Apache Flink in current research

Article *in* it - Information Technology · January 2016

DOI: 10.1515/itit-2016-0005

Jonas Traub
Technische Universität Berlin
19 PUBLICATIONS **110** CITATIONS
SEE PROFILE

Asterios Katsifodimos
Delft University of Technology
39 PUBLICATIONS **962** CITATIONS
SEE PROFILE

Volker Markl
Technische Universität Berlin
205 PUBLICATIONS **4,508** CITATIONS
SEE PROFILE

Tilmann Rabl*, Jonas Traub, and Volker Markl

Apache Flink in Current Research Projects

Abstract: Recent trends in data collection and the decreasing prices of storage result in constantly growing amounts of analyzable data. These masses of data cannot easily be processed by traditional database systems as these do not allow for a sufficient degree of scalability. Programs especially designed for parallel data analysis on large scale distributed systems are required. Developing such programs on clusters of commodity hardware is a complex challenge for even the most experienced system developers. Frameworks such as Apache Hadoop are scalable, but – when compared to SQL – extremely hard to program. The open-source platform Apache Flink is a link between conventional database systems and big data analysis frameworks. Flink is based on a fault tolerant runtime for data stream processing, which manages the distribution of data as well as communications within the cluster. A high diversity of use cases can be supported through various interfaces that allow for the implementation of data analysis processes. An active community is continually updating and further developing the platform. It is both the result and the foundation of a great number of research works in the field of big data and information management. In this paper, we present an overview of Apache Flink as well as some current research activities on top of the Apache Flink ecosystem.

1 Introduction

The amount of accessible data is growing rapidly due to ever decreasing costs in data storage, cloud-storage, and the intensified usage of the Internet. The general value of data analysis is out of question, yet data evaluation poses a huge challenge. Conventional database systems are no longer able to deal with such enormous amounts of data. Dynamic or missing structures in the data add to the problem.

The Stratosphere [3] research project aims at building a next generation big data analysis platform, which will make it possible to analyze massive amounts of data in a manageable and declarative way. In 2014 Stratosphere

***Corresponding Author: Tilmann Rabl, Jonas Traub, Volker Markl:** Technische Universität Berlin, FG DIMA

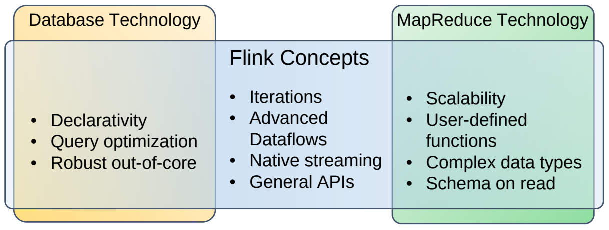

Fig. 1. Sources of technological concepts of the Apache Flink Platform

was open-sourced by the name Flink[1] as an Apache Incubator project. It graduated to Apache Top Level project in the same year.

In comparison with other distributed data analysis systems, Flink offers the user a reduced level of complexity through the integration of traditional database concepts such as declarative query languages and automatic query optimization. At the same time Flink allows for *schema on read*[2]. It further allows for user specific functions and is compatible with Apache Hadoop [3]. The platform offers a very high level of scalability. It was tested on clusters with several hundred nodes, on Amazon's EC2, and on Google's Compute Engine. Besides using concepts of existing database and MapReduce technologies, Apache Flink introduces additional concepts such advanced dataflows and native iterations. This is depicted in Figure 1.

The architecture of the Flink platform is described in Section 2. Libraries, interfaces, and a programming example are presented in Section 3. Section 4 addresses special features within the data stream analysis of Apache Flink as compared to other platforms. Section 5, presents several running research projects that use Apache Flink as a basis. We present related in Section 6, before concluding in Section 7.

1 http://flink.apache.org
2 With *schema-on-read* data are stored in their original format and without the definition of a data base schema. It is only on reading that the data will be transformed into a query specific schema. This allows for a high level of flexibility
3 http://hadoop.apache.org

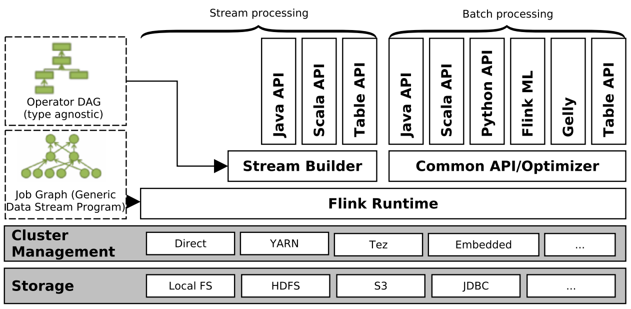

Fig. 2. Architecture and components of the Apache Flink platform

2 Architecture

Figure 2 gives an overview of the Apache Flink architecture. The foundation of Flink is a unified runtime environment in which all programs are executed. Programs in Flink are structured as directed graphs (JobGraphs) of parallelized operations that can further contain iterations [10]. A JobGraph consists of nodes and edges. There are two classes of nodes: (stateful) operators, and (logical) intermediate results (IRs). When running a program in Flink, operators are translated into various parallel entities, which consequently process partitions of intermediate results (or input files), offering data parallelism. Unlike Hadoop, programs in Flink are not divided into individual phases that are executed sequentially (Map and Reduce). Instead all operations are executed in parallel. The results of an operator are then directly forwarded to following operator to be processed, which results in a pipelined execution. Flink programs written using one of the many APIs, as described in the next section, are translated internally into abstract data flow programs. These are then transformed into execution plans using logical and physical cost-based optimization. These can then be executed in the engine. The scheduler decides on the operator placement and tries to exploit data locality where possible.

Flink provides a distributed runtime environment for clusters and also a local runtime environment. Programs can, therefore, be run and debugged right in a local development environment easing development. The distributed engine adapts the execution plan to the cluster environment and, thus, can run different plans based on the environment and data distribution. Flink is compatible with a number of cluster management and storage solutions, such as Apache Tez[4], Apache Kafka[5] [13], Apache HDFS[3] [14], and Apache Hadoop YARN[3] [16].

Stream Builder and *Common API* translate between the runtime environment and the interfaces (API) by transforming directed graphs of logical operations into generic data stream programs that are executed in the runtime environment. The automatic optimization of data flow programs is included in this process. The integrated optimizer for example chooses the best concrete join-algorithm for each respective used case, with the user only specifying an abstract join operation.

The following overview shows the upper layer of the Flink architecture. It consists of a wide spectrum of libraries and programming interfaces.

3 Libraries and Interfaces

Apache Flink users can specify their queries in various programming languages. A Scala and a Java API are available for the analysis of data streams and batch processing respectively. Batch data can further be processed via a Python API. All APIs offer the programmer generic operators such as Join, Cross, Map, Reduce, and Filter. In this Flink differs from Hadoop MapReduce, which only allows for complex operators to be implemented as a sequence of map and reduce phases. Furthermore, users can specify arbitrary user defined functions. Listing 1 shows a word count implementation with the Scala Stream Processing API. Analogous to this example an implementation to batch process is possible with the omission of the window specification.

```
1 case class Word (word: String, frequency: Int)
2 val lines: DataStream[String]
3     = env.fromSocketStream(...)
4 lines.flatMap{line => line.split(" ")}
5       .map { (_, 1) }
6       .keyBy(0)
7       .timeWindow(Time.of(5, TimeUnit.SECONDS))
8       .sum(1)
```

Listing 1. Wordcount implementation using Apache Flink's Scala stream processing API.

In the first line a tuple consisting of a string and an integer is defined. Line 2 indicates a Socket Stream, which reads a text data stream line by line. In Line 4,

4 http://tez.apache.org
5 http://kafka.apache.org

a FlatMap-Operator is applied, which obtains lines as input, divides these by blank spaces, and converts the resulting single words into the previously defined tuple format with the word as string and 1 as numeric value. Since this is a data stream query, a window is specified. This window is a sliding window with a duration of five seconds. Finally the words are grouped and the numeric values are added up within the various groups. The print method outputs the result on the console.

In addition to its classical interfaces the *FlinkML* library offers a number of algorithms and data analysis pipelines for machine learning. *Gelly* enables graph analysis with Flink. The *Table API* allows for declarative specifications of queries similar to *SQL*. It is available as Java and Scala version. Listing 2 shows a word count implementation with the Java Table API for batch processing.

```
1 DataSet<WC> input
2     = env.fromElements(new WC("Hello",1),
3       new WC("Bye",1),new WC("Hello",1));
4 Table table = tableEnv.fromDataSet(input)
5     .groupBy("word")
6     .select("word.count as count, word");
7 tableEnv.toDataSet(table, WC.class).print();
```

Listing 2. A word count implementation with the Java Table API for batch processing

Initially the input is explicitly created. Line 4 first converts the *DataSet* to a table to then group it according to the attribute *word*. Just like in SQL the select command chooses the word as well as the sums of numerators. The result table is finally converted back into a *DataSet* and printed.

4 Stream Processing on Flink

Data stream processing is significantly different form batch processing: programs have long (theoretically infinite) run times, they continually consume data of input streams and in return produce output streams. Aggregations can, however, only be calculated for closed blocks of data. In data stream programs these are, therefore, preceded by a discretization, which divides a data stream into closed, potentially overlapping windows. The aggregations proceed window by window.

In contrast to many other data analysis platforms, Flink, because of its runtime environment, is not bound to limitations resulting from micro-batching techniques [18]. Micro-batching interprets data streams as a sequence of data blocks of fixed size that are processed in separate batches; in order to calculate an overall result of the block results, the size of all windows have to be multiples of the block size. Flink offers far more flexible discretization options that are generalizations of IBM SPL's trigger and eviction policies [11]. A trigger policy indicates when a window ends and the aggregation for this window is processed. The eviction policy specifies the size of a window by determining when a tuple is removed from the window buffer.

Flink's Notion of Time. Flink distinguishes (both at the API and at the implementation level) between different two notions of time:

1. Event time is the time that an event happened (e.g., the time that a sensor emitted a signal, or the time that a person tapped on their smartphone). Event time is user-defined, and typically embedded in the data records themselves as a timestamp.
2. Processing time is the wall-clock time of the machine that is processing the data.

In distributed systems, there is an arbitrary lag between event time and processing time [2]. To compensate for arbitrary delays, Flink and other streaming systems that offer event time functionality rely on a notion of ”watermarks” or control events [1]. Flink programs that are based on processing time rely on the machine clocks and hence a less reliable notion of time, but exhibit the low latency. Programs that are based on event time provide reliable semantics, but may exhibit latency due to event time-processing time lag.

Users can choose from a number of pre-defined policies (for example based on time or counters) or implement custom policies. With a lower latency Flink achieves a higher degree of expressiveness as micro-batch-dependent systems and avoids the complexity of Lambda-architectures.

State and Fault Tolerance. Operators in Flink can be stateful. A snapshot algorithm ensures that even in the case of an error every tuple is represented only once within the operator status and will be processed accordingly. Flink offers a unique combination of batch processing, native data stream processing without the limits of micro-batching, stateful operators, expressive APIs, and *exactly-once* guaranties [6].

5 Current Research Projects

In this section, several research projects that have been proposed around the Apache Flink system are presented.

The presentation focuses on the applications and the technological advances that are defined in the projects.

5.1 Berlin Big Data Center

The Berlin Big Data Center (BBDC)[6] is a competence center for big data funded by the German Federal Ministry of Education and Research. Its goal is to enable large scale data analysis without requiring deep understanding of distributed systems. The reason for this is the talent gap in data science, where there are more professionals trained on doing data analysis than professionals trained on large scale distributed systems and only a small intersection of both groups that understand both. To fill this gap, the principal goal of the BBDC is to develop declarative ways of doing data analysis and machine learning and, thus, empowering data scientists with limited or no background in systems programming to do large scale data analysis. To this end, four concrete use cases that cover a broad range of data analysis tasks are adressed: video mining, text analytics, information-based medicine, and material science. While each of these use cases uses special methods and algorithms for data analysis, each poses a a big data challenge. Even though the processing of the data itself needs to be fast and scalable, the specification and adaption of data analysis programs requires to be fast as well to ensure overall efficiency. In Figure 3, the sources of latency in an end-to-end iterative data analysis pipeline are shown. It can be seen that data scientists efficiency, i.e., the human latencies, are a dominant factor on the critical path of data analysis. Unlike system latency, which can be improved by building scalable systems and using stronger hardware, human latency can be improved by making data analysis systems and tools available to a larger group of people, interactive, and easy to use.

The BBDC aims at building declarative languages and libraries for machine learning on big data systems and specifically Apache Flink. Furthermore, new ways of debugging, fault tolerance, and parallelization of big data analysis programs are researched. Finally, new network and file system models are incorporated to improve performance.

5.2 Proteus

Proteus[7] is a research project funded by the European Union, which aims at using Apache Flink for scalable online machine learning for predictive analytics and real-time interactive visualization in the area of smart industries, a.k.a., Industry 4.0. As a concrete use case steel manufacturing is used. Defects introduced in early processes of steel production have a great economic impact due to the costs of posterior transformations prior to detecting the defect. The sooner defects are detected, the sooner the process can be modified in order to stop producing defective subsequent coils and reassign new quality grade to already produced material.

A key phase of the steel production is performed in the hot strip mill. A hot strip mill is an installation where steel is transformed from slabs to coils after heating the material and then laminating it through rolls at high pressure and high temperature while keeping the steel under controlled tension, and finally cooling under in a pre-programmed cooling curve by using water showers in a continuous process. All processes are monitored using real-time sensors that produce extremely large and diverse structured and unstructured data streams. In this phase, it is necessary to deal with a continuous learning process as steel composition varies continuously, and so does its mechanical behavior. There are different types of steels and most of steel grades produced in 2015 did not exist five years earlier. Another problem that should be faced is the lack of data due to sensor malfunction. To address these challenges, new scalable online machine learning techniques have to be defined, implemented, and validated in an actual industrial scenario like this. Visualization methods for understanding the process are also needed. By relying on visual interactive interfaces, a better perception of the data and the analysis outcome will be enhanced. A clearly identified gap in the process executed in hot strip mill is to use data and quality parameters to understand the impact of different process parameters on dimensional defects. These will help to signal defects in real-time, while coils are still under production. By using appropriate massive online analysis techniques for mining the big data streams generated during the process, it is expected to achieve a reduction of 20% of defections coils and reducing rejected material by 15%.

One of the main contributions of Proteus is to develop innovative data analytics algorithms for massive online data to deal with various data engineering tasks, from

6 http://www.bbdc.berlin

7 http://www.proteus-bigdata.com/

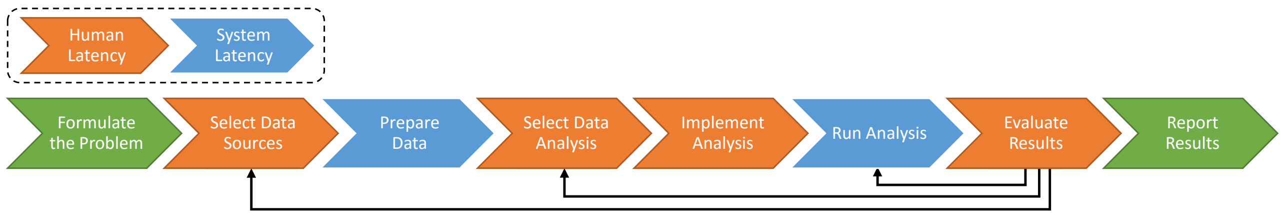

Fig. 3. Human Latency in Big Data Analysis

mining knowledge to learning from data streams. The project in particular investigates and develops a novel library on top of Flink that entails real-time analytics algorithms. Through the use of an optimized implementation of combined batch and streaming processing and building around this later scalable real time distributed online data analytics algorithms will be developed. A range of strategies is investigated in Proteus including pattern discovery, event detection, anomalies and novelty detection from streaming data.

5.3 Streamline

Today's big data analytics systems cater to either "data at rest" or "data in motion." As a result, enterprises are left to devise costly strategies to support and integrate disparate systems. To alleviate this burden, the STREAMLINE project[8], aims to reduce complexity, enable faster results, and reduce cost by supporting analysis on "big data at rest" and "fast data in motion" in a single system. STREAMLINE research and innovation actions include carrying out research in the areas of distributed systems, data management, and machine learning, with the key goal to arrive at sustainable innovation by technology transfer to an established and growing open source project. STREAMLINE targets four reactive and proactive analytics applications: customer retention, personalized recommendation, targeted advertisement, and multilingual Web processing.

More and more businesses require online prediction that goes beyond highly reactive systems management. Instead, they want to enable proactive event management and facilitate context-based recommendations and user profiling. Typical current setups handle data in batches from multiple origins, which cannot be extended trivially to allow for real-time exploration of heterogeneous, high-velocity data from various sources. Usually, data driven businesses all suffer from a heterogeneity of tools, all serving only small parts of their business needs, and all requiring high level expertise in the given programming paradigm that makes existing tools out of reach for their business. Contextualization promises to significantly increase the relevance of data analytics for a large number of businesses. However, currently, it remains hard to implement due to limits in streaming processing techniques.

The main STREAMLINE concept is the significant reduction of system and human latencies in big data analytics. Present big data deployments and tools could not yet overcome the system latency when combining huge data collections at rest with fast data in motion, resulting in very slow response to fresh data. Existing technologies for combing data streams and big data at rest are very complex and difficult to use, and most companies cannot find the expertise to solve their big data analysis needs, causing human latency in their business operations. The separation of offline and online processing on the optimization and application library / domain specific language level is the key issue, as data streams (real-time content) and data sets (historic content) are handled differently in the system and there are no means of interacting between them, which makes it very difficult, even for experienced professionals, to implement their application needs.

STREAMLINE aims at solving this issue for four concrete business cases: User modeling in quadruple play services (landline, mobile phone, internet, IPTV), recommendation generation in online video and music streaming, online analytics in mobile games, and web-scale data extraction.

5.4 Emma

Current data-parallel analysis languages and APIs like the one of Flink, suffer from a lack of declarativity or expressiveness to capture the complexity of today's data-parallel analysis requirements. Emma [4] strives to overcome these limitations by proposing a language for scalable data analytics deeply embedded in Scala. Emma's

8 https://streamline.sics.se/

approach argues that usability can be improved by reducing the amount of low-level parallelism constructs exposed to the programmer by platforms such as Flink, Spark, and Hadoop. Emma proposes a simplified parallel collection processing API that provides proper support for nesting and alleviates the need of certain second-order primitives through comprehensions – a declarative syntax akin to SQL. Emma's compiler is based on a metaprogramming pipeline that performs algebraic rewrites and physical optimizations which allow targeting parallel dataflow engines like Spark and Flink with competitive performance to hand-tuned low-level code.

The Emma project, currently focuses on an optimizer that can propagate interesting properties across different dataflows and generate the dataflows just in time. Moreover, a linear algebra API is specified and formal extensions that allow to mix the use of data bags, comprehensions, matrices, and vectors are investigated.

6 Research Results

Flink is the result as well as the foundation of a variety research projects. The most important publications are listed in the following. Warnecke et al. present the Nephele runtime environment [17], on which Flink's runtime was originally based. Battré et al. complement it with the PACT Model [5], an extension of MapReduce [8]. Alexandrov et al. offer a detailed description of the Stratosphere platform [3]. Hueske et al. consider the optimization of userdefined/custom functions [12]. Ewen et al. introduce the native support of iterations [10]. Current projects consider fault tolerance [9]. Spangenberg et al. compare the performances of Flink and Spark for various algorithms [15]. A recent analysis by Yahoo! compares the performance of Flink, Storm, and Spark [7].

7 Conclusion

Flink simplifies the parallel analysis of large amounts of data by using traditional database techniques such as automatic optimization and declarative query languages. Expressive, intuitive APIs allow for batch as well as data stream processing. Flink is scalable and versatile due to it high compatibility. Operators are processed in a pipeline, parallel, and free of limitations caused by micro-batching techniques.

Several active research projects use the Apache Flink platform to build next generation big data analysis methodologies. In this paper, we presented the Berlin Big Data Center, Proteus, Streamline, and EMMA, all of which use or extend Flink.

Acknowledgment: This work has been supported through grants by the German Ministry for Education and Research as Berlin Big Data Center BBDC (funding mark 01IS14013A) as well as by the DFG research group Stratosphere (FOR 1306) and also through grants by the European Union's Horizon 2020 research and innovation program under grant agreement 687691 for Proteus and 688191 for Streamline.

References

[1] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. Millwheel: fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044, 2013.

[2] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015.

[3] A. Alexandrov, R. Bergmann, S. Ewen, J. C. Freytag, F. Hueske, A. Heise, and D. Warneke. The stratosphere platform for big data analytics. *The VLDB Journal—The International Journal on Very Large Data Bases*, 23(6):939–964, 2014.

[4] A. Alexandrov, A. Kunft, A. Katsifodimos, F. Schüler, L. Thamsen, O. Kao, and V. Markl. Implicit parallelism through deep language embedding. In ACM, editor, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 47–61, 2015.

[5] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In ACM, editor, *Proceedings of the 1st ACM symposium on Cloud computing*, pages 119–130, 2010.

[6] P. Carbone, G. Fóra, S. Ewen, S. Haridi, and K. Tzoumas. Lightweight asynchronous snapshots for distributed dataflows. *arXiv preprint arXiv:1506.08603*, 2015.

[7] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky. Benchmarking streaming computation engines at yahoo! online, Dec 2015. http://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] S. Dudoladov, C. Xu, S. Schelter, A. Katsifodimos, S. Ewen, K. Tzoumas, and V. Markl. Optimistic recovery for iterative dataflows in action. In ACM, editor, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1439–1443, 2015.

[10] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *Proceedings of the VLDB Endowment*, 5(11):1268–1279, 2012.

[11] B. Gedik. Generic windowing support for extensible stream processing systems. *Software: Practice and Experience*, 44(9):1105–1128, 2014.

[12] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the black boxes in data flow optimization. *Proceedings of the VLDB Endowment*, 5(11):1256–1267, 2012.

[13] J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. In *In Proceedings of the NetDB*, pages 1–7, 2011.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In IEEE, editor, *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[15] N. Spangenberg, M. Roth, and B. Franczyk. Evaluating new approaches of big data analytics frameworks. In S. I. Publishing, editor, *Business Information Systems*, pages 28–37, 2015.

[16] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In ACM, editor, *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5, 2013.

[17] D. Warneke and O. Kao. Nephele: efficient parallel data processing in the cloud. In ACM, editor, *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, page 8, 2009.

[18] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In U. Association, editor, *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, page 10, 2012.