Quality model for evaluating and choosing a stream processing framework architecture

**Youness Dendane · Fabio Petrillo ·
Hamid Mcheick · Souhail Ben Ali**

2019 Jan

Abstract Today, we have to deal with many data (Big data) and we need to make decisions by choosing an architectural framework to analyze these data coming from different area. Due to this, it become problematic when we want to process these data, and even more, when it is continuous data. When you want to process some data, you have to first receive it, store it, and then query it. This is what we call Batch Processing. It works well when you process big amount of data, but it finds its limits when you want to get fast (or real-time) processing results, such as financial trades, sensors, user session activity, etc. The solution to this problem is stream processing. Stream processing approach consists of data arriving record by record and rather than storing it, the processing should be done directly. Therefore, direct results are needed with a latency that may vary in real-time.

In this paper, we propose an assessment quality model to evaluate and choose stream processing frameworks. We describe briefly different architectural frameworks such as Kafka, Spark Streaming and Flink that address the stream processing. Using our quality model, we present a decision tree to support engineers to choose a framework following the quality aspects. Finally, we evaluate our model doing a case study to Twitter and Netflix streaming.

1 Introduction

More and more data is produced today, and different techniques have been developed in order to process this data. Due to modern Big Data applications, like sensors, stock-trading or even user web traffic [6] data has to be processed

Universit du Qubec de Chicoutimi

Department of Mathematics and Computer science

555 boulevard de l'Universit

Chicoutimi, Canada

E-mail: dendaneys@gmail.com,fabio@petrillo.com,hamid_mcheick@uqac.ca,souhail.ben-ali1@uqac.ca

in real-time. The technique that can handle this problem is called : stream processing [5].

So we have assisted to the rise of Stream processing frameworks, such as Samza and Flink, which are becoming more and more popular, for offering a model to ingest and process data at near real-time [7].

However, with several stream processing frameworks and technologies associated available, a problem arise : how to choose the right framework ? Each framework has its own features and is more or less different from another framework.

So, depending on the context, you choose the best solution. But another problem occurs here : on what criteria are you basing on to answer this question ? In this paper, we provide a quality model for a decision taking. This model enforced by what we call variables/criteria, can help you through a decision and we see if it is suitable to choose stream processing framework.

We identify and explain in details four criteria that are important for the framework decision making. Further, we quickly present the selected frameworks with their pros and cons. The criteria and the frameworks have been chosen following a study of stream processing papers. We analyzed these papers, and picked based on an average, the most redundant.

The rest of the paper is organized as follow, we analyze the related work that has been done (ii), and then answer to the previous questions by identifying what are the different criteria you have to base (iii) and by introducing the different chosen stream processing frameworks (iv). We propose a decision model tree supported by the previous parts, that you can base on to choose the right framework technology (v).

2 State-of-the-art/ Related Work

A stream processing system requires four major elements: (1) Best understanding of the streaming applications architecture (2) identification of key requirements of distributed stream processing frameworks (DSPF) that can be used to evaluate such a system, (3) survey existing streaming frameworks, (4) evaluation and a comparative study of the most popular streaming platforms. We divide the related work based on the three elements mentioned above.

2.1 Architecture of streaming applications

Streaming applications architecture is not too much different from web architectures. Streaming sources are communicating using arbitrary protocols. So that, a gateway layer is set up to connect sources to streaming application and resolve the heterogeneity of sources protocols. A message queues are set up as a middleware to provide a temporary buffer and a routing layer to match the accepted event sources and the applications [11].

2.2 Requirements of distributed stream processing frameworks

There are eight rules [12] that serve to illustrate the necessary features required for any system that will be used for high-volume low-latency stream processing applications.

- Rule 1: Keep the Data Moving by achieving a low latency
- Rule 2: Query using higt level language like SQL on Streams (StreamSQL)
- Rule 3: Handle Stream Imperfections (Delayed, Missing and Out-of-Order Data)
- Rule 4: Generate Predictable Outcomes
- Rule 5: Integrate Stored and Streaming Data
- Rule 6: Guarantee Data Safety and Availability
- Rule 7: Partition and Scale Applications Automatically
- Rule 8: Process and Respond Instantaneously

2.3 Existing streaming frameworks

Several streaming frameworks have been proposed to allow real-time large scale stream processing. In this section sheds the light on the most popular big data stream processing frameworks:

2.3.1 Apache Spark [15]

Developed at UC Berkeley in 2009 [19], is a platform for distributed data processing, written in Java and Scala. In spark, streaming computation is treated as a series of deterministic batch computations on small time intervals.

2.3.2 Apache Storm [18]

is a real-time stream processor, written in Java and Clojure. Storm is a fault tolerant framework that is suitable for real time data analysis, machine learning, sequential and iterative computation.

2.3.3 Apache Flink [17]

is an open source processing framework supporting both stream and batch, It provides several benefits such as fault-tolerant and large scale computation [14]. Multy functionalities are offred by this plateform such us additional high level functions such as join, filter and aggregation it allows iterative processing and real time computation on stream data collected by different tools such as Flume [20] and Kafka [21].

	Spark	Storm	Flink	Samza
Data format	DStream	Tuples	DataStream	Message
Data sources	HDFS, DBMS, and Kafka	Spoots	HDFS, DBMS, and Kafka	kafka
Programming model	Transformation and action	Bolts	Actions functions (map,groupby..)	Mapreduce Job
Programming languages	Java, Scala and Python	Java	Java	java
Cluster manager	Hadoop YARN, Apache Mesos	Zookeeper	Hadoop YARN, Apache Mesos	YARN
Latency	Few seconds	Sub-second	Sub-second	Sub-second
Messaging	Exactly once	At least once	Exactly once	Exactly once
Machine learning compatibility	SparkMLLIB	Compatible with SAMOA API	FlinkML	Compatible with SAMOA API
Elasticity	Yes	Yes	No	No
Sliding windows/Windowing	time based	time based and count based	time based	time based and count based
Auto-parallelization	On demand	Pipelined processing	Pipelined processing	On demand
Streaming query	SparkSQL	No	No	Yes (Samza-SQL API)
Data Partitioning	Yes	No	No	Yes
API	Declaratif	Copositionnel	Declartaif	Copositionnel
Data transport	RPC	RPC	RPC	Kafka

Fig. 1 Frameworks comparative

2.3.4 Apache Samza [16]

is created by Linkedin to solve various kinds of stream processing requirements such as tracking data, service logging of data, and data ingestion pipelines for real time services [14]. It uses Apache Kafka as a distributed broker for messaging, and Hadoop YARN for distributed resource allocation and scheduling [14].

2.4 A comparative between processing frameworks

The comparison between those several frameworks listed above are data format, types of data sources, programming model, cluster manager, supported programming languages, latency and messaging capacities [14].

3 Paper Contribution

The work reported reported in this paper can be categorized under the class of decision help of choosing a stream processing framework. While there is a rich body of work in designing stream processing applications and huge comparative between these applications, a system that can help you to choose

the best application by criteria is still messing from contemporary stream processing systems.

In this paper we discuss some architectural frameworks such as Storm, Spark and others that resolve the Stream processing problem and we provide a a quality model to choose ans evaluate a stream processing framework basing on some criteria such us latency, guarantees, fault tolerance and data processing model.

4 Survey of Stream Processing Frameworks

In this section, we will present 4 frameworks that are used actually to resolve stream processing problem.

4.1 Storm

Storm integrates with any database (e.g: MongoDB) and any queuing system (e.g: RabbitMQ, Kafka).
Storm works with tuples. A tuple is a named list of values and can contain any type of object.
Its API is simple and easy to use due to only three abstractions :

1. Spout : A spout is a source of streams and reads from a queuing broker.
2. Bolt : Where most of computation's logic goes. Computation logic can be functions, filters, streaming joins, streaming aggregations etc. So basically, from an input, and with computation logic you can produce new output streams.
3. Topology : A network of spouts and bolts.

Storm is scalable, fault-tolerant and have an **at-least once** guarantee message semantic. The cons here are that there is not ordering guarantees and duplicates may occur.
Another of its strengths is if a node dies, the worker will be restarted on another node. If a worker dies, Storm will restart it automatically.
At the date of writing this article, with Storm SQL integration, queries can be run over streaming data, but it is still experimental.
Furthermore, Storm provides an **exactly-once** guarantee with Trident which is a high-level abstraction. This model is a micro-batch processing model that add a state and will increase latency.

4.2 Spark

Spark is an hybrid framework which means it can perform batch as well as stream processing.
Spark natively works with batch, but it has a library called Spark Streaming

that can allow to work with near real time data. It means that incoming data are regrouped into small batch and then processed without increasing the latency too much unlike Storm which provides true streaming processing.

One of its power is that the manner you write batch jobs is the same you write stream jobs. More than that, it is fault-tolerant and has an **exactly-once** semantics.

Spark has its own modules that you can combine :

- Spark SQL
- Spark Streaming
- Machine Learning
- GraphX (for graph programming)

Spark runs in Hadoop, Apache Mesos, Kubernetes, standalone or in the cloud and access diverse data sources such as HDFS, Cassandra, etc.

4.3 Samza

Samza is decoupled in three layers [8] :

1. Streaming
2. Execution
3. Processing

4.3.1 Streaming

For the message queuing system, Samza uses Kafka. Kafka is a distributed pub/sub and it has an at-least once message guarantees. Kafka consumers subscribe to topic, which allow them to read messages.

4.3.2 Execution

Samza uses YARN to run jobs. It allow to execute commands on a cluster of machines after allocating containers. This is made possible because of YARN, which is the Hadoop's next generation cluster scheduler. So, YARN provides a resource management and task execution framework to execute jobs.

4.3.3 Processing

It uses the two layers above; input and output come from Kafka brokers. YARN is used to run a Samza job and supervise the containers. The processing code the developer write runs in these containers. Samza's processing model is real time.

One of Samza's advantages is that the streaming and execution layers can be replaced with any other technologies. Also, because of the use of YARN,

Samza is fault tolerant; Samza works with YARN to transparently migrate tasks to another machine.

The processing model Samza provides are both batch and stream (real time). Whatever the code you write, it will be reusable whatever the model. Switching models needs config change; from HDFS to Kafka to pass from batch to stream processing.

4.4 Flink

Flink supports batch and real-time stream processing model. It has an exactly-once guarantee for both models. Flink is fault-tolerant and can be deployed to numerous resource providers such as YARN, Apache Mesos and Kubernetes; but also as stand-alone cluster.

One of the advantages of this framework is that it can run millions of events per seconds by using the minimum of resources, all of this at a low latency. Flink provides three layered API's :

1. ProcessFunction : It implements the logic, process individuals or grouped events and give control over time and state.
2. DataStream : Provides primitives for stream operations such as transformations. It is based on functions like aggregate, map and reduce.
3. SQL : To ease the writing jobs for analytics on real time data.

5 Criteria used in frameworks

To choose a stream processing framework, we have identified some criteria. These criteria don't give you the answer on whether you should use stream processing or batch processing, but rather helps you take the decision to pick the right framework. So this step assumes that you already identified the problem and you came to the idea that should use stream processing model over batch processing.

We first are going to give the criteria and explain them in details :

– Latency
– Message semantics (guarantees)
– Fault tolerance
– Data processing model (micro-batch or real-time)

5.1 Message semantics

Another term referring to this criteria is **Message guarantees**. The message guarantees can take three forms :

– At least-once : could be duplicates of the same message but we are sure that it has been delivered

– At most-once : the message is delivered zero or one time
– Exactly-once : the message is guaranteed to be delivered exactly one and only one time

Before providing message guarantees, system should be able to recover from faults. [6]

5.2 Fault tolerance

Streaming application run for an indefinite period, so it increases the chance of having faults. So this criteria is important, because despite the application has faults.
Fault tolerance guarantees that the system will be highly available, operates even after failures and has possibility to recover from them transparently. Flink has the highest availability.

5.3 Latency

Latency is the time between arrival of new data and its processing [10]. Latency goes hand in hand with recovery (fault tolerance) because, whenever the system has errors, it should recover fast enough so the latency doesn't decrease too much (i.e : the processing continue with minimal effect). Also, each framework can do do some optimization on data such as message batching, to improve the throughput, but the cost is sacrificing latency.

5.4 Data processing model

To do stream processing, there is two techniques :

– Micro-batch : based on batch processing but rather than processing data that have been collected over previous time, data is packaged into small batches and collected in a very small time intervals and then delivered directly to the batch processing. Spark for example does micro-batch.
– Real-time : data is processed on fly as individual pieces, so there is no waiting. Flink process data in real-time.

As messages are received directly the real-time processing technique has a lower stream processing latency than micro-batch but it become harder to have an exactly-once semantics. However, micro-batch provides better fault-tolerance and thus it can guarantees that the message has been received only once (i.e : Spark Streaming).

What we understand here is that message semantics are related to the fault tolerance and the data processing model, and according to how the fault tolerance is implemented the latency will increase or decrease.

Stream Processing Framework per Paper

Framework/Paper	An Evaluation of Data Stream Processing Systems for Data Driven Applications	Survey of Big Data Frameworks for Different Application Characteristics(++)	A Comparative Study on Streaming Frameworks for Big Data	On Big Data Stream Processing(++)	A New Architecture for Real Time Data Stream Processing	Pilot-Streaming: A Stream Processing Framework for High-Performance Computing (++)	REAL TIME DATA PROCESSING FRAMEWORKS	Survey of Distributed Stream Processing (++)	A Performance Comparison of Open-Source Stream Processing Platforms
Storm	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Samza	Yes	Yes	Yes	Yes	No	No	No	Yes	No
Spark Streaming	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Kafka Streams	No	No	No	No	No	No	No	No	No
Flink	No	Yes	Yes	No	No	Yes	No	Yes	Yes

Fig. 2 Frameworks per paper

Criteria per Paper

Criteria/Papers	A Performance Comparison of Open-Source Stream Processing Platforms	A Comparative Study on Streaming Frameworks for Big Data	REAL TIME DATA PROCESSING FRAMEWORKS	A New Architecture for Real Time Data Stream Processing	An Evaluation of Data Stream Processing Systems for Data Driven Applications	The 8 requirements of Real-Time Stream Processing	Survey of Big Data Frameworks for Different Application Characteristics(++)	Pilot-Streaming: A Stream Processing Framework for High-Performance Computing (++)	Survey of Distributed Stream Processing (++)
Latency	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Fault tolerance	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Message guarantees	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes
Data processing (micro-batch or real-time)	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Data storage	No	No	Yes	Yes	Yes	Yes	No	No	No
Streaming Query	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes
Input data source	No	Yes	Yes	No	No	No	No	Yes	No

Fig. 3 Criteria per paper

6 Quality Model for choosing and evaluating a SPF

After presenting the different frameworks and found the main characteristics/criteria, we came with a model. A model for evaluating the frameworks and choosing one given a set of criteria. In this section, we explain why we have chosen these particular frameworks and how we extracted certain criteria. Afterward, we explain how we have prioritized the criteria, and then, with all these information we present the quality model.

6.1 Methodology

There is several processing frameworks used in production today. But to find out what framework is used in which company is difficult and take time. So, our primary support was the research papers. We analyzed various papers about stream processing, and we defined **redundancy** as our benchmark. This means that we made a table with the papers and frameworks, and every time a paper cited a framework we gave a point to the paper. At the end, we had a table with the frameworks cited per paper.

We repeated the same process for the criteria. The result is on figure 3.

This paper is a first draft, and we plan to study more papers to have more criteria and frameworks, and thus, to have better average results.

6.2 Choosing and prioritizing the criteria

After finding the criteria, we had to prioritize them. Here is the criteria ranked by importance.

1. Data model
2. Fault tolerance
3. Message semantics
4. Latency

The first decision is what type of stream processing to choose, because this will have an impact on the other criteria. If you choose a micro-batch framework, it will be possible to have for each framework an exactly-once message semantics as opposite to a real-time model.

Latency is of great importance, but, a framework should be able to recover fast enough, so it does not affect the system too much (with minimum time). And before providing message semantics it also should be recover from faults automatically. Because it will influence the other criteria beneath it, this is why the fault tolerance is in second position.

Depending on whether it is exactly-once or at least-once message semantics, the latency will change depending this criteria.

6.3 Decision Model Tree

Based on the previous parts, we present the decision model tree to evaluate and choose a stream processing framework (fig. 4).

7 Case studies

In this section, we analyze some stream processing application cases. We go through two companies : Netflix and Twitter.
The goal of this section is to see if our contribution in this paper correspond to the reality (i.e: real world application). In analyzing how and why these companies use stream processing frameworks, we can identify the main underlying elements and compare them to our criteria. We get all information from papers and the companies tech blog.

7.1 Twitter

Twitter has actually an in-house framework called Heron. But before that, they were using Storm. We are going to detail framework evaluation for Storm, because Heron is an improvement but they are still using what we detail below.

The company that has made Storm was acquired by Twitter in 2011. Since, Twitter modified for their use.
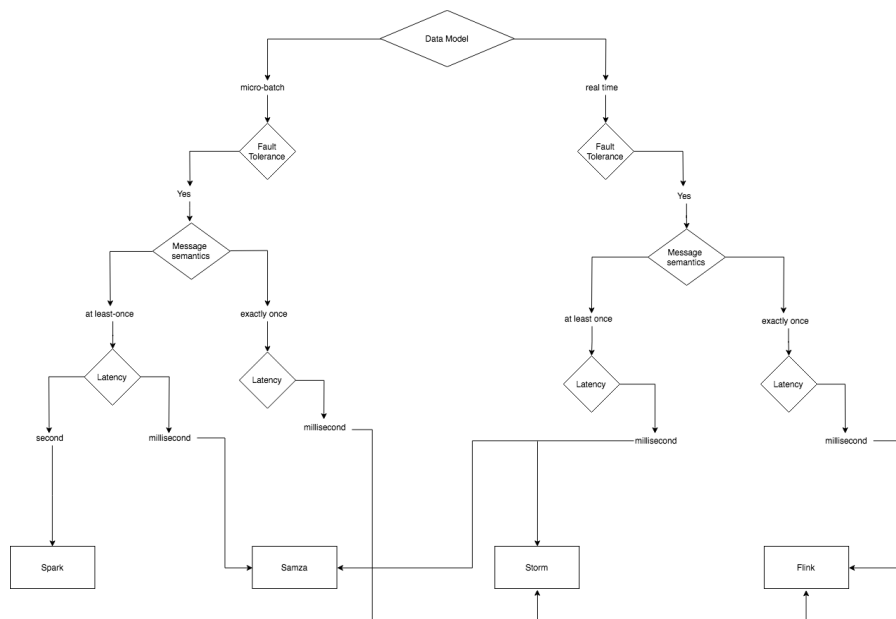
Fig. 4 The decision model tree

Let's begin with our first criteria : data processing model. At Twitter, due to choosing Storm, as we described it above, it has a micro-batch processing model. So, just by using it, the choice of data processing model has been made. We go now to our second criteria : fault tolerance. When Twitter describes Storm [18], they say that one of the argument chosen to design Storm is : resilient (i.e : fault tolerant); their second criteria and ours correspond. As they say in the article [18], on of the feature key is the processing semantics or message semantics. They describe that their solution has two guarantees : at least once and at most once. This characteristic correspond to our third criteria we have mentioned. Further in the article, Ankit et al. report some experiment they have made that had to show the latency results. As they calculated, their latency is close to 1ms 99% of the time. Our criteria are justified by the design and the use of Storm at Twitter.

In this first subsection, we can conclude that our criteria are match with the main characteristics of design and use of Storm at Twitter.

7.2 Netflix

In their article [22], they describe Keystone which is their stream processing platform. The solution chosen to do stream processing is Apache Flink. By choosing Flink, they automatically chosen the real-time processing for the data model criteria. Then, they gave a summary of common asks and trade-offs and one of them is failure recovery. This correspond with our criteria. One of the

asks was that the system is fault tolerant. If we follow our model, the next step is to choose the message semantics. In the post, their say that according to the use case loosing some events in the pipeline is acceptable while in other cases the event have to absolutely processed so it require a better durability. We see that this sentence is a synonym to our message guarantees criteria. In another post [23], they describe this time a real use case : to know what is trending on Netflix. In order to that, they need real-time data of what users watch, the event is then send to be processed. They describe that one of their challenges was having a low latency. This last criteria match with ours.

What we can conclude in this section is that these companies followed a path which correspond with our quality model. All our criteria had been taken into account by these companies and are part of the core decision on choosing and using stream processing framework architecture.

8 Discussion

In this section we will discuss the impact of our results, impact as well on engineers as on researchers. This quality model can be used as a guideline when wanting to choose a stream processing framework. Answering what type of criteria is important for a given context will end to the choice of the right solution; do I need absolutely only one instance of data or is it permissible to have duplicates ? (i.e: at least once vs exactly once semantics). Answering to these questions based on the criteria we identified will help the engineers make the right choice quicker. Further, the use case of our model is not limited to the choice only. Our model can be extended to serve to design a future stream processing framework architecture. When designing the solution, the model can help to see further steps on what will be implemented and thus the different dependencies it will have : when implementing the fault tolerance, the latency will increase or decrease given on how it is implemented. More over, thanks to the model, we see that the fault tolerance will also influence the message semantics. So based on what we want to have as message guarantees, we will implement the fault tolerance in a different manner. In the other hand, researchers can use this model when wanting to evaluate a framework architecture. Also, this model, can be reused in order to compare different frameworks. When wanted, as part of their research, they can have a quicker and a better view on the different solution and what brings to them and how they are different and also similar. More over, when wanted and depending on their need, they can easily extend this quality model in order to adapt it to their work : adding a criteria will add complexity, and thus a possible different path.

9 Conclusion & Future work

With the huge amount of data generated, and given a stream processing context, choosing the right framework architecture is major. In order to do that,

we first identified and explained what are the different criteria such as data model and latency... and presented some stream processing frameworks. We explained our methodology on how we came to choose the ideal framework architecture to fulfill user's needs. Given these, we provided a decision model tree which is a quality model to choose and evaluate a stream processing framework.

There is more work that has to be done, in order to have more criteria and frameworks, thus to have a more complete and complex model. We can base on this model to evaluate and choose a framework architecture, and not only that, this model can also serve as a guide to designing a new stream processing framework architecture. It can also be used as a support to have quickly a global view of the different solution and what brings to them depending on the different criteria.

References

1. http://storm.apache.org
2. http://spark.apache.org
3. A Framework for Real-time Streaming Analytics using Machine Learning Approach, Proceedings of National Conference on Communication and Informatics-2016
4. http://kafka.apache.org
5. Michael Stonebraker, Uur etintemel, Stan Zdonik. The 8 requirements of real-time stream processing. ACM SIGMOD Record Homepage archive, Volume 34 Issue 4, December 2005, Pages 42-47.
6. Supun Kamburugamuve and Geoffrey Fox : Survey of Distributed Stream Processing.
7. Fangjin Yang, Gian Merlino, Nelson Ray, Xavier Laut, Himanshu Gupta, Eric Tschetter : The RADStack: Open Source Lambda Architecture for Interactive Analytics.
8. http://samza.apache.org
9. http://flink.apache.org
10. Andre Luckow, George Chantzialexiou, Shantenu Jha. Pilot-Streaming: A Stream Processing Framework for High-Performance Computing
11. Supun Kamburugamuve, Geoffrey Fox : Survey of Distributed Stream Processing
12. Michael Stonebraker, Uur etintemel, Stan Zdonik: The 8 Requirements of Real-Time Stream Processing
13. Karan Patel, Yash Sakaria, Chetashri Bhadane : REAL TIME DATA PROCESSING FRAMEWORKS
14. Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, Engelbert Nguifo : A Comparative Study on Streaming Frameworks for Big Data
15. Apache Spark. Apache spark: Lightning-fast cluster computing, 2015
16. Apache Samza. Linkedins real-time stream processing framework by riccomini 2014
17. Apache Flink. Scalable batch and stream data processing, 2016
18. Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al : Storm @Twitter. In proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Pages 147-156
19. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. HotCloud, 10(10-10):95, 2010
20. Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R Henry, RobertBradshaw, andNathanWeizenbaum. Flumejava: easy, efcientdata-parallel pipelines. In ACM Sigplan Notices, volume 45, pages 363375. ACM, 2010
21. Nishant Garg. Apache Kafka. Packt Publishing Ltd, 2013
22. https://medium.com/netflix-techblog/keystone-real-time-stream-processing-platform-a3ee651812a
23. https://medium.com/netflix-techblog/whats-trending-on-netflix-f00b4b037f61