Big data techniques

Kording Lab, RIC
Take home message

• It is easier than ever before to analyze large datasets

• Amazon AWS:
  – $70 for 1 day access to:
    • 5 TB distributed file system
    • 250 GB (distributed) RAM

• Facebook:
  – 4800 cores, 12 PetaBytes (Sept 2009)
Contents

• MapReduce
• Hadoop
• Spark
• Thunder
MapReduce

• Google’s 2004 paper on how to process lots of data
• Commodity hardware
• Fault tolerant
Word count example

Wikipedia

Document 1
Document 2
Document 3

Map

Document 1

Line 1
Line 2

Map

Line

Line

Reduce

('the', 1)
('world', 1)

f(a, b) => a + b

Reduce

('a', 20)
('lab', 40)
('world', 40)

Reduce
Hadoop

- Open source implementation of MapReduce
- HDFS + MapReduce infrastructure

Data (HDFS) → Map → Map → Reduce → Reduce → Result

map(element) → reduce by key(a, b) => c
Hadoop

- Hadoop uses single programming model: MapReduce
- It only on hard drives
Spark

- RAM bandwidth has been increasing exponentially
- Spark can perform in-memory computations
Spark

- Spark can run arbitrary workflows
- It keeps track of transformation’s lineage
Spark

• Spark can access:
  – Local files
  – Hadoop distributed file system (HDFS)
  – Amazon’s Simple Storage Service (S3)

• Several interfaces:
  – Scala (Java functional programming)
  – Java
  – **Python**
Spark example

- Pyspark:

```
In [18]: rdd_readme = sc.textFile('README.md')

In [19]: rdd_readme
Out[19]: README.md MappedRDD[18] at textFile at NativeMethodAccessorImpl.java:-2

In [20]: rdd_readme.count()
Out[20]: 141

In [21]: rdd_readme.take(1)
Out[21]: [u'# Apache Spark']

In [22]: split_words = rdd_readme.flatMap(lambda x: x.split())

Out[22]: PythonRDD[21] at RDD at PythonRDD.scala:43

In [25]: initial_reduce = split_words.map(lambda x: (x, 1))

In [28]: initial_reduce.take(1)
Out[28]: [(u'#', 1)]
```

Lazy evaluation
Spark example (cont.)

• Pyspark:

In [31]: word_count = initial_reduce.reduceByKey(lambda a, b: a + b)

In [32]: word_count.count()

Out[32]: 347

In [33]: word_count.collect()

Out[33]: [(u'email', 1),
          (u'Contributing', 1),
          (u'webpage', 1),
          (u'when', 3),
          (u'<http://spark.apache.org/documentation.html>', 1),
          (u'not', 2),
          (u'including', 2),
          (u'computation', 1),
          (u'file', 1),
          (u'using', 1),
          (u'Scala', 1),
          ...
]

In [35]: rdd_readme.flatMap(lambda x: x.split()).map(lambda x: (x, 1)).reduceByKey(lambda a, b: a + b).collect()
Spark flexibility

- **MLlib:**
  - Basic statistics
    - summary statistics
    - correlations
    - stratified sampling
    - **hypothesis testing**
    - random data generation
  - Classification and regression
    - linear models (SVMs, logistic regression, linear regression)
    - decision trees
    - naive Bayes
  - Collaborative filtering
    - alternating least squares (ALS)
  - Clustering
    - k-means
  - Dimensionality reduction
    - singular value decomposition (SVD)
    - **principal component analysis (PCA)**
  - Feature extraction and transformation
  - Optimization (developer)
    - stochastic gradient descent
    - **limited-memory BFGS (L-BFGS)**
Spark flexibility

- **GraphX:**
  - Operations on big graphs
    - Page Rank
    - Connected components
    - Triangle counting
- **Spark SQL**
- **Spark streaming:**
  - Process data continuously
Thunder

- Develop by Freeman Lab
- Some algorithms specific for neural data analysis
Running Spark and Thunder on Quest

- Quest is good for high performance computing
- But it does not have large storage
Live example

• Running IPython notebook on QUEST
• SSH tunneling to Quest (run notebook)
• Install Spark and Thunder
• Simple example of Spark
• Simple example of Thunder (local computer)
Install Spark and Thunder

Add `.bashrc path a little bit`

```
>> module load python/anaconda
>> module load java
```

Install Spark

```
>> git clone https://github.com/apache/spark
>> ./sbt/sbt assembly
```

Install Thunder

```
>> pip install thunder-python --user
```

In `.bash_profile`

```
>> export SPARK_HOME=~/Installed_Program/spark
>> alias thunder='/home/<user>/.local/bin/thunder'
```
Run Ipython and Spark on Quest

Ask who is on quest?? (Pavan)
>> ./whosonquest

Log in to quest (login node or node)
>> msub -I -l nodes=2:ppn=10 -A b1024 -l walltime=5:00:00
>> ./sbin/start-all.sh
>> export IPYTHON_OPTS=’notebook --profile klab --no-browser --port=9999’
>> ./bin/pyspark --master spark://qnode0599:7077

Tunneling back from computer
>> ssh -L 9999:localhost:9999 tal138@quest.it.northwestern.edu ssh -L 9999:localhost:9999 -N tal138@qnode0599

At your local browser
>> http://localhost:9999/

And don’t forget to kill
>> ps aux | grep <net_id>
Run Thunder (local computer)

- At your local browser

```
>> export IPYTHON_OPTS='notebook --profile klab'
>> thunder
```

- Examples of Thunder (provided in Thunder webpage)