Harnessing the Complexity of Mobile Network Data with Smart Monitoring*

Aleksandr Suleykin[1] and Peter Panfilov[2]

[1]V. A. Trapeznikov Institute of Control Sciences,
Russian Academy of Sciences, Moscow, Russia
aless.sull@mail.ru
[2]National Research University – Higher School of
Economics, Moscow, Russia
ppanfilov@hse.ru

Abstract. Mobile (cellular) networks represent a fast evolving research field that take advantages of recent technological advances such as Big Data and distributed computing to provide extensive network monitoring for network operation planning and management purposes. Challenges related to making use of the large volume of streaming data generated by mobile networks include extracting relevant elements within massive amounts of signals possibly spread across different sources (data bases), reducing dimensionality, summarizing dynamic information in a comprehensible way and displaying it for interpretation purposes. The adequate network modeling provides both statistical data of network performance and important networking QoS related insights. Comprehensible mobile network information is needed that uncovers the role of each attribute and variable. To harness the complexity of mobile network data and to extract relevant information a dedicated distributed computing platforms and Big Data frameworks are needed, able to discover and deal with the inherent properties and complexities of these datasets. Smart network monitoring service plays a central role here.

Keywords: Smart Monitoring System · Cellular Networks · Distributed Computing · Big Data · Dynamic Data-Driven Application System · Lambda Architecture · DDSM · Roaming Users Detection and Monitoring.

1 Introduction

The number of smartphone users has already reached 4.61 billion users in 2016, and upward trend is forecast for the market with 5.07 billion users by the end of 2019 [1]. The rapid growth of a mobile user count immutably leads to the proportional increase of data being generated by mobile subscribers, user equipment, cellular network nodes and mobile networks as a whole. This is becoming more challenging for mobile operators to handle constantly increasing data volumes for many different protocols and mobile network interfaces using traditional solutions with standalone systems, relational databases, many different formats of data storage and transmission. To address these new challenges a new approaches such as Big Data, Internet of Things, Machine-to-Machine Communications, distributed computing platforms and paradigms find its application to cellular network data storage, aggregation, transformation, enhancement and transfer.

Real cellular networks exploit multiple different protocols for data transmission and corresponding interfaces. Each node of the mobile network is communicating with other node(s) and external environment according to the standard protocols such as 3GPP [2] and ITU [3]. Every protocol has its own parameters, which are different

from one network element to another. The complexity and variety of data protocols, very large volumes of data being transferred and importance of data have led to the need of new approaches to the cellular network data monitoring and analysis, using latest technology achievements such as Big Data and Machine Learning.

The smart monitoring system (SMS) vision relies on the use of ICT to efficiently manage and maximize the utility of mobile network infrastructures in order to improve the quality of service and network performance. Many aspects of SMS projects are dynamic data driven application systems where data from sensors monitoring the system state are used to drive computations that in turn can dynamically adapt the monitoring process as the complex system evolves. In this context, a research and development of a distributed Big Data driven framework for cellular network monitoring data entails the ability to dynamically incorporate more accurate information for mobile network controlling purposes through obtaining real-time measurements from the network meters, base stations, and other sensors.

Traditional network monitoring services, exemplified by Netboss XT [4] a sophisticated multipurpose telecommunication networks management system are designed to monitor networks, detect failures and provide network maintenance and performance analysis in an multi-vendor, multi-protocol, multi-service heterogeneous environments. The major advantage of such a system is its flexibility – the system comprises of hundreds of software intelligent agents that permit to manage the equipment of almost any known vendor and interface with major OSSs. It also provides an integrated development environment and programming language to create new software agents or enhance existing ones.

However, traditional approaches face problems of handling increasing complexity of network data because they do not scale well enough and are much slower when compared to the modern high-performance data analytics (HPDA) and distributed computing enabled solutions.

We propose a Distributed Big Data Driven Framework (DBDF) for cellular network monitoring data on the basis of the Dynamic Data-Driven Application System (DDDAS) paradigm [5] and a core concept of Lambda architecture [6-8], specifically targeted at scalable and secure real-time Big Data application systems, the comparison of different Big Data methods, techniques, and available tools. The DBDF consists of different components that realize SMS paradigm for the cellular network data in distributed Big Data driven fashion as it is presented in figure 1.

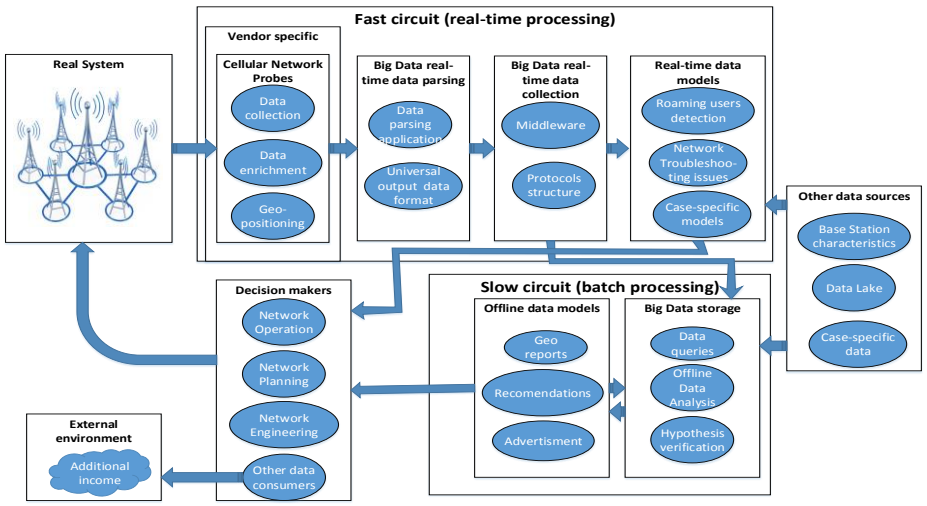

Fig. 1. The architectural overview of the DBDF – a concept

The new concept of DBDF for handling cellular network management and operation problems is targeted at network management representatives and decision makers in Mobile Engineering, Operating and Planning departments to help them in managing complexity of the cellular network data in real-time on the basis of smart network measurements, simulation and optimization models.

2 Distributed Computing and Big Data Issues

In fact, cellular network data is a streaming data coming from interfaces, base stations, billing system and other sources and represent highly-loaded systems with Gigabytes and Terabytes of data per second, even though this is a compressed data. As such, the CN monitoring data need to be parsed and processed on a cluster of computers with application of Big Data methods for processing of very large volumes of data. Distributed data processing methods and techniques play an important role in the whole CN data processing pipeline. All available and needed application systems and methods should be adapted for high performance and fully distributed computing environment and support scalable, reliable, and secure data processing. This is a key requirement for dealing with complexity in modern CN infrastructures.

Distributed computing techniques have been widely used by data scientists before the advent of Big Data concept. Thus, standard and time-consuming algorithms were successfully replaced by their distributed versions with the aim of agilizing the learning process. For many of current issues of Big Data, in particularly, in cellular network operation characterized with large volumes of streaming data, a distributed approach is becoming immutable nowadays.

The first distributed computing framework that enabled the processing of big volumes of data was the MapReduce paradigm. This tool was aimed at easily handling huge datasets in an automatic and distributed way using Map and Reduce task concepts, that enable user with building a distributed and scalable applications while hiding technical details as data partitioning, failure recovery or job communication. However, MapReduce concept is not designed to scale well when dealing with iterative and online processes, and usually deal with batch data tasks with relatively huge latency comparing with online data processing. In our suggested architecture this paradigm would be recommended for implementation in Batch layer for Big Data storage system component.

Another group of methods is based on distributed in-memory computing, micro-batch and real-time techniques. These methods are usually used for online Big Data processing with millions of tuples per second per node performance. In our proposed solution of the Smart Monitoring System architecture these methods are used in real-time data transformation system component, in real-time data parsing component and also might be used in a real-time data modeling component.

Message-driven applications are applications, that allow processing of future messages that will arrive after subscription. The main advantage of such systems is that many consumers can access the same data in independent way. Thus, a message-driven approach is suggested to be used in Big Data real-time data collection to enable many data consumers easily access cellular network data.

2.1 Lambda Architecture for Big Data

Building a reliable and efficient distributed Big Data application that satisfies a variety of end-user requirements is a challenging task. Lambda Architecture (LA) represents a useful framework for designing such applications. The appearance of the LA concept was inspired by the following motivations:

- the need for a robust and fault-tolerant system, both against human mistakes and hardware failures;
- the system should be linearly scalable scaling out rather than up;
- to serve a wide range of workloads and use cases, where low-latency reads and updates are required, with support of ad-hoc queries;
- the system should be extensible and features should be added easily.

Essentially, the Lambda Architecture (LA) comprises of 3 main parts:

- *Batch layer*. This layer has two functions: manage the main append-only raw data streams and pre-compute arbitrary query functions calling batch views. In our DBDF architecture this is a so called “slow circuit” layer, where cellular network data is coming from real-time data collection component

using batches.

- *Speed layer*. There are different fast and incremental algorithms which are used with low latency. The speed layer deals only with the most recent data. In the DBDF, these are key components that represent a "fast circuit", and all components inside this layer deal with online (real-time) data. Latency is a major concern there.
- *Serving layer*. This layer indexes the batch views in Batch layer, and data can be queried ad hoc with low latency. This layer is not included in current DBDF implementation and will be considered as a future work aimed at building DBDF Serving data layer [6-8].

3 Apache Spark and Lambda-enabled DBDF Architecture

Our implementation of the DBDF concept is based on Apache Spark open-source platform which supports all ranges of Big Data formats like batch data, text data, real-time streaming data, graphical data, etc. Apache Spark has already proved its huge potential in the Big Data industry because of its in-memory data processing that makes it high-speed data processing engine compare to Hadoop MapReduce.

Apache Spark provides a powerful data processing engine with development APIs to allow data scientists to execute streaming conveniently. Streaming is unstructured data that is generated continuously by thousands of data sources, including log files generated by customers using mobile or web applications, in-game player activity, information from social networks, financial trading and telemetry from connected devices or instrumentation in data centers, IoT sensors, etc.

With Spark running on Apache Hadoop YARN, developers can now create applications to consume and transform complex data streaming from Apache Kafka. Complex transformations like exactly-once event-time aggregation can be expressed using this API and the results can be output to a variety of systems. Apache Spark has the inbuilt support of over 80 high-level operators and lets programmers write applications using different programming languages as Python, Clojure, Scala or Java.

The low-latency in-memory data processing capability of Spark allows for handling many data analytics challenges using its machine learning libraries and graph analytics algorithms. With the help of Spark the CN providers can enable themselves to analyze data coming from various kinds of data sources, because Spark can easily process continuous streams of low-latency data. Thus, CN providers can create real-time dashboards and explore data in real-time to monitor and optimize the CN operation.

Figure 2 presents a high-level view of the Spark and Lambda-based implementation of the DBDF for the smart processing and analysis of streaming data.

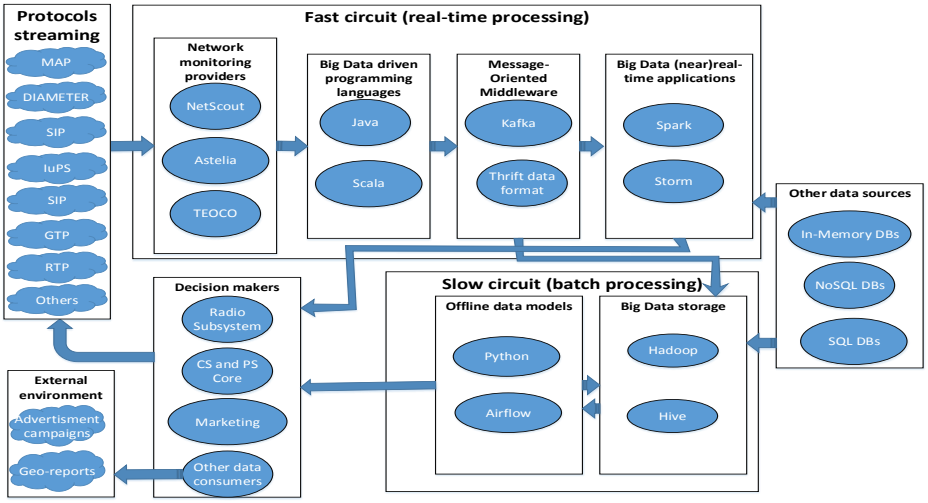

Fig. 2. Overview of the DBDF component architecture – an implementation

3.1 Real Cellular Network

Real CN consists of multiple network elements such as RNCs, BSs, NodeBs, eNodeBs, MMEs, GGSNs, and other nodes receiving and sending data via different cellular interfaces. Every new generation of mobile networks offers its own protocols for data exchange and interfaces which adds to the overall complexity of the real CN. The situation is aggravated by a variety of mobile device interfaces to cellular networks. Typically, a mobile device has both an application processor and a cellular processor, which each operate in different power states and data states. On most mobile devices, applications use multiple forms of connectivity with multiple transitions from one to the other. To manage connections to CN, the device itself usually has multiple cellular data modes with different data rates and different power usage. The cellular processor communicates with the CN in two different planes: the *control plane*, which manages the communication protocol, and the *data plane*, which carries the actual data packets representing different kinds of data – voice, SMS, MMS, data itself.

To provide good quality of service to the mobile user, CN providers attempt to deploy a CN that will never have its capacity exceeded. They can also be sensitive to the behavior of mobile applications which possess features that, intentionally or not, push the boundaries of the networks. As a result, carriers often need to restrict application behavior to keep them within the limitations of the networks. For example, a CN provider can limit the number of connections that an application triggers, especially in the control plane.

In addition, network measurements might be accompanied with the base stations characteristics, environmental (weather) condition data, and a mobile user's demands and/or complaints. These represent different attributes and properties of the real CN.

3.2 Cellular Network Events

Event-Driven network data are data records that appeared after specific network event. It means that different data items should be created and transferred after some event. Thus, these events might be presented as data communications between nodes (transactions), data on the base station parameters, user complaints, etc.

Base station data records are the characteristics of base station such as their geo-positioning, supportive technology, LAC code, Cells, address, vendor etc. User complaints are complaint data records that are collected in connection to the particular geographical location. Different transactions are data records that represent a part of a subscriber session. The data in transactions are being transferred according to the 3GPP and ITU standards in predefined format – CN data exchange protocols.

3.3 Network Data Parsing

Big Data driven programming environment is a special purpose application capable of processing huge volumes and streams of data (gigabit per second, Gbps) that parses all highly compressed protocol data (depends on vendor format) to the one unified data format, or processes data records received from real CN (BS parameters, complaints). The data parsing is implemented via programming language, the choice of which is highly important because different frameworks offer diverse functionality and more adopted for some particular cases.

3.4 Message Oriented Middleware

Message-Oriented Middleware is the Continuous Streaming data storage with predefined data structure according to different protocols. This is the middleware that provides opportunity for many data consumers to get data in real-time as well as send data for storage using batches. The issue here is that there might be many data consumers, including internal mobile provider departments and external partners. Actually, it hardly depends on the use cases – the more data use cases, the more data

consumers are. So the most important requirements for this layer is the capability of handling extremely fast data streams, high reliability, scalability and support to many different data consumers.

To meet requirements, it is proposed to use Apache Kafka messaging system as a distributed streaming platform. In a large distributed system, there are usually a lot of services that generate different events: logs, monitoring data, noticed attempts to access secret resources, etc. On the other hand, there are services that need these data. Kafka helps here as an interface between data producers and consumers: it collects data from former, then stores it in a distributed store using topics and distributes it at the latter via subscription. In other terms, Kafka is a hybrid of a distributed database and message queue [13].

3.5 Offline Data Models

It is proposed to use Apache Storm or Apache Spark applications for data transformations. Apache Storm is a free and open source distributed real-time computing system that performs processing of data streams. It has many use cases: continuous computation, ETL processes, real-time analytics, online machine learning and others. Storm is fast: a benchmark tests show that it can process at over a million tuples per second per node. It is fault-tolerant, scalable and simple to set up and operate [14]. Apache Spark is a fast and general-purpose cluster computing system. It has high-level APIs in Scala, Python, Java and R, optimized engine with support of general execution graphs, higher-level tools such as Spark SQL for SQL and structured data processing, GraphX for graph processing, Spark Streaming, MLlib for machine learning [15].

Also, for offline data processing mode we propose to use Python programming language for implementing machine learning and data mining algorithms. It has a clear syntax, is well developed and documented [17].

For offline mode we also propose to use Apache Airflow for scheduling. It is a library (or a set of libraries) for the development, planning and monitoring of work processes. With the Airflow we can use Python for coding work processes. Hence, there are advantages for organizing project and application development efforts. Implementation is fairly simple. One can use, for example, PyCharm plus Git [17].

3.6 Other Data Sources

These are data sources within a CN, which are mainly used in the network simulation. Usually, these data are collected and stored in data bases that are maintained by different departments of the CN provider. It can be SQL-, NoSQL-based, in-memory or other data structures, but the main concern is that data should be readily accesible by real-time data models. Other data sources component in the DBDF is considered an additional on-demand data source(s) for network simulation. In our previous work [9], we used data from Base stations data base, user complaints data base and population density data for the simulation and visualization of the so called “Problem Zones” in CN. In general, the particular use case and different requirements dictate the selection of the data bases and simulation tools.

3.7 Big Data Storage and Queries

This Big Data-driven layer in the general DBDF framework represents a component responsible for protocols data storage, report generation, validation of hypotheses and protocols data analysis. It can be done by using SQL queries to some database. Offline storage is a component that realizes reliable and scalable data storage of cellular network protocol data to be used in offline analysis, hypothesis validation and different kinds of on-demand reports.

For data storage solution we propose to use Hadoop NoSQL data base. It is open-source software for scalable, reliable, distributed computing, the framework that allows the distributed processing of large data sets across clusters of computers using

simple programming models [18]. It is highly scalable solution specially designed to scale up from the single servers to thousands of machines, where each server is offering storage and computation. It is created to detect and come up with failures at the application layer.

We propose to use Apache Hive for SQL queries to Hadoop. It supports analysis of large data sets stored in Hadoop in HDFS and compatible file systems, such as the Amazon S3 file system. Hive provides an SQL-like language called HiveQL with read schemes and with transparent mapping of MapReduce, Apache Tez and Spark jobs. All three execution engines run on Hadoop YARN. To speed up queries, it provides indexes, including bitmap indexes [18].

3.8 Decision Layer

Decision support layer composes the output from Big Data models and reports. It might be a streaming data in special predefined format, regular reports or triggered data events after filtering. The decision layer is the environment specially designed for Mobile Network Engineers, Planners, Operators, Managers and other data consumers. It includes all applications needed to aid decision process on the basis of model output data generated at previous stages of data pipeline. Decisions are made on the basis of this data: detection and prioritization of problem zones in the network [9], planning the network capacity and performance, base stations construction and deployment, managing Radio network elements and other.

4 Experiments and Results

As a proof of DBDF concept, a near real-time Big Data based prototype application for roaming users detection and monitoring was developed and tested using the real cellular network data of one of the largest mobile providers in Russia.

Mobile providers are keen to keep track of their subscribers who go abroad or migrate internally between regions. In fact, the significant percentage of all telecom operator's income is attributed to roaming users. Usually data usage, voice calls and SMS delivery are more expensive when traveling in other countries and regions, and telecom companies want to exploit solutions for detecting in real-time those users migrating abroad or in other regions in order to offer them special services.

Thus, telecom companies usually require:

- the ability to obtain data about the geolocation of the subscriber in real time for communication with the subscriber in Real-Time Marketing system,
- the ability to proactively detect the presence of blocking inconsistencies on the side of billing systems and HLRs (Home Location Registered).

The prototype system was developed using Apache Spark application and the Python programming language. The comparative analysis of the DBDF-based prototype system with traditional standalone monitoring services in cellular network was conducted to demonstrate benefits of the proposed framework such as scalability, reliability, speed and performance, as well as it's applicability to other similar use cases and possibility to check new hypotheses. A prototype application of roaming user detection and monitoring service demonstrated that Apache Spark streaming provides less delay, less processing time and as a result more time for decision makers in comparison to the traditional batch processing.

4.1 Experimental Setup

The prototype system was developed and executed on the basis of Apache Spark application, which is the largest open source project in data processing [15]. Since its release, Apache Spark, as the unified analytics engine, has seen rapid adoption by enterprises across a wide range of industries. Internet powerhouses such as Netflix,

Yahoo, and eBay have deployed Spark at massive scale, collectively processing multiple petabytes of data on clusters of over 8,000 nodes. It has quickly become the largest open source community in Big Data, with over 1000 contributors from 250+ organizations [15-16].

For our experiments, the YARN was chosen as a resource manager for Apache Spark. This manager is actually managing all cluster resources available for Spark jobs, which means that the capacity and performance of application are limited by resource manager YARN.

For messaging system, an Apache Kafka application was selected as a system with strong performance that can handle more than 100 000 events per second [14].

All installations of Spark, YARN and other support applications were done by Hortonworks (HDP version is 2.6.3.0-235).

Common configuration parameters are:

- Java version is 1.8.0_77 (Oracle Corporation);
- Scala version is 2.11.8;
- Operational system is Linux;
- Operational system version is 3.10.0-514.21.1.el7.x86_64.

The experiments were conducted on a powerful cluster. The characteristics of cluster and YARN resources available for all Spark jobs that run on cluster are as follows:

- 3 nodes for YARN allocated;
- Memory allocated for all YARN containers on a node is 306Gb. Total memory is 918 GB;
- Minimum Container Size (Memory) is 2048Mb;
- Maximum Container Size (Memory) is 100Gb;
- Number of virtual cores is 32;
- Percentage of physical CPU allocated for all containers on a node is 80%;
- Minimum Container Size (VCores) is 1;
- Maximum Container Size (VCores) is 32. VCores total is 96;
- 1 second interval between jobs;
- 2 Spark Executors selected on default.

4.2 Prototype Application Data and Code

Following to the specifications 3GPP and ITU, to analyze the subscribers relocation to other countries and/or regions inside a country we have looked at MAP (Mobile Application Part) protocol for 2G and 3G technologies (if user is currently using 2G or 3G network), or Diameter for 4G generation. The following attributes of the data would have been organized in messaging system entities (table 1):

Table 1. Filtered MAP and Diameter attributes.

Protocol	Event type	Filtered Attributes
MAP	CancelLocation/UpdateLocation	MSISDN
		IMSI
		VLR
		Timestamp
		SccpCallingDigits
		SccpCalledDigits
MAP	UpdateGPRSLocation	MSISDN
		IMSI
		Back Calling Address
		Timestamp
Diameter	UpdateLocation/Cancel Location	IMSI
		MSISDN
		OriginRealm

We have searched for Cancel location and Update location events only. From the CN's business logic perspective, we would be searching for the following events:

- start of another country visit;
- change of operator;
- change of country;
- return to country of origin;
- change of VLR (Visitor Location Registered).

The data streaming of MAP and Diameter protocols was organized using one of the largest telecom company in Russia, and data were received in Kafka application in thrift data format, during the job implementation converted in Json data format and finally sent to another Kafka messaging system to topics according to event types described above (table 1).

The average size of the streaming dataset is really challenging with an average number of 37300 records per second for MAP protocol and an average number of 24200 records per second for Diameter protocol, for a data stream of 61500 records per second in total.

To run the application it is necessary to have 3 files in the folder:

- startup script (run.sh is in the project in the ./src/main folder);
- main code implementation file (stream-1.0-SNAPSHOT-jar-with-dependencies.jar);
- folder with configuration file.

The fragment of the main code of a prototype application is shown in Figure 4.

```
import scala.util.parsing.json_
import scala.collection.JavaConversions._

object Main extends Serializable {
  def main(args: Array[String]): Unit = {

    val sparkConf = new SparkConf()
    sparkConf.set("spark.streaming.stopGracefullyOnShutdown","tru
    e")
    val sc = new SparkContext(sparkConf)
    val prop = new StreamingContext(sc, Seconds(1))
    val topicsSetGSM = prop.gsmmapTopicsSet
    val topicsSetDia = prop.diameterTopicsSet
    val kafkaParamsInGsm = prop.kafkaParamsIn.toMap + (
     "key.deserializer" -> classOf[StringDeserializer],
     "value.deserializer" -> classOf[GsmMapDecoder]
    )
    val kafkaParamsInDia = prop.kafkaParamsIn.toMap + (
     "key.deserializer" -> classOf[StringDeserializer],
     "value.deserializer" -> classOf[DiameterDecoder]
    )
    val metricsAccumulator =
     MetricsAccumulator.getInstance(prop.elasticHosts,
     prop.elasticIndexName, prop.elasticDateFormat)
    val kS = KafkaUtils.createDirectStream[String,
     TGsmMapRecord](
     ssc,
     PreferConsistent,
     Subscribe[String, TGsmMapRecord](
     topicsSetGsm, kafkaParamsInGsm)).
     mapPartitions(part => {
         part.map(_.value(()
```

Fig. 4. The fragment of Spark code.

As a result of Spark job implementation, the following results of YARN resources are allocated for this particular Spark job:

- Allocated memory is 6144Mb;
- Allocated CPU VCores is 3;
- 3 containers running;
- average Scheduling Delay is 14 ms;
- average Processing Time is 464 ms;
- total Delay is 478 ms.

The data can be viewed using Kafka UI tool (Fig. 5). It shows the messages in topic. The result of the application is data streaming with fields as it is shown in figure. Notice that IMSI and MSISDN fields that represent “private” information are cut from the image, but the other informative fields are here.

Fig. 5. Result messages in Kafka application after Spark job implementation

The approach was validated by comparison of amount of filtered messages for particular period. For this purpose, we were using code in Python to connect to Kafka application and to take two regarded data streams – Map and Diameter protocols. We analyzed the same period of time of messages in Kafka and filtered them using the same rules that were used for Spark application. The amount of filtered messages using Python code and Spark was equal.

4.3 Experimental Results

The comparison with traditional cellular network monitoring system and batch processing of CN data shows the advantages of the prototype system based on Apache Spark application.

The Spark streaming has many advantages comparing with usual batch streaming:

- average scheduling delay (ASD) for batch processing is much longer than Spark delay. Apache Spark streaming runs its jobs with only 0.015 seconds delay, while traditional batch processing has 0.5 seconds delay in average;
- average processing time (APT) shows that the same amount of data might be processed in 45 seconds intervals, while Spark streaming process data in 0.464 time intervals. It is achieved because Spark jobs runs each second, and the data processing is really fast, in-memory and efficient;
- average interval time (AIT) between Spark jobs is 1 second, while interval between batch jobs is usually 1 minute. Batch processing cannot run faster because of overheads before job start. Each start of job takes some additional resources and needs some time to start job itself. For batch processing it is larger than for streaming;

- average available time (AAT) for decision makers to trigger roaming users. It shows that time to make a decision about some action against "caught" users is larger with Spark due to its faster computation comparing to the traditional batch processing. Usually telecom providers are interested in users with no more than 15 minutes delay when the event occurred that is the user crossed the country border and this event has been caught by system. After we have this event in messaging system, all the rest is depending on us – how fast we process data and filter it for triggering and sending notifications. Thus, if we consider average time between appearance of event in messaging system and this event filtered - Spark shows only 1 second delay on average, while traditional batch requires more than 1 minute and 15 seconds. It means that decision makers can have more time to understand this user, his behavior and to decide on sending any notifications.

The results of comparative study of the DBDF-based Spark streaming monitoring versus traditional batch-based monitoring service can be described as follows (Fig. 6):

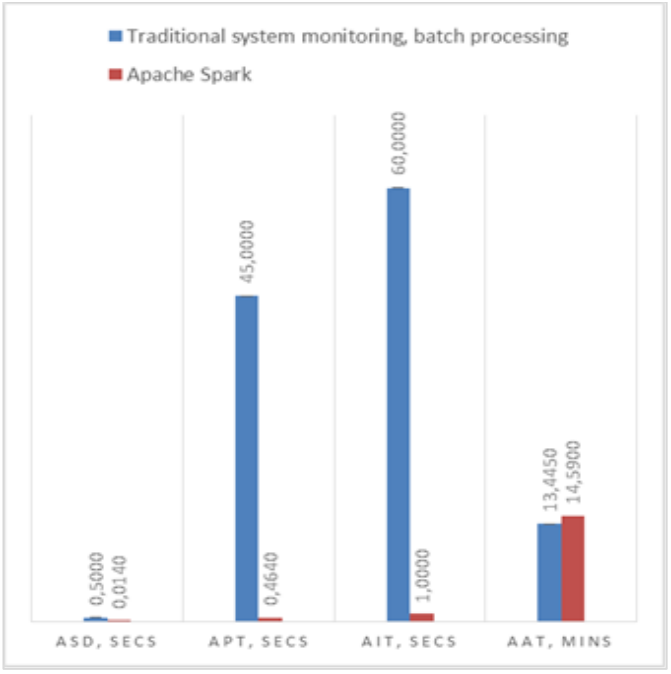

Fig. 6. The performance of the roaming users monitoring service implemented on the basis of Apache Spark (red) and traditional batch processing (blue)

In total, a new DBDF framework has the following advantages over traditional monitoring systems (Table 2):

Table 2. DBDF vs traditional monitoring system.

metric/system	**Traditional system monitoring**	**DBDF**
Scalability	low	high, not limited
Reliability	low	high
Speed and performance	low	high
Amount of possible use cases	one	unlimited
Data access	strict, within a department	non-strict, company wide
New hypothesis check	N/A, not enough data	Easily, all monitoring data

Thus, all DBDF components are scalable, and with adding more compute nodes to a cluster more performance gains can be obtained:

- storage and processing memory are scalable for all DBDF components that represents a significant advantage over traditional standalone monitoring systems. The parameter is important because of constant traffic growth worldwide;
- reliability of the solution is supported by the fact that all data are replicated in a cluster that makes framework reliable. In case of node failures the data is not lost;
- speed and performance shows the huge difference. Because of the cluster mode and in-memory computations, DBDF is processing data very fast, while traditional standalone systems usually perform much slower;
- amount of possible use cases is not limited with DBDF – all monitoring data are collected and stored, and many new use cases can be created and discovered. In traditional system usually one system is covering one use case, or one department. With DBDF, new use cases can be easily implemented with all company departments based on processing rules (online streaming) or new hypotheses validation (offline streaming);
- data access is usually strict in traditional monitoring systems, while with DBDF all departments can have access to all monitoring data and achieve synergy effect all together. It means that departments can work together for new use cases adaptation and verification;
- new hypotheses checks are almost not possible with traditional systems because of not all monitoring protocols are presented in place. In contrast, DBDF open up new horizons with petabytes of data exploration.

5 Conclusion

To manage the complexity of the cellular network's large volumes of streaming data it is extremely important to deploy a powerful framework for data processing, aggregation, enhancement, enrichment and storage. The adaptation of the distributed Big Data driven framework for the smart monitoring services in telecom provider environment and deployment of proposed architectural components will help to achieve effective, reliable, scalable, high-speed and secure processing of the cellular network data. The proposed DBDF framework is fully capable of dealing with high-loaded CN data streams and can be considered a foundation for future models creation, making sure that all data are reliably saved and not lost.

The proof of concept was achieved by creating a near real-time Big Data based prototype application for roaming users detection with processing performance of above 60 000 events per second. The prototype monitoring system has been created using Apache Spark application and the adequacy of the model was checked by the test application in Python programming language. The created prototype system has revealed that Apache Spark is capable of handling thousands and even more events per second and may be considered a foundation for real-time Big Data hub creation.

The comparisons of the prototype DBDF-based smart monitoring system with traditional standalone monitoring systems in cellular network demonstrates many benefits of the proposed framework such as its scalability, reliability, speed and performance, possibility to check new hypotheses. Apache Spark streaming facility in sample application for roaming user detection demonstrated less delay, less processing time and more time for decision makers comparing with traditional batch processing.

The implementation of the DBDF framework can be repeated for any telecom operator using the same protocols, provided the cluster has the same performance to maintain stable work of application. Or, the data stream can be also proportionally decreased along with the amount of nodes in a cluster and their capacity.

As future developments of the DBDF for the cellular network data processing and analysis we plan to develop, integrate and test a *Serving layer* of the Lambda-enabled implementation of the smart monitoring system for the CN data.

References

1. https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide (2017). Number of mobile phone users worldwide from 2013 to 2019 (in billions), Accessed on: 2017-10-11.
2. http://www.3gpp.org/about-3gpp. Accessed on: 2018-01-31.
3. https://www.itu.int/en/itutelecom/Pages/default.aspx. Accessed on: 2018-01-31.
4. http://www.pictogramdesign.com/websites/nti_overview/docs/ServiceAssurance_OV_082610.pdf NetBoss Technologies Integrated Service Assurance. Pictogram Digital Design. Accessed on: 2019-03-17.
5. Darema, F. (2004). Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements. International Conference on Computational Science. M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 662–669, 2004. © Springer-Verlag Berlin Heidelberg 2004.
6. Marz, Nathan and Warren, James (2015). Big Data: Principles and best practices of scalable realtime data systems, 1st ed.. Manning Publication Co., 2015.
7. The Lambda architecture: principles for architecting realtime Big Data systems, blog post by James Kinle. Available at: http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for Accessed on: 2018-02-05.
8. Lambda Architecture: A state-of-the-art, post by Pere Ferrera. Available at: http://www.datasalt.com/2014/01/lambda-architecture-a-state-of-the-art/ Accessed on: 2018-02-05.
9. A. Suleykin, P. Panfilov (2017). The Simulation-Based Smart Management Approach for Cellular Network Operation and Planning, in: Annals for DAAAM for 2017 & Proceedings, DAAAM International, Viena, 2017, pp.0423-0432.
10. http://window.edu.ru/catalog/pdf2txt/503/80503/60870, p. 1-20. Accessed on: 2018-03-21.
11. https://kafka.apache.org/documentation.html#introduction. Kafka 1.0 Documentation. Accessed on: 2018-02-04.
12. http://spark.apache.org. Apache Spark. Accessed on: 2018-02-04.
13. https://databricks.com/spark/about/. Accessed on: 2018-03-08.
14. https://kafka.apache.org/documentation.html#introduction. Kafka 1.0 Documentation. Accessed on: 2018-02-04.
15. http://storm.apache.org. Apache Storm. Accessed on: 2018-02-04.
16. http://spark.apache.org. Apache Spark. Accessed on: 2018-02-04.
17. https://www.python.org/. Accessed on: 2018-03-08.
18. http://airflow.apache.org/. Accessed on: 2018-03-08
19. http://hadoop.apache.org. What Is Apache Hadoop? Accessed on: 2018-02-04.
20. http://libreportal.net/data-warehousing/apache-hive.html. Accessed on: 2018-03-07.