BATCH AND STREAM BIG DATA PROCESSING PLATFORM: CASE OF NETWORK TRAFFIC ANALYSIS

S.D. Belov [1,2 a], **I.S. Kadochnikov** [1,2 b], **V.V. Korenkov** [1,2 c], **R.N. Semenov** [1,2 d], **P.V. Zrelov** [1,2 e]

[1] *Joint Institute for Nuclear Research, 6 Joliot-Curie St, Dubna, Moscow Region, 141980, Russia*

[2] *Plekhanov Russian University of Economics, Stremyanny lane, 36, Moscow, 117997, Russia*

E-mail: [a] belov@jinr.ru, [b] kadivas@jinr.ru, [c] korenkov@jinr.ru, [d] roman@jinr.ru, [e] zrelov@jinr.ru

While scientific data is usually processed with specialized software, more and more institutions make use of new developments in Big Data for processing all the ancillary data that accompany experimental research and scientific computing. One common use case is monitoring of the experiments and the IT infrastructure, but a well-realized Big Data platform can be used to extract useful knowledge from many available data streams. This work describes our efforts to lay out and implement a general-purpose Big Data processing framework capable of both batch and stream operation for use in our institute. As the core, we chose Apache Spark running on cluster resources controlled by Apache Mesos. To demonstrate the functionality of the platform in both modes, we chose a sample problem of local network packet analysis. TCP and IP header fields were aggregated by the source IP and converted to the AGgregate and Mode (AGM) form. After encoding into numerical features, clusterization and principal component analysis was performed. Thus, it was shown that the platform works and can be used for batch processing. The same algorithm will be implemented for online network packet analysis to test the stream processing capability of the platform.

Keywords: traffic analysis, Big Data, stream processing

1. Project goals

Big Data tools and methods provide an increasingly attractive opportunity to extract new knowledge from arbitrary data sources. Scientific experiments are engineered to generate rigidly structured datasets that are processed using specialized tools. More general-purpose tools, such as those considered part of the “Hadoop ecosystem”, are more often used by the scientific institutions for processing and analysis of ancillary data, in particular for IT infrastructure management, experiment hardware monitoring, computational resource accounting, data management, etc [1, 2].

We aimed to build a prototype Big Data platform using modern frameworks and tools. In addition to batch processing and analysis, it should be capable of stream processing and visualizing analysis results. To test the platform, we use as an example a problem of network traffic analysis by methods proposed for darknet packet analysis. That is, extracting features from TCP and IP metadata to categorize IP addresses based on aggregated network traffic.

2. Platform structure

The core of the platform is the Apache Mesos cluster that manages computing resources [3]. It is designed to run different frameworks, including Apache Spark [4]. We chose to configure the executor containers to use prebuilt Docker images with the desired libraries. This allowed to include all desired libraries and dependencies for Spark, including Python user-defined functions support, using the Conda package manager for Python packages.

Other services that constitute the platform are run more directly using Docker in swarm mode [5]. As resources are limited, the swarm only contains one node for now. The key service running on docker is Apache Zookeeper for distributed coordination of other services [6]. Mesos uses Zookeeper to share information and build the cluster. It also serves as the entry point for Spark clients (drivers) to start execution on the cluster. To allow users to run analysis and visualization jobs, the platform includes the Zeppelin notebook-style web interface [7], running as a docker service with the same base dependency image as the Spark executors to ensure version compatibility.

All the components required for batch mode operation of the platform are now integrated and working. For storage of bulk results the MooseFS distributed storage system is mounted to all processing nodes of the cluster [8]. The sample problem of network packet analysis was implemented to test the functioning and performance of the platform in batch processing mode.

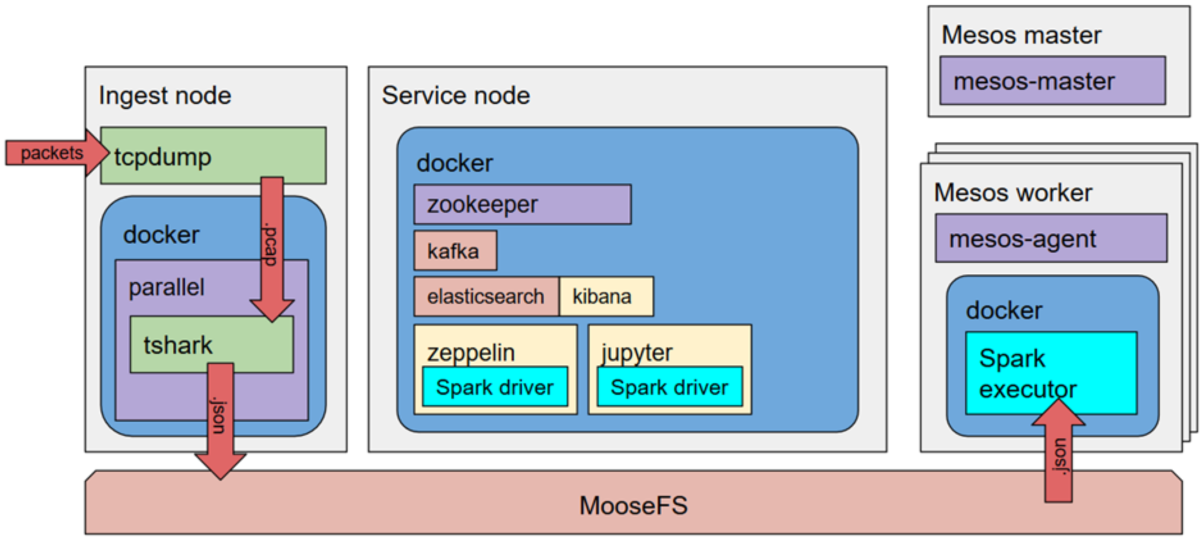

Fig.1. Platform nodes and component services

Other services are required to implement stream processing, most important of which is the Apache Kafka event streaming platform [9]. It is integrated with the same Zookeeper instance for distributed configuration and metadata exchange. For processed result storage and exploratory visualization, we intend to run Elasticsearch document database with a Kibana front-end. For time series intermediate results – a time series database should be included, there are many options [10]. With the development of Zeppelin slowing down, it is prudent to provide an alternative notebook interface, such as Jupyter. The last key service we expect to need for the platform is a stream dispatching, formatting and prefiltering service, such as Apache Flume or Apache Nifi. In our experiments with Flume, we found the compatibility with modern versions of our platform insufficient and are now looking for a suitable alternative. In a similar project using Nifi for dataflow management was proposed and evaluated [11]. The final machine learning and analysis we expect to perform with minimal modification with the same methods as batch analysis, using the Spark Streaming mode [12].

The platform hardware consisted of six physical Supermicro servers with double 4-core Intel Xeon E5420 CPUs, 16 Gb of DDR2 RAM and 1 Tb local hard disk running Centos 7.8 Linux operating systems. The nodes were connected over gigabit Ethernet with the ingest node receiving network packets for processing over a dedicated second Ethernet port.

3. Traffic analysis method

To test the capability of our Big data processing framework, a suitable data streaming problem was required. We selected a task of ingesting, processing and analysis of network packets from the local network of our Institute (JINR). If successful, the results could be used for anomaly detection and possible network security applications. We used an approach for aggregating packet header metadata that is often employed for darknet traffic analysis [13].

3.1. Packet capture and parsing

All the input data for this project arrive at the separate ingest node of the platform in the form of raw network packets. They are cloned and forwarded from the router from one local subnetwork by the JINR networking team. The traffic from one 256-address subnetwork creates enough data to test both streaming and batch operation of the platform under sufficient load without overwhelming available resources.

For the initial testing of batch processing, a dataset of raw packet headers was captured for 24 hours with tcpdump and default snapshot length of 262144 bytes. The pcap files were split every minute for a total of 1440 files containing 383.3 Gb. Packet header data was then extracted in json format by tshark, the terminal executable of Wireshark. Wireshark contains many protocol dissectors, but for our chosen method only TCP and IP are necessary. The resulting json dataset contains only TCP and IP metadata, however dissection results contain redundant fields, json is text-based and not binary, and as a result the dataset after dissection grew to 2.2 Tb. However, it is easily mitigated by compressing the data on the fly by gzip, resulting in size reduction to 213 Gb. In order to use the latest version of tshark that supports json without having to build from source, it was installed and run in an Arch Linux docker container. For performance, we used GNU parallel to run 7 processing pipelines of tshark into gzip [14]. This packet parsing step took 8.3 hours.

As one can see, the packet parsing step is the most time-consuming, and while there is enough resources to perform packet ingestion in s tream mode (24 hours of data is processed in 8 hours), throughput is close to the limit and the method for parallel processing is not easily scalable. In the future to validate our platform’s stream processing capability, we will have to completely redesign this step.

3.2. AGgregate and Mode format

The basic AGgregate and Mode format is designed to allow statistical analysis of captured packets with darknet destinations by aggregating them into a 22-feature vector describing a given source IP address. It uses as input only 8 fields of each packet: source IP, destination IP, source TCP port, destination TCP port, TTL, protocol, TCP flags and packet length. Each field X is described by 3 aggregated features

- #X: the number of unique values of X

- M(X): the mode of X
- #pkts[M(X)]: the number of packets for which the field is equal to the mode

All the aggregation is done by source IP, and as a final feature the total packet count #pkts is added. For machine learning applications [13] propose a 29-parameter Numerical AGM format by dropping or encoding all non-quantitative parameters of AGM. The choice of features to be dropped is dictated by the nature of the dataset and problem. For similar reasons, we chose to drop the M(dst_ip) and all the protocol features and only consider TCP packets. M(src_port), M(dst_port) and M(flag) were encoded into 10, 10 and 16 dummy variables, respectively.

3.3. Spark bulk processing and analysis

Packet data was ingested from the distributed MooseFS filesystem in json format to be prefiltered and processed into parquet columnar files. Due to memory limitations of json support in spark, raw packet data was reprocessed by tshark to split larger pcap files into 100 Mb chunks and the parsing step was repeated with split files. The Zeppelin Spark interpreter was configured to start the Spark framework executors with 12 Gb memory and 1 CPU core to further mitigate the memory requirements of ingesting large json files. After filtering only TCP packets and projecting the dataset onto fields relevant for the Numeric AGM format, the processed dataset was written as parquet. This processing step produced 835 Mb of data in 3.5 hours.After restarting the Spark framework with executors having 12 Gb memory and 7 cores each, the final processing and machine learning step of the analysis was performed with SparkML [15]. Network packet data was aggregated by source IP into AGM format, the processing having taken just 10 minutes. The most frequently occurring mode of source port and destination port were computed to use for one-hot encoding these parameters into features (Fig. 2).

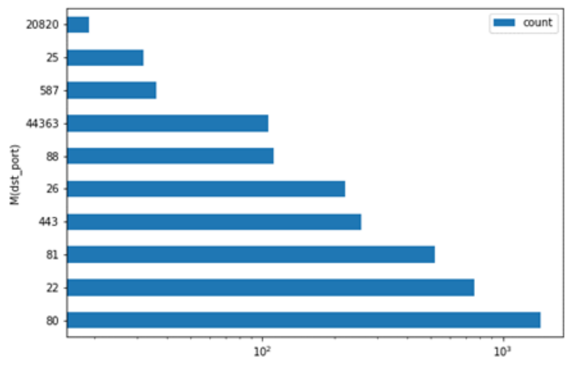

Figure 2a: number of addresses in the dataset with a given dst_port, logarithmic scale

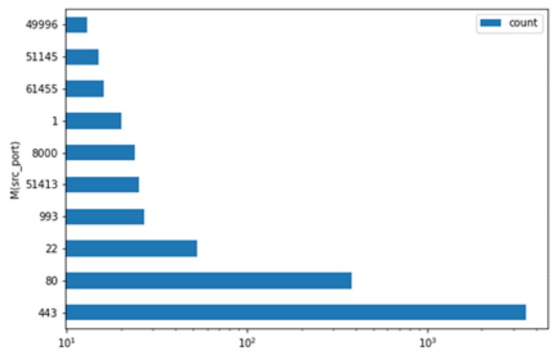

Figure 2b number of addresses in the dataset with a given src_port, logarithmic scale

The encoding of categorical features to produce the NAGM dataset was done with Spark ML pipelines, StringIndexer and OneHotEncoderEstimator models. The resulting numeric-only features were combined into a 51-component feature vector and scaled with a StandardScaler. As a demonstration of possible transformation and clustering, PCA and BisectingKMeans with k=4 were applied. The resulting projection on the two principal components is shown on Fig. 3.

It is apparent that the simple machine learning methods which were applied, showed no visibly interesting results in the limited dataset that was collected. However, we were able to show that our platform prototype can be used to replicate all steps of the analysis proposed in [13].

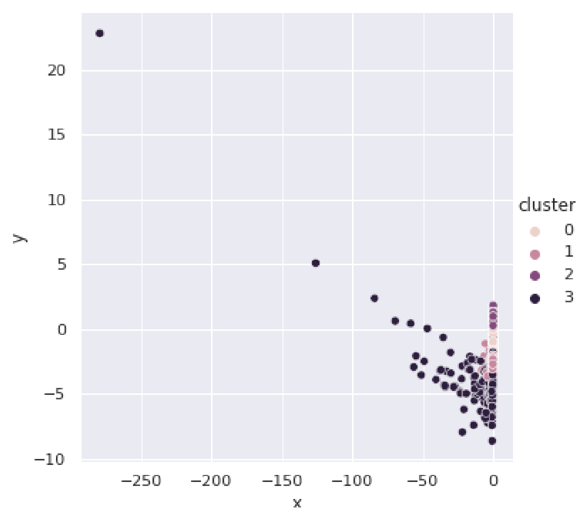	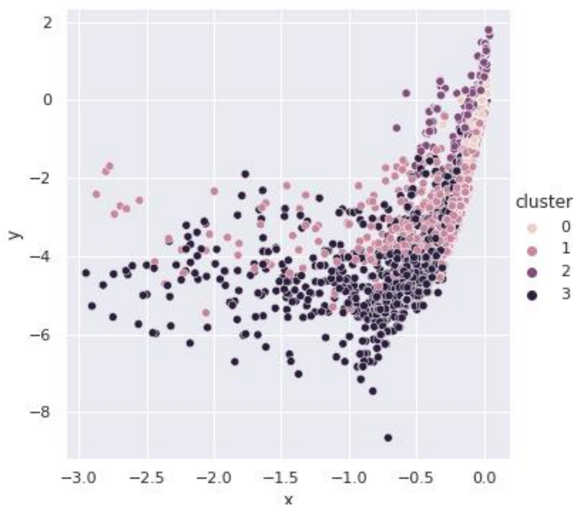
Figure 3a. IP address features projected as their 2 top principal components as x, y. Bisecting k-means for k=4 clusterization results	Figure 3b: IP address features projected as their 2 top principal components as x, y. Bisecting k-means for k=4 clusterization results), showing the more densely populated region of x>-3.

4. Conclusion

The Big Data processing and analysis platform was implemented, based on Apache Spark, Apache Mesos and Docker. Batch operation of the platform was shown by processing an existing method of network packet analysis as the example. The next steps in this project aim to run and test the streaming operation mode of the platform, to compare performance and usability with the batch processing mode.

The discovered major performance bottlenecks of packet parsing and json ingestion limit the practicality of application of this platform to larger-scale network traffic analysis. One possible solution is to integrate a stream pre-processing and filtering framework similar to Flume or Nifi to allow for easier horizontal scaling of both pcap -> json and json -> parquet transformation.

With more input data and better feature selection, and possibly more sophisticated machine learning models, we expect to extract more interesting knowledge at the end of network analysis process.

Acknowledgement

The study was carried out at the expense of the Russian Science Foundation grant (project No. 19-71-30008).

References

[1] Z. Baranowski *et al.*, “Evolution of the Hadoop Platform and Ecosystem for High Energy Physics,” *EPJ Web Conf.*, vol. 214, p. 04058, 2019, doi: 10.1051/epjconf/201921404058.

[2] A. Aimar *et al.*, “Unified Monitoring Architecture for IT and Grid Services,” *J. Phys. Conf. Ser.*, vol. 898, p. 092033, Oct. 2017, doi: 10.1088/1742-6596/898/9/092033.

[3] B. Hindman *et al.*, “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center,” p. 14.

[4] M. Armbrust *et al.*, "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, Melbourne, Victoria, Australia, 2015, pp. 1383–1394, doi: 10.1145/2723372.2742797.

[5] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, p. 2:2, Mar. 2014.

[6] P. Hunt, M. Konar, Y. Grid, F. Junqueira, B. Reed, and Y. Research, "ZooKeeper: Wait-free Coordination for Internet-scale Systems," *ATC USENIX*, vol. 8, Jun. 2010.

[7] Y. Cheng, F. C. Liu, S. Jing, W. Xu, and D. H. Chau, "Building Big Data Processing and Visualization Pipeline through Apache Zeppelin," in *Proceedings of the Practice and Experience on Advanced Research Computing*, New York, NY, USA, Jul. 2018, pp. 1–7, doi: 10.1145/3219104.3229288.

[8] J. Yu, W. Wu, and H. Li, "DMooseFS: Design and implementation of distributed files system with distributed metadata server," in *2012 IEEE Asia Pacific Cloud Computing Congress (APCloudCC)*, Nov. 2012, pp. 42–47, doi: 10.1109/APCloudCC.2012.6486509.

[9] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a Distributed Messaging System for Log Processing," p. 7.

[10] A. Bader, O. Kopp, and M. Falkenthal, *Survey and Comparison of Open Source Time Series Databases*. 2017.

[11] H. Isah and F. Zulkernine, "A Scalable and Robust Framework for Data Stream Ingestion," in *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, Dec. 2018, pp. 2900–2905, doi: 10.1109/BigData.2018.8622360.

[12] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," Nov. 2013, pp. 423–438, doi: 10.1145/2517349.2522737.

[13] R. Niranjana, V. A. Kumar, and S. Sheen, "Darknet Traffic Analysis and Classification Using Numerical AGM and Mean Shift Clustering Algorithm," *SN Comput. Sci.*, vol. 1, no. 1, p. 16, Aug. 2019, doi: 10.1007/s42979-019-0016-x.

[14] O. Tange, *GNU Parallel 20150322 ('Hellwig')*. Zenodo, 2015.

[15] X. Meng *et al.*, "MLlib: machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, Jan. 2016.