Apache Storm Based on Topology for Real-Time Processing of Streaming Data from Social Networks

Anatoliy Batyuk, Volodymyr Voityshyn
Lviv Polytechnic National University, 12 Bandera street, Lviv, 79013, Ukraine,
abatyuk@gmail.com, voytyshyn@gmail.com

***Abstract* — In recent years Big Data has become one of the hottest topic in software engineering. Necessity to deal with huge amount of data brings new challenges and, of course, new opportunities.**

Big Data requires revising approaches to designing software architecture. In particular, the Lambda Architecture pattern distinguishes the components that process recent data only in real-time (“speed layer”). In practice, such data items come continuously from so-called data streams.

The aim of this paper is to represent concept of the implemented by the authors Apache Storm based topology for real-time processing of data streams from social networks.

Keywords* — *big data; data stream; Apache Storm; .NET; Lambda Architecture.

I. Introduction

One of the most challengeable software development tasks over the last few years is to deal with huge amount of data which is significantly greater then it was whenever before. We are accustomed to refer such tasks to the relatively new software engineering field called “Big Data”. In particular, streaming data are considered as one of the most important sources for big data.

In general, considering the fact that the amount of data is huge, it seems difficult to process the data altogether. That is why real-time data stream processing has been a considerable scientific and engineering task for more than a decade. Traditionally this task contains two ingredients: algorithms (that are accountable for obtaining analytical information) and software (that implements processing infrastructure).

Current paper is devoted to the second ingredient. The concept of a real-time data processing topology based on Apache Storm with social networks as the input data streams is represented below.

The rest of the paper is organized as follows. In section II the architectural concept of the entire system is briefly described. Formulation of the engineering task is represented in section III. In section IV we provide some details regarding implemented by the authors real-time data processing topology based on Apache Storm. The .NET based components are described in section V. Since Apache Storm is not the only option to implement such a system we provide a brief overview of some alternative approaches in section VI. Existing known analogues, their pros and cons are briefly described in section VII. Results of experiments with the implemented system are presented in section VIII. Some architecturally significant future tasks are listed in section IX. We conclude in section X.

II. System Architecture Overview

To show the overall picture it is necessary to introduce architecture concept for the entire system.

The purpose of the system is to accumulate data from social media services (e.g. GitHub, Twitter etc.) and build analytical models upon the data. A predictive model for software development technologies trends has been implemented as a sample task to test the describing topology.

It should be emphasized that Big Data requires revising of the software architecture patterns. A particular system that deals with Big Data has to meet high level requirements called “3V” [1]. To ensure those requirements the approach called “Lambda Architecture” [2] has been chosen to design the describing system (Fig. 1).

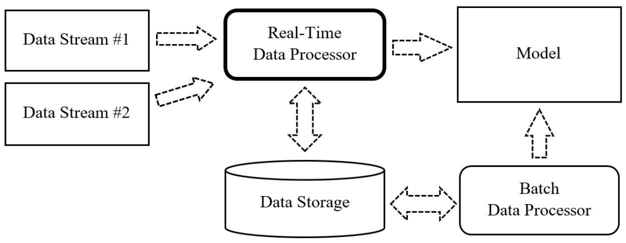

Figure 1. Architectural Concept of the Entire System

In particular, one of the “3Vs” called “Velocity” is crucial within the scope of current paper. “Velocity” means that the data items come continuously and have to be processed in real-time. The “Real-Time Data Processor” component (Fig. 1) is accountable for it.

III. Formulation of the Task

Formal definition of the implemented task is represented in current section.

The task is to implement the "Real-Time Data Processor" component that fits the architecture described above (Fig. 1). Therefore, the component has to meet the following high-level functional requirements:

a. The analytical part is included. The purpose of the analytical part is to keep the model up-to-date.

b. The data and results of their processing are persisted into the data storage.

To ensure appropriate quality level the following non-functional requirements (base on the quality attributes [3]) have to be satisfied (the requirements are arranged by their importance):

a. Performance: 1) high throughput (amount of data items processed per second) and 2) low latency (the average time taken for a data item).

b. Interoperability: 1) supporting more than one data source, which require different communication protocols and data formats; 2) possibility of using different technologies to develop components of the system.

c. Reliability: 1) the system has to be functional even through a particular data tuple has failed in processing; 2) the system has to keep functioning when the "Model" component is temporary unavailable.

d. Maintainability: 1) aptitude to continuous improvements and smooth evolution; 2) the system has to provide enough information about failures to ensure effective repair.

e. Security: the communication points between the system and data streams, model and data storage have to be protected.

The next sections describe how those functional and non-functional requirements from the lists above have been addressed.

IV. Real-Time Data Processing Topology

A. *Choosing of the Platform*

The real-time data processing topology has been implemented by means of Apache Storm. This platform belongs to the Hadoop ecosystem. The main reason of choosing Storm is that it addresses most part of the non-functional requirements from section III. In particular, the distributed computations model ensures high performance, wide range of flexibility in implementing spouts allows to support different data streams, the fault-tolerant approach to handle failures provides the needed level of reliability. Another important advantage of Storm is supporting different technologies (some details about .NET-based components implementation are represented in section V).

It should be noted that Apache Storm is not the only option to implement the "Real-Time Data Processor" component. Some alternatives are briefly discussed in section VI.

B. *Entry Points*

In general, the system (Fig. 2) takes data from different data streams. The final goal of the system is to obtain some analytics from the data in real-time in order to keep the model up-to-date.

Following the Apache Storm terminology, the entry points of the topology are called "spouts". The spouts are adapters between the data sources and real-time data processing infrastructure. The key point is that each spout deals with a specific data stream which allows to produce tuples from streams with different protocols and data formats.

C. *Data Processing Infrastructure*

The data processing layers compose the core of the "Real-Time Data Processor" component (Fig. 2). Within the scope of current paper, the "data processing layer" (or simply "layer") term means a logical set of Storm bolts consolidated by their purpose. High-level description (without disclosing the implementation details) of each layer is provided in the rest of current subsection.

1) *Formatting Layer*

Once the tuples are emitted into the topology, they should be converted into the unified format. The "Formatting Layer" is accountable for it. The challenge in implementing corresponding bolts is that the data items are obtained from streams in different formats. Consequently, current stage simplifies and unifies data processing for the next ones.

2) *Filtering Layer*

The next layer aims to filter the data in order to drop the part, which is not important from the processing goal standpoint. Obviously, this stage decreases data size and reduces complexity which is significant for topology performance and effectiveness of the data processing algorithms.

3) *Aggregating Layer*

After filtering, the data items are aggregated. In fact, this layer decreases the number of tuples to deal with on the analytics layer. In general, such a component requires stateful stream processing since some amount of data items should be accumulated before aggregation. To implement current layer Apache Trident is employed. The used Apache Trident topology is configured in the way that state of the processing data streams is stored in memory (one of the advantages of this decision is that it allows to avoid a dependency to an external storage like Cassandra).

4) *Analytics Layer*

Referring to the general concept of the entire system (Fig. 1) the model component has to meet the following requirements: 1) consider historical data and 2) be updatable in real-time mode [2]. The analytics layer addresses requirement #2. According to Lambda Architecture terminology, this component is called "speed layer".

Of course, machine-learning algorithms make the core of current layer. Such class of algorithms are called "incremental" because they are highly optimized to process streaming data in real-time.

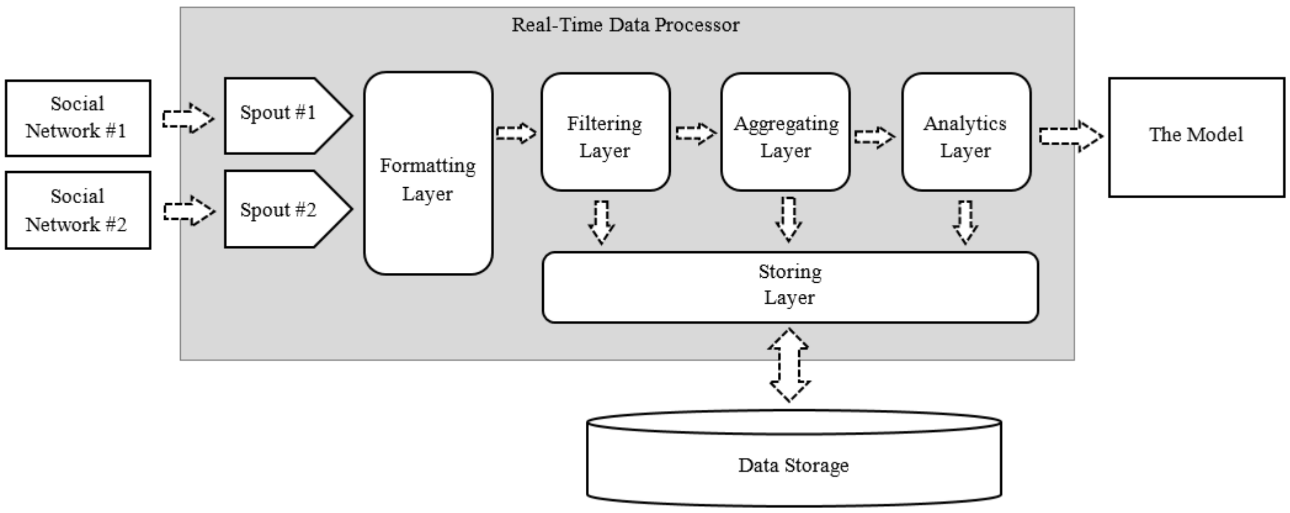

Figure 2. Real-Time Data Processing Topology

In current version of the system, this layer is implemented by means of Apache Mahout. From one hand, this library supplies effective implementations of standard online machine learning algorithms, from the other hand, it provides a flexible framework to build custom extensions.

5) Storing Layer

The storing layer is concerned with persisting the data and processing results into the data storage (Fig. 2). The bolts from current layer are involved on the filtering, aggregating and analysis stages. In fact, the respective bolts are connection points to the database which is important for the system reliability standpoint (some thought regarding this question can be found in the rest of current section and also in section IX).

D. Addressing Reliability

In general, reliability is an important quality attribute for a distributed system. As specified in section IV, the attribute includes two requirements.

The first of them is concerned with handling failures (e.g. exceeding of tuple processing timeout). This requirement is addressed by the Storm built-it guaranteeing message processing mechanisms.

The second requirement specifies that the "Real-Time Data Processor" component has weak dependency on the model so that the processing infrastructure is fully functional even through the model has temporary downgraded. This one is achieved thanks to the storing layer which ensures saving the processed data into the data storage. Once the model has become available, the data will be picked up by the batch data processor. The described process ensures keeping the model up-to-date.

However, there is a strong dependency between the real-time data processing infrastructure and data storage. When the data storage has stopped working the real-time data processor is not functional. Easing this dependency is an improvement to be implemented in the future (see section IX for details).

V. Implementation of the .NET-Based Components

Apache Storm is a flexible platform that supports different technologies (first of all Java, and also .NET, Python etc.). Furthermore, it supports mixture of them.

The spouts and bolts of the topology described above (except the bolts from the analytics layer) are implemented by means of .NET framework. To ensure using C# the topology is deployed on an HDInsight cluster (which is integrated in Windows Azure).

One of the main reason behind the decision to implement the most part of the topology with C# and deploy the system on Azure is to take advantages from .NET platform and Microsoft cloud computing infrastructure. Some alternatives are listed in section VI.

VI. Overview of Alternative Approaches to Implementation

Undoubtedly, the represented concept is not the only approach to implement the task from section III. Some alternatives to the made key architectural decisions are briefly discussed in current section.

A. Choosing of a Real-Time Data Processing Platform

As we can see from the previous sections, choosing of a real-time processing platform is one of the key decisions within the scope of current task.

In particular, there are a few commercial solutions similar to Apache Storm, for example:

1. Azure Stream Analytics
2. Amazon Kinesis
3. IBM InfoSphere Streams.

Obviously, Apache Storm's main advantages in comparison with those ones are free of charge and independency from a particular cloud-computing platform.

An alternative that could be used instead of Storm is Spark Streaming (extension to Apache Spark API). The reason behind choosing Storm is that it is more adapted to deal with real-time data [4].

B. *Basic Library for the Analytics Layer*

Effective implementation of the analytics layer is crucial for the system because this module is accountable for the most complex calculations. That is why it is important to consider various existing libraries to achieve the highest performance.

R is a free platform for statistical computing. The CRAN project includes the "stream" package that focuses on data stream mining tasks. In particular, it provides implementation of clustering and classification algorithms.

Another option is Spark MLib. This is a library for a distributed computing environment. Wide range of statistical and machine learning algorithms are supplied by it.

C. *Setting up a Storm Cluster*

In general, there are 2 main options to set up a Storm cluster: 1) cloud computing platform and 2) your own servers. The main difference between them is that the first approach avoids the need to use your own hardware.

Currently implemented version of the topology is deployed on Azure (see section V). However, another cloud provider (e.g. Amazon Web Services or Google Cloud) can be used as well.

VII. Existing Similar Projects

Of course, the idea of social media mining is not a new one. Accumulating and analysis data from a wide range of social services has been being done for the recent few years. Here are some examples of such projects:

1. GitHub Archive (https://www.githubarchive.org/) aims to record and archive public GitHub events in order to make them easily accessible for analysis.

2. Gnip (https://gnip.com/) is an official Twitter's API enterprise platform, which provides real-time and historical data.

3. Data Sift (http://datasift.com/) gets access to historical social posts from several data sources (e.g. Facebook, Yammer, Google+, YouTube, blogs etc).

The main difference between the listed above existing projects and the one represented in current paper is that those projects supplies data and some standard analytics whilst current project aims to build specialized models upon the data. That is why, except the models themselves, it is extremely important to have own data processing infrastructure optimized to meet the project requirements.

VIII. Experiments With System's Performance

Performance is one of the most important qualities of the implemented real-time processing system (section III). That's why measurement and monitoring performance is vitally important for its effective functioning. In particular, currently we are interested in the following two performance characteristics: throughput and latency.

A. *Experimental Environment*

As mentioned in section V the Apache Storm real-time data processing infrastructure is deployed on Windows Azure. The following configuration parameters are applied to the cluster:

- Operating system – Windows Server 2012 R2 Datacenter
- Cluster type – Storm
- HDInsight version – 3.4
- Apache Storm version – 0.10.0
- HDInsight cluster – Standard_A1 (CPU Cores: 1, Memory: 1.75 GB, Max. disk size: 70 GB).

Deployment model of the cluster is shown on Fig. 3.

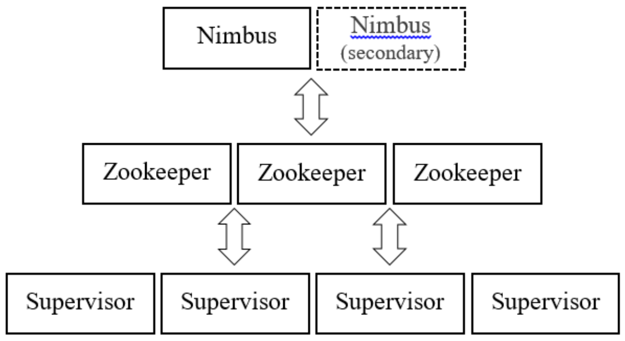

Figure 3. Storm Cluster Based on 1 Node

It should be noted that the secondary Nimbus instance starts working when the primary one temporary fails.

The experiments from the rest of the section were performed on this cluster configuration.

B. *Measuring Performance*

For sake of experiments, social networks were replaced with fake data streams (of data items similar to the ones received from social networks), which allowed us to set up various levels of load upon the system. In order to test performance of the real-time data processing infrastructure in isolation from the other parts of the system the "Storing Layer" component was not involved (in other words, data saving functionality was turned off).

The test dataset contained 100000 items. The experiments were done for a few versions of the dataset which contained 25%, 100%, 200% and 400% of data items (200% and 400% mean that the initial dataset was self-concatenated 2 and 4 times respectively).

In the performed experiments parallelism in the Apache Storm topology was controlled in 2 levels: number of workers and number of threads per worker (columns "Node" and "Parallelism" in Table 1 respectively). In particular, 4 nodes means horizontal scaling of the cluster from Fig. 3.

The results of performance measurement are listed in Table 1. The column "Elapsed Time" contains the overall time spent on processing the whole dataset (in terms of performance measurement it is called "latency"). The number of data items (tuples) processed per 1 sec is represented in the "Throughput" column. The "CPU Time" column shows the overall time spent by CPUs on processing the whole dataset (it was measured by means of Windows Azure dashboard).

TABLE I. PEFORMANCE MEASUREMENT RESULTS

Nodes	Dataset	Parallelism	Elapsed Time (s)	Throughput (tuple/sec)	CPU Time (s)
1	25%	x1	12.4	2412	10.4
1	25%	x2	10.9	2304	9.9
1	25%	x4	5.4	2663	4.4
1	100%	x1	45.7	2340	42.7
1	100%	x2	43.1	2432	42.1
1	100%	x4	32.0	2779	30.0
4	100%	x4	18.7	5353	39.2
4	100%	x8	15.5	6456	33.7
4	200%	x4	34.3	5363	78.2
4	200%	x8	31.5	6340	73.9
4	400%	x4	79.5	5226	196.4
4	400%	x8	59.9	6571	144.7

The experiments results shown that increasing number of parallel thread (the "Parallelism" column) improved throughput. In particular, from x1 to x4 the average gain of throughput was about 15%. The graph of dependency between "Parallelism" and throughput for one node is represented on Fig. 4.

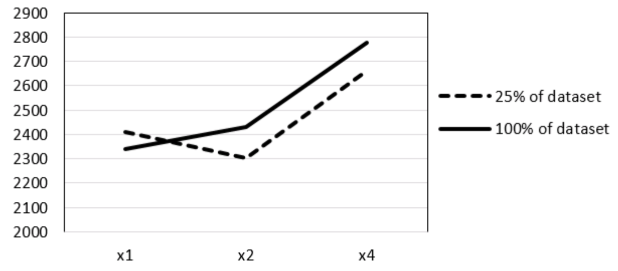

Figure 4. Dependency Between "Parallelism" and throughput (tuples / sec)

As we can see from Table 1 throughput was significantly improved by horizontal scaling of the cluster. It proves the fact that the implemented topology supports parallel computations which can be used for performance improvements in case it is necessary in practice.

IX. FUTURE TASKS

Currently implemented version of the "Real-Time Data Processor" component has left some unresolved architectural tasks. Here are two of them.

A. *Easing Dependency From the Data Storage*

The represented real-time data processing topology has a strong dependency to the data storage: when data storage is not available for some reason, the real-time data processor cannot store the received data and the results of processing. This fact, obviously, affects reliability of the system as a whole since some part of real-time data will be lost during the downgrade time.

Possible ways to resolve this issue can be:

1. Increasing reliability of the data storage itself
2. Using a buffer on the real-time processing component side for temporary saving the data before persisting.

B. *Preventing Losing Data From Input Streams*

The integration points with the input streams have a potential weakness: in case the amount of input data exceeds the topology's processing capability, there is a risk of losing part of the data. This pessimistic scenario may take place only within a certain periods of time (e.g. when social network users are the most active).

To reduce the risk the following tactics can be applied:

1. Increasing capability of the topology by means of the flexible load planning (a cloud infrastructure feature).
2. Using a messages queue in front of the spouts.

X. CONCLUSIONS

In this paper we represented architectural concept of the Apache Storm based real-time data processing topology. The described system was implemented by the authors. To prove effectiveness of the designed topology a sample big data analytics task was solved (the task was to build up a predictive model for programming technologies trends based on the data stream from GitHub and Twitter).

Experiments with the system allowed concluding the following:

1. The chosen toolset (mostly based on Apache Storm) was convenient in usage and shown its effectiveness on the implementation and testing stages.
2. The implemented topology (Fig. 2) demonstrated enough flexibility for the sample task. Therefore, it can be evolved in order to be applied for resolving more complex and valuable problems.
3. Apache Mahout provides effective machine learning algorithms adopted to data stream mining purposes. Furthermore, we can take advantages of its framework to build custom improvements and extensions.

REFERENCES

[1] Wei Fan, Albert Bifet (2016, Apr 22). Mining Big Data: Current Status, and Forecast to the Future [Online] – Available: http://www.kdd.org/exploration_files/V14-02-01-Fan.pdf
[2] N. Marz and J. Warren Big Data: Principles and best practices of scalable realtime data systems, New York: Manning Publications, 2015.
[3] M. Barbacci and others "Quality Attributes", SEI, Pittsburgh, PA, Tech. Rep. ESC-TR-95-021, Dec. 1995.
[4] Kenny Ballou (2016, Apr 21). Apache Storm vs. Apache Spark [Online] – Available: http://zdatainc.com/2014/09/apache-storm-apache-spark/