Low latency analytics for streaming traffic data with Apache Spark

Altti Ilari Maarala, Mika Rautiainen, Miikka Salmi, Susanna Pirttikangas and Jukka Riekki
Department of Computer Science and Engineering, University of Oulu, FInland
Email: {amaarala, tiainen, misalmi, msp, jpr}@ee.oulu.fi

***Abstract*—Demand for new efficient methods for processing large-scale heterogeneous data in real-time is growing. Currently, one key challenge in Big Data is performing low-latency analysis with real-time data. In vehicle traffic, continuous high speed data streams generate large data volumes. Harnessing new technologies is required to benefit from all the potential this data withholds. This work studies the state-of-the-art in distributed and parallel computing, storage, query and ingestion methods, and evaluates tools for periodical and real-time analysis of heterogeneous data. We also introduce a Big Data cloud platform with ingestion, analysis, storage and data query APIs to provide programmable environment for analytics system development and evaluation.**

I. Introduction

Digitalization has led to exponential information growth and brought forth the 3Vs of the Big Data paradigm where high velocity data streams pose one of the key challenges for real-time analysis and decision making. In many applications, the value of data decreases in proportion to the time that has passed since the data is produced. For example in traffic scenarios, a delay counted in minutes is inevitably too long when decisions about the quickest driving routes from a metropolitan traffic congestion have to be made immediately. Heterogeneity i.e. variety of the high velocity data sources sets another challenge for Big Data processing. The explosive growth of different data sources from sensors to social media messaging demands data to be processed and interpreted in varying data representations.

Typically, common large scale distributed and parallel data processing frameworks such as Apache Hadoop are designed for batch processing of static data sets. However, in recent years, new efficient technologies such as Apache Spark and Apache Storm have been developed for real-time analysis of dynamic data streams. This paper evaluates what are the best practices to perform real-time analysis of high velocity heterogeneous traffic data. We develop Apache Spark driven analytics system for performing low latency real-time traffic data analysis in Hadoop Ecosystem. We examine how MapReduce based real-time and batch analysis workflow could be carried out flexibly and efficiently and evaluate the developed system and used technologies. Next section gives short background and related work. Section III introduces the developed platform and its architecture. Section IV shows the experiments and their results. Finally, we conclude our paper with discussion in the section V.

II. Background

Sensor technology is changing traffic monitoring and controlling by bringing interconnected sensor devices into crossings and cars [1]. Traffic data sources include sensors installed on vehicles, such as GPS and RFID sensors, built-in sensors mounted on the road infrastructure, such as road condition and weather sensors, laser radars, digital video and still cameras, as well as various mobile devices and applications. This transition will lead to significant increase in the amount of traffic data. The data generated by these sources varies in their data representation, content, data rate, and volume. These heterogeneous sources produce significant amounts of information at high rates that need to be ingested, fused, stored, analyzed and queried efficiently and rapidly. Moreover, the analytics system has to be flexible, configurable and scalable both vertically and horizontally. Mian et al. propose a conceptual framework for managing traffic data based on Hadoop Ecosystem [2]. Aydin et al. have evaluated clustering algorithms on static GPS data using Apache Spark as their processing architecture [3].

Numerous applications have been developed for road and vehicular traffic services. Rich sets of sensors are used for traffic monitoring globally in big cities. For example, Intel has developed an efficient system to process data from hundreds of traffic monitoring devices in Hangzhou city based on the Apache Hadoop and Xeon E5 processors [4]. Nomura Research Institute developed Zenryoku Annai! application for predicting real-time traffic jams and searching the fastest driving routes using large data sets. Google Traffic Layer of Google Maps provides real-time information of the prevailing traffic conditions by crowdsourcing with GPS devices of mobile phones.

III. Analytics system architecture

We identify seven functional phases from data ingestion to data provisioning and develop analytics platform based on these phases (Figure 1) : 1) Ingestion and aggregation, 2) Data pre-processing and fusion, 3) Real-time analysis, 4) Storage and metadata creation, 5) Periodic batch analysis, 6) Archiving, querying and post analysis and 7) Data provisioning. Cloudera CDH 5 distribution of Hadoop framework is chosen as it provides comprehensive set of components, management interface and API to access management functions. We developed our analytics platform as inspired by the Lambda architecture [5], considering flexible composition of data ingestion, storage and analytics components. In Lambda

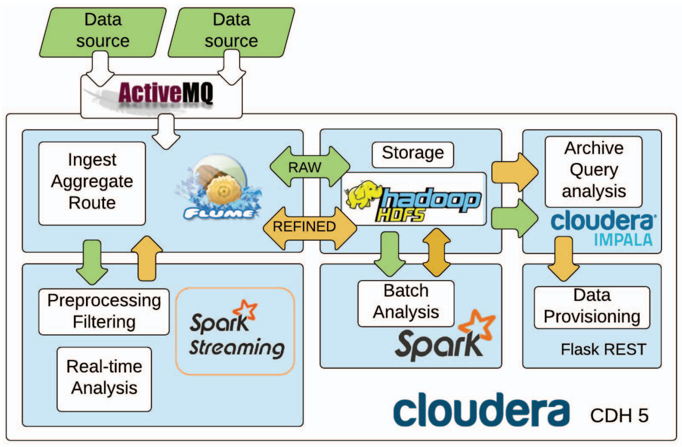

Fig. 1. Analytics system architecture

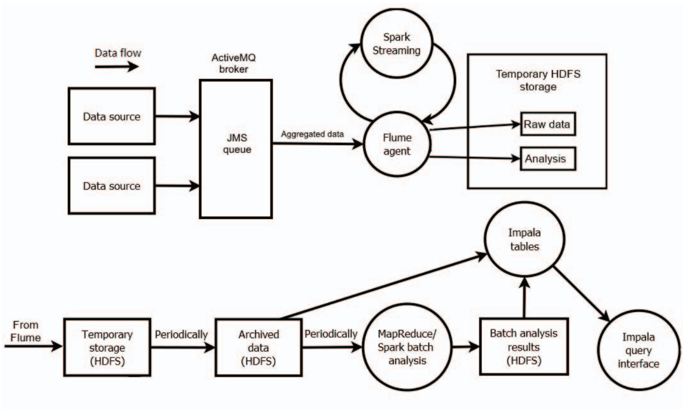

Fig. 2. Data flow in the analytics system.

architecture raw data is delivered separately to batch and speed processing layers. Correspondingly in our system, data can be flexibly aggregated and transferred between these layers using Apache Flume.

Figure 2 explains the data flow between the system components within our current traffic analysis scenario. Apache Spark Streaming is chosen as the core component for real-time analysis because it is efficient in iterative computing tasks, supports a variety of data sources and programming languages and can be run on Hadoop. Data aggregation is done with Apache Flume in data ingestion pre-stage together with Apache Active MQ broker. This solution provides message routing, aggregation, queuing, load balancing and flexible configuration and integration to other systems. Flume agents ingest high velocity GPS data from taxi cabs and road weather data from weather stations, and forward data to Spark for pre-processing. Data storage is split into two separate tasks: storing the raw data and storing the analysis results. Each data set is stored in a separate HDFS subdirectory. Periodic analysis tasks read the preprocessed and ingested data from the HDFS subdirectories and write results to separate locations. Impala database is chosen since it has good query performance with HDFS (Figure 6). Impala daemons are run on each datanode in Hadoop cluster and Impala query engine retrieves data straight from HDFS directory structure using Parquet storage method. This makes both the raw data and the result data instantly available. We developed Python API to support developers in creating custom Spark analysis scripts, to manage Impala database and HDFS storage, as well as to configure Flume agents and the REST based data provisioning interface. Python Flask based REST interface is implemented for data provisioning which transforms requests to Impala queries and transfers responses to JSON format.

The computing cluster is deployed on six server machines where each server act as a Hadoop datanode and one of them acts as a namenode. The cluster has been set up on Amazon EC2 cloud service with paravirtual machine instances and General Purpose Elastic Block Storage. The namenode server is deployed on Amazon EC2 c3.2xlarge instance which has 8 CPUs, 15GB of main memory and 100GB of storage space. Each datanode is c3.xlarge instance having four CPUs, 7,5GB of main memory and 70GB of storage space. Spark is configured to use four executor cores per datanode having 1GB of main memory for each executor.

IV. Experiments

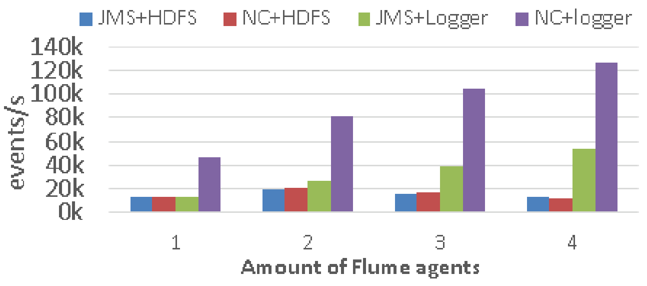

Fig. 3. Event throughput with different Flume agent configurations. (JMS = Java Message Service, NC = Netcat)

1) Ingestion: We experiment Flume data ingestion with different agent configurations. First we deploy one to four Flume agents with Netcat sources and measure event ingestion rate with and without HDFS sink (Figure 3). Flume file channel capacity is set to 1 million events, transaction capacity to 100,000 events, file channel checkpoint interval to 30 seconds, and HDFS replication factor to minimum value of one as these were considered to be proper values for streaming analysis. One Netcat source is able to ingest events at the rate of 46K events per second. Netcat source with HDFS sink suffers performance degradation. That is, if events are produced at higher rate than the HDFS sink can drain, the I/O write performance decreases throughput and causes Flume channel to overflow. The maximum ingestion rate with netcat source and HDFS sink is reached with two agents that ingest 18K events per second with throughput of 1,22 MB/s. Increasing channel capacity resulted in channel overflow. Here, Flume consumed 50 percent of the overall CPU time of an 8 core machine, which indicates that high rate of requests to the Netcat source overloads server resources. For avoiding channel overflow, we implement Java Messaging Service (JMS) source with Active MQ queue. Each JMS source with HDFS sink can ingest near 14K events per second and with four agents, event ingestion rate is near 55K events per second.

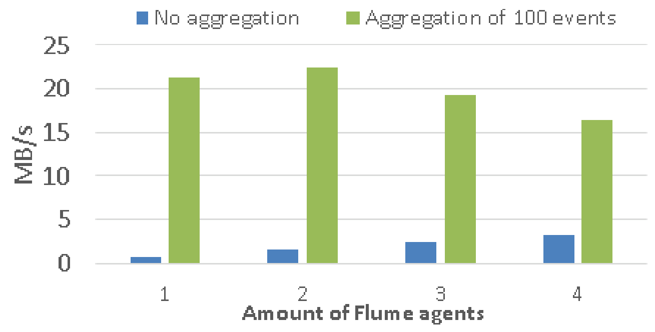

Fig. 4. Throughput with and without aggregation. JMS source and HDFS sink used.

$$ind = \frac{(\sum_{i=1}^{n} v_{recent})/n}{(\sum_{i=1}^{m} v_{ref})/m}$$

Eq. IV-2. Formula for calculating traffic flow indicator.

2) Real-time analysis: We experimented with Spark Streaming analysis using Flume JMS source configuration. The analysis takes GPS events as an input, calculates traffic flow indicator values in real-time and stores results to HDFS. Analysis is based on Spark‘s map, reduce and filter operations. Traffic Fluency indicator values (Equation IV-2) are calculated for each predefined geographical area from location, time stamp, and speed. Area ids are used as keys in MapReduce based distributed analysis while the time stamp and speed are combined and used as values. Analysis script merges consecutive data samples from each area and calculates average speeds between predefined batch intervals.

After calculating average speed for an are, it is compared to the reference average value of the current area. The ratio of the current average area speed and reference area speed becomes the current traffic flow indicator value. Results are routed back to Flume agent which delivers them to HDFS sink. Spark analysis is configured to use one second batch interval, which denotes how frequently Spark analysis is executed over ingested data. Without JMS event aggregation, maximum throughput with four agents stays below 5 MB/s (Figure 4). To increase analysis throughput, we aggregate events on JMS queue by area ids. Aggregation of events increased analysis throughput remarkably compared to ingesting of individual events reaching 23 MB/s with two agents and aggregation size of 100 events. When more agents are deployed throughput decreases as more I/O writes are executed simultaneously into HDFS. Clearly, design of the analysis affects to the throughput.

3) Periodic batch analysis: Batch analysis is carried out in the system at regular time intervals, such as hourly, daily

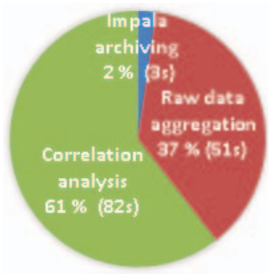

Fig. 5. Latencies experienced in different batch analysis phases.

$$\rho = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{\sqrt{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}}$$

Eq. IV-3. Pearson formula for estimating traffic flow correlation.

monthly or yearly basis. Batch analysis is performed sequentially as parallel execution of multiple analysis could reduce performance. Analysis is executed in a separate child process to improve error resiliency. To demonstrate periodic analysis, we wrote an analysis routine for calculating correlation between road condition scores, read from the road weather data source, and traffic flow indicator values which were computed in real-time analysis. Analysis takes monthly collected road condition data and traffic flow indicator values as an input, and calculates hourly averages of road condition scores for geographical areas as well as hourly average values for the traffic flow indicator. All incomplete and erroneous data values are filtered out. Pearson correlation coefficient is then used to estimate a linear dependency between average road condition and traffic flow indicator values, which is presented in Equation IV-3.

We measured latencies separately for Spark batch analysis, aggregating raw data in HDFS and archiving data to Impala as all of these phases are closely related to batch analysis process in the system. Experiments were executed three times and latencies for each phase were calculated. Figure 5 shows percentages of total latency in each phase. As we can see, the analysis and HDFS raw data aggregation produces most of the latency, whereas storing data to Impala is fast even though it includes merging of data tables.

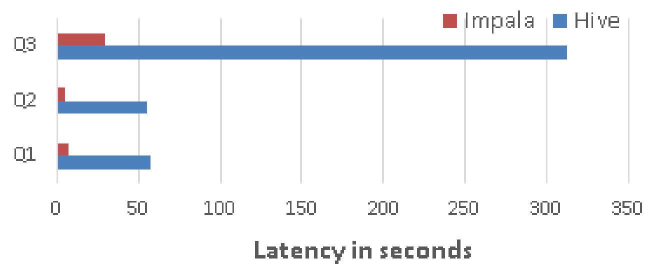

Fig. 6. Latencies of different queries with Apache Hive and Cloudera Impala.

4) Querying: Query performance was tested with Hive and Impala queries over randomly generated data in HDFS. Data rows contain an integer, a floating point and a string. Ten million rows of raw data is generated to HDFS and stored in ten separate files totalling to 2.2GB of raw data. Data tables and metadata are created into both Hive and Impala databases respectively. Experiment was carried out with following SQL queries: 1) SELECT SUM (x) FROM t, 2) SELECT MAX (y) FROM t WHERE x BETWEEN 100 AND 200, and 3) SELECT COUNT (tx) FROM t INNER JOIN t2 WHERE AND t2.y ty = tz = '1'. Queries were executed three times in Hive and Impala respectively and latencies were measured. The average latencies are shown in Figure 6. Results show that Impala outperforms Hive in all queries.

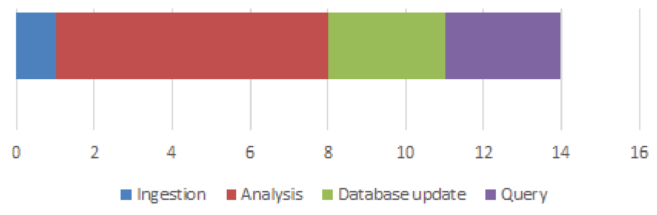

Fig. 7. Overall latency experienced in the analytics system.

5) Overall latency: We analysed the overall latency from data ingestion to retrieving real-time analyzed data from Impala database (Figure 7). The analysis is based on latencies measured during the experiments. We examined how latency accumulates in typical scenario where GPS events arrive from taxi cabs to the Flume agent, proceed through real-time Spark analysis and HDFS storage, and eventually translate into traffic flow indicator queries in Impala database. We assume data to be produced at 50K events per second, which is the maximum rate that Spark Streaming is able to analyse steadily according to our experiment. Results show that ingesting data to analys one batch of data with Spark Streaming takes at maximum one second. Pre-processing, analysis and HDFS write operations both add three seconds due to batch interval settings. Overall, analysis contribute seven seconds the total latency. From the Impala archiving test (Figure 5), we estimate that updating Impala data tables takes approximately three seconds. Query latency depends on the size of the data partitions. Based on the query experiment (Figure 6), we estimate that performing type three query takes approximately three seconds when data is partitioned to tables with 10 million rows at maximum. According to previous estimates, the overall latency adds approximately 14 seconds in a typical scenario using maximum Spark analysis throughput (23 MB/s). Near one second latencies could be reached with lower ingestion rates, proper system configuration and optimization.

V. Discussion

In this paper, we present a Big Data platform for low latency traffic flow analysis based on real-time, high velocity data. Based on our experiments, Apache Flume is suitable technology for ingesting real-time data to Spark Streaming analysis when properly configured. Configuration involves choosing appropriate Flume sources and sinks for different purposes as well as configuring their batch intervals, channel capacities, channel types, checkpoint intervals etc. Moreover, Flume can be integrated into message brokers such as Active MQ or Kafka for more advanced message routing, queuing, load balancing and availability. However, since transmission is based on TCP protocol, small packet size causes large overhead and too many simultaneous requests when ingesting individual events. Thus, data aggregation and lightweight protocols such as MQTT and CoAP should be considered. HDFS I/O write performance can become bottleneck if multiple Flume agents are writing result data from simultaneous Spark Streaming analysis. Spark Streaming and Spark batch analysis did not have any performance issues in during our experiments. However, we noticed that load balancing for all Spark executors lead to uneven results and some tasks ran longer than others. This could be caused by either virtualization or Flume and Spark Streaming configurations and should be examined more thoroughly in future work. Virtualization certainly has its disadvantages such as parallel usage of disk I/O and network resources, which can reduce performance compared to dedicated server clusters. However, performance of computing cluster on Amazon EC2 can also be increased by using I/O and memory optimized instances and enhanced networking.

Our future work will cover scalable real-time analysis and decision making for the Internet of Things applications. We will continue our platform evaluation by integrating alternative state-of-the-art technologies to our platform such as Apache Kafka, MQTT and CoAP. Moreover, we will study how semantic technologies can be integrated with our Big Data platform to facilitate interoperability, scalable knowledge discovery and data mining tasks. This work has been already started in our previous research [6] where we studied distributed reasoning of real-time traffic events.

References

[1] C.-Y. Chong and S. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247–1256, Aug 2003. doi: 10.1109/JPROC.2003.814918

[2] R. Mian, H. Ghanbari, S. Zareian, M. Shtern, and M. Litoiu, "A data platform for the highway traffic data," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th International Symposium on the*, Sept 2014, pp. 47–52.

[3] G. Aydin, I. R. Hallac, and B. Karakus, "Architecture and implementation of a scalable sensor data storage and analysis system using cloud computing and big data technologies," *Journal of Sensors*, vol. 2015, 2015. doi: 10.1155/2015/834217

[4] H. Chen, "Improving traffic management with big data analytics," 2013. [Online]. Available: http://www.intel.com.br/content/dam/www/public /us/en/documents/case-studies/big-data-xeon-e5-trustway-case-study.pdf

[5] N. Marz, *Big data : principles and best practices of scalable realtime data systems*. O'Reilly Media, 2013.

[6] A. Maarala, X. Su, and J. Riekki, "Semantic data provisioning and reasoning for the internet of things," in *International Conference on the Internet of Things*, Oct 2014, pp. 13–18.