Parallel Frameworks & Big Data

*Hadoop and Spark on BioHPC*

[web] portal.biohpc.swmed.edu
[email] biohpc-help@utsouthwestern.edu

*Updated for 2016-06-15*
What is ‘Big Data’?

Big data & parallel processing are hand-in-hand

Models of parallelization

Apache Hadoop

Apache Spark
Datasets too big for conventional data processing applications.

In business, often *Predictive Analytics*.

"Your recent Amazon purchases, Tweet score and location history makes you 23.5% welcome here."
Complex, long running analyses on GB datasets ≠ Big Data

Small number of genomic datasets (100s of GBS) no longer ‘big data’

*Easy to manage on individual modern servers with standard apps.*

Cross ‘omics’ integration often is a big data problem.

*Thousands of datasets of different types*  
*genomics, proteomics, metabolomics, imaging...*

*Different formats – import, storage, integration problems*

*Common questions fit the ‘predictive analytics’ model of Big Data in business.*
A Big Data Example – Proteomics DB

Partnership between academia and industry (SAP)

Data size is 7.21TB (not huge) but contains 43 million MS/MS spectra.

Biggest challenge is presentation and interpretation – search, visualization etc.

Web environment uses SAP HANA
160 cores and 2TB RAM behind the scenes

Mass-spectrometry-based draft of the human proteome.

Big Data…. light?

Looks like ‘Big Data’…..

TB scale input data

40,000 core hours of processing on TACC

BUT – all achieved using a traditional single relational database, algorithms etc.

No flexible search / query architecture.


Confetti: a multiprotease map of the HeLa proteome for comprehensive proteomics.

Guo X1, Trudgian DC1, Lemoff A1, Yadavalli S1, Mirzaei H2.
Consider a Big Data problem....

I have 5,000 sequenced exomes with associated clinical observations and want to:

• Find all SNPs in each patient
• Aggregate statistics for SNPs across all patients
• Easily query for SNP associations with any of my clinical variables
• Build a predictive model for a specific disease

A lot of data, a lot of processing – need to do each step in parallel
Parallelization – Shared Memory

1 system, multiple CPUs

Each CPU has portion of RAM attached directly.

CPUs can access their, or other’s RAM.

Parallel processes or threads can access any of the data in RAM easily.
Parallelization – Distributed Shared Memory

RAM is split across many systems that use a special interconnect.

The entire RAM across all machines is accessible anywhere, in a single address space.

Still quite simple to program, but costly and now uncommon.
Many systems, multiple CPUs each. Can only directly access RAM inside single system.

If needed data from other system must pass as a message *across network*.

Must be carefully planned and optimized.

Difficult!
Map Reduce

Impose a rigid structure on tasks. Always map, then reduce
Distributed Computing - Hadoop

A different way of thinking

Specify problem as mapping and reduction steps.

Run on subsets of data on different nodes.

Use special distributed filesystem to communicate data between steps.

The framework takes care of the parallelism

https://en.wikipedia.org/wiki/Apache_Hadoop
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
{
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
Because of the way the cluster is configured, we need to reserve slave nodes for our hadoop / spark jobs before starting them. This is so hadoop/spark have permission to ssh to these nodes.

The dummy job script ‘reserve_myhadoop_node.sbatch’ is provided.

Run....

    sbatch reserve_myhadoop_node.sbatch

...as many times as the number of slave nodes you want to use.
When you are finished working with hadoop/spark cancel these slave reservation jobs...

```
scancel -u $USER -n myhadoop-reserved
```

BUT don’t do this until your main hadoop & spark jobs have finished and cleaned up!
Running a Hadoop Job on BioHPC – sbatch script

#!/bin/bash
# Run on the super partition
#SBATCH -p super

# With a 1h time limit
#SBATCH -t 1:00:00

module add myhadoop/0.30-spark

export HADOOP_CONF_DIR=$PWD/hadoop-conf.$SLURM_JOBID

myhadoop-configure.sh -s /tmp/$USER/$SLURM_JOBID -i 's/Nucleus[0]*/10\10\10/'

$HADOOP_HOME/bin/start-all.sh
$HADOOP_HOME/bin/hadoop dfs -mkdir data
$HADOOP_HOME/bin/hadoop dfs -put ./pg2701.txt data/
$HADOOP_HOME/bin/hadoop dfs -ls data
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-examples-* .jar wordcount data wordcount-output
$HADOOP_HOME/bin/hadoop dfs -ls wordcount-output
$HADOOP_HOME/bin/hadoop dfs -get wordcount-output ./

$HADOOP_HOME/bin/stop-all.sh

myhadoop-cleanup.sh
Hadoop Limitations on BioHPC

Inefficiency
(Small) wait for everything to startup
Workers are solely dedicated to you
Sit idle during portions of job not highly parallelized

HDFS
Uses /tmp on each compute node
Slow HDD – but lots of RAM for caching
Not persistent – deleted after job ends *

Old Hadoop
Running Hadoop 1.2.1
Update to 2.2.0 possible
More up to date if a lot of demand!

* Looking for interested users to try out persistent HDFS on /project
### General Hadoop Limitations

<table>
<thead>
<tr>
<th>Model</th>
<th>Rigid map-&gt;reduce framework hard to model some problems. Iterative algorithms can be difficult (<em>lot of scientific analysis</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Java is only 1st class language. Wrappers / frameworks are other languages available but generally slower.</td>
</tr>
<tr>
<td>HDFS</td>
<td>Always write results to disk after map/reduce. Architecture not good for small files/random reading.</td>
</tr>
</tbody>
</table>

*Many things are alleviated by additional Hadoop projects – Hive, Pig, Hbase etc.*
Apache Spark™ is a fast and general engine for large-scale data processing.

- In-memory computing model
- Load/save data using HDFS or standard file system
- Scala / Java / Python 1st class language support
- Interactive shells for exploratory analysis
- Libraries for database work, machine learning & linear algebra etc.
- Can leverage hadoop features (HDFS, HBASE) or run independently

Far easier and better suited to most scientific tasks than plain Hadoop
Singular Value Decomposition

Challenge: Visualize patterns in a huge assay x gene dataset

Solution: Use SVD to compute eigengenes, visualize data in few dimensions that capture majority of the interesting patterns

\[ X = USV^T \]
import org.apache.spark.rdd.RDD
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.linalg.Vector
import org.apache.spark.mllib.linalg.distributed.RowMatrix

val input = sc.textFile("file:///home2/dtrudgian/Demos/hadoop/matrix.txt")

val rowVectors =
  input.map(
    _.split("\t")
    .map(_.toDouble)
    ).map( v => Vectors.dense(v) ).cache()

val mat=new RowMatrix(rowVectors)

val svd: SingularValueDecomposition[RowMatrix, Matrix] =
  mat.computeSVD(20, computeU = true)

Simple scripting language - This is scala, can also use python
#!/bin/bash
# Run on the super partition
#SBATCH -p super
# With a 1h time limit
#SBATCH -t 1:00:00

module add myhadoop/0.30-spark
export HADOOP_CONF_DIR=$PWD/hadoop-conf.$SLURM_JOBID

myhadoop-configure.sh -s /tmp/$USER/$SLURM_JOBID -i 's/Nucleus[0]*/10\10\10\10/'

$HADOOP_HOME/bin/start-all.sh

source $HADOOP_CONF_DIR/spark/spark-env.sh
myspark start

spark-shell -i large_svd.scala

myspark stop

$HADOOP_HOME/bin/stop-all.sh

myhadoop-cleanup.sh
Must be on a cluster node to connect to spark workers (login node or GUI session)

```bash
# Launch a spark cluster
sbatch slurm_spark_interactive.sh

# Wait for spark to start, then load settings
source hadoop-conf.<jobid>/spark/spark-env.sh

# Connect using interactive scala session
spark-shell    # scala

# ...or interactive python session
pyspark        # python

# And shutdown when done
scancel <jobid>
```
Some python example code – compute Pi

```python
from random import random

NUM_SAMPLES = 1000000

def sample(p):
    x, y = random(), random()
    return 1 if x**2 + y**2 < 1 else 0

count = sc.parallelize(xrange(0, NUM_SAMPLES)) \ 
    .map(sample) \ 
    .reduce(lambda a, b: a + b)

print "Pi is roughly %f" % (4.0 * count / NUM_SAMPLES)
```
What do you want to do?

Discussion

What big-data problems do you have in your research?

Are Hadoop and/or Spark interesting for your projects?

How can we help you use Hadoop/Spark for your work?