Learning Spark

Bridging HPC and Big Data Analytics
About me

- Félix-Antoine Fortin
- Computer eng. (M. Sc., ~PhD)

Main interests
- Advanced Computing
- Data Analytics
- Python

Projects at Compute Canada
- Digital Humanities
- Interactive Computing
- 3D Rendering

@CmdNtrf
Survey

By the end of the workshop, share your opinion at

http://tinyurl.com/hpcs-spark

According to the anonymous online employee survey, you don’t trust management. What’s up with that?

Oh. Right.

It is anonymous... I swear!
Objectives

- Understanding the fundamentals of Spark
- Interactively working with Spark
Outline

1. Big Data
2. Introduction to Apache Spark
   2.1. Hands-on
   2.2. Manipulating arbitrary objects
3. Dataframes and Spark SQL
   3.1. Hands-on
Setup check

https://jupyter.calculquebec.ca/
What Qualifies Big Data?

The Big Data * Vs in Images
Big Data - Volume

40 Zettabytes
[43 Trillion Gigabytes]
of data will be created by 2020, an increase of 300 times from 2005

Volume
SCALE OF DATA

It's estimated that
2.5 Quintillion Bytes
[2.3 Trillion Gigabytes]
of data are created each day

6 Billion People
have cell phones

World Population: 7 Billion

Most companies in the U.S. have at least
100 Terabytes
[100,000 Gigabytes]
of data stored

Source: http://www.ibmbigdatahub.com/infographic/four-vs-big-data
Big Data - Velocity

The New York Stock Exchange captures 1 TB of trade information during each trading session.

Modern cars have close to 100 sensors that monitor items such as fuel level and tire pressure.

By 2016, it is projected there will be 18.9 billion network connections – almost 2.5 connections per person on earth.

Source: http://www.ibmbigdatahub.com/infographic/four-vs-big-data
Big Data - Variety

As of 2011, the global size of data in healthcare was estimated to be 150 EXABYTES (161 BILLION GIGABYTES).

By 2014, it's anticipated there will be 420 MILLION WEARABLE, WIRELESS HEALTH MONITORS.

4 BILLION+ HOURS OF VIDEO are watched on YouTube each month.

30 BILLION PIECES OF CONTENT are shared on Facebook every month.

400 MILLION TWEETS are sent per day by about 200 million monthly active users.

Source: http://www.ibmbigdatahub.com/infographic/four-vs-big-data
Big Data - Veracity

1 in 3 business leaders don't trust the information they use to make decisions.

27% of respondents in one survey were unsure of how much of their data was inaccurate.

Poor data quality costs the US economy around $3.1 trillion a year.

Why Data Analytics?

Clive Humby, mathematician:

“Data is the new oil. It’s valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc. to create a valuable entity that drives profitable activity; so must data be broken down, analyzed for it to have value.”
Why Data Analytics?

Eric Schmidt: "One day we had a conversation where we figured we could just try and predict the stock market... and then we decided it was illegal. So we stopped doing that."

http://www.bloombergview.com/articles/2015-01-23/capital-one-fraud-researchers-may-also-have-done-some-fraud
Main issue with Big Data

Problem: Too much data for one computer
- The data does not fit in memory
- The data does not fit on disk

Solution: use parallelism
Types of parallelism

Task Parallelism

Data Parallelism

Input Data

Parallel Processing

Result Data

Aggregation Task
Map-Reduce Paradigm

Paradigm popularized by Hadoop
Map-Reduce example

Counting words in a set of documents
✓ multiples files
✓ a lot of words in each files

We want to compute the frequency of each unique word

• Map: associate a frequency to a word:
  dad : 1

• Reduce: sum frequencies by word:
Hadoop

Framework allowing the distributed processing of a large quantity of information using a simple programming model based on data.

HDFS + Map-Reduce
Hadoop - architecture

Hadoop regroups the processing and the storage of data on the same node (data locality)
Beyond Map-Reduce

As interest in Big Data grows, complex workflows emerge and usage of iterative and interactive pipeline become the norm. Such pipelines require **frequent access to the data**.

Complex and interactive tasks require something Hadoop cannot offer: **an efficient primitive for sharing data**.
Beyond Map-Reduce

Hadoop's Primitive for data sharing: **storage**!

Serialisation and I/O = up to 90% of job time!
Introducing Spark

- Open Source Project since 2010
- Over 1000 developers contributing in 2016
- Founding principles:
  - Ease data scientist's task
  - Provide a rich function library
  - Access diverse data sources
  - **Use cache to avoid data moving**
Principles behind Spark

Source: http://fr.slideshare.net/tsailiming/spark-meetup1-intro-to-spark
Resilient Distributed Dataset (RDD)

- Immutable, partitioned collection of records
- Can only be created through deterministic operations on
  - data in stable storage
  - other RDDs
- Distributed
- Can be cached in memory
- Fault tolerance:
  - Partition replication on multiple nodes
  - Memorization of RDD transformations
Spark Runtime

- **Driver:**
  - defines and invokes actions on RDDs
  - tracks the RDDs’ lineage

- **Worker:**
  - store RDD partitions
  - perform RDD transformations

Source: https://www.cs.cmu.edu/~pavlo/courses/fall2013/static/slides/spark.pdf
## Spark: performance

### Terasort contest

<table>
<thead>
<tr>
<th></th>
<th>Hadoop Record</th>
<th>Spark 100TB</th>
<th>Spark 1PB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td>100TB</td>
<td>100TB</td>
<td>1000 TB</td>
</tr>
<tr>
<td><strong>Duration</strong></td>
<td>72 minutes</td>
<td><strong>23 minutes</strong></td>
<td>234 minutes</td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>2100</td>
<td>206</td>
<td>190</td>
</tr>
<tr>
<td><strong># Cores</strong></td>
<td>50,400</td>
<td>6592</td>
<td>6080</td>
</tr>
<tr>
<td><strong>Instance type</strong></td>
<td>Dedicated</td>
<td>EC2 (i2.x8large)</td>
<td>EC2 (i2.x8large)</td>
</tr>
</tbody>
</table>

- Up to 100x faster than Hadoop MapReduce in memory
- Up to 10x faster on disk

Spark: ecosystem

- Spark SQL
- Spark Streaming
- MLlib (machine learning)
- GraphX (graph)

Apache Spark

Supporting languages:
- Java
- Scala
- Python
- R
Spark RDD: creation

RDD can be created from different external sources. Files can be multiples and compressed

`textFile`

It is also possible to parallelize arbitrary objects*.

`parallelize`
Spark RDD: transformations

- RDDs are immutable.
- These functions create a new RDD lazily (lazy evaluation).

map  distinct  mapPartitions
filter groupByKey  union
flatMap reduceByKey  intersection
cogroup sample sortByKey  cartesian
join
Spark RDD: lazy transformations

- RDD transformations are represented as a directed acyclic graph (DAG).
- The graph represents the lineage of an RDD.
- The transformations are not applied until the execution of an action (LAZY).
- A stage is a series of transformations that don't require a shuffle.
Spark RDD: transformations*

Narrow dependencies

Wide dependencies
Spark RDD: transformations*

Wide dependencies imply partition shuffling between nodes (costly operation)
Spark RDD: actions

Actions produce an **immediate** result that needs to fit in the memory of the driver or on disk.

- `count`  
- `take`  
- `first`  
- `collect`  
- `takeSample`  
- `reduce`  
- `foreach`  
- `saveAsTextFile`  

To execute an action on an RDD:

- scheduler decides the stages from the RDD’s lineage graph
- each stage contains as many pipelined transformations with narrow dependencies as possible
Spark RDD: Lifecycle

transformation → RDD → action → value
Spark RDD: persistence

RDD are not necessarily kept in memory.

- cache

If there is not enough memory to store the whole RDD, different schemes of persistence can be used.

- persist

To evict the RDD from persistent memory.

- unpersist
Spark Program in 4 steps

1. Create input RDDs from external data or parallelize data from in the driver program.
2. Transform RDDs to define new ones using methods such as: `map()`, `filter()`, `join()`, etc.
3. `cache()` the RDDs that will be accessed frequently.
4. Trigger an action such as `reduce()` or `collect()` to execute optimized parallel computation with Spark.
sc = pyspark.SparkContext()

file = sc.textFile("...")

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

counts.saveAsTextFile("...")
def sample(p):
    x, y = random(), random()
    return 1 if x*x + y*y < 1 else 0

sc = pyspark.SparkContext()
count = sc.parallelize(range(0, NUM_SAMPLES))
    .map(sample)
    .reduce(lambda a, b: a + b)

pi_approx = 4.0 * count / NUM_SAMPLES
Hands-on the basics

Using it is the hardest part.
Recap

So far we have

- Created a resilient distributed dataset from unstructured data
- Transformed the data and add value to it
- Saved the data on disk and aggregated results

What can we do if the data is already semi-structured or structured?
**DataFrame**

1. A table, or two-dimensional array-like structure, in which each column contains measurements on one variable, and each row contains one case.

Spark DataFrame

- DataFrame in Spark are RDDs
  - Immutable
  - Lineage tracking
  - Distributed
  - Lazily transformed
  - etc.

- Created by reading data (numerous formats) from storage, a Pandas DataFrame or a Python collection.
Spark DataFrame: creation

DataFrames can be created by reading from different external sources.

```python
ctx.read.format("csv")
ctx.read.format("parquet")
```

DataFrames can also be created from Python lists or Pandas DataFrame

```python
ctx.createDataFrame
```
DataFrames are immutable.
Transformation are evaluated lazily.
Catalyst optimizes the required calculations

- select
- where
- groupBy
- join
- limit
- agg
- sample
- replace
- drop
- dropna
- dropDuplicates
- freqItems
User can defined its own column transformation with the function \texttt{udf}

\begin{verbatim}
slen = udf(lambda s: len(s), IntegerType())
df.select(slen(df.name).alias('slen'))
\end{verbatim}
DataFrame: actions

Actions produce an **immediate** result that needs to fit in the memory of the driver or on disk.

- count
- collect
- foreach
- head
- take
- takeSample
- describe
- approxQuantile
- first
- show
- corr
- cov
DataFrame: SQL

DataFrames can be transformed using a subset of Structured Query Language (SQL). To do so, one first need to register the DataFrame as a table.

```python
registerTempTable('tbl1')
```

SQL queries can then be applied by specifying the table.

```python
ctx.sql('FROM tbl1 SELECT *')
```

The result is a new DataFrame
Spark DataFrame: persistence

DataFrame are not necessarily kept in memory.

```scala
cache
```

If there is not enough memory, different schemes of persistence can be used.

```scala
persist
```

To evict the DataFrame from persistent memory.

```scala
unpersist
```
Interoperability

Spark DataFrame can read from a variety of source and write back the results in the same format.

{ JSON }  JDBC  Parquet  Since 2.0

HIVE  MySQL  PostgreSQL

HDFS  S3  H2

Dataframe can also transformed into an RDD.
PySpark: averaging a column

RDD

data = sc.textFile("...").map(lambda x: x.split("\t"))
data.map(lambda x: (x[0], (int(x[1]), 1)))
    .reduceByKey(lambda x,y: map(sum(zip(x,y))))
    .mapValues(lambda x: x[1][0] / x[1][1])
    .collect()

DataFrame

df = spark.read.format("csv")
    .options(sep="\t", header=True)
    .load("...")

df.groupby("name").agg("name", avg("age")).collect()
Dataframe: performance

- DataFrame SQL
- DataFrame R
- DataFrame Python
- DataFrame Scala
- RDD Python
- RDD Scala

Time to Aggregate 10 million int pairs (secs)
Hands-on DataFrames

Using it is the hardest part.
Comparison with MPI

- Not a lot of effort have been put so far to compare both technologies.
- Alex Gittens et al. recently took interest in comparing performances for computing scientific matrix decompositions.
- Implemented in Spark and MPI the following
  - Principal Component Analysis (PCA)
  - Negative Matrix Factorization (NMF)
  - Randomized Column Subset Selection (CX)

https://amplab.cs.berkeley.edu/scientific-matrix-factorizations-in-spark-at-scale/
## Comparison with MPI

<table>
<thead>
<tr>
<th>Science Area</th>
<th>Format/Files</th>
<th>Dimensions</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSI</td>
<td>Parquet/2880</td>
<td>8,258,911 × 131,048</td>
<td>1.1TB</td>
</tr>
<tr>
<td>Daya Bay</td>
<td>HDF5/1</td>
<td>1,099,413,914 × 192</td>
<td>1.6TB</td>
</tr>
<tr>
<td>Ocean</td>
<td>HDF5/1</td>
<td>6,349,676 × 46,715</td>
<td>2.2TB</td>
</tr>
<tr>
<td>Atmosphere</td>
<td>HDF5/1</td>
<td>26,542,080 × 81,600</td>
<td>16TB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nodes / cores</th>
<th>MPI Time</th>
<th>Spark Time</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 / 1,600</td>
<td>1 min 6 s</td>
<td>4 min 38 s</td>
<td>4.2x</td>
</tr>
<tr>
<td>100 / 3,200</td>
<td>45 s</td>
<td>3 min 27 s</td>
<td>4.6x</td>
</tr>
<tr>
<td>300 / 9,600</td>
<td>30 s</td>
<td>70 s</td>
<td>2.3x</td>
</tr>
<tr>
<td>PCA (2.2TB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 / 3,200</td>
<td>1 min 34 s</td>
<td>15 min 34 s</td>
<td>9.9x</td>
</tr>
<tr>
<td>300 / 9,600</td>
<td>1 min</td>
<td>13 min 47 s</td>
<td>13.8x</td>
</tr>
<tr>
<td>500 / 16,000</td>
<td>56 s</td>
<td>19 min 20 s</td>
<td>20.7x</td>
</tr>
<tr>
<td>PCA (16TB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI: 1,600 / 51,200</td>
<td>2 min 40 s</td>
<td>69 min 35 s</td>
<td>26x</td>
</tr>
<tr>
<td>Spark: 1,522 / 48,704</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Key points

- Computation time is similar, but
- Spark mechanisms to schedule tasks, fault-tolerance, etc. introduces overheads
Why Spark then?

Short answer: ease of use without HPC knowledge

1. Accessibility: while MPI may be faster for now, Spark is easier to use for scientists without prior experience with distributed computing.

2. Checkpointing: regardless of the algorithm, computation done in Spark can be checkpointed using the same mechanism and be restarted with more ease than BLCR.

3. IO support: Spark algorithms could be applied on a large variety of data sources without effort.
Workshop Summary

1. Introduction to Big Data
   a. Map-Reduce Paradigm
2. Introduction to Apache Spark
   a. Resilient Distributed Dataset (RDD)
   b. API
3. Apache Spark SQL
   a. Working with structured data
   b. DataFrames
4. Quick comparison with MPI
Survey

Thanks for letting us know your impressions on the day

http://tinyurl.com/hpcs-spark