Review Paper on Batch Processing and Stream Processing

Dilraj bhullar¹, Nitin Verma²

¹Research Scholar, M.Tech (CSE), ²Assistant Professor (CSE)

¹CGC Technical Campus, I.K.Gujral Punjab Technical University, Jalandhar
²bhullardilraj@gmail.com, ²nitinverma.cgc@gmail.com

Abstract: Big data is a term that describes the large volume of data both structured and unstructured that inundates a business on a day-to-day basis. But it’s not the amount of data that’s important. It’s what organizations do with the data that matters. Big data can be analyzed for insights that lead to better decisions and strategic business moves.

I. BIG DATA HISTORY & CURRENT CONSIDERATIONS

While the term “big data” is relatively new, the act of gathering and storing large amounts of information for eventual analysis is ages old. The concept gained momentum in the early 2000s when industry analyst Doug Laney articulated the now-mainstream definition of big data as the three Vs:

Volume Organizations collect data from a variety of sources, including business transactions, social media and information from sensor or machine-to-machine data. In the past, storing it would’ve been a problem – but new technologies (such as Hadoop) have eased the burden.

Velocity Data streams in at an unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time.

Variety Data comes in all types of formats—from structured, numeric data in traditional databases to unstructured text documents, email, video, audio, stock ticker data and financial transactions. At SAS, we consider two additional dimensions when it comes to big data:

Variability In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Is something trending in social media? Daily, seasonal and event-triggered peak data loads can be challenging to manage.

Complexity Today’s data comes from multiple sources, which makes it difficult to link, match, cleanse and transform data across systems. However, it’s necessary to connect and correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control.

II. WHY BIG DATA IS SO IMPORTANT

The importance of big data doesn’t revolve around how much data you have, but what you do with it. You can take data from any source and analyze it to find answers that enable cost reductions, time reductions, new product development and optimized offerings, and smart decision making. Combine big data with high-powered analytics, you can accomplish business-related tasks such as:[1]

• Determining root causes of failures, issues and defects in near-real time.

• Generating coupons at the point of sale based on the customer’s buying habits.

• Recalculating entire risk portfolios in minutes.

• Detecting fraudulent behavior before it affects your organization.

III. WHAT IS STREAMING DATA

Streaming Data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes). Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers [3].

This data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows, and used for a wide variety of analytics including correlations, aggregations, filtering, and sampling. Information derived from such analysis gives companies visibility into many aspects of their business and customer activity such as-service usage (for metering/billing), server activity, website clicks, and geo-location of devices, people, and physical goods—and enables them to respond promptly to emerging situations. For example, businesses can track changes in public sentiment on their brands and products by continuously analyzing social media streams, and respond in a timely fashion as the necessity arises [2].

IV. EXAMPLES OF STREAMING DATA

⇒ Sensors in transportation vehicles, industrial equipment, and farm machinery send data to a streaming application. The application monitors performance, detects any potential defects in advance, and places a spare part order automatically preventing equipment down time[4].

⇒ A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements.

⇒ A real-estate website tracks a subset of data from consumers’ mobile devices and makes real-time property recommendations of properties to visit based on their geo-location.

⇒ A solar power company has to maintain power throughput for its customers, or pay penalties. It implemented a streaming data application that monitors of all of panels in the
field, and schedules service in real time, thereby minimizing the periods of low throughput from each panel and the associated penalty payouts.

A media publisher streams billions of click stream records from its online properties, aggregates and enriches the data with demographic information about users, and optimizes content placement on its site, delivering relevancy and better experience to its audience.

An online gaming company collects streaming data about player-game interactions, and feeds the data into its gaming platform. It then analyzes the data in real-time, offers incentives and dynamic experiences to engage its players.

V. COMPARISON BETWEEN BATCH PROCESSING AND STREAM PROCESSING

Before dealing with streaming data, it is worth comparing and contrasting stream processing and batch processing. Batch processing can be used to compute arbitrary queries over different sets of data. It usually computes results that are derived from all the data it encompasses, and enables deep analysis of big data sets. Like Amazon EMR, Spark, Hadoop are examples of platforms that support batch jobs. In contrast, stream processing requires ingesting a sequence of data, and incrementally updating metrics, reports, and summary statistics in response to each arriving data record. It is better suited for real-time monitoring and response functions [16].

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Batch Processing</th>
<th>Stream Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data scope</td>
<td>Queries or processing over all or most of the data in the dataset.</td>
<td>Queries or processing over data within a rolling time window, or on just the most recent data record.</td>
</tr>
<tr>
<td>Data size</td>
<td>Large batches of data.</td>
<td>Individual records or micro batches consisting of a few records.</td>
</tr>
<tr>
<td>Performance</td>
<td>Latencies in minutes to hours.</td>
<td>Requires latency in the order of seconds or milliseconds.</td>
</tr>
<tr>
<td>Analyses</td>
<td>Complex analytics.</td>
<td>Simple response functions, aggregates, and rolling metrics.</td>
</tr>
</tbody>
</table>

Table 1: Batch processing vs Stream Processing

VI. ENGINES TO PROCESS BATCH PROCESSING AND STREAM PROCESSING

A. Apache Spark

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

B. Evolution of Apache Spark

Spark is one of Hadoop’s sub project developed in 2009 in UC Berkeley’s AMPLab by Matei Zaharia. It was Open Sourced in 2010 under a BSD license. It was donated to Apache software foundation in 2013, and now Apache Spark has become a top level Apache project from Feb-2014.

C. Features of Apache Spark

Apache Spark has following features.

- **Speed** – Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.

- **Supports multiple languages** – Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.

- **Advanced Analytics** – Spark not only supports ‘Map’ and ‘reduce’. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

D. Spark Built on Hadoop

The following diagram shows three ways of how Spark can be built with Hadoop components.

There are three ways of Spark deployment as explained below.

- **Standalone** – Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.
Hadoop Yarn – Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

Spark in MapReduce (SIMR) – Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

E. Components of Spark

The following illustration depicts the different components of Spark.

a. Apache Spark Core

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

b. Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called Schema RDD, which provides support for structured and semi-structured data.

c. Spark Streaming

Spark Streaming leverages Spark Core’s fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

d. MLlib (Machine Learning Library)

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of Apache Mahout (before Mahout gained a Spark interface).

e. GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

F. Apache Flink

This Apache Flink tutorial will help us in understanding what is Apache Flink along with Flink definition, Flink ecosystem components and various Flink APIs and libraries like Flink dataset API, DataStream API of Flink, Flink Gelly API, Flink CEP and Flink Table API. Apache Flink is an open source distributed data stream processor. Flink provides fast, efficient, consistent and robust handling of massive streams of events that can handle both batch processing and stream processing. Flink is the first and only open source framework that has been demonstrated to deliver (1) throughput of millions of events per second in moderate clusters, (2) sub-second latency of milliseconds, (3) exactly once semantics for application state and delivery with supported sources and sinks and (4) accurate results through its support for event time. Refer this guide to learn about the Major differences between Flink vs Spark vs Hadoop. There are different layers in the ecosystem diagram:

a. Storage / Streaming

Flink is only a computation engine that does not have any storage system. It is designed to read and write data from different storage systems as well as can consume data from streaming systems. Below are some of the storage / streaming systems from which Flink can read write data:

- HDFS – Hadoop Distributed File System
- Local-FS – Local File System
- HBase – NoSQL Database in Hadoop ecosystem
- MongoDB – NoSQL Database
- RBDBMs – Any relational database
- S3 – Simple Storage Service from Amazon
• Kafka – Distributed messaging Queue
• Flume – Data Collection and Aggregation Tool
• RabbitMQ – Messaging Queue

b. Deployment
Deployment / resource management is the second layer in Flink eco system. Flink can be deployed in following modes:
• Local mode – On single node, in single JVM
• Cluster – On multi-node cluster, with following resource manager
  1. Standalone – This is the default resource manager which is shipped with Flink
  2. YARN – This is very popular resource manager, it is part of Hadoop, introduced in Hadoop 2.x
• Mesos – This is a generalized resource manager.
• Cloud – on Amazon or Google cloud
c. Flink Kernel
The third layer is Runtime – the Distributed Streaming Dataflow, which is also called as kernel of Apache Flink. This layer provides distributed processing, fault tolerance, reliability, native iterative processing capability, etc.
d. APIs and Library
This API handles the data at the rest that is generated at various sources like by reading text or CSV files or from local collections and allows user to implement transformations like mapping, filtering, joining, grouping, etc. post which output is returned via sinks to write data to text or CSV files or to return the output to the client. It is mainly used for distributed processing. The batch application is also executed on the streaming runtime. Some best practices to follow while working in DataSet API are as follows:
• Use print() to fastly printing a dataset
• Use collect() for fast retrieval of dataset
• Use name() on an operator for easy searching in logs
It handles transformations on continuous stream of the data. To process live data stream it provides various operations like filtering, updating states, defining windows, aggregating, etc. it can consume the data from various streaming source like message queues, socket streams, files etc and can return the data via sinks for writing data to files or standard output like command line terminal. It supports both Java and Scala. Some best practices to follow while working in DataStream API are as follows:
• Use print() to fastly printing a datastream
• Use env.fromelements(..) or env.fromCollection(…) for getting data stream to start working with.

Now let’s discuss some DSL (Domain Specific Library) Tools
e. Table API
Table API enables users to perform ad-hoc analysis through language like SQL for relational stream and batch processing. It can be embedded in DataSet and DataStream APIs of Flink in both java and Scala. Actually it allows users to run SQL queries on the top of Flink that saves them from writing complex code for data processing. The key concept of the Table API is a Table that represents a table with relational schema. Tables can be created from a DataSet or DataStream, converted into a DataSet or DataStream, or registered in a table catalog using a Table Environment. Registered tables can be queried with regular SQL queries. A Table is always bound to a specific Table Environment. It is not possible to combine Tables of different Table Environments.
f. Gelly
It is the graph processing engine which allows users to run operations for creating, transforming and processing the graph. It also provides the library of algorithm to simplify the development of graph applications. Gelly can transform and modify graph using high level functions that are similar to those provided by batch processing APIs. Its APIs are available in both Java and Scala. Scala methods are implemented as wrappers on top of Java operations. Dataset of vertices and edges are used to represent graph in Gelly. A vertex is defined by its unique ID and a value and an edge is defined by its source ID, target ID, and value. Transformations and Utilities are the methods of Graph class that include transformations, graph metrics, mutations and neighborhood aggregations. End to end data analysis can be done through gelly on single system. Gelly API can be seamlessly mixed with DataSet Flink API for implementing applications that use both record based and graph based analysis.
g. FlinkML – Machine Learning for Flink
It is the machine learning library which provides intuitive APIs and efficient algorithm to handle machine learning applications in Apache Flink. As we know machine learning algorithms are iterative in nature flink provides native support for iterative algorithm to handle the same quite effectively and efficiently. It is written in the Scala FlinkML supports Supervised learning (Optimization framework, Multiple linear regression and SVM algorithms) and Unsupervised learning (k-Nearest neighbors join) along with Data Pre processing.
One of the key concepts of FlinkML is its scikit-learn inspired pipelining mechanism that allows building complex data analysis pipelines.
h. FlinkCEP – Complex event processing for Flink
FlinkCEP allows easy detection of complex event patterns in streams of data that is used for finding matching sequences to get insights of data. For comparing and finding matching events, it is necessary for datastream to implement equals() and hashcode() methods. Complex event patterns can be defined easily by pattern API. The patterns have different states in which user can define conditions for events. Various pattern operations include begin, next, followedBy, within etc. that can be written in both java and scala. CEP is used nowadays for large number of applications like for financial
applications such as stock market trend and credit card fraud detection. It is also used in RFID-based tracking and monitoring, for example, for detecting thefts in a warehouse where items are not properly checked out. It can also find its usage in detecting network intrusion by specifying patterns of suspicious user behaviour. To learn about more Flink use cases, refer Flink use case tutorial to get real time use cases of Apache Flink and how industries are using Flink for their various purposes.

REFERENCES

https://issues.apache.org/jira/browse/HIVE