A Short History

- Started at UC Berkeley in 2009
- Open Source: 2010
- Apache Project: 2013
- Today: most popular big data processing engine

What Is Spark?

- Parallel execution engine for big data processing
- Easy to use: 2-5x less code than Hadoop MR
  - High level APIs in Python, Java, and Scala
- Fast: up to 100x faster than Hadoop MR
  - Can exploit in-memory when available
  - Low overhead scheduling, optimized engine
- General: support multiple computation models

General

- Unifies batch, interactive computations

Spark SQL

Spark Core
General

- Unifies batch, interactive, streaming computations

Spark SQL  Spark Streaming  Spark Core

Easy to Write Code

WordCount in 50+ lines of Java MR

WordCount in 3 lines of Spark

Large-Scale Usage

- Largest cluster: 8000 nodes
- Largest single job: 1 petabyte
- Top streaming intake: 1 TB/hour
- 2014 on-disk sort record
Fast: Time to sort 100TB

2013 Record: Hadoop
- 2100 machines
- 72 minutes

2014 Record: Spark
- 207 machines
- 23 minutes

Also sorted 1PB in 4 hours

RDD: Core Abstraction

Write programs in terms of **distributed datasets** and **operations** on them

- **Resilient Distributed Datasets** (RDDs)
  - Collections of objects distr. across a cluster, stored in RAM or on Disk
  - Built through parallel transformations
  - Automatically rebuilt on failure

- **Operations**
  - Transformations (e.g. map, filter, groupBy)
  - Actions (e.g. count, collect, save)

Operations on RDDs

- Transformations f(RDD) => RDD
  - Lazy (not computed immediately)
  - E.g. “map”

- Actions:
  - Triggers computation
  - E.g. “count”, “saveAsTextFile”

Working With RDDs

```scala
val textFile = sc.textFile("SomeFile.txt")
```
Working With RDDs

```python
linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
```

**Driver**

**Worker**

**Example: Log Mining**

Transformed RDD

```python
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

**Driver**

**Worker**
Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```

Example: Log Mining
Load error messages from a log into memory, then interactively search for various patterns.

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()

messages.filter(lambda s: "mysql" in s).count()
```
Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[-2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[-2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[-2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[-2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Example: Log Mining

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

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[-2])
messages.cache()
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Cache your data ➔ Faster Results
Full-text search of Wikipedia
- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk
Language Support

Pyhon
```python
lines = sc.textFile(...) lines.filter(lambda s: "ERROR" in s).count()
```

Scala
```scala
val lines = sc.textFile(...) lines.filter(x => x.contains("ERROR")).count()
```

Java
```java
JavaRDD<String> lines = sc.textFile(...); lines.filter(new Function<String, Boolean>() {
    Boolean call(String s) {
        return s.contains("error");
    }
}).count();
```

Expressive API

• map
• reduce

Interactive Shells
• Python & Scala

Performance
• Java & Scala are faster due to static typing
• …but Python is often fine

Fault Recovery

RDDs track lineage information that can be used to efficiently reconstruct lost partitions
Fault Recovery Example

- Two-partition RDD A={A₁, A₂} stored on disk
  1) filter and cache → RDD B
  2) join → RDD C
  3) aggregate → RDD D

Fault Recovery Example

- C₁ lost due to node failure before reduce finishes

Fault Recovery Example

- C₁ lost due to node failure before reduce finishes
- Reconstruct C₁, eventually, on different node

Spark Streaming: Motivation

- Many important apps must process large data streams at second-scale latencies
  - Site statistics, intrusion detection, online ML
- To build and scale these apps users want:
  - Integration: with offline analytical stack
  - Fault-tolerance: both for crashes and stragglers
How does it work?

- Data streams are chopped into batches
  - A batch is an RDD holding a few 100s ms worth of data
- Each batch is processed in Spark

Data streams → Spark → batches

Streaming Word Count

```scala
val lines = context.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
ssc.start()  // start processing the stream
ssc.awaitTermination()  // print some counts on screen
```

Word Count

```scala
object NetworkWordCount {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("NetworkWordCount")
    val context = new StreamingContext(sparkConf, Seconds(1))
    val lines = context.socketTextStream("localhost", 9999)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```
• Midterm 3 coming up on Wen 11/29 6:30-8PM
  – All topics up to and including Lecture 24
    » Focus will be on Lectures 17 – 24 and associated readings, and Projects 3
    » But expect 20-30% questions from materials from Lectures 1-16
  – Closed book
  – 2 sides hand-written notes both sides
BREAK

From RDDs to DataFrames

Spark early adopters

Users

Understands
MapReduce
& functional APIs

Data Engineers
Data Scientists
Statisticians
R users
PyData …

DataFrames in Spark

Distributed collection of data grouped into named columns
(i.e. RDD with schema)

Domain-specific functions designed for common tasks
– Metadata
– Sampling
– Project, filter, aggregation, join, …
– UDFs

Available in Python, Scala, Java, and R
Spark DataFrame

Similar APIs as single-node tools like Pandas, R, i.e., easy to learn

- `head(filter(df, df$waiting < 50))` # an example in R
- `## eruptions waiting
##1  1.750  47
##2  1.750  47
##3  1.867  48`

*Old Faithful geyser data from http://www.stat.cmu.edu/~larry/all-of-statistics/data/faithful.dat

Spark RDD Execution

Opaque closures (user-defined functions)

Spark DataFrame Execution

Intermediate representation for computation

Simple wrappers to create logical plan
Performance

Runtime for an example aggregation workload (sec)

Python
Java/Scala

Benefit of Logical Plan:
Performance Parity Across Languages

Runtime for an example aggregation workload (sec)

SQL
R
Python
Java/Scala

Further Optimizations

- Whole-stage code generation
  - Remove expensive iterator calls
  - Fuse across multiple operators

Optimized input / output
- Parquet + built-in cache

Automatically applies to SQL, DataFrames, Datasets

TPC-DS Spark 2.0 vs 1.6 – Lower is Better

Runtime (seconds)

Time (1.6)
Time (2.0)
From Streaming to Structured Streaming

Data
Late arrival, varying distribution over time, …

Processing
Business logic change & new ops (windows, sessions)

Output
How do we define output over time & correctness?

The simplest way to perform streaming analytics is not having to reason about streaming

Structured Streaming
High-level streaming API built on DataFrames
- Event time, windowing, sessions, sources & sinks

Also supports interactive & batch queries
- Aggregate data in a stream, then serve using JDBC
- Change queries at runtime
- Build and apply ML models

Not just streaming, but “continuous applications”
Example: Batch Aggregation

```scala
logs = ctx.read.format("json").open("s3://logs")
logs.groupBy("userid", "hour").avg("latency")
  .write.format("jdbc")
  .save("jdbc:mysql://...")
```

Example: Continuous Aggregation

```scala
logs = ctx.read.format("json").stream("s3://logs")
logs.groupBy("userid", "hour").avg("latency")
  .write.format("jdbc")
  .startStream("jdbc:mysql://...")
```

Example

- Kafka
- ETL
- Database
- Reporting
- Applications
- ML Model
- Ad-hoc Queries

Goal: end-to-end continuous applications

Apache Spark Today

- > 1,500 contributors
- > 400K meetup members worldwide
- > 300K students trained worldwide
- 1,000s deployments in productions
  - Virtually every large enterprise
  - Available in all clouds (e.g., AWS, Google Compute Engine, MS Azure, IBM, ...)
  - Distributed by IBM, Cloudera, Hortonworks, MapR, Oracle, ...
- Databricks, startup to commercialize Apache Spark
Data Sources
Spark Core
DataFrames
ML Pipelines

Spark: unified engine across *data sources, workloads and environments*

**Summary**

- Server → Datacenter
- OS → Datacenter OS (e.g., Apache Mesos)
- Applications → Big data / ML applications (e.g., Apache Spark)

- AMPLab
  - Massive success in industry,...
  - and, academia:
    » Faculty at MIT, Stanford, CMU, Cornell, etc
    » Two ACM Dissertation Awards

- New lab starting: RISELab

**RISELab**
(Real-time Intelligent Secure Execution)

**RISELab**
From *live data to real-time decisions*

**AMPLab**
From *batch data to advanced analytics*
RISE stack

- Ray
- Clipper
- Tegra
- Secure Analytics & ML
- scheduler
- object store
- optimizer
- RISE μkernel

Ground (data context service)

- cassandra
- S3
- MongoDB
- Kafka
- SGX

Time Machine