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 (TGI Fridays)
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**?
(Formerly 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**
Project History [latest: v1.1]

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, ...

http://en.wikipedia.org/wiki/Apache_Spark
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?

Google Dumps MapReduce in Favor of New Hyper-Scale Analytics System


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. After digging further, this headline and article is rather inaccurate.
Data Sharing in MapReduce

Input → HDFS read → iter. 1 → HDFS read → iter. 2 → HDFS read → ... → HDFS read

- query 1 → result 1
- query 2 → result 2
- query 3 → result 3

...
Data Sharing in MapReduce

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

Input → iter. 1 → iter. 2 → ...

one-time processing → Distributed memory

Input → query 1 → ...

Input → query 2 → ...

Input → query 3 → ...

Input
Data Sharing in Spark

- Input
  - Distributed memory
  - iter. 1
  - iter. 2
- one-time processing
  - query 1
  - query 2
  - query 3
- 10-100× 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, R
Functional programming in D3: [http://sleptons.blogspot.com/2015/01/functional-programming-d3js-good-example.html](http://sleptons.blogspot.com/2015/01/functional-programming-d3js-good-example.html)

Example: Log Mining

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

http://www.slideshare.net/normation/scala-dreaded
Example: Log Mining

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

http://www.slideshare.net/normation/scala-dreaded
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()
```

http://www.slideshare.net/normation/scala-dreaded
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()
```

http://www.slideshare.net/normation/scala-dreaded
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()
```

http://www.slideshare.net/normation/scala-dreaded
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()
```

http://www.slideshare.net/normation/scala-dreaded
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()
```

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
Example: Log Mining

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

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

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

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
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
```

http://www.slideshare.net/normation/scala-dreaded
Example: Log Mining

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

```scala
driver = spark.sparkContext
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
```

http://www.slideshare.net/normation/scala-dreaded
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
...
```

http://www.slideshare.net/normation/scala-dreaded
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)

http://www.slideshare.net/normation/scala-dreaded
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)

http://www.slideshare.net/normation/scala-dreaded
Fault Tolerance

RDDs track the series of transformations used to build them (their *lineage*) to recompute lost data.

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

Load data in memory once
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

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)
Logistic Regression Performance

Running Time (s)

Number of Iterations

127 s / iteration

first iteration 174 s
further iterations 6 s

1
5
10
20
30

Number of Iterations

Hadoop
Spark
Supported Operators

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

CON VIVA  foursquare
quantifind  KLOUT  Yahoo!
University of California  Princeton University  UCSF
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
Spark SQL Architecture

[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.

Row Storage

<table>
<thead>
<tr>
<th></th>
<th>john</th>
<th>4.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mike</td>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
<td>sally</td>
<td>6.4</td>
</tr>
</tbody>
</table>

Column Storage

<table>
<thead>
<tr>
<th></th>
<th>john</th>
<th>mike</th>
<th>sally</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.1</td>
<td>3.5</td>
<td>6.4</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**.

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

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%’;
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

![Bar chart showing iteration time (s) vs. % of working set in memory with cache disabled and fully cached scenarios.]

- Cache disabled: 68.8 ± 1
- 25%: 58.1 ± 1
- 50%: 40.7 ± 1
- 75%: 29.7 ± 1
- Fully cached: 11.5 ± 1
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