SDMiner: A Tool for Mining Data Streams on Top of Apache Spark

[B.Sc. Thesis Proposal Abstract]

Sina Sheikholeslami
Amirkabir University of Technology
(Tehran Polytechnic)
424 Hafez Avenue, Tehran, Iran
sinash@aut.ac.ir

Amir H. Payberah*
Swedish Institute of Computer Science
Kista, Sweden
amir@sics.se

ABSTRACT

Stream processing has always been an important topic in data mining research. However, with recent improvements in cloud computing technologies and the need for more efficient Big Data Analytics tools, it has garnered even more attention due to its exclusive applications and challenges that require different computation models and tools. Existing platforms such as Apache Spark, Apache Storm and Apache Flink have provided tools for efficient stream processing, but still many common data mining and stream processing algorithms are not available to be used by these platforms. We are proposing SDMiner, an open-source software for mining stream data that includes a graphical interface for managing stream mining jobs, and a library of algorithms for mining data streams. SDMiner is designed on top of Apache Spark as a state-of-the-art distributed computing platform.

Keywords

Stream Mining, Data Mining, Apache Spark, Big Data Analytics

1. INTRODUCTION

Recent advances in computer hardware technologies have made the process of continuous data gathering a simple and regular one [1]. Everyday activities such as web search, posting to social networks, and purchasing from web stores continuously generate a huge amount of data, and we can yield interesting and useful results by processing and mining this data. As another example, transportation traffic control systems normally face a massive amount of data in the form of streams, and performing low-latency analytics on them can be beneficial to decision makers. Moreover, with real-time mining and analysis of network packets, attacks or anomalies can be detected more efficiently. In all of the aforementioned examples, a certain type of data labeled as "Stream Data" are involved.

Stream data in comparison to other types of data have unique properties that turns their mining and processing tasks into challenging ones [1; 2; 10; 7]. These include, but are not limited to, the need for one-pass algorithms, the need for low-latency processing, impossibility of storing data in conventional mass storages or databases, variability in size and input rate of data streams, and the evolution of data over time.

During past few years, many distributed computing platforms with support for stream data processing have been developed [8]. Examples of these systems include Apache Spark [12; 16], Apache Storm [15], and Apache Flink [6]. Although these systems have provided tools for efficient stream processing, but still many common data mining and stream processing algorithms are not available to be used by them. Therefore, we aim to design and implement an easy to use tool that includes a library of common algorithms for stream data mining. To address the above stream data processing challenges we build our tool on top of Apache Spark. We have chosen Apache Spark as a state-of-the-art and stable distributed computing platform that provides fault tolerance, scalability, and information flow control, and also includes a standard API for stream data processing called Spark Streaming [14; 17]. Our contribution includes:

- A graphical tool for defining and managing stream data mining jobs, managing stream data sources, and presenting results to the end user, and
- A library of common algorithms for mining data streams, including algorithms for sampling and histogram construction.

Related works include MLlib [11], which is Spark's scalable machine learning library. Albert Bifet et al. [4] have proposed StreamDM, a system for advanced data mining in Spark Streaming. StreamDM is similar to our contribution as both projects are built on top of Apache Spark Streaming. However, they have mainly implemented algorithms for clustering and classification of stream data, such as Multinomial Naive Bayes classifier and CluStream clusterer.

In this document, we will present a more detailed review of SDMiner, including the graphical tools, the library of stream data mining algorithms, and Apache Spark as the distributed computing platform for running stream mining jobs.

2. SYSTEM AND IMPLEMENTATION

In this section, we will first briefly review Apache Spark, then we will have a more detailed discussion of SDMiner

*Thesis Advisor

and its components, including the graphical tool and the streaming data mining algorithms library.

2.1 Apache Spark

Apache Spark is a fast and general engine for large-scale data processing [12]. Spark is considered as the processing engine in the Berkeley Data Analytics Stack (BDAS)[3]. Programs can be developed on top of Spark using Java, Scala, Python, and R.

Spark has a number of APIs for common Big Data Analytics tasks, including Stream Processing (Spark Streaming), working with Structured Data (SparkSQL), Machine Learning (MLlib), and Graph Processing (GraphX). Apache Spark has been regarded as a Top-Level Project of Apache Foundation since February 2014.

Spark's primary memory abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD) [16]. RDDs are immutable collections of objects spread across a cluster. Spark processes incoming data streams by creating small batches of data, and facilitates its batch processing engine's mechanisms to control the flow of information. Spark provides fault tolerance through logging lineages, which are transformations used to build an RDD. Users can also choose to checkpoint some RDDs to stable storages for faster recovery.

2.2 SDMiner Description

SDMiner's architecture consists of:

- A graphical user interface for defining jobs and presenting the results,
- A library of common stream mining algorithms,
- Apache Spark as the distributed stream processing platform, and
- A stream generator for defining arbitrary data streams.

Figure 1 shows SDMiner's layered architecture. Our contribution includes the user interface, the stream mining library, and the stream generator. Each of these components will be discussed in the following sections.

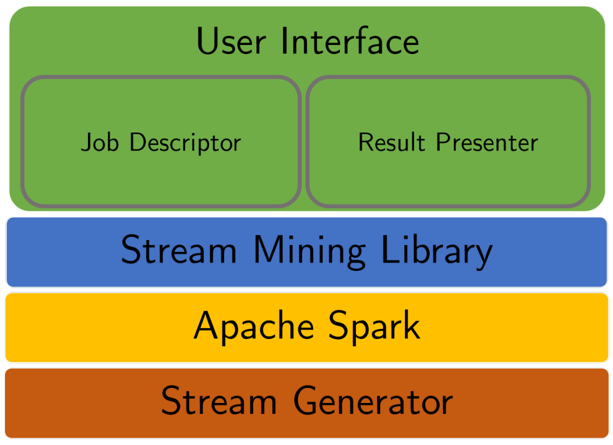

Figure 1: SDMiner's Layered Architecture

2.2.1 *User Interface*

To make stream mining tasks easier for end users, SDMiner includes a graphical user interface, consisting of a job descriptor and a result presenter. Using the job descriptor module, the users can easily:

- Select external data stream sources, such as Apache Kafka [9] or Twitter
- Pick the desired algorithm from the stream mining library
- Set needed parameters for running the stream mining job, including Apache Spark's connection settings and algorithm's other parameters

After the stream mining job is finished, users can have a visualized view of the results using the result presenter module.

2.2.2 *Stream Mining Library*

The stream mining library includes a number of common data stream mining algorithms as to be used for running mining jobs on top of Apache Spark. We will implement sampling algorithms, such as random sampling with a reservoir and concise sampling. Moreover, algorithms for histogram construction, including one-pass construction of equi-depth histograms and construction of v-optimal histograms will be implemented [1; 10].

2.2.3 *Connection to Apache Spark*

SDMiner will use Spark Streaming API to run stream mining jobs on Spark's core engine [13; 5]. The connection parameters and other needed constraints can be set using the job descriptor module.

2.2.4 *Stream Generator*

Besides external stream data sources such as Apache Kafka or Twitter, the users of SDMiner can also define their own data streams using the stream generator module. Having this module will result in a more integrated system and provides better flexibility to the users for defining stream mining jobs.

3. CONCLUSION

In this document, we discussed SDMiner, an open-source tool for mining stream data on top of Apache Spark. We started by describing the usage and challenges of processing and mining stream data that lead us to the need for specific methods and tools. SDMiner tries to solve this problem by providing a user-friendly graphical user interface to define and run stream mining jobs, and a library of common stream mining algorithms.

4. REFERENCES

[1] C. C. Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007.

[2] H. C. Andrade, B. Gedik, and D. S. Turaga. *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press, 2014.

[3] BDAS, the Berkeley data analytics stack. `http://amplab.cs.berkeley.edu/software/`. [Web. 31 Jan. 2016.].

[4] A. Bifet, S. Maniu, J. Qian, G. Tian, C. He, and W. Fan. Streamdm: Advanced data mining in spark streaming. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1608–1611. IEEE, 2015.

[5] T. Das, M. Zaharia, and P. Wendell. Diving into Spark Streaming's execution model. `https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html`. [Web. 31 Jan. 2016.].

[6] Apache Flink: Scalable batch and stream data processing. `http://flink.apache.org/`. [Web. 31 Jan. 2016.].

[7] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

[8] S. Kamburugamuve and G. Fox. Survey of distributed stream processing. 2013.

[9] J. Kreps, N. Narkhede, J. Rao, et al. Kafka: A distributed messaging system for log processing. NetDB, 2011.

[10] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[11] MLlib | Apache Spark. `http://spark.apache.org/mllib`. [Web. 31 Jan. 2016.].

[12] Apache Spark - Lightning-fast cluster computing. `http://spark.apache.org`. [Web. 31 Jan. 2016.].

[13] Spark Streaming programming guide. `http://spark.apache.org/docs/latest/streaming-programming-guide.html`. [Web. 31 Jan. 2016.].

[14] Spark Streaming | Apache Spark. `http://spark.apache.org/streaming`. [Web. 31 Jan. 2016.].

[15] Apache Storm. `http://storm.apache.org`. [Web. 31 Jan. 2016.].

[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[17] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM, 2013.