Performance Analysis of Spark and Hadoop

Christian Rachmaninoff and Unni Krishnan

Problem

- Set out to reproduce the Spark and Hadoop benchmarks from the Spark Paper
- Potential Bottlenecks within Hadoop as compared to Spark:
  - Repeated Disk reads
  - Object Serialization Cost
  - Inefficient Job Scheduling/Resource Management
- Chose to focus primarily on iteration times for Logistic Regression and PageRank

Spark vs Hadoop Overview

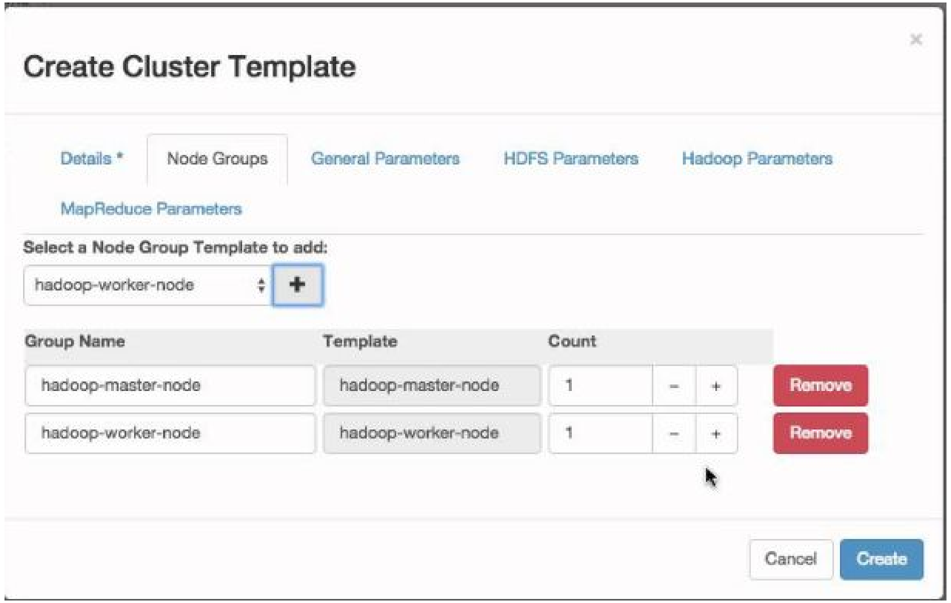
- RDD(Resilient Distributed Datasets) - Spark's in memory datasets (now called DataFrames)
  - In order to get true iteration times in Spark must perform some type of action (i.e. collect) within loop
- Modes of execution of job - Hadoop
  - Single Node
  - Cluster
- Modes of execution in spark
  - Standalone cluster
  - Apache Mesos
  - Hadoop Yarn

Project Plan

- We set out to measure iteration times for the following configurations:
    1. Spark, in-memory storage, with default settings
    2. Spark, no persistence whatsoever, RDDS are recomputed every time
    3. In-memory RDDS, with java serialization
    4. In-memory RDDS, with kryo serialization
    5. On-disk RDDs, with java Serialization
    6. On-disk RDDs with kryo Serialization
    7. Mapreduce with on disk text file
    8. Mapreduce with in memory text file
    9. Mapreduce, with on disk binary file
    10. MapReduce with in memory binary file

Initial Configuration Attempt

- Openstack Mitaka with Sahara Plugin
  - Streamlined Cluster Deployment and job submission
  - http://sahara-files.mirantis.com/images/upstream/
  - Unfortunately, clusters would not successfully, attributed to keystone authentication error
  - After significant effort, looked into other alternatives



Create Cluster Template

Select a Node Group Template to add: hadoop-worker-node

Group Name	Template	Count
hadoop-master-node	hadoop-master-node	1
hadoop-worker-node	hadoop-worker-node	1

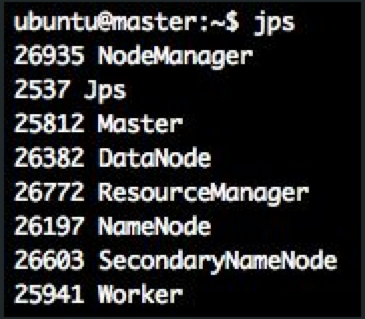
Google Cloud Platform

- Advantages
  - Again, easier Cluster Deployment and job submission
  - Essentially no setup/configuration required
- Drawbacks
  - Limited Resources for free trial, 4GB RAM per machine and 6 total cores available
  - Limited configuration options
  - http://apacheignite.gridgain.org/v1.2/docs/gce-configuration

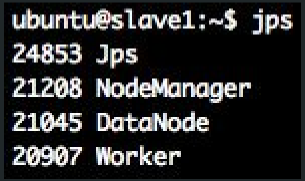
Manual Installation

- Created 5 VM's on Openstack Mitaka (1 Master, 3 slaves, 1 ResourceManager)
- Downloaded hadoop-2.7.1 and replicated configuration across all nodes
- Ran into several connection errors
- Configuration lost manually lost when experiment was swapped out
- To prevent another major setback, automated installation and configuration of hadoop and spark using TCL and shell scripts

Cluster Setup



```
ubuntu@master:~$ jps
26935 NodeManager
2537 Jps
25812 Master
26382 DataNode
26772 ResourceManager
26197 NameNode
26603 SecondaryNameNode
25941 Worker
```



```
ubuntu@slave1:~$ jps
24853 Jps
21208 NodeManager
21045 DataNode
20907 Worker
```

- 4 Cloudlab Compute nodes
  - 4 OpenStack VMs
  - Ubuntu 14.04
  - 3 Hadoop slave nodes
  - One node acting as both the ResourceManager and NameNode.
  - Each VM allocated 80 GB of Storage, 8GB RAM, and 4 virtual CPUs.
- Installed Spark on top of this cluster
  - Allowed for access to both YARN, the Hadoop ResourceManager, and HDFS.

Spark Logistic Regression

- Iterative machine learning algorithm, so Spark should have a significant advantage over Hadoop given that it only has to load data once

```
var w = new DenseVector(a)

 for (i <- 1 to 10) {
   val gradient = data.map(p =>
       p._2 * (1/(1+exp(-p._1*(w dot p._2)))-1)*p._1
   ).reduce((a,b) => a+b)
   for( i <- 0 to 27 ){
     w(i) -= gradient(i)
   }
 }
```

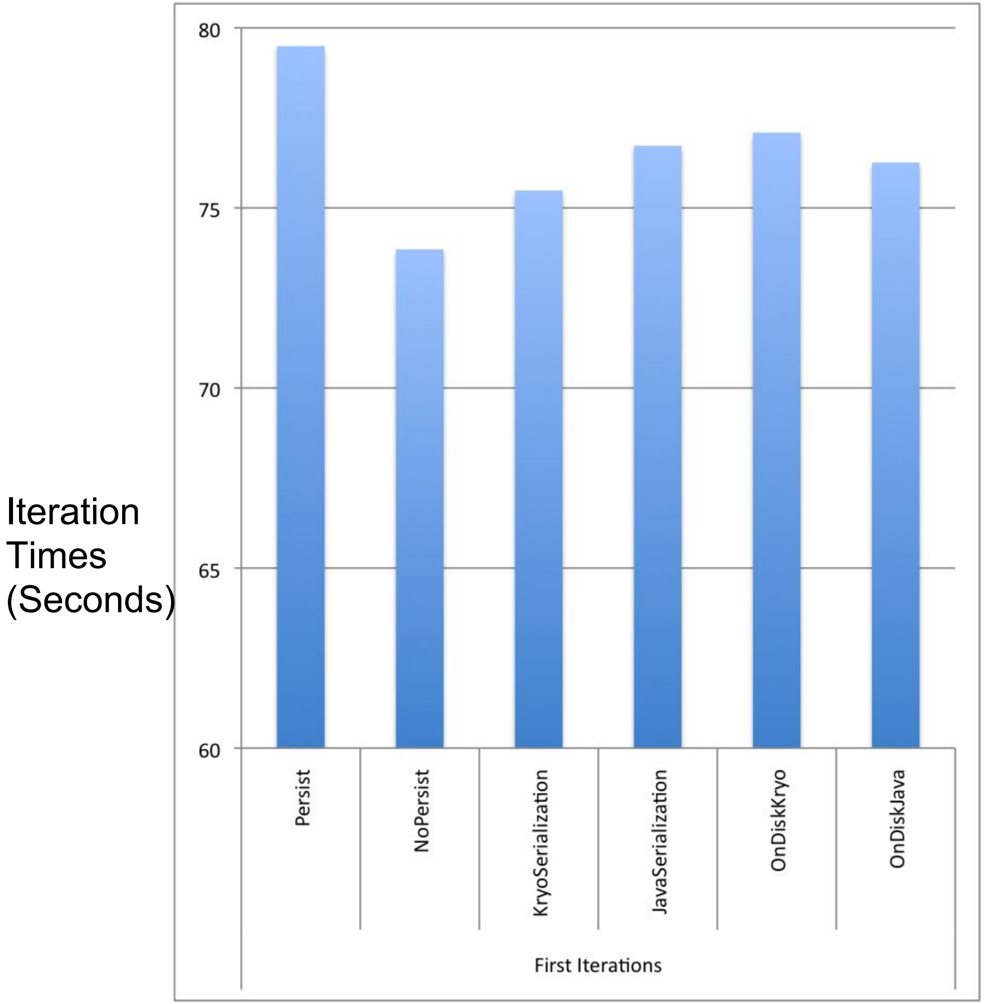
Experiment

- HIGGS Dataset
  - 11 million data points
  - Each point consists of 28 features, of type double and a label
  - ~8GB uncompressed, stored on HDFS with default settings
  - https://archive.ics.uci.edu/ml/datasets/HIGGS
- Clear page cache and swap space before each run
- Vary Persistence and serialization strategies across runs

```
val data = MLUtils.loadLabeledPoints(sc, args(0))
    .map(p => (p.label, new DenseVector(p.features.toArray)))
    .persist(StorageLevel.MEMORY_ONLY_SER)
```

First Iteration Times

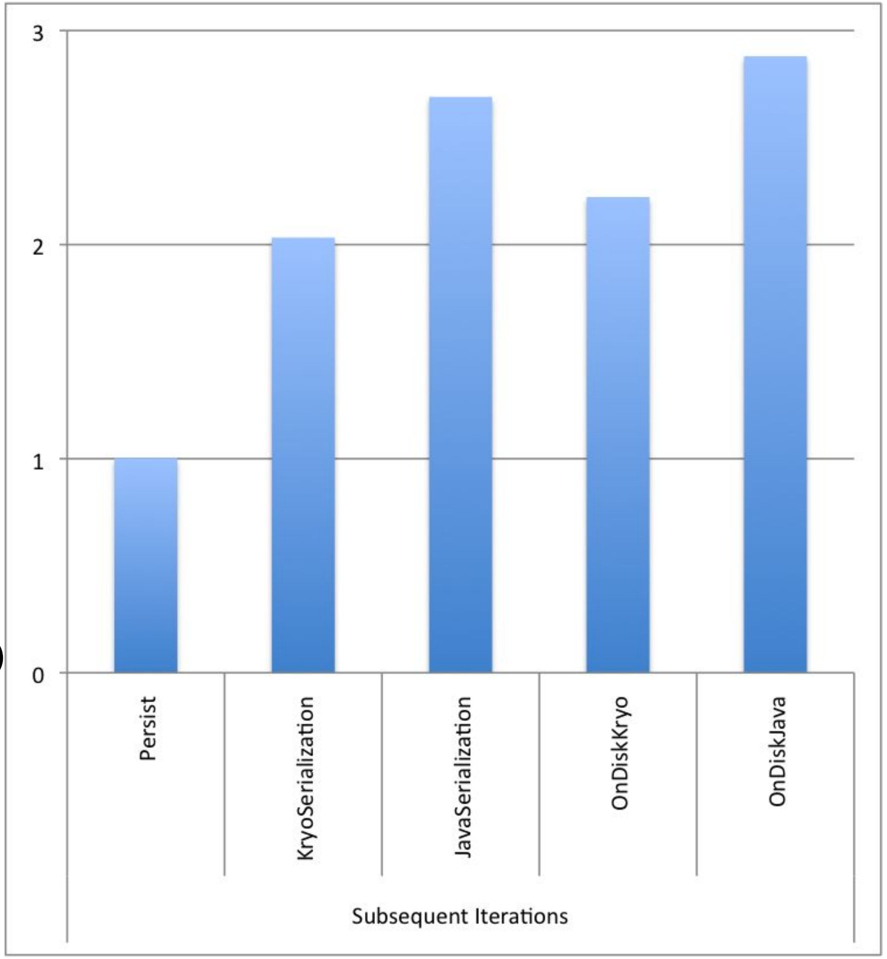
- Persist was slowest
- No Persistence was fastest
- For the most part, no significant difference between strategies
- Reported times are averaged across 3 runs



Subsequent Iterations

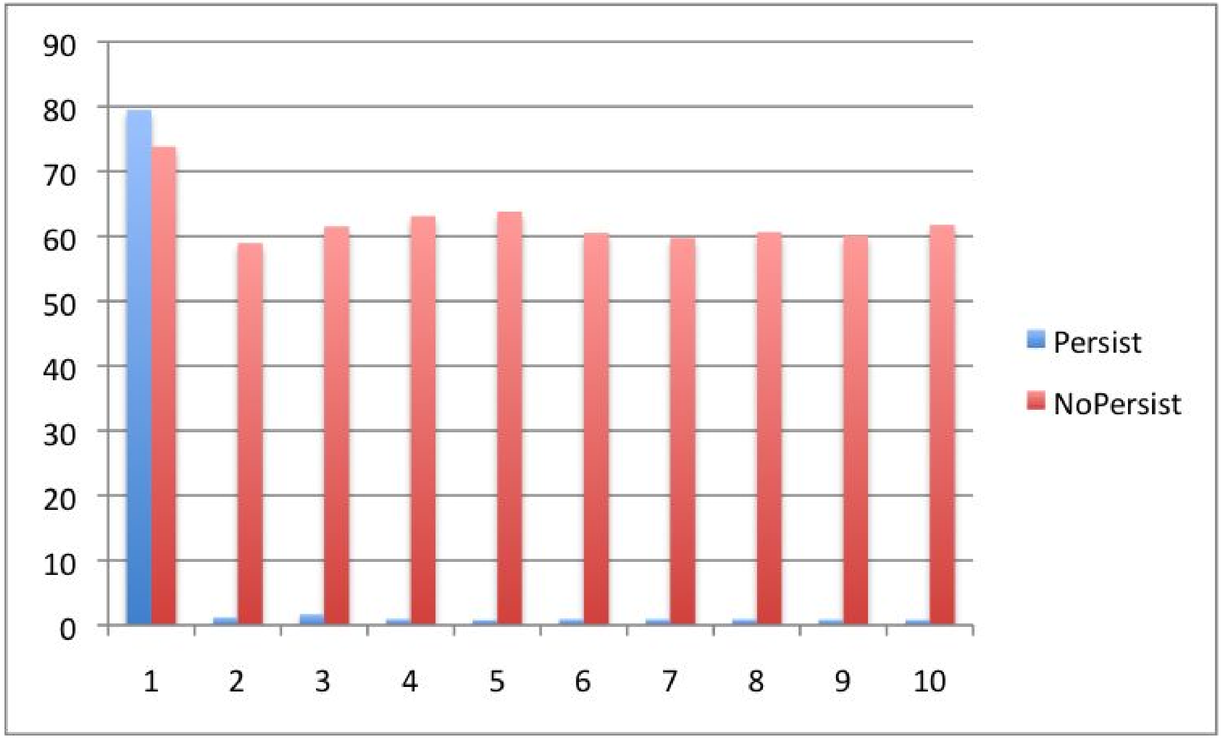
- Persist was fastest
- On disk was only slightly slower for both serialization strategies
- Standard persist was twice as fast as all other strategies

Iteration Times (Seconds)



Persist vs No Persistence

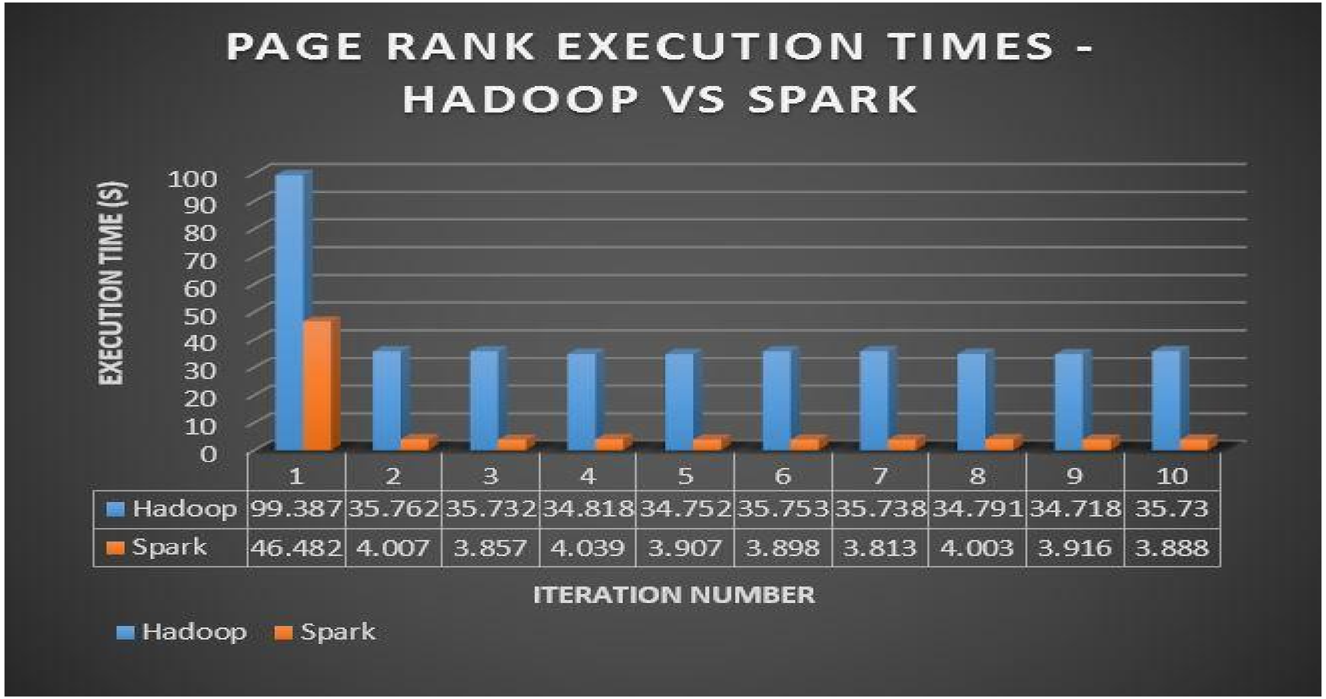
Iteration Times



PageRank - Hadoop vs Spark

- Compared the performance using ~33 MB dump from stanford web-graph, and stored file on HDFS
- The dump contains directed edges of ~2.3 million, ~283k nodes Weblink - http://snap.stanford.edu/data/#web
- PageRank job on spark had memory persistence enabled
- Each spark and hadoop job were executed 10 times iteratively
- Cleared swap space using our script for every job run on spark
- Each spark job was executed with 5G of executor memory and 4 executor cores
- Hadoop job was not executed with any external settings but we observed that not all cores and memory were utilized fully

Spark vs Hadoop(Execution Time)



	1	2	3	4	5	6	7	8	9	10
Hadoop	99.387	35.762	35.732	34.818	34.752	35.753	35.738	34.791	34.718	35.73
Spark	46.482	4.007	3.857	4.039	3.907	3.898	3.813	4.003	3.916	3.888

- **Spark outperformed hadoop by factor of ~8-9x except for the first iteration**

Pending/Future Work

- Already Installed Apache Ignite(hadoop in memory) system on the cluster but unable to start Ignite due to configuration errors
- Run PageRank using ignite and compare the execution times with spark and hadoop
- Compute micro benchmarks like file loading times, time taken for sharding files across the cluster, job starting time in hadoop and spark
- Test spark jobs with limited memory
- Test both with binary files to improve read performance
- Test hadoop pagerank with more number of containers per hadoop job
- Streamline installation of hadoop and spark across 25 nodes using parallel ssh and rsync instead of TCL

Conclusion

- Spark is significantly faster(~8-10 times on average) than hadoop when using iterative algorithms like PageRank(as reported in paper as 7.4x)
- In memory RDD gives an execution advantage compared to MapReduce jobs in Hadoop
- Serialization costs were as not as prominent as initially expected
- Java serialization is not really that slow

Q&A

Thank You!