Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 103 (2017) 536 – 542

Procedia Computer Science

XIIth International Symposium «Intelligent Systems», INTELS'16, 5-7 October 2016, Moscow, Russia

The Big Data approach to collecting and analyzing traffic data in large scale networks

L.U. Laboshin*, A.A. Lukashin, V.S. Zaborovsky

Peter the Great Saint-Petersburg Polytechnic University, 29, Polytechnicheskaya street,, St.Petersburg, 195251, Russia

Abstract

Information processing is currently one of the most vital tasks. With the growth and development of information and telecommunication technologies increased the volume of data transmitted over the Internet. Simultaneously with the processing large amount of information raises the question of its protection. The given paper proposes a distributed cloud-computing framework based on the Big Data approach where both storage and computing resources can be scaled out to collect and process traffic from a large-scale network in a reasonable time.

Keywords: Network traffic analysis; actor model; mapreduce; DFS; akka; scala; netgraph

1. Introduction

Currently, computer technology continues to evolve. The increase of computational power leads to a change of content, for example, recent progress in display technology has resulted in the ultra high-definition video. Actively developing the internet of things (IoT) means more devices utilizing a data networks. Most of these changes lead to an increase in the volume of information transmitted over networks. Throughput is constantly increasing in order to provide an effective service. Currently, the technology GigabitEthernet (1 Gbit/s) has almost completely replaced FastEthernet (100Mbit/s) on personal computers. In ISP networks is widely used 10 Gbit/s, is the transition to 40 Gbit/s and more. The task of information security is still relevant. Malicious software and attacks on information

* Corresponding author.
E-mail address: laboshinl@neva.ru

systems can have a major impact not only on the commercial companies but also for public service. And with the growth of information influence will continue to increase. There are many tools for information security: antivirus software, firewalls, intrusion detection system (IDS), and others. And if the tools are working in real-time continue to evolve along with the threats to information security, needing to analyze previously accumulated data, in particular, network intrusion detection system, should be considered separately.

As already noted, such funds shall require the prior accumulation of a data packet. If you install network IDS in a channel with a bandwidth of 1Gbps it turns out that every minute it is necessary to analyze up to 15 GB of data. In the case of channel 10Gbps, this amount rises up to 150 GB of data. And if there is a need to analyze the traffic for an hour, then the volume is 900 and 9000 Gigabytes respectively. Storing and processing it becomes challenging. The analysis time of such a volume of data makes it difficult to quickly identify an attack or a policy violation access. These problems can be solved using new approaches for data storing and processing relating to the "big data" computing task class. The most famous approach is the use of distributed filesystems to store very large amounts of data and use computation models such as MapReduce or BSP for analysis. There are ready software and hardware platforms for these purposes. However, these tools are not suited for collecting and analyzing network traffic.

In this paper, we propose a distributed framework for collecting and analyzing network traffic that provides:

1. Load scalability: The ability to easily expand and contract its resource pool to accommodate heavier or lighter loads or a number of inputs;
2. Reliable storage capacity to handle large amounts of data;
3. Provides analysis of the entire amount of data in reasonable time.

On the basis of this framework, it is possible to build a new type of intrusion detection system which is able to handle very large amounts of data. Recent development in cloud computing enables the ability of such system to scale out if necessary, for example, to reduce the analysis time, by adding the desired number of compute nodes. Due to architecture scalability, the system will be flexible enough to meet the continuous growth of data volumes.

The aim of this work is to develop a system architecture able to retrieve, store, and process network traffic of high-speed data channels in a distributed manner.

2. Recent work

Various tools such as Wireshark network protocol analyzer or CoralReef are available for monitoring, measurement and analysis passive Internet traffic data. However, most of these tools designed to run on single high-performance server which is not capable of handling huge amount of traffic data captured at very high-speed links. Lee et al. [1, 2] proposes a Hadoop-based traffic analysis system with a limitation of the only collecting periodic statistic like total traffic and host/port count. RIPE library does not consider the parallel-processing capability of reading packet records from distributed filesystem which leads to performance degradation. Vierira et al. [3, 4] have developed a parallel packet processing system only for JXTA-based applications. Jethoe [5] in his paper proposed a method for detected DDOS attacks using Hadoop. In our papers [6, 7] we proposed a MapReduce approach to performing deep packet inspection (DPI) and application-aware network content inspection with TCP-flow extraction on large traffic traces.

3. Scalable data processing technologies

3.1. MapReduce

MapReduce is the model of parallel programming developed by the Google company as a solution to the information search tasks. Main opportunities of this technology are

- automatic multi-sequencing of the task on a cluster of standard architecture servers;
- load balancing between cluster nodes;
- protection against failures of the equipment by restarting of the task on the other node of a cluster;
- the distributed filesystem for data storage on internal disks of servers of a cluster.

These opportunities are reached due to use of the single algorithm – MapReduce which is based on list processing and consists of two steps: Map and Reduce. Map receives the list of couples <key,value> on an input and gives out transformed list of couples <key, value>. Reduce receives the list of couples <key, value> with an identical key on an input and produces only one value [Fig. 1]. Processing of the elements can be executed independently of each other in a parallel manner. Algorithm contains only one phase of communication. After completion of a Map step values with the same key are sent to single cluster node where they are processed on Reduce step. The user only defines the Map and Reduce functions processing one value, and parallel execution over the large volume of data is then done by MapReduce system. In MapReduce cluster each node is used at the same time for both data storage and computation. The MapReduce scheduler tries to distribute jobs for those nodes of a cluster where the data is stored. Such distribution is known as data locality principle and allows significantly increase productivity.

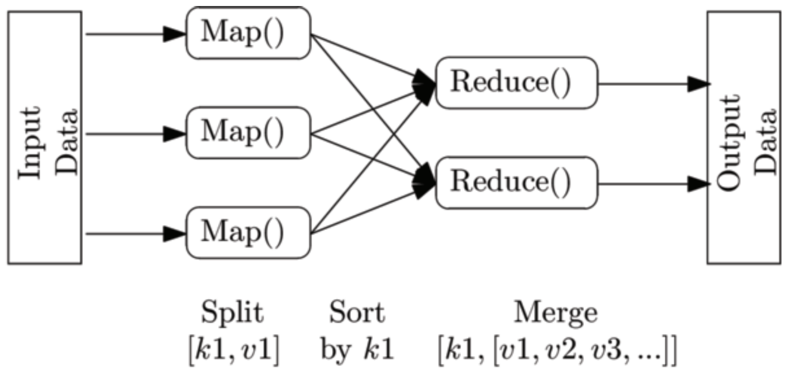


Fig. 1 MapReduce programming model.

Since MapReduce applications mostly used to process large sets of text data such as log files or web documents, main input file format is text format where input treats each line as a <key, value> pair: the key is the offset in the file and the value is the contents of the line. LibPcap binary file format (discussed in detail in section 3) has no mark between packet records like carriage return in text file. Input file is chunked into fixed size blocks a packet record is usually located across two neighbouring filesystem blocks. Our sliding window algorithm we have proposed in [6] allows detecting the beginning of packet record in the filesystem chunk (Fig. 2) and makes it possible to use DFS such as CEPH as underlying storage for MapReduce jobs on LibPcap files.

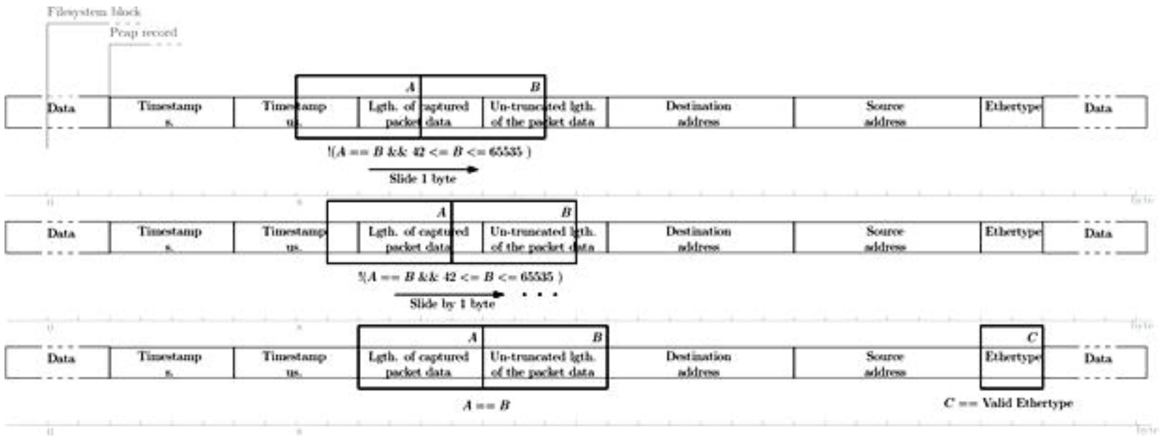


Fig. 2 Pcap packet record detection algorithm for distributed filesystem.

3.2. Actor model

The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation[3]. The Actor model consists of independent communicating computational agents that share nothing and respond to messages by:

1. Generating more messages;
2. Modifying their behaviour;
3. Creating other actors.

Actors communicate with each other using asynchronous communication (Fig.3). Actors are stateful, and modify their behaviour in response to messages. The creation of actors on-the-fly is a key element in supporting dynamic scalability and fault tolerance.

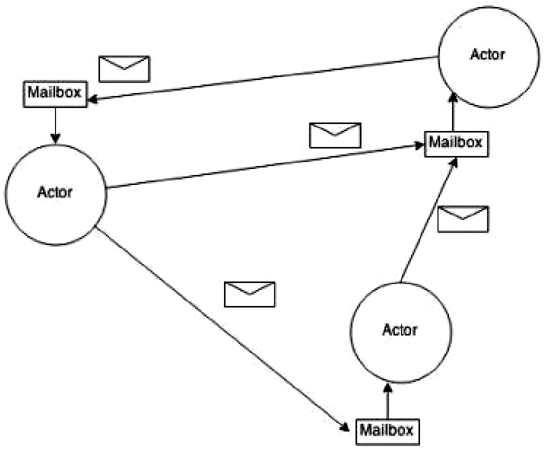


Fig. 3 An actor model of computations.

As was mentioned by Mohindra at al. [9] for task parallelism, we needed a framework that provided a unified local and distributed programming model – from threads to processes to data centers. Increasingly, we need to support heterogeneous platforms (multi-cores, GPUs, function accelerators, etc.) The actor model of concurrent computation satisfied these constraints.

4. Collecting network traffic with graph based networking subsystem

The important part of the system is the ability to copy traffic in high-speed networks to the file store without making any delays. It is also important to hide the fact of copying traffic making it invisible to other components. There are different mechanisms of implementing this capability with both physical devices and software methods.

The Most effective way is to use a specialized physical device, but in order to run our system on a commodity hardware, we will use Netgraph [10] subsystem of FreeBSD operating system for transparent traffic mirroring.

Netgraph is a modular networking subsystem of the FreeBSD kernel, based on the graphs. Netgraph builds a graph from the nodes of different types. Each node has several inputs and outputs and allows a user to perform certain actions on a package passing through it. Due to the fact that Netgraph is a kernel subsystem, it provides a high performance.

Our implementation uses a graph consisting of 6 vertices:

• Three nodes of type *ng_ether* associated with real network interfaces;
• Single *ng_tee* node type that copies packages;
• Two *ng_ksocket* type nodes that implement BSD sockets.

The replication and further traffic transmission occur as follows (Fig.2):

1. An Ethernet packet arrives on interface *em0* (external interface) or *em1* (internal interface). Using the hook (the edge) *lower*, they are transmitted to the *ng_tee* node via hook *left* or *right*.
2. Packages that came with the hook left to pass on to the hook right and copied to left2right, the packages came to the right hook are on the hook is copied to left and right2left. Thus incoming and outgoing traffic are divided.
3. Nodes of *ng_ksocket* type are used for interprocess communication by creating BSD sockets. In our implementation we use "*local*" sockets type, the transmission is carried out using datagrams. Other types can also be used and. Two *ng_ksocket* nodes are connected to the *left2right* and *right2left* hooks and configured to transfer packets came from *ng_tee* to the specified socket.

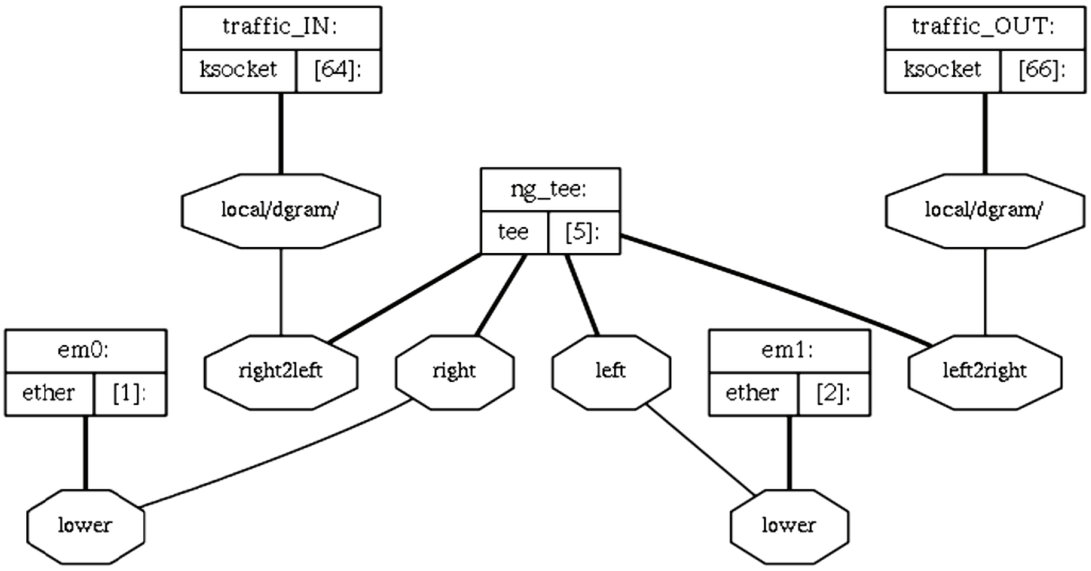


Fig. 4 Diagram of the traffic mirroring subsystem.

The data passed from the external network interface via ng_tee node to the internal network interface and back directly and this a tunnel is invisible either from external or internal network. Access exists only from the used network to receive the copied data.

Therefore, all passing through ng_tee packets are copied and sent to the nodes that implement interprocess communication through sockets. It remains only to write a program that will read the data from there. There are several file formats specially designed for this purpose. One of the most widely used is the Packet CAPture format, "PCAP". It is fairly easy to implement, and there various tools such as tcpdump, windump, wireshark, and others utilize this format. The PCAP format is a part of LibPcap library written in C. It is designed for storing network traffic dumps and implements functions such as the separation of one package from another within a snapshot, and record timestamps with nanosecond precision. File format PCAP is divided as follows (Fig 4).

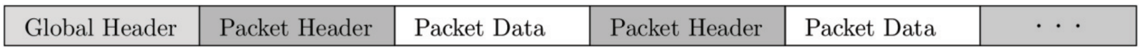


Fig. 5 LibPcap file structure.

5. Design of the system

Akka [11] is a toolkit and runtime based on actor model for building highly concurrent, distributed, and resilient message-driven applications on the JVM. We have implemented a Scala and Akka based library to run map-reduce jobs with all "map" and "reduce" tasks running in parallel, using Akka actors in Scala.

The client actor can work in 3 modes. To process the file stored in the local filesystem of the computer that it is running, the file from a remote server specifying the URL, or a data stream from packet sniffer, as described in the previous paragraph. The file is divided into blocks of a specified size and distributed among the filesystem actors using Akka Stream.

Filesystem actor saves received chunk to cluster node's filesystem. Then, with the method proposed in the article[6] the beginning of the first LibPcap packet record is detected. For each block created a database record with the following structure (Fig. 7).

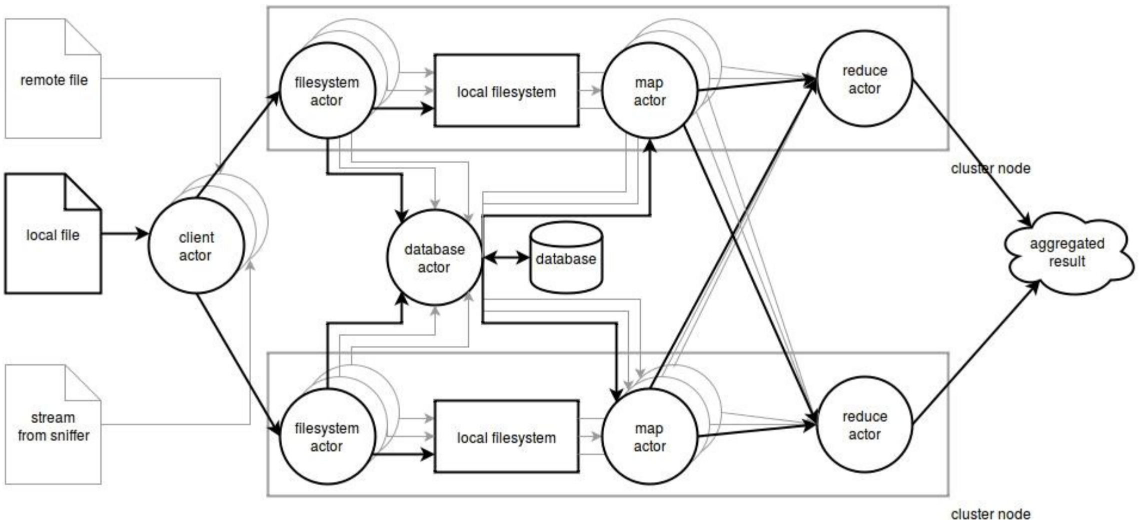
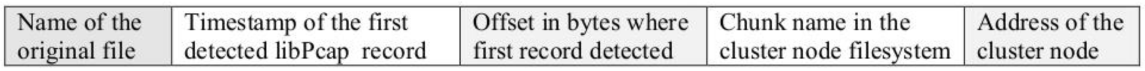


Fig. 6. Schematic view of the framework.

When the map task is started, each node of the cluster makes a request to the database and gets the list of blocks lying in its local filesystem and then processes them independently. If a few bytes are missed to complete the last packet record performed a database query on the location of the next unit in the sorted table, and a message is sent asking for the necessary number of bytes to the specified address of Akka cluster node. This ensures the principle of data locality.



Name of the original file	Timestamp of the first detected libPcap record	Offset in bytes where first record detected	Chunk name in the cluster node filesystem	Address of the cluster node

Fig. 7. Database record structure.

6. Conclusion

Within our work, we offered a web traffic evaluation technique that is centred on Map-Reduce paradigm. We suggest making use of Akka cluster within cloud infrastructures and dynamically reconfigure depend on the quantity of information needed to be processed. For collecting network traffic we have also build a module for a FreeBSD networking subsystem that makes possible to intercept and log traffic that passes over a high-speed digital network in a hidden way when no changes to topology and configuration of other devices needed. Future work is to develop prototype queries to network traffic dumps which include expressions for various network protocol fields and for transmitted data signatures to evaluate proposed systems performance and scalability.

Acknowledgements

The applied scientific investigations are carried out with the financial support of the Russian Federation represented by the Russian Ministry of Education and Science. The unique identifier of the applied scientific investigations is RFMEFI57814X0046.

References

1. Lee Y, Kang W, Lee Y. A hadoop-based packet trace processing tool. In International Workshop on Traffic Monitoring and Analysis; 2011. pp 51-63.
2. Lee Y, Lee Y. Toward scalable internet traffic measurement and analysis with hadoop. ACM SIGCOMM Computer Communication Review; 2013. pp 5-13.
3. Vieira T, Soares P, Machado M, Assad R, Garcia, V. Evaluating performance of distributed systems with mapreduce and network traffic analysis. *Computing;* 2012. pp 4-6.
4. Vieira T, Soares P, Machado M, Assad R, Garcia V. Measuring Distributed Applications through MapReduce and Traffic Analysis. In ICPADS; 2012. pp. 704-705.
5. Jethoe S. Detecting DDOS attacks using distributed processing frameworks.
6. Lukashin A, Laboshin L, Zaborovsky V, Mulukha V. Distributed packet trace processing method for information security analysis. *In International Conference on Next Generation Wired/Wireless Networking* (pp. 535-543). Springer International Publishing.
7. Laboshin L, Lukashin A, Zaborovsky V. Applying MapReduce and network traffic; 2015.
8. Hewitt C .Actor model of computation: scalable robust information systems. analysis to control access to information resources. *St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunication and Control Systems*; 2010. pp. 34-40.
9. Mohindra S, Hook D, Prout A, Sanh A, Tran A,Yee C. Big Data Analysis using Distributed Actors Framework.
10. Elischer J & Cobbs A. The Netgraph networking system. Technical report, Whistle Communications. 2000. http://www.elischer.com/netgraph/ (Accessed 07 November 2016)
11. Bonér, J, Klang V, Kuhn R, Nordwal P, Antonsson B, Varga E. Akka scala documentation. *Tech. rep., Typesafe Inc.* 2014.