Batch & Stream Processing with Apache Spark

Pramod Bhatotia

http://homepages.inf.ed.ac.uk/pbhatoti/

Credits for the lecture material:

Spark NSDI paper and presentation
Spark Streaming SOSP paper and presentation
Apache Flink Website



Why Spark?

MapReduce greatly simplified "big data" analysis on large, unreliable clusters

- But as soon as it got popular, users wanted more:
 - More complex, multi-stage applications
 (e.g. iterative machine learning & graph processing)
 - More interactive ad-hoc queries

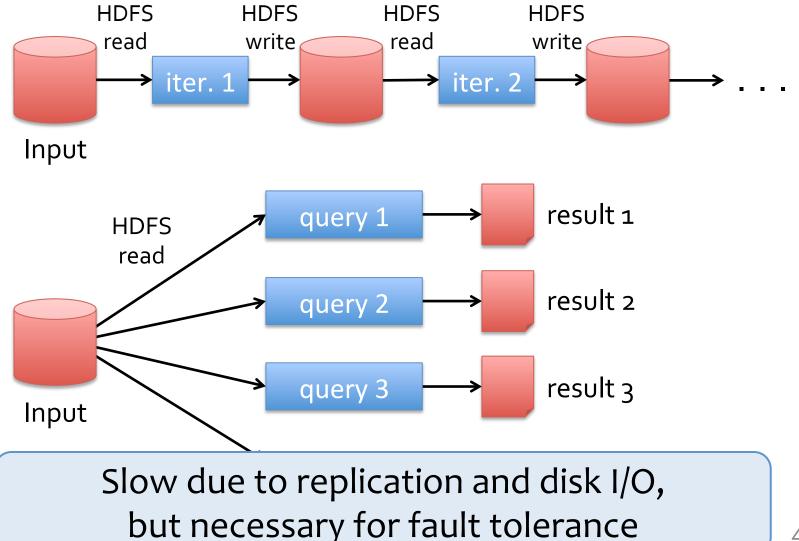
Why Spark?

Complex apps and interactive queries both need one thing that MapReduce lacks:

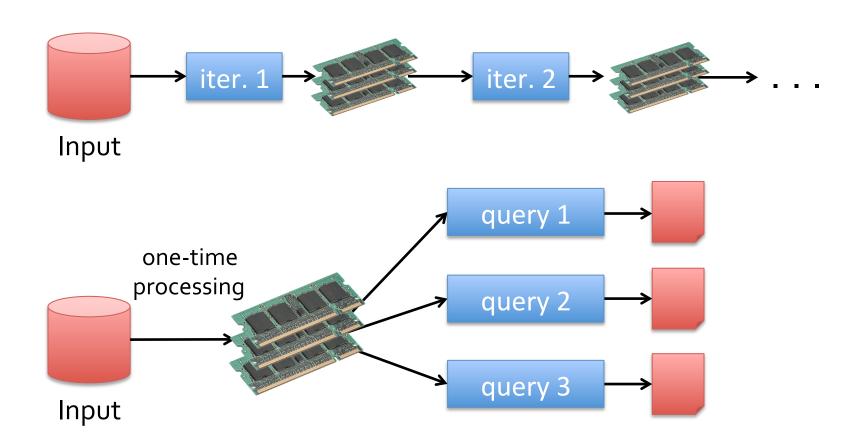
Efficient primitives for data sharing

In MapReduce, the only way to share data across jobs is stable storage → slow!

Example

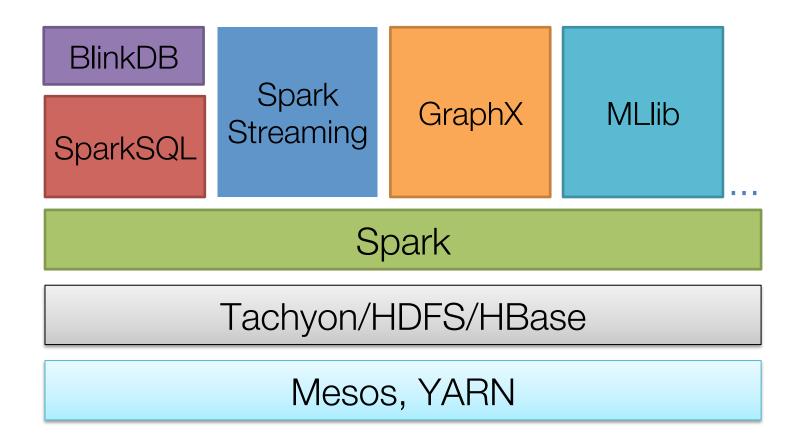


Goal: In-memory data sharing



Spark project

An in-memory cluster computing framework



Pitfall of in-memory computing

10-100× faster than network/disk, but how to get FT?

How to design a distributed memory abstraction that is both **fault-tolerant** and **efficient**?

Key contribution

Resilient Distributed Datasets (RDDs)

In this lecture

- 1. What are RDDs?
- 2. How Spark uses RDDs to achieve efficiency and fault-tolerance?

RDDs

- Restricted form of distributed shared memory
 - Immutable, partitioned collections of records
 - Can only be built through coarse-grained deterministic transformations (map, filter, join, ...)
- Efficient fault recovery using lineage
 - Log one operation to apply to many elements
 - Re-compute lost partitions on failure
 - No cost if nothing fails

Spark programming interface

- DryadLINQ-like API in the Scala language
- Usable interactively from Scala interpreter
- Interface for Java/Python/Scala

Provides:

- Resilient distributed datasets (RDDs)
- Operations on RDDs: transformations (build new RDDs), actions (compute and output results)
- Control of each RDD's partitioning (layout across nodes) and persistence (storage in RAM, on disk, etc)

Example: Log mining

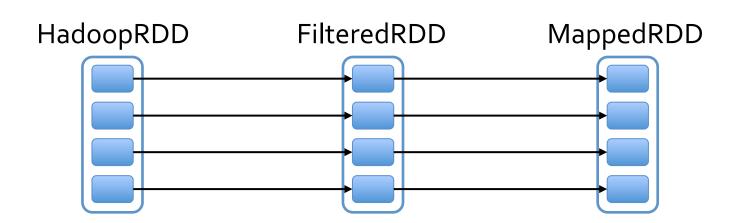
Load error messages from a log into memory, then interactively search for various patterns

```
Transformed RDD
lines = spark.textFile("hdfs://...")
                                                                  Worker
                                                        results
errors = lines.filter(_.startsWith("ERROR"))
                                                             tasks
messages = errors.map(_.split('\t')(2))
                                                                   Block 1
                                                    Master
messages.persist()
                                                  Action
messages.filter(_.contains("foo")).count
messages.filter(_.contains("bar")).count
                                                                 Worker
                                                                  Block 2
                                                 Worker
                                                 Block 3
                                                                        12
```

Fault tolerance

RDDs track the graph of transformations that built them (their lineage) to rebuild lost data



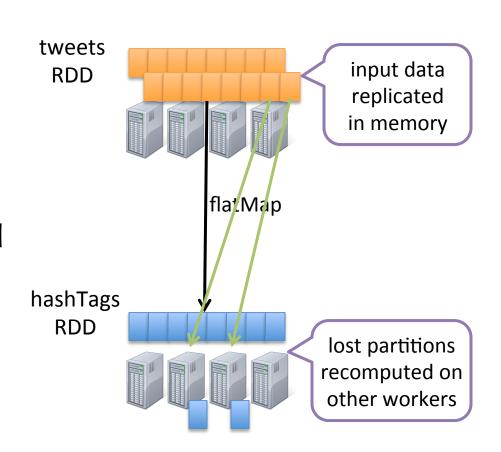


Fault-tolerance

Batches of input data are replicated in memory for fault-tolerance

Data lost due to worker failure, can be recomputed from replicated input data

All transformations are fault-tolerant, and exactly-once transformations



RDD operations

Transformation

- Map
- Filter
- GroupBy
- Union
- Intersect
- Join
- Aggregate
- •

Actions

- Reduce
- Collect
- Count
- First
- •

Summary

 RDDs offer a simple and efficient programming model for a broad range of applications

 Leverage the coarse-grained nature of many parallel algorithms for low-overhead recovery

Resources: https://spark.apache.org/

Why stream processing?

Many big-data applications need to process large data streams in real-time

Website monitoring Overview O Refresh Now Last day Last week Last month All time Fraud detection 0101010101010110101010000000100101 Ad monetization Google* bird houses Search Advanced Search Preferences Results 1 - 10 of about 22,100,000 for big houses. (0.19 Web Show options... Traffic Sources Posts Blogger integrates with Amazon Associates Sponsored Links Dec 16, 2009 www.Scotts.com Scotts attracts colorful birds to your backyard! http://buzz.blogger.com/ **Bird Houses** Find All Types Of Bird Feeders And Specialty Bird Houses Houses At Lowe's® New Lower Price www.birds-out-back.com Roosting Boxes, Purple Martins, Bat Chalets. www.Lowes.com Create Pages in Blogger Bird Houses at BestNest Feb 3, 2010 Bird Houses Sale www.bestnest.com Over 225 different houses in stock, Free shipping! Authorized Dealer - New Designs. Show off your Followers Low Price Guarantee- Free Shipping www.OutdoorLivingShowroom.com Bird House: JustBirdHouses.net learn more about blue bird houses and Blogger Template Designer Now Available To Everyon. birdhouses, along with our blue bird house and purple martin bird houses. ww.justbirdhouses.net/ - Cached - Similar -High Quality Bird Houses Bird Houses 🕋 🗵 Nesting boxes & decorative houses Bird Houses and Bird Feeders for north american bird species. 5-Star Service. Free Shipping \$75+ ww.birdhouses101.com/ - Cached - Similar - 🦃 www.backyardbird.com Bird Feeders, Bird Houses - The Backyard Bird Company Decorative Bird Houses Bird Feeders - The Backyard Bird Company has a variety of bird feeders Beautify Your Garden With Our will accent your landscape and attract wildlife. Wooden Bird Houses at a Discount. Bird Houses - Birdfeeders - Decorative Bird Houses BirdHouse Station.com ww.backvardbird.com/ - Cached - Similar -Google Checkout

Spark streaming

A stream processing framework

Scales to hundreds of nodes

Achieves second-scale latencies

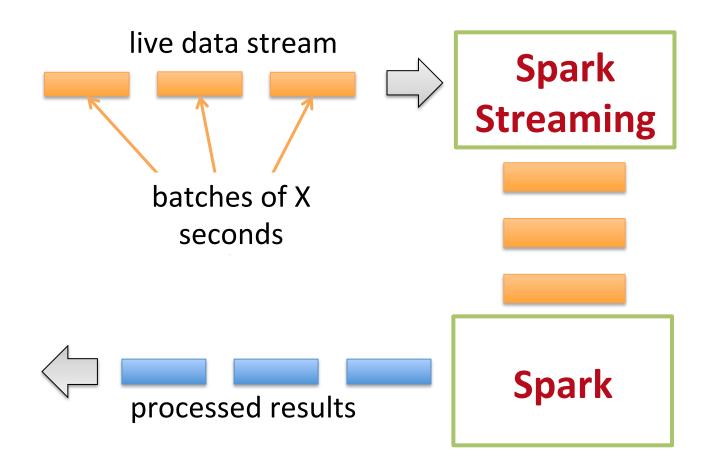
Efficiently recover from failures

Integrates with batch and interactive processing

Consistent "exactly-once" semantics

Spark streaming

 Run a streaming computation as a series of very small, deterministic batch jobs



Input source

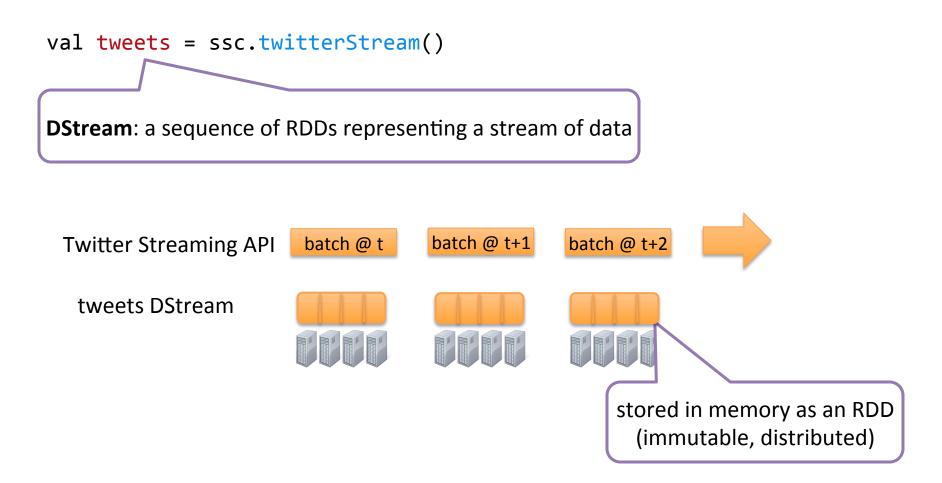
Spark streaming provides input from

Kafka, HDFS, Flume, Akka Actors, Raw TCP sockets, etc.

Very easy to write a receiver for your own data source

Also, generate your own RDDs from Spark, etc. and push them in as a "stream"

Example: Get hashtags from Twitter



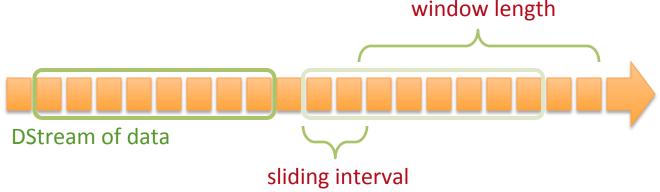
Example: Get hashtags from Twitter

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
                    transformation: modify data in one DStream to create
new DStream
                                     another DStream
                                                  batch @ t+2
                         batch @ t
                                     batch @ t+1
  tweets DStream
                             flatMap
                                          flatMap
                                                       flatMap
  hashTags Dstream
                                                              new RDDs created
  [#cat, #dog, ...]
                                                                for every batch
```

Sliding windows

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()

sliding window
operation
window length
window length
```



A primer on Apache Flink



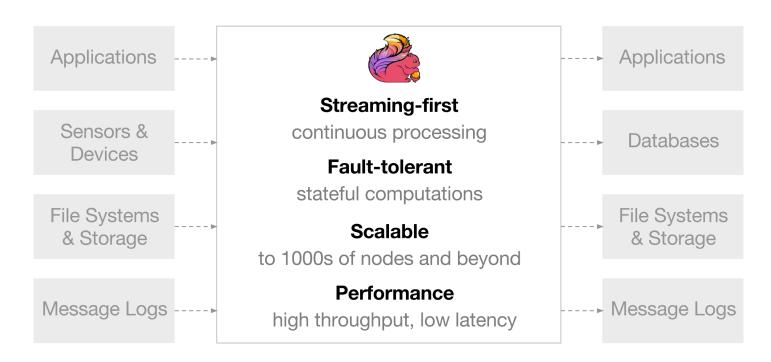
Pramod Bhatotia

http://homepages.inf.ed.ac.uk/pbhatoti/



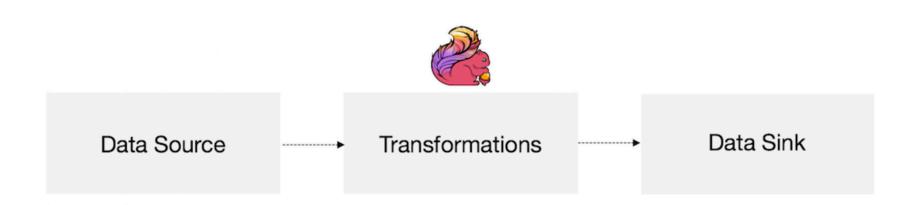
Why Flink?

- Apache Spark follows a batched streaming model
 - Geared towards for high throughput
 - Streaming application requires low-latency too!



Flink Programs

Continuous execution model



Flink APIs

SQL

High-level Language

Table API

Declarative DSL

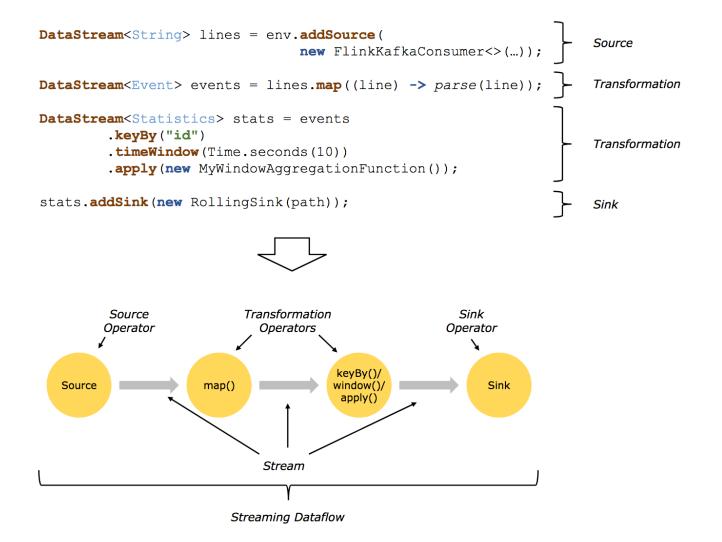
DataStream / DataSet API

Core APIs

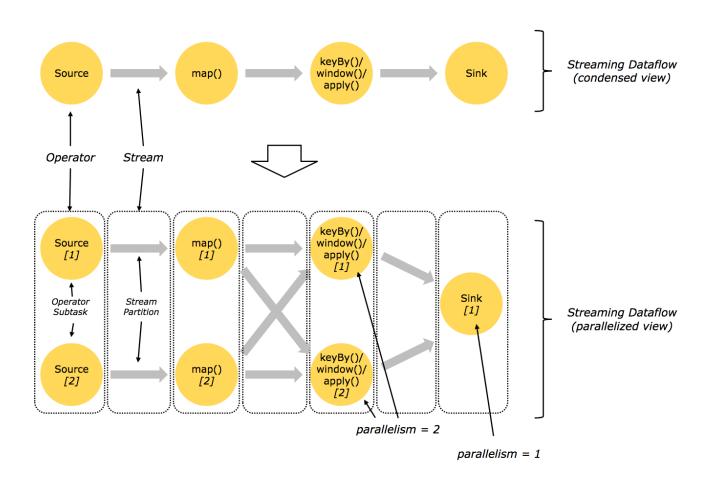
Stateful Stream Processing

Low-level building block (streams, state, [event] time)

Example program



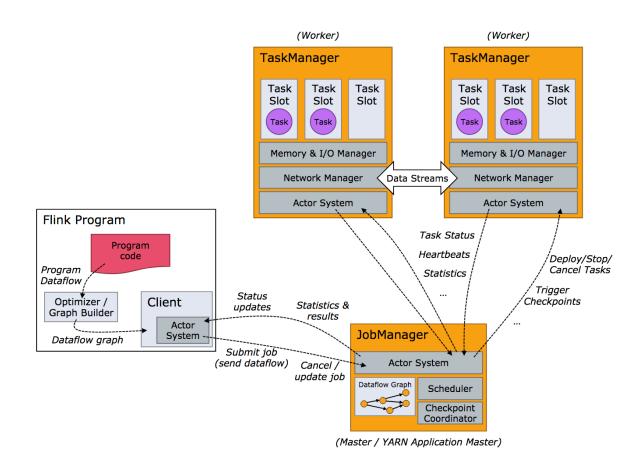
Data-parallel execution



Other features in Flink

- Window
 - Tumbling or sliding windows
 - Time-based or data-driven
- Time
 - Event time (Watermarks)
 - Ingestion time
- Stateful operators
 - Key-value store
- Fault-tolerance
 - Checkpointing
 - Stream replay

Distributed run-time



Summary

Apache Spark and Flink

- Unified data engines for batch and stream processing
- Expose a data-parallel programming model
- Designed to be scalable, fault-tolerant, strong semantics

Resources:

- Spark: https://spark.apache.org/
- Flink: https://flink.apache.org/

Thanks!

http://homepages.inf.ed.ac.uk/pbhatoti/