What is Apache Spark?

Fast and general cluster computing engine that generalizes the MapReduce model

Makes it easy and fast to process large datasets

- High-level APIs in Java, Scala, Python, R
- Unified engine that can capture many workloads
A Unified Engine

Spark SQL
structured data

Spark Streaming
real-time

MLlib
machine
learning

GraphX
graph

Spark
A Large Community

Contributors / Month to Spark

Most active open source project for big data
Overview

Why a unified engine?

Spark programming model

Built-in libraries

Applications
MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with
MapReduce

A *general* engine for batch processing

We wrote the first version of the MapReduce library in February of 2003, and made significant enhancements to it in August of 2003, including the locality optimization, dynamic load balancing of task execution across worker machines, etc. Since that time, we have been pleasantly surprised at how broadly applicable the MapReduce library has been for the kinds of problems we work on. It has been used across a wide range of domains within Google, including:
Beyond MapReduce

MapReduce was great for batch processing, but users quickly needed to do more:

• More complex, multi-pass algorithms
• More interactive ad-hoc queries
• More real-time stream processing

Result: specialized systems for these workloads
Big Data Systems Today

General batch processing

Specialized systems for new workloads

MapReduce

Pregel

Giraph

Dremel

Drill

Impala

Presto

Storm

S4

...
Problems with Specialized Systems

More systems to manage, tune, deploy

Can’t easily combine processing types

- Even though most applications need to do this!
- E.g. load data with SQL, then run machine learning

In many cases, data transfer between engines is a dominant cost!
Big Data Systems Today

MapReduce

Pregel
Dremel
Impala
Storm

Giraph
Drill
Presto
S4

...
Overview

Why a unified engine?

Spark programming model

Built-in libraries

Applications
Background

Recall 3 workloads were issues for MapReduce:

• More complex, multi-pass algorithms
• More interactive ad-hoc queries
• More real-time stream processing

While these look different, all 3 need one thing that MapReduce lacks: efficient data sharing
Data Sharing in MapReduce

Input

HDFS read → iter. 1 → HDFS write → iter. 2 → HDFS write → ... 

HDFS read → query 1 → result 1
HDFS read → query 2 → result 2
HDFS read → query 3 → result 3

Slow due to replication and disk I/O
What We’d Like

Input

Distributed memory

iter. 1

iter. 2

... one-time processing

Input

Distributed memory

query 1

query 2

query 3

... 10-100x faster than network and disk
Spark Programming Model

Resilient Distributed Datasets (RDDs)
- Collections of objects stored in RAM or disk across cluster
- Built via parallel transformations (map, filter, …)
- Automatically rebuilt on failure
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: "Redis" in s).count()
...
```

Example: full-text search of Wikipedia in 0.5 sec (vs 20s for on-disk data)
Fault Tolerance

RDDs track lineage info to rebuild lost data

```python
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```
Fault Tolerance

RDDs track *lineage* info to rebuild lost data

```python
file.map(lambda rec: (rec.type, 1))
  .reduceByKey(lambda x, y: x + y)
  .filter(lambda (type, count): count > 10)
```
Example: Logistic Regression

Running Time (s) vs. Number of Iterations

- Hadoop: 110 s / iteration
- Spark: first iteration 80 s, further iterations 1 s
On-Disk Performance

Time to sort 100TB

2013 Record: 2100 machines
Hadoop
72 minutes

2014 Record: 207 machines
Spark
23 minutes

Source: Daytona GraySort benchmark, sortbenchmark.org
Libraries Built on Spark

- Spark SQL: structured data
- Spark Streaming: real-time
- MLlib: machine learning
- GraphX: graph
Combining Processing Types

// Load data using SQL
points = ctx.sql("select latitude, longitude from tweets")

// Train a machine learning model
model = KMeans.train(points, 10)

// Apply it to a stream
sc.twitterStream(...)
  .map(lambda t: (model.predict(t.location), 1))
  .reduceByWindow("5s", lambda a, b: a + b)
Combining Processing Types

Separate systems:

Spark:
Performance vs Specialized Systems

**SQL**
- Hive
- Impala (disk)
- Impala (mem)
- Spark (disk)
- Spark (mem)

**Throughput (MB/s/node)**
- Storm
- Spark

**ML**
- Mahout
- GraphLab
- Spark

**Response Time (sec)**
- 0
- 10
- 20
- 30
- 40

**Response Time (min)**
- 0
- 5
- 10
- 15
- 20
- 25
- 30
- 35
- 40
- 45
- 50
- 55
- 60

---

[Databricks](https://www.databricks.com)
Some Recent Additions

Dataframe API (similar to R and Pandas)
  • Easy programmatic way to work with structured data

R interface (SparkR)

Machine learning pipelines (like SciKit-learn)
Overview

Why a unified engine?

Spark programming model

Built-in libraries

Applications
Spark Community

Over 1000 deployments, clusters up to 8000 nodes

Many talks online at spark-summit.org
Top Applications

- Business Intelligence: 68%
- Data Warehousing: 52%
- Recommendation: 44%
- Log Processing: 40%
- User-Facing Services: 36%
- Faud Detection / Security: 29%
Spark Components Used

- Spark SQL: 69%
- DataFrames: 62%
- Spark Streaming: 58%
- MLlib + GraphX: 58%

75% of users use more than one component.
Learn More

Get started on your laptop: spark.apache.org

Resources and MOOCs: sparkhub.databricks.com

Spark Summit: spark-summit.org
Thank You