Flink vs. Spark

Big data frameworks

- Big data started as the multiple frameworks focused on specific problems
- Every framework defined its own abstraction
- Though combining multiple frameworks in powerful in theory, putting together is no fun
- Combining right set of frameworks and making them work on same platform became non trivial task
- Distributions like Cloudera, Hortonworks solved these problems to some level

Dawn of big data platforms

- Time of big data processing frameworks is over, it’s time for big platforms
- More and more advances in big data shows people want to use single platform for their big data needs
- Single platform normally evolves faster compared to a distribution of multiple frameworks
- Spark pioneering the platform and others are following their lead

Framework vs Platform

- Each framework is free to define it’s own abstraction but platform should define single abstraction
- Frameworks often embrace multiple runtime whereas platform embraces single runtime
- Platforms normally supports libraries rather than frameworks
- Frameworks can be in different versions, normally all libraries on a platform will be on a single version

Abstractions in frameworks

- Map/Reduce provides file as abstraction as it’s a framework focused on batch processing
- Storm provided realtime stream as abstraction as it’s focused on real time processing
- Hive provides table as abstraction as it’s focused on structured data analysis
- Every framework define their own abstraction which defines their capabilities and limitations

Abstraction in platform

- As we moved into platform, abstraction provided by platform becomes crucial
- Abstraction provided by platform should be generic enough for all frameworks/ libraries built on top of it.
- Requirements of platform abstraction
  - Ability to support different kind of libraries/ framework on platform
  - Ability to evolve with new requirements
  - Ability to harness advances in the hardware

Abstraction in Spark platform

- RDD is the single abstraction on which all other API’s have built
- RDD allowed Spark to evolve to be platform rather than just another framework
- RDD currently supports
  - Batch processing
  - Structure data analysis using DF abstraction
  - Streaming
  - Iterative processing - ML

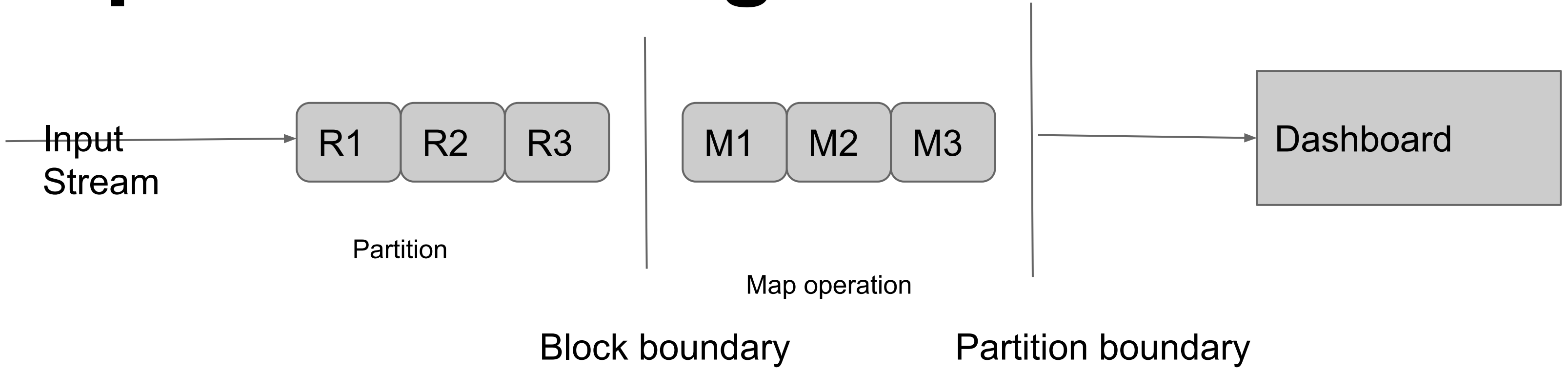
RDD is the assembly of big data?

- RDD abstraction as platform abstraction was much better than any other framework abstractions that came before that.
- But as platform idea becomes common place, there is been question is RDD is the assembly of big data?
- Databricks has convinced it is, but other disagree
- RDD seems to start showing some cracks in it’s perfect image, particularly in
  - Streaming
  - Iterative processing

RDD in streaming

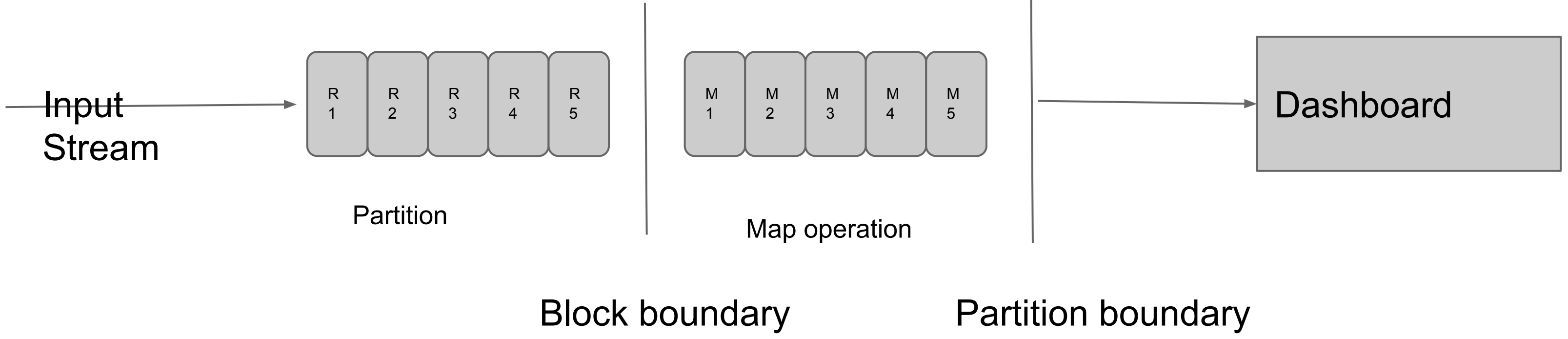
- RDD is essentially a map/reduce file based abstraction that sits in memory
- RDD file like behaviour makes it difficult to amend to the streaming latencies
- This is the reason why spark streaming in a near realtime not realtime system
- So this is holding platform back in API’s like custom windows like window based on record count, event based time

Spark streaming



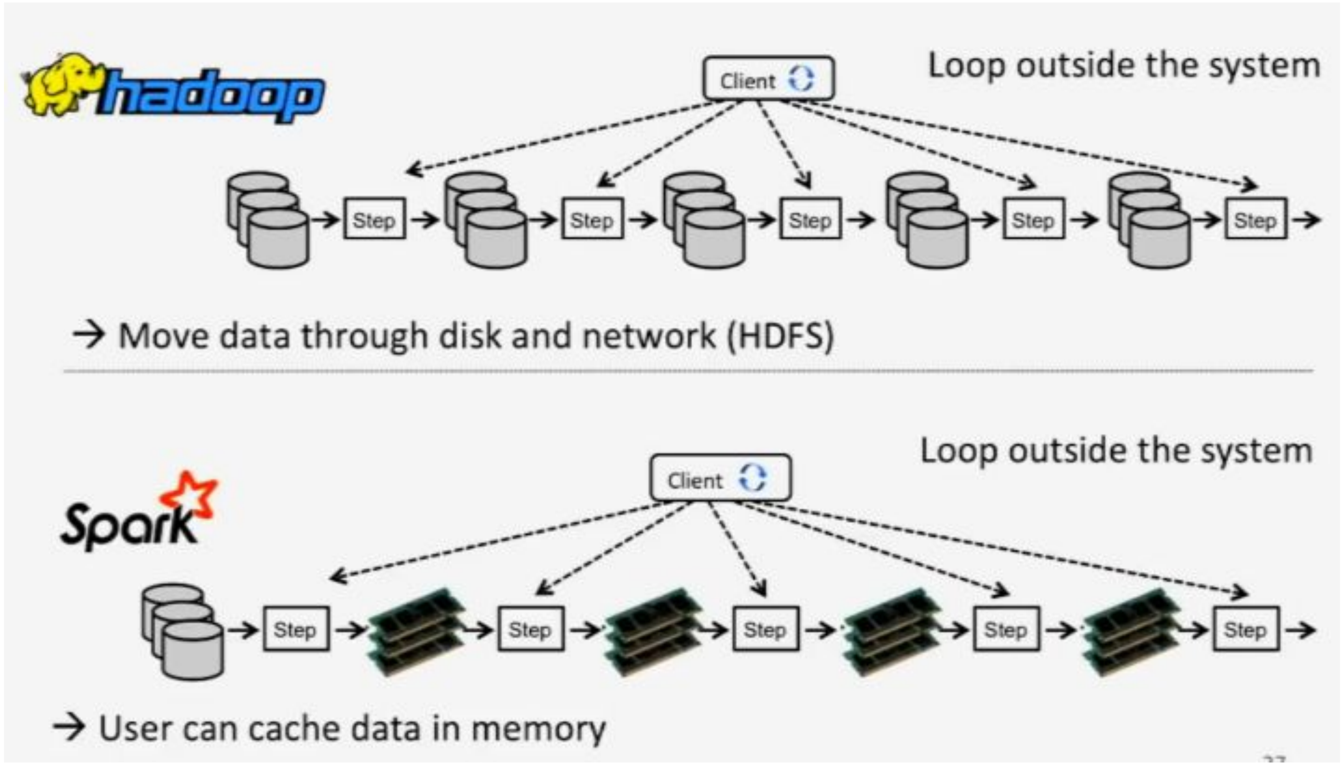
- RDD is an abstraction based on partitions
- To satisfy the RDD abstraction, spark streaming inserts blocking operations like block boundary, partition boundary to streaming data
- Normally it's good practice to avoid any blocking operation on stream but spark streaming we can't avoid them
- Due to this nature, dashboard our example has to wait till all the records in the partition to be processed before it can see even first result.

Spark streaming in spikes



- In case of spikes, as spark streaming is purely on time, data in single block increases.
- As block increases the partition size increases which makes blocking time bigger
- So again it puts more pressure on the processing system and increases latency in downstream systems

Spark iterative processing



→ Move data through disk and network (HDFS)

→ User can cache data in memory

RDD in iterative processing

- Spark runtime does not support iteration in the runtime level
- RDD caching with essential looping gets the effect of iterative processing
- To support iterative process, the platform should support some kind of cycle in its dataflow which is currently DAG
- So to implement advanced matrix factorization algorithm it’s very hard to do in spark

Stream as the abstraction

- A **stream** is a sequence of data elements made available over time.
- A stream can be thought of as items on a conveyor belt being processed one at a time rather than in large batches.
- Streams can be unbounded (streaming application) and bounded (batch processing)
- Streams are becoming new abstractions to build data pipelines.

Streams in last two years

- There's been many projects started to make stream as the new abstraction layer for data processing
- Some of them are
    - Reactive streams like akka-streams, akka-http
    - Java 8 streams
    - RxJava etc

Streams are picking up steam from last few years now.

Why stream abstraction?

- Stream is one of most generic interface out there
- Flexible in representing different data flow scenarios
- Stream are stateless by nature
- Streams are normally lazy and easily parallelizable
- Streams work well with functional programming
- Stream can model different data sources/sinks
  - File can be treated as stream of bytes
- Stream normally good at exploiting the system resources.

Why not streams?

- Long running pipelined streams are hard to reason about.
- As stream is by default stateless, doing fault tolerance becomes harder
- Stream change the way we look at data in normal program
- As there may be multiple things running parallely debug becomes tricky

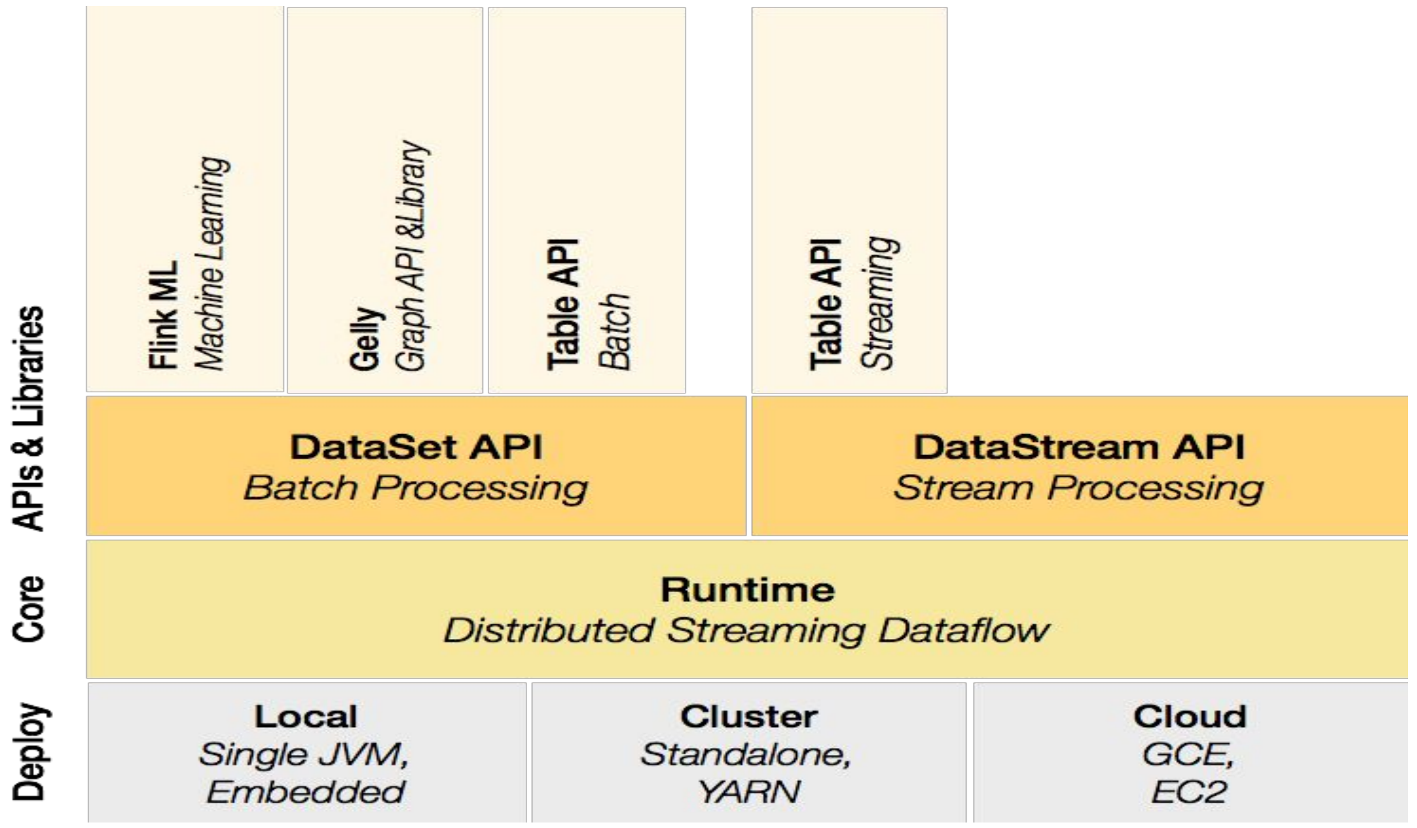
Stream abstraction in big data

- Stream is the new abstraction layer people are exploring in the big data
- With right implementation, stream can support both streaming and batch applications much more effectively than existing abstractions.
- Batch on streaming is new way of looking at processing rather than treating streaming as the special case of batch
- Batch can be faster on streaming than dedicated batch processing

Apache flink

- Flink’s core is a **streaming dataflow engine** that provides data distribution, communication, and fault tolerance for distributed computations over data streams.
- Flink provides
  - Dataset API - for bounded streams
  - Datastream API - for unbounded streams
- Flink embraces the stream as abstraction to implement it’s dataflow.

Flink stack


APIs & Libraries
Core
Deploy
Flink ML
Machine Learning
Gelly
Graph API &Library
Table API
Batch
Table API
Streaming
DataSet API
Batch Processing
DataStream API
Stream Processing
Runtime
Distributed Streaming Dataflow
Local
Single JVM,
Embedded
Cluster
Standalone,
YARN
Cloud
GCE,
EC2

Flink history

- Stratosphere project started in Technical university, Berlin in 2009
- Incubated to Apache in March, 2014
- Top level project in Dec 2014
- Started as stream engine for batch processing
- Started to support streaming few versions before
- DataArtisians is company founded by core flink team
- Complete history is here [1]

Spark vs Flink

- Spark was the first open source big data platform. Flink is following it’s lead
- They share many ideas and frameworks
    - Same collection like API
    - Same frameworks like AKKA, YARN underneath
    - Full fledges DSL in Scala
- It is very easy to port code from Spark to Flink as they have very similar API. Ex : **WordCount.scala**
- But there are many differences in the abstractions and runtime.

Abstraction

Spark	**Flink**
RDD	Dataflow (JobGraph)
RDD API for batch and DStream API for streaming	Dataset for batch API and DataStream API for streaming
RDD does not go through optimizer. Dataframe is API for that.	All API’s go through optimizer. Dataset is dataframe equivalent.
Can combine RDD and DStream	Cannot combine Dataset and DataStream
Streaming over batch	Batch over streaming

Memory management

Spark	Flink
Till 1.5, Java memory layout to manage cached data	Custom memory management from day one.
Many OOM errors due to inaccurate approximation of java heap memory	Fewer OOM as memory is pre allocated and controlled manually
RDD does not uses optimizer so it can’t use benefits of Tungsten	All API’s go through optimizer. So all can depend upon the controlled memory management
No support for off heap memory natively	Support for off heap memory natively from 0.10 version
Supports explicit data caching for interactive applications	No explicit data caching yet

Streaming

Spark	**Flink**
Near real time / microbatch	Realtime / Event based
Only supports single concept of time i.e processing time	Support multiple concepts of time like event time, process time etc
Supports windows based on process time.	Supports windows based on time, record count, triggers etc
Great support to combine historical data and stream	Limited support to combine historical data and stream
Supports most of the datastreams used in real world	Limited supported for streams which are limited to kafka as of now

Batch processing

Spark	Flink
Spark runtime is dedicated batch processor	Flink uses different optimizer to run batch on streaming system
All operators are blocking aka partitioned to run in flow	Even some of the batch operators are streamed as it's transparent to program
Uses RDD recomputation for fault tolerance	Chandy-Lamport algorithm for barrier checks for fault tolerance

- As spark has dedicated batch processing, it's expected to do better compare to flink
- Not really. Stream pipeline can result in better performance in even in batch[2].

Structured Data analysis

Spark	Flink
Excellent supports for structured data using Datasource API	Only supports Hadoop InputFormat based API's both for structured and unstructured data
Support Dataframe DSL and SQL interface for data analysis	Limited support for DSL using Table API and no SQL support yet
Integrates nicely with Hive	No integration with Hive yet
Support for statically types dataframe transformation is getting added in 1.6	Supports statically typed structured data analysis using Dataset API
No support for Dataframe for streaming yet	Table API is supported for streaming

Maturity

- Spark is already 5 years old in Apache community where as flink is around 2 year old
- Spark is already in version 1.6 whereas flink is yet to hit 1.0
- Spark has great ecosystem support and mature compared to flink at this point of time
- Materials to learn, understand are far more better for spark compared to Flink
- Very few companies are using flink in production as of now

References

1. http://www.slideshare.net/stephanewen1/flink-history-roadmap-and-vision
2. http://www.slideshare.net/FlinkForward/dongwon-kim-a-comparative-performance-evaluation-of-flink
3. http://www.slideshare.net/stephanewen1/flink-history-roadmap-and-vision

References

- http://flink.apache.org/
- http://flink.apache.org/faq.html
- http://flink-forward.org/
- http://flink.apache.org/news/2015/05/11/Juggling-with-Bits-and-Bytes.html
- https://cwiki.apache.org/confluence/display/FLINK/Data+exchange+between+tasks