Spark & Spark SQL

High-Speed In-Memory Analytics over Hadoop and Hive Data

Instructor: Duen Horng (Polo) Chau

Slides adopted from Matei Zaharia (MIT) and Oliver Vagner (Manheim, GT)
What is Spark?

http://spark.apache.org

Not a modified version of Hadoop

Separate, fast, MapReduce-like engine
  » In-memory data storage for very fast iterative queries
  » General execution graphs and powerful optimizations
  » Up to 40x faster than Hadoop

Compatible with Hadoop’s storage APIs
  » Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc
What is **Spark SQL**?
(Formally called Shark)

Port of Apache **Hive** to run on **Spark**

Compatible with existing Hive data, metastores, and queries (HiveQL, UDFs, etc)

Similar speedups of up to **40x**
Spark project started in 2009 at UC Berkeley AMP lab, open sourced 2010.

Became Apache Top-Level Project in Feb 2014.

Shark/Spark SQL started summer 2011.

Built by 250+ developers and people from 50 companies.

Scale to 1000+ nodes in production.

In use at Berkeley, Princeton, Klout, Foursquare, Conviva, Quantifind, Yahoo! Research, ...
Why a New Programming Model?

MapReduce greatly simplified big data analysis

But as soon as it got popular, users wanted more:

» More complex, multi-stage applications (e.g. iterative graph algorithms and machine learning)
» More interactive ad-hoc queries
Why a New Programming Model?

MapReduce greatly simplified big data analysis

But as soon as it got popular, users wanted more:

» More complex, multi-stage applications (e.g. iterative graph algorithms and machine learning)

» More interactive ad-hoc queries

Require faster data sharing across parallel jobs
Up for debate… as of 10/7/2014

Is MapReduce dead?


http://www.reddit.com/r/compsci/comments/296aqr/on_the_death_of_mapreduce_at_google/

As an employee, I was surprised by this headline, considering I just ran some mapreduces this past week.

Cloud DataFlow is the external name for what is internally called Flume.

Flume is a layer that runs on top of MapReduce that abstracts away the complexity into something that is much easier.
Data Sharing in MapReduce

Input

HDFS read

iter. 1

HDFS write

iter. 2

HDFS read

HDFS write

query 1

result 1

HDFS read

query 2

result 2

query 3

result 3

...
Data Sharing in MapReduce

Input

HDFS read → iter. 1 → HDFS read → iter. 2 → HDFS read → iter. 3 → HDFS write → ... → HDFS write

Input

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

Slow due to replication, serialization, and disk IO
Data Sharing in Spark

Input

iter. 1

iter. 2

iter. 3

one-time processing

Distributed memory

query 1

query 2

query 3

...
Data Sharing in **Spark**

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

Key idea: resilient distributed datasets (RDDs)
» Distributed collections of objects that can be cached in memory across cluster nodes
» Manipulated through various parallel operators
» Automatically rebuilt on failure

Interface
» Clean language-integrated API in Scala
» Can be used interactively from Scala, Python console
» Supported languages: Java, Scala, Python
Object-Oriented Meets Functional

Have the best of both worlds. Construct elegant class hierarchies for maximum code reuse and extensibility, implement their behavior using higher-order functions. Or anything in-between.

LEARN MORE
Example: Log Mining

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

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

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()
```
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(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
```
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(_.startsWith("ERROR"))
messages = errors.map(_.split("\t"))(2)
cachedMsgs = messages.cache()
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t\')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

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

```scala
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count

Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
```
Example: Log Mining

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

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t\")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...

Result: scaled to 1 TB data in 5-7 sec  
(vs 170 sec for on-disk data)
Fault Tolerance

RDDs track the series of transformations used to build them (their \textit{lineage}) to recompute lost data.

E.g: \texttt{messages = textFile(...).filter(_.contains("error")).map(_.split(\textbackslash t')(2))}

- HadoopRDD
  - \texttt{path = hdfs://...}
- FilteredRDD
  - \texttt{func = _.contains(...)}
- MappedRDD
  - \texttt{func = _.split(\textbackslash t')(2)}
Example: Word Count (Python)

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

file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API
Example: Logistic Regression

val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
Example: Logistic Regression

```scala
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Repeated MapReduce steps to do gradient descent
Logistic Regression Performance

Running Time (s)

Number of Iterations

Hadoop
Spark

127 s / iteration
first iteration 174 s
further iterations 6 s
Supported Operators

map  reduce  sample
filter  count  cogroup
groupBy  reduceByKey  take
sort  groupByKey  partitionBy
join  first  pipe
leftOuterJoin  union  save
rightOuterJoin  cross  ...

...
Spark Users

CON VIVA
foursquare
quantifind
KLOUT
Yahoo!
University of California Berkeley
PRINCETON UNIVERSITY
UCSF
Use Cases

In-memory analytics & anomaly detection (Conviva)
Interactive queries on data streams (Quantifind)
Exploratory log analysis (Foursquare)
Traffic estimation w/ GPS data (Mobile Millennium)
Twitter spam classification (Monarch)

...
Conviva GeoReport

Group aggregations on many keys w/ same filter

40× gain over Hive; avoid repeated reading, deserialization, filtering
Mobile Millennium Project

Estimate city traffic from crowdsourced GPS data

Iterative EM algorithm scaling to 160 nodes

Credit: Tim Hunter, with support of the Mobile Millennium team; P.I. Alex Bayen; traffic.berkeley.edu
Spark SQL: Hive on Spark
Motivation

Hive is great, but Hadoop’s execution engine makes even the smallest queries take minutes.

Scala is good for programmers, but many data users only know SQL.

Can we extend Hive to run on Spark?
Hive Architecture

- **Meta store**
- **HDFS**
- **Client**
  - SQL Parser
  - Query Optimizer
  - Physical Plan
  - Execution
  - CLI
  - JDBC
- **MapReduce**
- **HDFS**
Spark SQL Architecture

Meta store

Client
- CLI
- JDBC

Driver
- Cache Mgr.

SQL Parser

Query Optimizer

Physical Plan

Execution

Spark

HDFS

[Engle et al, SIGMOD 2012]
Efficient In-Memory Storage

Simply caching Hive records as Java objects is inefficient due to high per-object overhead.

Instead, Spark SQL employs column-oriented storage using arrays of primitive types.

<table>
<thead>
<tr>
<th>Row Storage</th>
<th></th>
<th>Column Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 john 4.1</td>
<td>1 john 2 3</td>
<td></td>
</tr>
<tr>
<td>2 mike 3.5</td>
<td>2 mike 3 3</td>
<td></td>
</tr>
<tr>
<td>3 sally 6.4</td>
<td>3 sally 4 3</td>
<td></td>
</tr>
</tbody>
</table>
Efficient In-Memory Storage

Simply caching Hive records as Java objects is inefficient due to high per-object overhead.

Instead, Spark SQL employs column-oriented storage using arrays of primitive types.

**Row Storage**

| 3 | Sally | 6.4 |

**Column Storage**

| 4.1 | 3.5 | 6.4 |

**Benefit:** similarly compact size to serialized data, but >5x faster to access.
Using Shark

CREATE TABLE mydata_cached AS SELECT ... 

Run standard HiveQL on it, including UDFs
  » A few esoteric features are not yet supported

Can also call from Scala to mix with Spark
Benchmark Query 1

SELECT * FROM grep WHERE field LIKE '%%XYZ%';
Benchmark Query 2

SELECT sourceIP, AVG(pageRank), SUM(adRevenue) AS earnings
FROM rankings AS R, userVisits AS V ON R.pageURL = V.destURL
WHERE V.visitDate BETWEEN '1999-01-01' AND '2000-01-01'
GROUP BY V.sourceIP
ORDER BY earnings DESC
LIMIT 1;
What’s Next?

Recall that Spark’s model was motivated by two emerging uses (interactive and multi-stage apps)

Another emerging use case that needs fast data sharing is **stream processing**

» Track and update state in memory as events arrive
» Large-scale reporting, click analysis, spam filtering, etc
Streaming Spark

Extends Spark to perform streaming computations

Runs as a series of small (~1 s) batch jobs, keeping state in memory as fault-tolerant RDDs

Intermix seamlessly with batch and ad-hoc queries

tweetStream
  .flatMap(_.toLowerCase.split)
  .map(word => (word, 1))
  .reduceByWindow("5s", _ + _)

[Zaharia et al, HotCloud 2012]
Streaming Spark

Extends Spark to perform streaming computations

Runs as a series of small (~1 s) batch jobs, keeping state in memory as fault-tolerant RDDs

Intermix seamlessly with batch and ad-hoc queries

Result: can process 42 million records/second (4 GB/s) on 100 nodes at sub-second latency

[Zaharia et al, HotCloud 2012]
Streaming Spark

Extends Spark to perform streaming computations

Runs as a series of small (~1 s) batch jobs, keeping state in memory as fault-tolerant RDDs

Intermix seamlessly with batch and ad-hoc queries

tweetStream
  .flatMap(_.toLowerCase.split)
  .map(word => (word, 1))
  .reduceByWindow(5, _ + _)

[Zaharia et al, HotCloud 2012]
Spark Streaming

Create and operate on RDDs from live data streams at set intervals

Data is divided into batches for processing

Streams may be combined as a part of processing or analyzed with higher level transforms
Behavior with Not Enough RAM

Iteration time (s)

% of working set in memory

Cache disabled

25%

50%

75%

Fully cached

68.8

58.1

40.7

29.7

11.5
MLlib

Scalable machine learning library

Interoperates with NumPy

Available algorithms in 1.0
  » Linear Support Vector Machine (SVM)
  » Logistic Regression
  » Linear Least Squares
  » Decision Trees
  » Naïve Bayes
  » Collaborative Filtering with ALS
  » K-means
  » Singular Value Decomposition
  » Principal Component Analysis
  » Gradient Descent
GraphX

Parallel graph processing

Extends RDD -> Resilient Distributed Property Graph
  » Directed multigraph with properties attached to each vertex and edge

Limited algorithms
  » PageRank
  » Connected Components
  » Triangle Counts

Alpha component
Commercial Support

Databricks

» Not to be confused with DataStax
» Found by members of the AMPLab
» Offering
  • Certification
  • Training
  • Support
  • DataBricks Cloud
Commercial Support

Databricks Cloud

https://databricks.com