**CS595 - Assignment 9**

***Reading Assignment:***

1. Read and provide a half page summary and analysis of this article available on the blackboard in the ‘Articles’ section: Beyond Batch Processing: Towards Real-Time and Streaming Big Data.

**Answer:** Following is the summary of “Beyond Batch Processing: Towards Real-Time and Streaming Big Data”

article:

* The article discusses two categories of non-batch workloads solutions in Big data domain:
  + **Real-time processing:**

The goal of real-time processing is to provide solutions that can process big data very fast and interactively. An example for real-time processing is fast and interactive queries on big data warehouses.

* + **Stream processing of big data**:

Stream processing deals with problems that their input data must be processed without being totally stored. The use case for for stream processing include online machine learning, and continuous computations.

* For each category the article highlights and presents paradigms, strengths and differences to Hadoop.
* **Real-Time Big Data Processing:** Solutions in this sector can be classified into two major categories:

* + **In-Memory Computing:**

Solutions that try to reduce the overhead of MapReduce and make it faster to enable execution of jobs in less than seconds.

It is based on using a distributed main memory storage system that can be used either as a standalone input source or as a caching layer for disk-based storages to store and process big data in real-time.

When the input totally fits in distributed memory or when the job has multiple iterations over input, in-memory computing can significantly reduce execution time.

* + **Real-Time Queries over Big Data:**

Solutions that focus on providing a means for real-time queries over structured and unstructured big data using new optimized approaches.

These solutions mostly use custom storage formats and well-known techniques from parallel DBMSs to join and aggregation, and hence can respond to queries in less than a few seconds. In the stream-processing sector, there are two popular frameworks: Storm, and S4.

* **Streaming Big Data:**

Today’s applications need more stream-like demands in which the input data is not available completely at the beginning and arrives constantly. Also, sometimes an application should run continuously.

Although MapReduce does not support stream processing, but it can partially handle streams using a technique known as micro-batching (implemented in Apache Spark Streaming). There are a few stream processing frameworks that are inherently designed for big data streams such as Storm from Twitter, and S4 from Yahoo.

**Business case for Real-time processing, and Stream processing of big data:**

* Hadoop is inherently designed for batch and high throughput job execution and it is suitable for jobs that process large volumes of data over a long time.

* Although Hadoop is very suitable for batch jobs, there is an increasing demand for non-batch requirements like: interactive jobs, real-time queries, and big data streams. Since Hadoop is not suitable for these non-batch workloads, new solutions are proposed to these new challenges.

* The article introduces two categories: real-time processing and streaming big data to deal with real-time (interactive) and streaming requirements in Big Data.

**Conclusion:**

* Considering high demands for interactive queries and big data streams, in-memory computing stands out as a notable solution that can handle both real-time and stream requirements. Apache Spark is a good example which supports in-memory computing using RDDs, real-time and interactive querying using Shark, and stream processing using fast micro-batching.

1. Read and provide a half page summary and analysis of this article available on the blackboard in the ‘Articles’ section: Real-time stream processing for Big Data.

**Answer:** Following is the summary of “Real-time stream processing for Big Data” article:

* The article gives an overview over the state of the art of stream processors for low-latency Big Data analytics and conduct a qualitative comparison of the most popular contenders, namely Storm and its abstraction layer Trident, Samza and Spark Streaming.
* It provides an overview over some of the most popular distributed stream processing systems currently available and highlight similarities, differences and trade-offs taken in their respective designs.
* **Real-time analytics:** 
  + In contrast to traditional data analytics systems that collect and periodically process huge – static – volumes of data, streaming analytics systems avoid putting data at rest and process it as it becomes available, thus minimizing the time a single data item spends in the processing pipeline.
* **Real-time processors:**
  + An important distinction between the individual systems that directly translates to the achievable speed of processing, i.e. latency, is the processing model. Handling data items immediately as they arrive minimizes latency at the cost of high per-item overhead (e.g. through messaging), whereas buffering and processing them in batches yields increased efficiency, but obviously increases the time the individual item spends in the data pipeline.
  + Purely stream-oriented systems such as Storm and Samza provide very low latency and relatively high per-item cost, while batch-oriented systems achieve unparalleled resource-efficiency at the expense of latency that is prohibitively high for real-time applications.
  + Systems like Storm Trident and Spark Streaming employ micro-batching strategies to trade latency against throughput: Trident groups tuples into batches to relax the one-at-a-time processing model in favor of increased throughput, whereas Spark Streaming restricts batch size in a native batch processor to reduce latency.
* **Storm:**
  + Its programming model provided an abstraction for stream-processing similar to the abstraction that the MapReduce paradigm provides for batch-processing.
  + Storm is scalable, fault-tolerant and even elastic as work may be reassigned during runtime. Storm provides reliable state implementations that survive and recover from supervisor failure.
  + A data pipeline or application in Storm is called a topology and is a directed graph that represents data flow as directed edges between nodes which again represent the individual processing steps: The nodes that ingest data and thus initiate the data flow in the topology are called spouts and emit tuples to the nodes downstream which are called bolts and do processing, write data to external storage and may send tuples further downstream themselves.
* **Storm Trident:**
  + Trident was released as a high-level API with stronger ordering guarantees and a more abstract programming interface with built-in support for joins, aggregations, grouping, functions and filters.
  + In contrast to Storm, Trident topologies are directed acyclic graphs (DAGs) as they do not support cycles, which makes them less suitable for implementing iterative algorithms and is also a difference to plain Storm topologies which are often wrongfully described as DAGs, but actually can introduce cycles.
  + Also, Trident does not work on individual tuples, but on micro-batches and correspondingly introduces batch size as a parameter to increase throughput at the cost of latency which, however, may still be as low as several milliseconds for small batches
* **Samza:**
  + It is a stream processor with a one-at-a time processing model and at-least-once processing semantics.
  + Samza is designed to handle large amounts of state in a fault-tolerant fashion by persisting state in a local database and replicating state updates to Kafka. By default, Samza employs a key-value store for this purpose, but other storage engines with richer querying capabilities can be plugged in.
* **Spark streaming:**
  + It offers several benefits in comparison to Hadoop, most notably a more concise API resulting in less verbose application logic and significant performance improvements through in-memory caching.
  + Spark Streaming shifts Spark’s batch-processing approach towards real-time requirements by chunking the stream of incoming data items into small batches, transforming them into RDDs and processing them as usual.

**Business case for Real-time processing, and Stream processing of big data:**

* The shift towards more dynamic and user-generated content in the web and the omnipresence of smart phones, wearables and other mobile devices, have led to an abundance of information that are only valuable for a short time and therefore have to be processed immediately.
* Only a part of the potential of today’s Big Data repositories can be exploited using traditional batch-oriented approaches as the value of data often decays quickly and high latency becomes unacceptable in some applications.
* Distributed data and stream processing systems have emerged that deviate from the batch-oriented approach and tackle data items as they arrive, thus acknowledging the growing importance of timeliness and velocity in Big Data analytics. The main achievement of these new systems is abstraction from scaling issues and thus making development, deployment and maintenance of highly scalable systems feasible.

**Conclusion:**

* Batch-oriented systems have done the heavy lifting in data-intensive applications for decades, but they do not reflect the unbounded and continuous nature of data as it is produced in many real-world applications.
* Stream oriented systems process data as it arrives and thus are often times a more natural fit, though inferior with respect to efficiency.
* Whether at the core of novel system designs such as the Kappa Architecture or as a complement to existing architectures such as the Lambda Architecture, horizontally scalable stream processors are gaining momentum as the requirement for low latency has become a driving force in modern Big Data analytics pipelines.