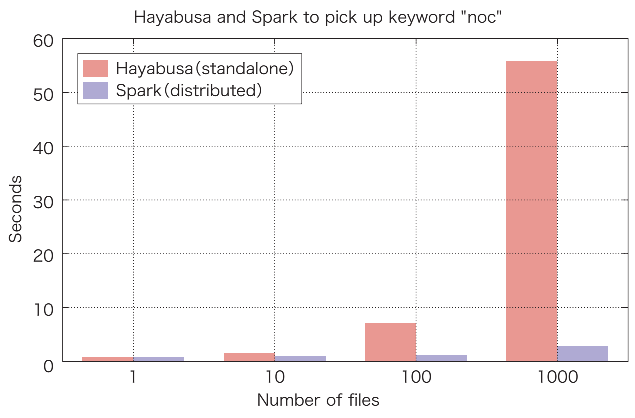


Figure 3: Comparison of Hayabusa and Apache Spark Performance

*3 SQLite3 (https://www.sqlite.org/).
*4 GNU Parallel (https://www.gnu.org/software/parallel/).
*5 Apache Spark (https://spark.apache.org/).
*6 H. Abe, K. Shima, Y. Sekiya, D. Miyamoto, T. Ishihara, and K. Okada. Hayabusa: Simple and fast full-text search engine for massive system log data. In Proceedings of the 12th International Conference on Future Internet Technologies, CFI'17, pages 2:1-2:7, New York, NY, USA, 2017. ACM.

3.4 Hayabusa Distributed Processing

However, stand-alone environments have hardware performance limitations, and it is estimated that performance will be overtaken by other larger scale distributed processing cluster sytems sooner or later. In light of this, with Hayabusa we aimed to remove the constraints of a stand-alone environment, and implemented an architecture capable of scaling out search performance by constructing a distributed processing environment for Hayabusa using multiple hosts.

We redefined the stand-alone version of Hayabusa as a distributed processing system and conducted tests for scaling out search processing capability. To leverage the stand-alone processing capability of Hayabusa, each processing request is sent from the client to the target host as a high-speed RPC (Remote Procedure Call). Parallel searches are implemented on the processing host using GNU Parallel, and results are returned to the client using RPC, and then aggregated. This enables the fast searches in stand-alone environments that Hayabusa originally made possible, while implementing high-speed requests and responses via RPC. To implement a search function that scales out in a distributed host environment, we designed the parallel data storage and distributed search functions separately.

3.4.1 Parallel Storage and Distributed Search

To evaluate the scale out performance of search processes, we configured all the processing hosts to retain the same data, so that searches were possible regardless of which processing host would receive the processing requests. That means syslog data is copied and sent to all of the processing hosts. This ensures that the same result is returned no matter which processing host handles the processing request.

We used RPC to implement search requests to a set of distributed processing hosts. Distributed processing for Hayabusa assumes that processing hosts that receive requests from clients will issue search queries to multiple database files saved to a local disk, much like a stand-alone Hayabusa implementation. It is possible to pass SQLite3 commands to a processing host as a search request parameter, but to achieve a simple distributed process while making full use of the performance of a stand-alone Hayabusa, similar GNU Parallel commands are passed as request parameters in this proposal. In a distributed search, processing requests are issued to multiple processing hosts, so we selected a producer-consumer model where clients queue processing requests, and workers running on processing hosts acquire queued processing requests and return the results. Workers running on each processing host execute the search process after acquiring a processing request.

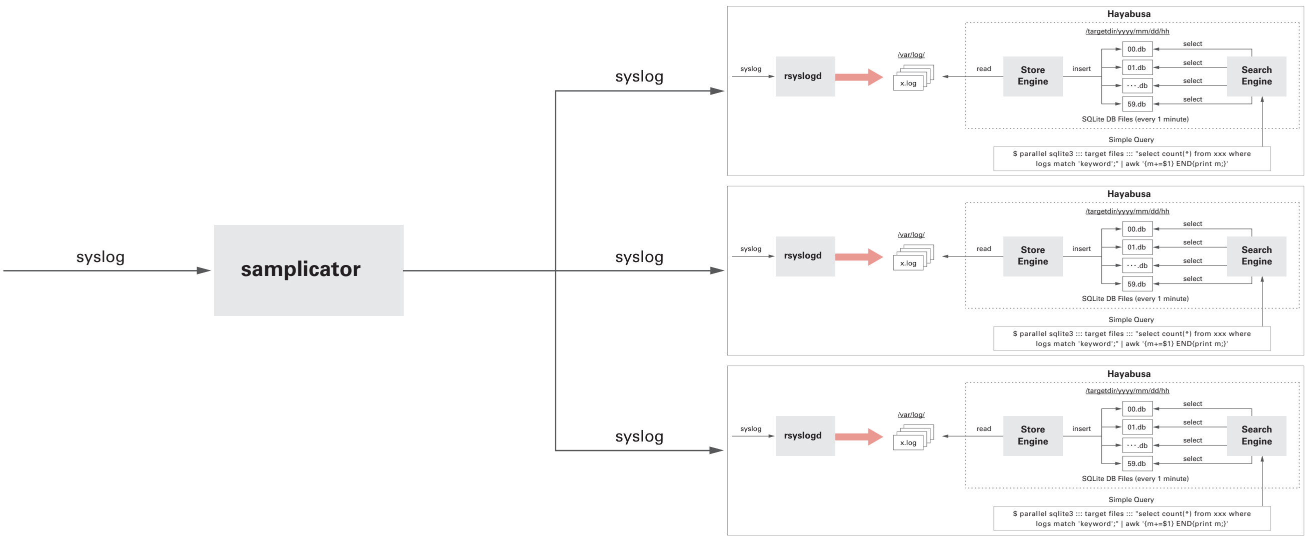
In this proposal, data is copied to all processing hosts, so the same results are returned no matter which processing host receives a processing request. We also performed load balancing so that processing requests from clients are not concentrated on certain processing hosts, meaning requests are distributed evenly.

3.4.2 Implementation

■ Parallel Storage

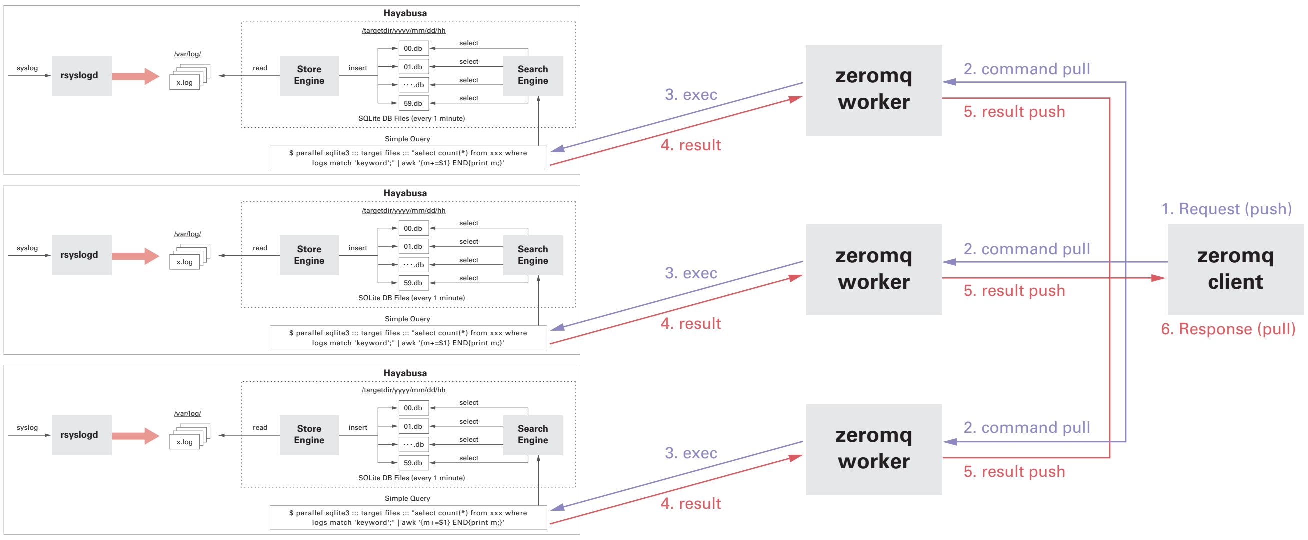
In our research, we used the open-source software UDP Samplicator*7 to copy syslog data to all the processing nodes. Figure 4 shows an illustration of the syslog replication process. UDP Samplicator can forward incoming UDP packets to a designated target host without changing the source address. This technique makes it possible for the destination host to receive UDP packets as if the data has been received directly from the source.

Normally, UDP Samplicator performs UDP forwarding using a single process. However, the process load increases when a large volume of syslog data is received, sometimes causing CPU usage to hit 100%. This load can cause packet forwarding to be delayed, leading to dropped packets. In these tests, we patched the UDP Samplicator source code so that it can be run as multiple



```
$ parallel sqlite3 ::: target files ::: "select count(*) from xxx where
logs match 'keyword';" | awk '{m+=$1} END{print m;}'
```

Figure 4: Syslog Replication Using UDP Samplicator



```
$ parallel sqlite3 ::: target files ::: "select count(*) from xxx where
logs match 'keyword';" | awk '{m+=$1} END{print m;}'
```

Figure 5: ZeroMQ Push/Pull Implementation Used With UDP Hayabusa

*7 UDP Samplicator (https://github.com/sleinen/samplicator).

processes. More specifically, multiple processes can use the same listening port by adding "SO_REUSEPORT" as a socket option. In this proposal, the incoming syslog port UDP 514 is shared, with packets sent to this port being automatically load balanced between multiple UDP Samplicator processes. This technique makes it possible to scale the replication and forwarding of syslog packets across CPU cores when a large volume of syslog data is received, which tends to become a bottleneck when using only a single CPU core.

■ Distributed Search

The producer-consumer model can be implemented using a wide range of software, but for this research, we used ZeroMQ*8 because it enables high-speed RPC processing, and client and worker processes can be implemented using libraries. ZeroMQ is used as a distributed message queue that runs at high speed, and it is easy to implement a wide range of messaging patterns such as Request/Response, Publish/Subscribe, and Push/Pull. In this proposal, we implemented a producer-consumer model using the Push/Pull pattern.

As shown in Figure 5, the clients in this proposal were implemented to fulfill both the Push and Pull roles. This enables the client program to perform everything from the issuing of requests, queueing, and the acquisition and aggregation of results through a single client process.

3.5 Evaluation

To investigate the scale-out performance of Hayabusa as a distributed system, we tested whether processing time could be reduced when the number of processing hosts is increased. While increasing the number of processing hosts incrementally from one to ten, the client repeatedly executes 100 requests against one day of data. The size of the data requested 100 times is 14.4 billion records (assuming a syslog flow rate of 100,000 records per minute).

As shown in Figure 6, search processing time for one host was about 468 seconds. As the number of hosts increased, the result was roughly this amount of time divided by the number of hosts. The processing time of ten hosts was 48 seconds.

While 30 million records can be scanned per second when a single host is used, with ten hosts this increases to 300 million, ten times that value. This experiment demonstrates that when the number of hosts is increased from one to ten, search processing performance scales out linearly with the number of hosts.

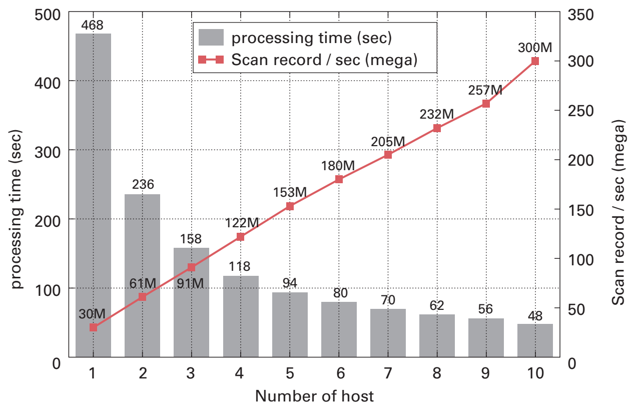


Figure 6: Search Host Scale-Out Performance

*8 ZeroMQ (http://zeromq.org/).

Next, we increased the number of workers incrementally from one process through 16 processes, while processing on ten hosts. We settled on the number 16 because it was the number of physical cores on the CPU of the server used for this test, and we wanted to check whether it would be possible to scale out to the maximum number of physical cores. Running a single worker process on ten hosts took 48 seconds, while running ten worker processes on ten hosts attained a peak processing performance of 6.1 seconds, providing around eight times the processing speed. Processing speed saturated after adding the tenth worker, so we believe attaining maximum performance with sixteen processes may require some additional effort.

When combining the scale-out of both the number of hosts and the number of workers, a search that took 468 seconds when using one worker process on one processing host, dropped to 6.1 seconds when using ten worker processes on ten hosts, which is about 78 times faster.

3.6 Future Challenges

In this research, we copied the same syslog data to each processing host to perform distributed queries in an attempt to improve search performance. This involves the mass replication of data, meaning that as the amount of data increases, there is a waste of network bandwidth and the data to be retained. It is possible to define the number of times that data is replicated, then distribute and retain data across multiple hosts as with HDFS in Hadoop[*9]. However, in this case, a metadata management system manages the data, and data access is provided through this system as well, which may decrease performance when accessing storage, and lower the performance of the search process. Considering the advantages and disadvantages of Hayabusa and distributed storage, and implementing a distributed storage solution suitable for Hayabusa at the same time, will be a major challenge.

Because Hayabusa operates as a search infrastructure system, it has the potential to provide more benefits when combined with specific application software that runs on it.

It enables high-speed syslog searches, so we believe it will be possible to create high-speed anomaly detection applications when combining with prior research on anomaly detection[*10] using syslog data in event networks.

*9 Apache Hadoop (http://hadoop.apache.org/).

*10 Hiroshi Abe and Mikifumi Shikida. Proposal of the anomaly detection method analyzing syslog data using Bollinger Bands algorithm on event network. In Proceedings of the Internet and Operation Technology Symposium 2016, Volume 2016, Pages 57-64, Dec 2016.

3.7 Hayabusa Applications

One of the goals of research in the security field is to predict and detect attacks.

To counter the increasing threat of cyber attacks, it would help to be able to analyze signs of attacks in real time and assess potential attacks as well as their severity and the extent of the impact. This would make it possible to utilize machine learning and deep learning to support responses to security incidents that have up until now been largely dependent on the individual skills of the staff in charge.

The IIJ Research Laboratory is taking part in the "Real-time Attack Detection and Prediction via Cyber Threat Big Data Analysis"*11 project in collaboration with the University of Tokyo, Tokyo Institute of Technology, and Nara Institute of Science and Technology. In this project, Hayabusa will be incorporated as part of the infrastructure for analyzing signs of attacks in real time. It will also operate as infrastructure for gathering statistics and applying machine learning analysis to signs of attack in collaboration with other data analysis and machine learning software (R, Chainer, Pandas, etc.).

3.8 Conclusion

In this report, we provided an overview of the Hayabusa open-source software and experiment results that used distributed processes. The fact that we were able to perform a full scan across 14.4 billion records in approximately six seconds means that we have achieved a full data scan speed on the same level as Google's BigQuery. The ability to realize such a high-speed scan with ten processing hosts demonstrates this is also a reasonable and high-performance distributed processing system from a cost perspective.

A more detailed explanation of the contents in this report have been published in the paper*12.

Part of this research has been carried out with the support of the "Strategic Creation Research Promotion Project (CREST) JPMJCR1783" research and development project implemented by the Japan Science and Technology Agency (JST).

Author:
Hiroshi Abe
Researcher, IIJ Research Laboratory

Author:
Keiichi Shima
Senior Researcher, IIJ Research Laboratory

*11 "Real-time Attack Detection and Prediction via Cyber Threat Big Data Analysis" (https://www.jst.go.jp/kisoken/crest/project/1111094/1111094_13.html) (in Japanese).
*12 Hiroshi Abe and Yoichi Shinoda. Research of the scalable syslog search engine and evaluation. In Proceedings of the Internet and Operation Technology Symposium 2017, Volume 2017, Pages 73-80, Nov 2017.