Scientific Research Publishing

Journal of Software Engineering and Applications, 2017, 10, 843-853
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

# Research on Establish an Efficient Log Analysis System with Kafka and Elastic Search

**Yuanzhou Wei[1], Manman Li[2], Binsen Xu[1]**

[1]Beihang University, Beijing, China
[2]Henan University of Economics and Law, Zhengzhou, China
Email: weiyuanzhou@buaa.edu.cn

**How to cite this paper:** Wei, Y.Z., Li, M.M. and Xu, B.S. (2017) Research on Establish an Efficient Log Analysis System with Kafka and Elastic Search. *Journal of Software Engineering and Applications*, **10**, 843-853.
https://doi.org/10.4236/jsea.2017.1011047

**Received:** January 19, 2017
**Accepted:** October 27, 2017
**Published:** October 30, 2017



## Abstract

Search logs in a timely and efficient manner are an important part of SRE (Site Reliability Engineer). Logs help us solve the problems during our development work. In this paper, we will introduce you a way how to build an efficient logs analysis system based on kafka and Elastic Search. We hope you can learn something through the iteration of the Version and get some inspiration with your own log analysis system.

## Keywords

Log Analysis System, Kafka, Elastic Search, Software Applications, Site Reliability Engineer

## 1. Introduction

This article shared with you the method how we built ourselves' log analysis system step by step, and introduced the version from the beginning to evolve into this version that we are now using. If you want to use Elastic Search and Kafka doing log analysis, there will be some inspiration. The full text mainly focuses on the following Topics:

1) The basic needs of log analysis system;
2) The evolution of our log system;
3) Our experiences and thoughts;

First, we needed to understand what is log? In simple, log is just a structured data with stamp. Log is there when the computer was appeared, from that time, we created a wide variety of tools to help us analyse, interpret or search logs.

At beginning, many teams feel that was not necessary to make this thing, engineers can log in to the server to do some cat or grep operate and other simple

expression handling, you can find the information you need through these operations. If not enough, such as in many machines, then, you can use other tools like mssh and cssh.

If you found it was also not enough for you, you can write your own tools by yourself, one time I found a tool on our production server, there was an engineer who wrote a system from their own desktop, did a ssh tunnel to the actual production system which made a remote code call, remote to take those files back, this was a level of safety accidents, and it was very irresponsible, but it also showed that we do have this demand.

When we had 50,000 servers, or more than 500 micro-services, you can't expect everyone to be very skilled to solve such things. Development or operation and maintenance often encounter such a demand, for example, take a time between two points of all the logs, only to see WARN or ERROR or FATAL message, and there are a dozen errors are known to be ignored.

This service was running on several data centers which contain hundreds of servers, but we needed to care about whether there got no new errors, this error was not due to a particular user caused, or some specific user behavior caused, for example, what post someone did? Whether the length of the request was more than a fixed length? Whether the error message on this server was associated with the error message on the other server? Give me 30 minutes I have the possibility to write a four or five lines grep order to hundreds of servers to log down, but if at three o'clock in the morning, this is an unlikely task.

1) For important logs, we needed to meet the index, retrieval, sorting, classification, and to provide a certain degree of visualization, analysis of the log function;

2) Can be scaled horizontally according to the size of the data. Because the development of the Internet is very fast, we hope to find a solution, not a year or even six months, when the server or the number of users doubled, the solution is completely unavailable. So, we need to find a program, when the number of users doubled, simply add a few machines or servers can continue to use;

3) The system can be easily extended, because many companies already have a lot of alarm or monitoring system. Can it be easily accessed through the API or through extended access to the existing monitoring, alarm, or other systems inside [1].

There were a number of other scalability requirements, including logging samples, improving security, and protecting the information contained in the logs.

Our log system evolution back to four years ago, from established in 2012 we had a system called Splunk, very easy to use, only one problem, too expensive. In 2012, we had three or four thousand servers in the production environment, and when we renewed our contract for the second year, their offer was $20 million a year. This was really unacceptable, and that time was 2012, we now server number, the number of user requests had turned almost ten times, then the price if it

was $20 million at 2012, now more, because it was based on the amount of data Operator to calculate price.

From 2012 to 2014, we decided not use Splunk, and we entered a chaotic period, this time was very painful, we all had needs, but no one has the method, many people began to engage in their little tricks, make some small tools. I had looked at the internal tool before the library, which had 20 or 30 python or shell written with a small tool that is used to find a small period of time log or log of a particular user, but there is a great waste, Many tools was repeated, and also very difficult to maintain.

So we made a determined effort at 2014 to 2015, we must build a Log system which can stretch all logs across LinkedIn, and extended to the entire LinkedIn. ELK was selected, its advantages were: open source, release cycle was very fast, of course, there were some shortcomings, it was very new and there were many small problems.

I believed many people had already know what ELK is—Elastic Search, Logstash and Kibana. Elastic Search was based on the Lucene storage, indexing, search engine; Logstash is to provide input and output and conversion processing plug-in log standardization of the pipeline; Kibana provide visualization and query ES user interface, as shown in Figure 1.

## 2. Related Work

### 2.1. Fundamentals and Strengths of Kafka

Apache kafka was a distributed push-subscribe-based messaging system, it has a fast, scalable, and sustainable features. It was now an open source Apache system, as Hadoop ecosystem as a part of a wide range of commercial companies was widely used. Its biggest feature was the ability to process large amounts of data in real time to meet a variety of needs scenarios: such as hadoop-based batch processing system, low-latency real-time systems, storm/Spark streaming engine.

Kafka has these basic fundamentals: High throughput, low latency: kafka can handle hundreds of thousands of messages per second, with latency of only a few milliseconds; Scalability: kafka clusters support thermal expansion; Persistence, reliability: messages were persisted to a local disk, and data backup is supported to prevent data loss fault tolerance: Allows nodes in a cluster to fail ($n$ – 1 nodes are allowed if the number of replicas is $n$) High concurrency: supports thousands of clients reading and writing at the same time [2].


Figure 1. How to use log data.

### 2.2. Fundamentals and Strengths of ElasticSearch

ElasticSearch (ES) was an open source, distributed, RESTful full-text search engine built on top of Lucene.

However, ElasticSearch was not just a full-text search engine, it was also a distributed real-time document storage, where each field was indexed data and can be searched; was a real-time analysis of the distributed search engine, and Can be expanded to hundreds of servers to store and process petabytes of data [3].

The main advantages of ES can see the following aspects:

1) Horizontal scalability: only need to add a server, do a little configuration, start the ES process can be incorporated into the cluster;

2) Fragmentation mechanism to provide better distribution: the same index into multiple sharding (sharding), which is similar to the HDFS block mechanism; divide and conquer the way to improve processing efficiency, I believe we are not unfamiliar;

3) High Availability: Provides a replica mechanism that allows multiple replicas to be set up in a single fragment, so that when one server goes down, the cluster can still function as usual and restore replication lost due to server downtime to another Available on the node; this is also similar to the HDFS replication mechanism (HDFS default is 3 copies).

## 3. Evolution of Our Log Analysis System

Everyone can spend 30 minutes in their own computer or production environment to take such a thing: Log through Logstash read out, into the ElasticSearch, then Kibana to read. This step can be achieved after the fact that very good results. All business groups will require an exception panel, for example, payment system business groups; it was probably about 10 different small services [4].

When the alarm system found that the payment system has a variety of problems, our first step was to look at the abnormal panel to find through the log, according to the timeline, make sure that if there are new services get new log in the near future or error log was different. And it will be based on different exception/java stack out to do count, which also brings great help to the analysis, you can also write a lot of complex query.

The first version was very simple (as shown in **Figure 2**), we only applied it to one or two very critical system. We made a comparison after this system is done, the average time to resolve the fault from the previous 50 minutes to less than 30 minutes. Our online systems typically took 5 to 10 minutes at the earliest to have

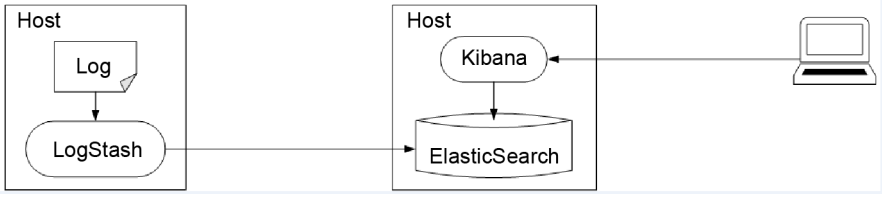


**Figure 2.** (Version 1).

a non-critical alarm. If we can quickly find out where the problem was, solve the problem, such as a simple rollback operation on hundreds of machines it will take 5 to 10 minutes, the real time left to determine the problem under the log was only a short period of 5 to 10 minutes.

If there was not an abnormal panel can see all the information, such as whether there were any server's anomalies more than other servers, whether there was an anomaly sudden appeared many times today, or whether there was a server appeared many anomalies Today. And it will be at least take us 20 to 30 minutes to see the log manually.

The first version got a few problems, first, the Logstash Agent's maintenance was not very good, Logstash was based on Ruby, it will have dozens of megabytes of memory eaten by jvm when it started, we wanted to avoid every machine from running a Logstash. And we found Logstash was very unstable during the process we use it, sometimes inexplicable to die and it also needed a daemon to protect it.

The second problem was Log standardization, also a very troublesome problem.

The first problem with Logstash Agent was solved by introducing Kafka, which does not require an agent on every host after the introduction of Kafka [5].

We get an internal SRE team to maintain Kafka, kafka was very cheap and do not need to spend money to maintain. As for log standardization, we spent a lot of time to research, 99% of the services were java service in LinkedIn, there were more than 15 kinds of logs, most important was the access log and application log. One thing we did was writing directly to Kafka via the java Container logger normalization. Some procedures directly wrote in kafka, not on the disk, some procedures have to write to the disk inside, this was configurable.

These were rollout through our standard container to all services. Developers do not have to control anything, as long as the process of writing logger, info/logger, error, the information will be directly into the Kafka. For the program log, the default warning above the level of access to Kafka, can be controlled through the jmx online. For the access log, we had 10% of the sample, can be dynamically controlled through the ATS entrance [6].

As shown in **Figure 3**, this was the second version, you can see in the production environment of the Java service side, Host has no Logstash, all the log was directly written in Kafka, Logstash consume these logs from Kafka and write inside ElasticSearch [7].

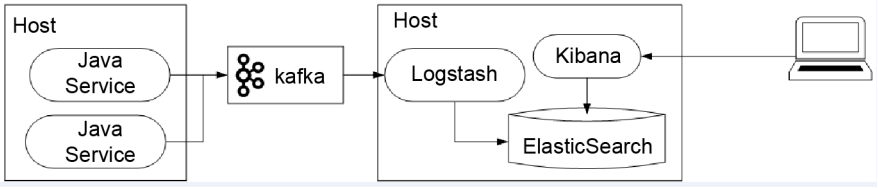


**Figure 3.** (Version 2).

The second version also got some problems, for example, a service problem will affect the entire ELK cluster. We met such a situation, a team wrote a new service, which defined all the level of logs as errors, the whole ElasticSearch was down by it. In many cases there will be other problems like network saturation.

Solutions: Very simple, the second step was to split it to optimize: The decision we made to solve the second version's problem was to split it to optimize: First, split ELK cluster by business functions, for example, payment system, money-related system use a same cluster; Systems landing with the user-related, security-related use a cluster; Next, separately Run Logstash and ElasticSearch. ElasticSearch was a disk-intensive operation, Logstash was a CPU-intensive operation, we put them on a same physical machine, but found the influence between each was quite big, so we decided to use Logstash mixed with other systems, separately from ElasticSearch [8].

For each Kafka topic, the number of Logstash was not less than the number of topic partitions. We had more than 500 services; each service will produce two Kafka topics: access log and program log. When Logastash produce Kafka, if the number of consumers less than the partition numbers, it will trigger a hidden vulnerability of Kafka, resulting in uneven Kafka partition, and appeared all kinds of strange problems. Our proposal was that under normal circumstances, each topic has four to eight partitions, and then set the appropriate number of Logstashs according to the specific circumstances.

According to business needs, we split out about 20 of this same cluster, as shown in **Figure 4**, in this version, there were also some problems. The First problem was querying from cross-business clusters. Although a problem can be found in a business group under normal circumstances, but there were still many cases we have to find out the problem cross to other clusters inside.

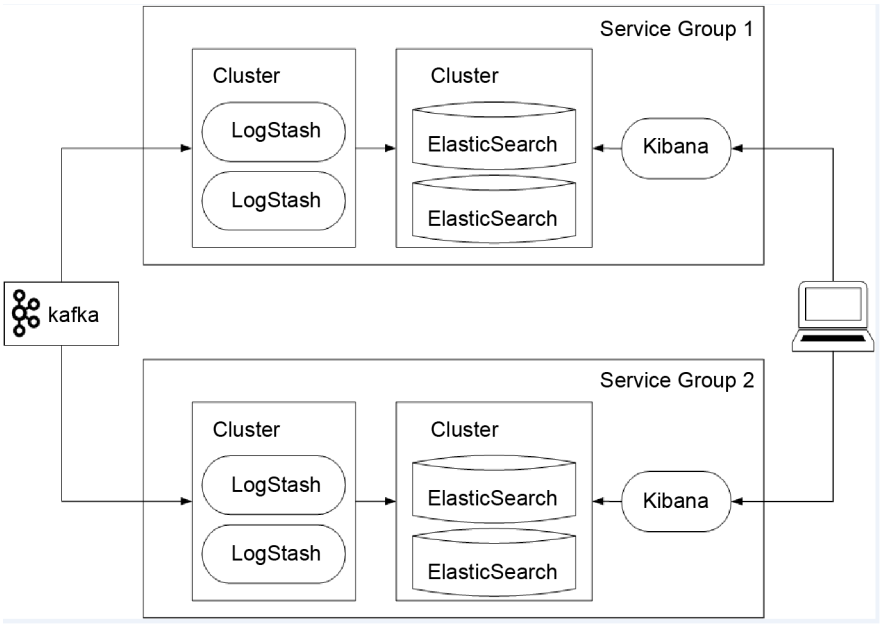


**Figure 4.** (Version 3).

Second, do not make the ElasticSearch clusters cross the data center, it was very poor and simply do not run up. Even if the two data centers were very close, especially when the amount of data comes up, there will be some very strange problems. The ElasticSearch cluster, which had a very large data volume, will require it to be all in one rack, and if there was a server in another rack, there would be a timeout problem.

In order to solve the problem just said, we introduced the Tribe, with the down feeling can be used, but this is obviously not a function they support. Tribe is a good idea, but it does not support tiered, we need two different Tribes, the first to be able to cross-business group, but also across the data center.

The best situation was to make a hierarchical structure, the data center in the outermost, business group in the innermost, however, as for the design concept, it was another one, so it did not make sense. In a data center, there will be a Tribe across all of the business groups. There will also be a Tribe across the data center for the same business group. There were two different types of Tribe that can be queried [9].

As shown in **Figure 5**, most basic functions were achieved in this version. We probable spent a half and a month to put more than 500 services into kafka and found that consume can't keep up, we encountered a performance bottleneck. ElasticSearch with more than 50 or 100 servers, will encountered many such bottlenecks. We spent almost three months doing various performance tuning.

This step was the last step, first, understand what our business logic was, the thing we were going to do was very clear, very single, was the need for real-time, or quasi-real-time log to do online trouble shooting, basically do not use the data before 14 Days. There was no time to consider the problems a few months ago unless today's problems are solved. The actual business status was most log queries

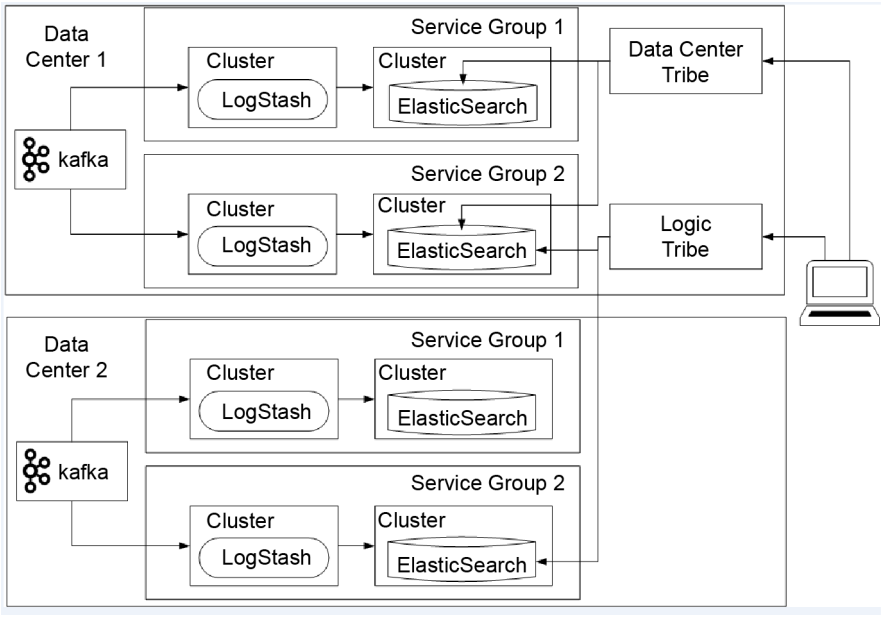


**Figure 5.** (Version 4).

up within 24 hours, very little log queries up before 14 days. The query speed requirements should be within 15 seconds, if more than 30 seconds, it would be timeout. The index speed was acceptable for 30 seconds or less but triggers an alarm while more than 5 minutes.

So how to solve the questions we met in version 4? As shown in **Figure 6**, we made improvements by using hot and cold partition to solve this question in version 5. We tested a lot of different hardwares, and finally selected SSD as thermal index on the cluster which has very important and very large amount of data, all indexes were on the SSD Machine within 24 hours, moved to the cold cluster if more than 24 hours. After doing this, the system became more stable, and function was also normal, internal system would retain 7 to 14 days of data according to needs.

After this step, we extended it to the entire company LinkedIn, the next day I received a phone call from Ministry of Justice and Security. When we did a very easy system, we showed all the logs in front of everyone. If we can easily search out 400 million users of the user name, password, mail, this thing was very serious, so we made a few adjustments [10].

The first was to scan all the ES regularly, according to a specific keyword to prevent sensitive information into the log, if in, alarm immediately. There was also the issue of user privacy. All ElasticSearch records were also sent to Kafka, where access to sensitive business units was isolated and all access logs were regularly reviewed. Our approach was to add an Nginx for access control can be done through the nginx, and there was a scanning process regularly scanning a variety of keywords while all the user access logs back to Kafka.

## 4. Results and Evaluation

This was the state of our production system, as shown in **Figure 7**, there were more than 20 modules for different business ELK clusters, 1000 + servers, mainly ElasticSearch. We can search the production system log we want on this side within 1 minute of, all the log reserves 7 to 14 days. There were now about 50 billion index files, 500 to 800 T, and this system can normally work when pushed to 1500 to 2000 T.

Because we had more than 500 services, more than 20 clusters, there is was no way to maintain so many clusters, so each business module SRE team needs to

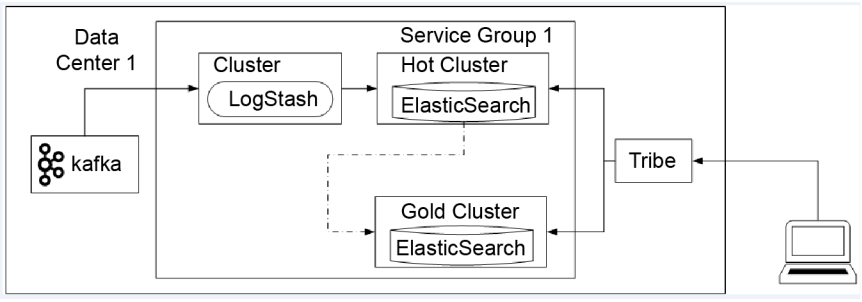


**Figure 6.** (Version 5).

maintain their own ElasticSerach cluster. The Virtual Team mode ensures that the ELK is up-to-date. There was also another critical point, you needed to ensure that your ES would not be accessed by users who had no permission. It does not accept SSL connections by default. You can use SearchGuard or Sheild to solve this problem.

I would like to focus on sampling, this is more interesting, but also a more general way. Sampling method is 10% + specific users, why did we sample like this? As shown in Figure 8, we had transferred a different proportion, and found that does not affect, if there is a problem, it can be found through 10% of the sample. Why were user specific users? Most of the time, there were some active users often give you the error, gave you feedback, you needed to see in what happened in time.

There were only about a few percent of the power users on the site are very active, did a lot of things, we needed to add their logs in the sample. All the user requests came into the data center through the Apache Traffic Server, in which it would visit Lix, asking whether to label the current user, if labeled, then put the label inside Invocation Context.

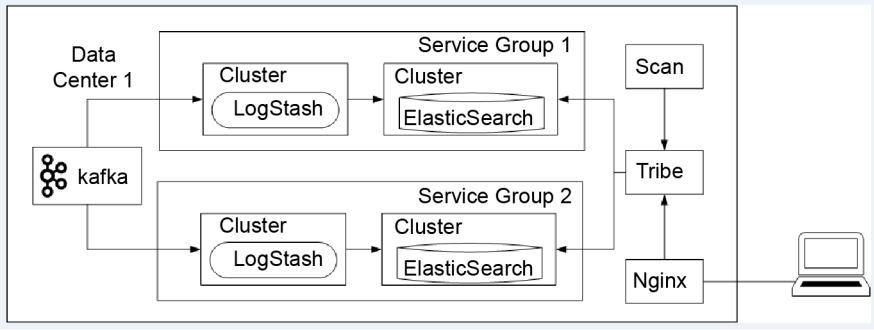


Figure 7. (Final version).

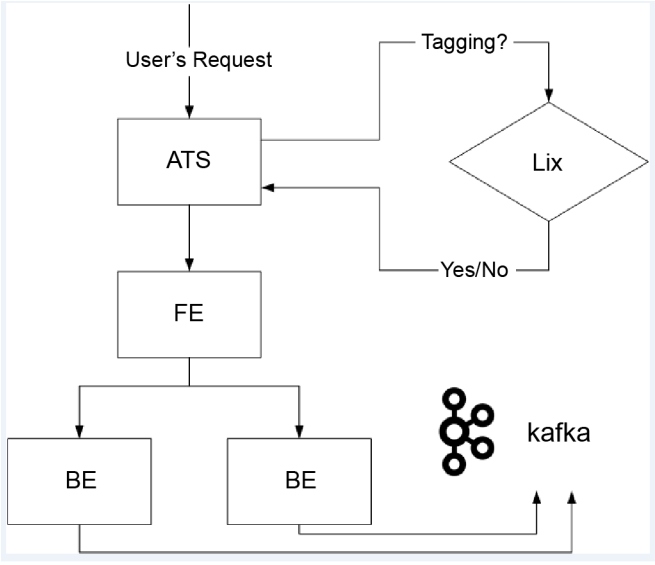


Figure 8. User request and process flow.

From the front to the background, as long as all the servers the touch to the request, we would see the request ID in the Invocation Context. Got the request ID through the java container by default, and made a sample by sending the log of that visit to Kafka.

This way had two advantages, that was, if there was any problem, we only needed to put his member ID or request ID on top layer of the Tribe, the system would appear all the service logs about this request. We can find out the problem at once at 99.9% of the situation [11].

## 5. Conclusions

Efficient log analysis system can provide effective and efficient help for engineers' daily development activities, it can help the engineers find the problem and solve it in time, through the continuous exploration, we finally found an efficient log analysis system based on kafka and ElasticSearch.

After our system was finished, we can easily found that it do have a lot of advantages for our team's daily development activities. This system not only significantly improved the development efficiency of our team of engineers, but also let us find a lot of unexpected surprises, so that our development process more standardized, more development process clear and thorough, in the face of urgent problems, we can solve faster and more efficient, and, a lot of potential problems in our development process has been resolved.

All in all, the log analysis system was for our development activities, how to find a suitable log analysis system for their own team's development activities was based on the specific circumstances of each team.

With the continuous development of computer technology, we believe that there will be many more excellent tools in the future, log analysis system will be more intelligent and humane, computer science is a fascinating subject, and one of the important features is that you can use the computer to complete many of the original impossible tasks. Let us explore the future of computers, explore the unknowns possible, and create more useful value.

## References

[1] Bai, J. and Guo, H.B. (2014) Software Integration Research of Large-Scale Logs Real-Time Search Basedon ElasticSearch. *Journal of Jilin Normal University* (*Natural Science Edition*), **2014**, No. 2.

[2] Kreps, J., Narkhede, N. and Rao, J. (2011) Kafka: A Distributed Messaging System for Log Processing. http://notes.stephenholiday.com/Kafka.pdf

[3] Gormley, C. and Tong, Z. (2015) Elasticsearch: The Definitive Guide. O'Reilly Media, Inc., Sebastopol.

[4] Bagnasco, S. and Berzano, D. (2015) Monitoring of IaaS and Scientific Applications on the Cloud Using the Elasticsearch Ecosystem. *Journal of Physics: Conference Series*, **608**, No. 1. https://doi.org/10.1088/1742-6596/608/1/012016

[5] Narkhede, N., Shapira, G. and Palino, T. (2017) Kafka: The Definitive Guide. O'Reilly Media, Inc., Sebastopol.

[6] Jiang, K., Feng, J., Tang, Z.X. and Wang, C. (2015) The Metadata Searching and

Sharing Platform Based on ElasticSearch. *Computer and Modernization*, No. 2.

[7] Xu, D.H. (2014) Application Research in Vehicle License Plate Recognition System Based on Elaticsearch. *Computer Era*, No. 12.

[8] Lei, X.F., Wang, Z. and He, Y.Z. (2015) Log Real-Time Management Scheme Based on International Workshop on Materials Engineering and Computer Sciences. *The International MultiConference of Engineers and Computer Scientists* 2015, Hong Kong, 18-20 March 2015.

[9] Baker, J., *et al.* (2011) Megastore: Providing Scalable, Highly Available Storage for Interactive Services. *Fifth Biennial Conference on Innovative Data Systems Research*, Asilomar, January 9-12 2011.

[10] Nguyen, C.N., Kim, J.-S. and Hwang, S. (2016) KOHA: Building a Kafka-Based Distributed Queue System on the Fly in a Hadoop Cluster. *IEEE* 1*st International Workshops on Foundations and Applications of Self* Systems* (*FAS*W*), Augsburg, 12-16 September 2016, 48-53. https://doi.org/10.1109/FAS-W.2016.23

[11] Chen, J.J. and Huang, G.F. (2015) Reconstruct Library Search Engine Based on Elasticsearch. *Information Research*, No. 11.