REAL TIME DATA PROCESSING FRAMEWORKS

Karan Patel, Yash Sakaria and Chetashri Bhadane

Department of Computer Engineering, University of Mumbai, Mumbai

ABSTRACT

On a business level, everyone wants to get hold of the business value and other organizational advantages that big data has to offer. Analytics has arisen as the primitive path to business value from big data. Hadoop is not just a storage platform for big data; it's also a computational and processing platform for business analytics. Hadoop is, however, unsuccessful in fulfilling business requirements when it comes to live data streaming. The initial architecture of Apache Hadoop did not solve the problem of live stream data mining. In summary, the traditional approach of big data being co-relational to Hadoop is false; focus needs to be given on business value as well. Data Warehousing, Hadoop and stream processing complement each other very well. In this paper, we have tried reviewing a few frameworks and products which use real time data streaming by providing modifications to Hadoop.

KEYWORDS

Data Mining; real time data analytics; apache storm; apache spark streaming; TIBCO StreamBase

1. INTRODUCTION

In every area of business, Business intelligence, analytics and data warehouse technologies are used to support strategic and optimum decision-making. These technologies provide users the ability to access large volumes of business data and thus visualizing results. Systems are limited by the underlying data management infrastructure, which is nearly always based on periodic batch updates, causing the system to lag real-time by hours and often a full day and almost always require pull-based querying, so new results are provided and submitted only when a client requests them. However, as the need for real time data started increasing, this inability to work on real time data starting becoming a major cause of concern for companies. This was when the need for a framework arose that could handle real time processing of data.

Having a lot of data pouring into your organization is one thing, being able to store it, analyze it and visualize it in real-time is a whole different scenario. More and more organizations want to have real-time insights in order to fully understand what is going on within their organization. This is where the need for a tool arises that can handle real time data processing and analytics.

DOI : 10.5121/ijdkp.2015.5504

2. HADOOP AND ITS CHALLENGES WITH RESPECT TO STREAM PROCESSING

Hadoop for the enterprise is driven by several upcoming needs. On a technology level, many organizations need data platforms to scale up to handle bigger than ever data volumes. They also need a scalable extension for prevailing IT systems in warehousing, archiving, and content management. Others need to finally get Business Intelligence value out of non-structured data. However, Hadoop was never built for real-time processing. Hadoop initially started with MapReduce, which offers batch processing where queries take hours, minutes or at best seconds. This is great for complex transformations and computations of big data volumes. However, it is not suitable for ad hoc data exploration and real-time analytics. The key to this problem is to pair Hadoop with other technologies that can handle true real-time analytics. Thus the most optimum solution is a combination of stream processing and Hadoop. Multiple vendors have thus made improvements and added capabilities to Hadoop that make it capable of being more than just a batch framework.

Let us now consider some of the frameworks and products developed in order to implement data stream mining.

3. APACHE STORM

3.1 Core Concept

YARN forms the architectural epicenter of Hadoop and allows multiple data processing engines such as interactive SQL, real-time data streaming, data science and batch processing to manage data stored in a single platform, giving a completely new direction to analytics. It forms the basis for the new generation of Hadoop. Having YARN at the center of its architecture, Hadoop draws attention of various new engines to run as organizations want to efficiently store their data it a single repository and work with it simultaneously in different ways. They want SQL, streaming, machine learning, along with traditional batch and more all in the same cluster. Apache Storm adds reliable real-time data processing capabilities to Enterprise Hadoop.

Storm is a distributed real-time computational system for processing and handling large volumes of high-velocity data. Storm on YARN is powerful for scenarios that require real-time analytics, machine learning and incessant monitoring of operations. Some of specific new business opportunities include: real-time customer service management, data monetization, or cyber security analytics and risk detection. Because Storm integrates with YARN via Apache Slider, YARN manages Storm while also considering cluster resources for data governance, security and operations components of a modern data architecture.

Storm is extremely quick, with the ability to compute over a million records per second per node on a cluster of median size. In order to prevent unwanted events or to maximize positive outcomes, Enterprises harness this speed and integrate it with other data access applications in Hadoop. A Hadoop cluster can efficiently process and compute a full range of workloads from real-time to interactive to batch with Storm in Hadoop on YARN.

The following characteristics make Storm most suitable for real-time data processing workloads:

- **Fast** – Ability to process one million 100 byte messages per second per node
- **Scalable** – with the facility of parallel calculations that run across a cluster of machines
- **Fault-tolerant** – when workers die, Storm will restart them automatically. If a node dies, the worker will be restarted on another node
- **Reliable** – Storm assures that each unit of data (tuple) will be processed at least once or exactly once. Messages are only replayed when there are failures
- **Easy to operate** – standard configurations are suitable for production on day one. Once deployed, Storm is easy to operate

3.2 Architecture [1]

A storm cluster has three types of nodes:

- **Nimbus node** (master node):
  - Uploads vital computations for execution
  - Distributes code across the cluster
  - Launches workers across the cluster
  - Monitors computation and reallocates workers wherever needed
- **ZooKeeper nodes** – communicates with the Storm cluster
- **Supervisor nodes** – communicates with Nimbus through Zookeeper and also starts and stops workers according to signals from Nimbus node

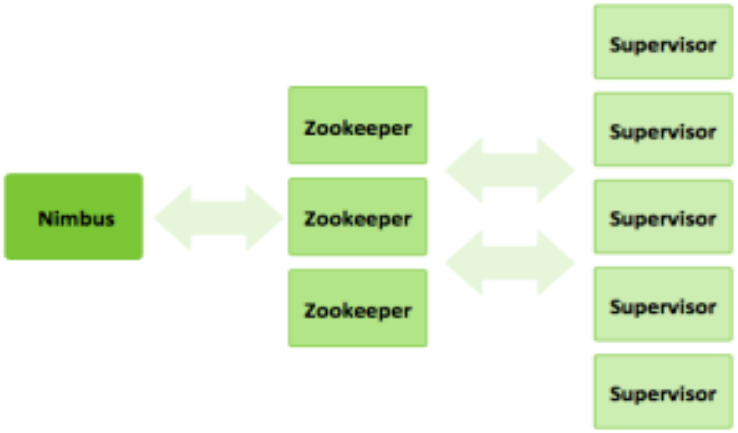

Figure 1 [1]

Five key abstractions which can help us to understand how Storm processes data are:

- **Tuples**– an ordered list of elements. For example, a “4-tuple” might be (7, 1, 3, 7)

- **Streams** – an unbounded sequence of tuples.
- **Spouts** –sources of streams in a computation (e.g. a Twitter API)
- **Bolts** – process input streams and produce output streams. They can: run functions; filter, aggregate, or join data; or talk to databases.
- **Topologies** – the overall calculation, represented visually as a network of spouts and bolts

When data comes streaming in from the spout, Storm users define topologies for how to process the data.

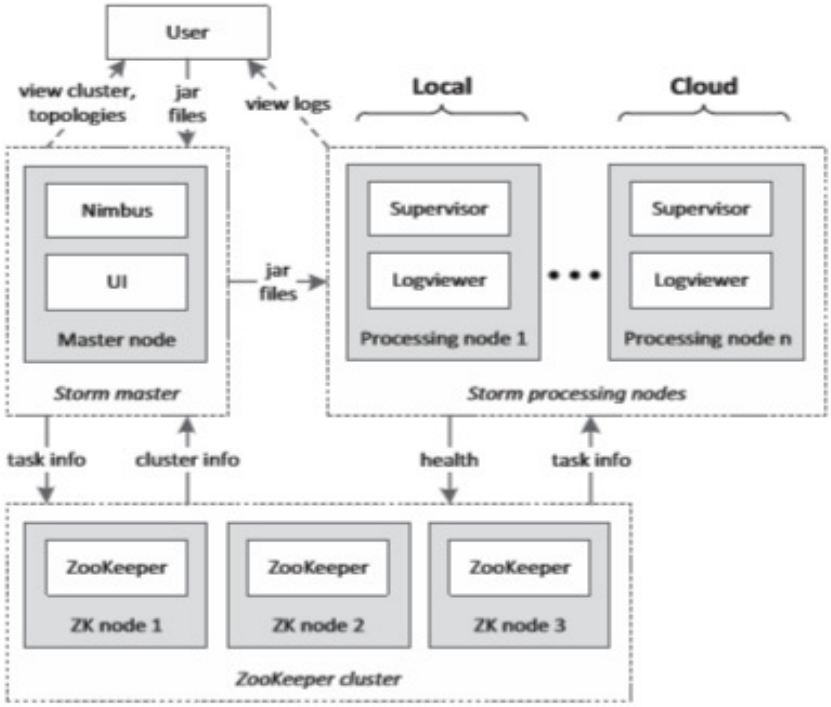

Figure 2. Apache Storm Architecture [2]

When the data comes in, it is processed and the results are passed into Hadoop.

3.3 Scaling Storm [2]

A Storm cluster contains a single master node, multiple processing nodes and a ZooKeeper cluster. No matter what the size of the cluster is, the master node is always fixed. The ZooKeeper cluster consists of an odd number of nodes as a majority of nodes need to be running for the ZooKeeper service to be available. A single machine is used for development, while a cluster of three or five is used for medium to large Storm clusters. The scaling of Storm is done by adding processing nodes to the cluster when more execution power is needed and removing them when

they are no longer needed. Storm provides the ability to users to add processing nodes to the platform. It directly finds these extra nodes when they are added, but doesn't use them until new topologies are created in the platform or existing ones are rebalanced.

The various functions of Storm scaling are a) the detection of overprovisioning and under-provisioning, b) the automatic addition of nodes in case of under-provisioning and rebalancing of existing topologies, and c) the automatic removal of nodes in the case of over-provisioning. In the field of autonomic computing, it is common to use the Monitor, Analyze, Plan, Execute (MAPE) loop to control a system. This loop is used to make Storm flexible, i.e. growing and shrinking of the Storm cluster based on the processing power that is needed by the topologies. The process starts with a sensor that checks how the Storm cluster is performing. The MAPE loop then monitors the output of the sensor, analyzes the measured values to detect problems, plans on a set of actions to solve the problem, and executes the selected actions. The process ends with an effector to influence the cluster.

4. TIBCO STREAMBASE

TIBCO StreamBase is a superior framework for quickly fabricating applications that investigate and follow up on continuous gushing real time information. The objective of StreamBase is to offer an item that assists designers in quickly assembling ongoing frameworks and deploying them effortlessly.

4.1 TIBCO StreamBase Live Data Mart

StreamBase Live Data View is a constantly live data shop that devours information from spilling ongoing data sources, makes an in-memory information distribution center, and gives push-based inquiry results and cautions to end clients.

Conventional examination and information distribution center frameworks have worked by configuration without-of-date data. In TIBCO Live Datamart, a convincing framework is produced without these restrictions, giving clients access to live, exact information, with related advantages in trust and responsiveness.

TIBCO Live Datamart is another way to deal with continuous investigation and data warehousing for situations where expansive volumes of information oblige an administration by special case way to deal with business operations. TIBCO Live Datamart joins systems from complex occasion handling (CEP), dynamic databases, online systematic preparing (OLAP), and data warehousing to make a live information distribution center against which nonstop questions are executed. [3] The subsequent framework empowers clients to make impromptu inquiries against a huge number of live records, and get push-based upgrades when the consequences of their questions change. TIBCO Live Datamart is utilized for operational and danger observing in high recurrence exchanging situations, where contingent alarming and mechanized remediation empower a modest bunch of administrators to oversee a great many transactions for each day, and make the aftereffects of trading obvious to several clients progressively. TIBCO Live Datamart is likewise utilized as a part of retail to track stock, and in discrete assembling for deformity administration. [3]

TIBCO Live Datamart is built on top of the TIBCO StreamBase Event Processing Platform and benefits from the CEP capabilities of that platform. The major components of TIBCO Live Datamart are as follows:

- Connectivity
- Continuous query processing
- Aggregation
- Alerting and notification
- Client connectivity

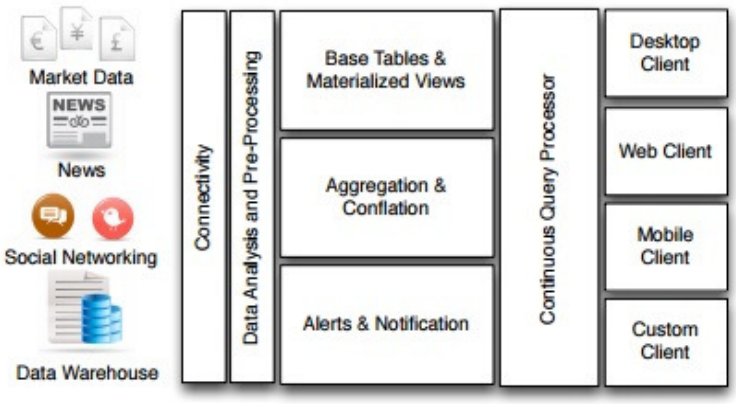

Figure 3 [4]

4.2 Architecture

The StreamBase LiveView Continuous Query Processor (CQP) is at the heart of the framework, effectively overseeing a large number of simultaneous queries against a live information set. The objective of the CQP is to take a predicate and the present substance of a table and first to give back a preview containing the columns that match the predicate. The processor then conveys persistent redesigns as the qualities in the table change. Upgrades incorporate supplements, erases, set up redesigns of columns, overhauls that bring new lines into the perspective, and redesigns that take lines out of the perspective. A key configuration rule of the CQP is to proficiently bolster substantial prejoined, de-standardized tables normally found in information warehousing, overseeing information in a fundamental memory database. The CQP works by sharding so as to index both questions and information, and records over different CPU centers. Preview queries are executed utilizing ordered as a part of memory tables, with various strings all the while filtering distinctive scopes of the file. Nonstop redesigns are conveyed utilizing indexing, however for this situation the questions are filed, not the information. Given an inquiry predicate, for example, "symbol='IBM' && amount > 1000", the question may be set into a table of enrolled inquiries ordered by image, under the key 'IBM'. At the point when an overhaul to a record comes in, the pre-and postimage of that record are checked against this table to check whether matches the key "IBM." Given a match whatever is left of the predicate (i.e. "amount > 1000") is checked and a redesign is dispatched to customers of that query. [4]

4.3 Materializing Views

These are the basis of complex analysis in TIBCO Live Datamart. Materialized views in TIBCO StreamBase LiveView are pre-computed, and then continuously or periodically updated using calculations based on one or more live data tables or on other materialized views. Materialized views include aggregations, joins, or alerts.

Aggregations are creator time or specially appointed. Creator time Aggregations can be arranged into a live information table based upon basic accumulations that clients will need to queries. A solitary table can be designed with different collections. Impromptu accumulations are issued in an inquiry. Both sorts of conglomeration influence the full force of the implicit question dialect, LiveQL, which incorporates SQL-like expressions, as well as extra inquiry modifiers for gushing information.

Joins, for example, joining evaluating data with positions, can be driven by any CEP rationale, which incorporates combinations with different dialects like Java, Matlab, or R, prompting almost unbounded adaptability.

Alerts empower administration by-exemption. The framework will ready administrators of chances or looming issues, permitting them to invest a large portion of their energy reacting to these cautions. Cautions are characterized by conditions against a base table, and are another sort of emerged perspective, with columns coordinating the conditions being held on in ready tables until the alarm is cleared somehow. A few alerts, for example, stuck requests, will clear naturally when the issue is determined. Different alarms, for example, an over the top number of business sector information abnormalities, must be cleared physically. Alarms then commute warnings, messages conveyed to a client through some interface. Warnings can come complete with setting and prescribed remediation. Connection is sufficient data for the administrator to comprehend the warning, by and large the consequences of inquiries that the administrator would be relied upon to require. Connection may likewise incorporate authentic and additionally live data. Alarms can likewise encourage into case administration. Criticism from the client interface or other framework can advise the server when a caution is being taken care of, who is taking care of it, and when it is determined. Determination conditions can be controlled physically by the client, or with mechanization, for example, when the framework recognizes the ready no more applies. Alerts are auditable and may sustain different frameworks for case or ticket following.

5. APACHE SPARK STREAMING

5.1 Core Concepts

Spark Streaming is a fascinating augmentation to Spark that includes support for continuous stream processing to Spark. Sparkle Streaming is in dynamic improvement at UC Berkeley's amplab alongside whatever remains of the Spark venture. Most Internet scale frameworks have real time information necessities alongside their developing batch processing needs. Spark Streaming is composed with the objective of giving close constant live data processing with more or less one second latency to such projects. Probably the most ordinarily utilized applications with such prerequisites incorporate site measurements/investigation, interruption identification frameworks, and spam channels. Spark Streaming looks to bolster these queries while keeping up adaptation to non-critical failure like cluster frameworks that can recoup from both out and out

failures and stragglers. They moreover tried to augment cost-viability at scale and give a simple programming model. Finally, they perceived the significance of supporting incorporation with batch and ad-hoc query frameworks. With a specific end goal to comprehend Spark Streaming, it is vital to have a decent comprehension on Spark itself.

5.2 Spark Architecture [5]

Sparkle is an open source cluster registering framework created in the UC Berkeley AMP Lab. The framework plans to give quick calculations, quick composes, and profoundly intuitive queries. Spark fundamentally beats Hadoop MapReduce for certain issue classes and gives a straightforward Ruby-like translator interface.

Spark beats Hadoop by giving primitives to in-memory cluster processing; along these lines staying away from the I/O bottleneck between the individual jobs of an iterative MapReduce work process that over and again performs reckonings on the same working set. Spark makes utilization of the Scala language, which permits appropriated datasets to be controlled like nearby accumulations and gives quick improvement and testing through it's intuitive interpreter (like Python or Ruby).

Spark was likewise created to bolster intuitive data mining, (notwithstanding the undeniable utility for iterative calculations). The framework likewise has appropriateness for some broad figuring issues. Past these specifics Spark is appropriate for most dataset change operations and computations.

Moreover, Spark is based to keep running on top of the Apache Mesos cluster administrator. This permits Spark to work on a cluster one next to the other with Hadoop, Message Passing Interface (MPI), HyperTable, and different applications. This permits an association to create crossover work processes that can profit by both dataflow models, with the expense, administration, and interoperability worries that would emerge from utilizing two free groups. As "they" regularly say, there is no free lunch! How does Spark give it's speed and stay away from Disk I/O while holding the alluring adaptation to internal failure, territory, and versatility properties of Existing appropriated memory deliberations, for example, key-quality stores and databases allos fine-grained redesigns to impermanent state. This powers the bunch to duplicate data or log redesigns crosswise over machinces to keep up adaptation to internal failure. Both of these methodologies bring about significant overhead for an data serious workload. RDDs rather have a limited interface that just permits coarse-grained upgrades that apply the same operation to numerous data things, (for example, guide, channel, or join).

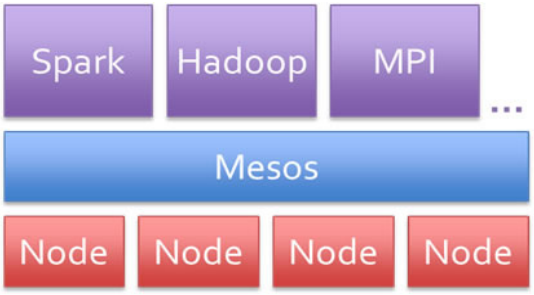

Figure 4 [5]

This permits Spark to give adaptation to internal failure through logs that just record the changes used to construct a dataset rather than all the data. This abnormal state log is known as a lineage. Since parallel applications, by their exceptionally nature, ordinarily apply the same operations to an extensive part of a dataset, the coarse-grained confinement is not as restricting as it may appear. Truth be told, RDDs can productively express programming models from various separate systems including MapReduce, Dryad LINQ, SQL, Pregel, and HaLoop.

Furthermore, Spark likewise gives extra adaptation to non-critical failure by permitting a client to determine a perseverance condition on any change which makes it promptly keep in touch with plate. Information region is kept up by MapReduce? The answer from the Spark group is a memory reflection they call a Resilient Distributed Dataset (RDD).

Permitting clients to control information apportioning taking into account a key in every record. (One sample utilization of this dividing plan is to guarantee that two datasets that will be joined are hashed in the same way.) Spark keeps up adaptability past memory constraints for output based operations by putting away segments that develop too expansive on circle.

As expressed beforehand, Spark is essentially intended to bolster cluster changes. Any framework that needs to make offbeat fine grain overhauls to shared state, (for example, a datastore for a web application) ought to utilize a more conventional framework, for example, a database, RAMCloud, or Piccolo. This figure gives a case of Spark's clump changes and how the different RDDs and operations are assembled into stages.

5.3 Spark Streaming Architecture [5]

Most traditional streaming systems involve an update and pass methodology that handles a single record at a time. These systems can handle fault tolerance by replication (fast, 2x hardware cost) or upstream backup/buffered records(slow,~ 1x hardware cost), but neither approach scales to hundreds of nodes.

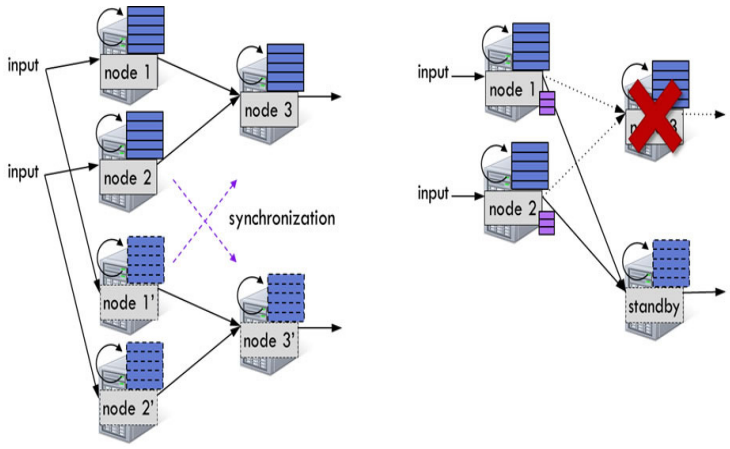

Figure 5 [5]

5.4 Observations from Batch Systems [5]

On account of these constraints the Spark Streaming group hoped to batch frameworks for motivation on the best way to enhance scaling and noticed that:

- batch frameworks dependably partition handling into deterministic sets that can be effectively replayed
- falied/moderate undertakings are re-keep running in parallel on different hubs
- lower clump sizes have much lower inactivity.

This lead to the thought of utilizing the same recuperation instruments at a much littler timescale. Spark turned into a perfect stage to expand on as a result of it's in memory stockpiling and streamlining for iterative changes on working.

5.5 Streaming Operations

Spark Streaming provides two types of operators to let users build streaming programs:

- Transformation operators, which produce a new DStream from one or more parent streams. These can be either stateless (i.e., act independently on each interval) or stateful (share data across intervals).
- Output operators, which let the program write data to external systems (e.g., save each RDD to HDFS).

Spark Streaming backs the same stateless changes accessible in run of the mill bunch structures, including guide, reduce, groupBy, and join (all administrators upheld in Spark are stretched out to this module).

Fundamental real-time streaming is a valuable apparatus in itself, however being capable the capacity to take a gander at totals inside of a given window of time is likewise imperative for most business analysis. Keeping in mind the end goal to bolster windowing and aggregation Spark Streaming advancement has made two assessment models.

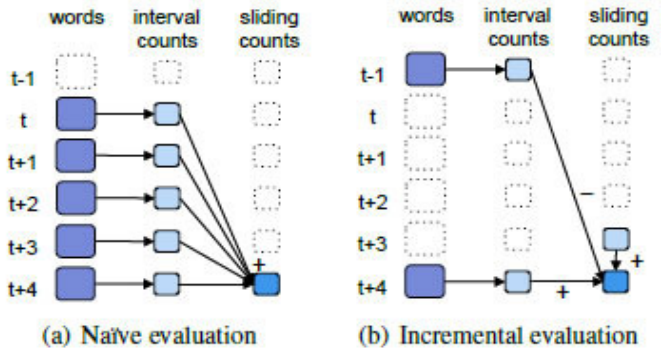

Figure 6 [5]

Figure 6 analyzes the credulous variation of reduceByWindow (a) with the incremental variation for invertible functions (b). Both variants register a for every interval tally just once, yet the second evades re-summing every window. Boxes signify RDDs, while arrows demonstrate the operations used to figure the window.

Spark likewise contains an administrator that backs time-skewed joins that permits clients to join a stream against its own particular RDDs from some time in the past to process patterns, for example, how current site visit checks contrast with those from five minutes prior.

For output Spark Streaming permits all RDDs to be composed to an upheld stockpiling framework utilizing the save administrator. Then again, the for each operator can be utilized to prepare each RDD with a custom client code snippet.

5.6 Integration with Batch Processing [5]

Since Spark Streaming is based on the same central data structure (RDDs) as Spark incorporating streaming data with other batch handling for analysis is conceivable and clients will have the capacity to create calculations that will be possibly versatile over their whole analysis stack.

The principle weakness in such manner is that Spark can't as of now share RDDs crosswise over procedure limits, as of now, no arrangement exists other than composing these RDDs to capacity. However an apparatus to empower the same is as of now being developed, yet there is no time allotment for a release.

6. COMPARATIVE STUDY OF THE THREE FRAMEWORKS

After studying all the three frameworks and discussing them in detail above, we have now done a comparative study of the three frameworks based on certain parameters. This comparison is as shown below:

Property	Apache Storm	StreamBase	Spark Streaming
Input Data Sources	There are connectors available for Event Hubs, Service Bus, Kafka, etc. Unsupported connectors may be implemented via custom code) [7]	JDBC Data Source [8]; Input View offers two ways to enter values to be sent to a stream: 1. forms-based input panel 2. JSON input panel	1. *Basic sources*: Sources directly available in the StreamingContext API. Examples: file systems, socket connections, and Akka actors. 2. *Advanced sources*: Sources like Kafka, Flume, Kinesis, Twitter, etc. are available through extra utility classes. These require linking against extra dependencies as discussed in the linking section. [8]

Data Mobility	Pull based, no blocking operations [9]	Push based, real time operations [7]	Configured with Flume to receive data using two approaches: 1. Push-based Receiver 2. Pull-based Receiver [10]
HA & Data Recovery	Highly available with rollback recovery using upstream backup. At least once processing. [9]	Highly available; system will continue to operate under the following conditions: • Hardware failure of a server running TIBCO StreamBase • Unexpected software failure in the TIBCO StreamBase event processing platform • Administrative error on a server, such as accidental shutdown Failure of network links • Server maintenance/removal from production [11]	Highly available with rollback recovery using DStreams to utilize the capability of RDD [12]
Deterministic or Nondeterministic	Doesn't specify [9]	Doesn't specify	Deterministic batch computation [8]
Data Partition and Scaling	Handled by user configuration and coding [9]	Handled automatically by the system based on sharding [4]	Handled automatically by the system [13]
Data Storage	None [9]	StreamBase LiveView Data mart	HDFS File System
Data Querying	Trident [9]	StreamBase LiveView	Spark-EC2 [8]

7. USE CASES

7.1 Apace Storm

Let us now discuss some of the real world use cases and how Apache Storm has been used in these situations

1. Twitter

Storm is used to power various Twitter systems like real-time analytics, personalization, search, revenue optimization and several others. Apache Storm integrates with the rest of Twitter's infrastructure which includes, database systems like Cassandra, Memcached, etc, the messaging

infrastructure, Mesos and the monitoring & alerting systems. Storm's isolation scheduler makes it suitable to utilize the same cluster for production applications and in-development applications as well. It provides an effective way for capacity planning.

2. Yahoo!

Yahoo! is working on a next generation platform by merging Big Data and low-latency processing. Though Hadoop is the main technology used here for batch processing, Apache Storm allows stream processing of user events, content feeds, and application logs.

3. Flipboard

Flipboard is a single place to explore, collect and share news that interests and excites you. Flipboard uses storm for a wide range of services like content search, real-time analytics, custom magazine feeds, etc. Apache Storm is integrated with the infrastructure that includes systems like ElasticSearch, Hadoop, HBase and HDFS, to create highly scalable data platform.

There are many more organizations implementing Apache Storm and even more are expected to join this game, as Apache Storm is continuing to be a leader in real-time analytics.

7.2 TIBCO StreamBase [14]

The various use cases of StreamBase are as follows:

- Intelligence and Security
    - Intelligence and Surveillance
    - Intrusion Detection and Network Monitoring
    - Battlefield Command and Control
- Capital Markets
    - FX
    - Market Data Management
    - Real Time Profit and Loss
    - Transaction Cost Analysis
    - Smart Order Routing
    - Algorithmic Trading
- Multiplayer Online Gaming
    - Interest Management
    - Real-Time Metrics
    - Managing User-Generated Content
    - Web Site Monitoring

- Retail, Internet and Mobile Commerce
    - In Store Promotion
    - Website Monitoring
    - Fraud Detection
- Telecommunications and Networking
    - Network Monitoring and
    - Bandwidth and Quality-of-Service Monitoring
    - Fraud Detection
    - Location-based Services

7.3 Apache Spark Streaming [15]

- Intrusion Detection
    - Malfunction Detection
    - Site Analytics
    - Network Metrics Analysis
- Fraud Detection
    - Dynamic Process Optimzation
    - Recommendations
    - Location Based Ads
- Log Processing
    - Supply Chain Planning
    - Sentiment Analysis

8. CONCLUSION

In this paper, we have tried to explain in brief about real-time data analytics and have discussed some of the tools used to implement real time data processing. We started off by explaining the limitations of Hadoop and how it is meant for batch processing and stream processing. Back in 2008, when Hadoop had established its foothold in the market of data analysis, it was eBay which stated "eBay doesn't love MapReduce." At eBay, analysts thoroughly evaluated Hadoop and came to the clear conclusion that MapReduce is absolutely the wrong technology long-term for big data analytics. Hadoop's shortcomings in the field of live data streaming have been pretty obvious and well known today. After evaluating the limitations of Hadoop, we then discussed some of the frameworks which are used for real time data streaming. These frameworks include Apache Storm, Apache Spark Streaming and TIBCO StreamBase. This discussion is followed by a comparative study of the three frameworks wherein we've compared the performance of these frameworks with respect to certain parameters. Lastly, we've provided certain use cases wherein these frameworks are suitable for usage.

REFERENCES

[1] http://hortonworks.com/hadoop/storm/.
[2] Jan Sipke van der Veen, Bram van der Waaij, Elena Lazovik, Wilco Wijbrandi, Robert J. Meijer "Dynamically Scaling Apache Storm for the Analysis of Streaming Data," IEEE, 2015
[3] TIBCO, TIBCO Live Datamart: Push-Based Real-Time Analytics.
[4] Richard Tibbetts, Steven Yang, Rob MacNeill, David Rydzewski, StreamBase LiveView: Push-Based Real-Time Analytics.
[5] http://www.cs.duke.edu/~kmoses/cps516/dstream.html.
[6] https://azure.microsoft.com/en-in/documentation/articles/stream-analytics-comparison-storm/.
[7] http://docs.streambase.com/sb75/index.jsp?topic=/com.streambase.sb.ide.help/data/html/samplesinfo/SBD2SBDInput.html.
[8] http://www.apache.org/.
[9] S. Kamburugamuve, "Survey of Distributed Stream Processing for Large Stream Sources ", 2013
[10] http://datametica.com/integration-of-spark-streaming-with-flume/).
[11] R. Tibbetts, TIBCO StreamBase High Availability Deploy Mission-Critical TIBCO StreamBase Applications in a Fault Tolerant Configuration.
[12] https://www.sigmoid.com/fault-tolerant-stream-processing/).
[13] https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html.
[14] http://www.streambase.com/industries/.
[15] http://www.planetcassandra.org/apache-spark-streaming/.