Hadoop

- Hadoop is a software platform for running applications that process large sets of data.
- Hadoop is basically two components:
  - MapReduce – a programming paradigm for processing and generating data sets composed of a Map function followed by a Reduce funciton
    - Map –function that runs on all data pieces to generate a new data chunk
    - Reduce – function to merge data chunks from map step
  - Hadoop Distributed File System (HDFS) - creates multiple copies of the data and stores it in a distributed file system.
    - HDFS is designed to be reliable and to run on commodity hardware (that may fail)
- These work in concert because Map Reduce runs the appropriate Map function where the data lives (rather then sending the data)

Hadoop

- Write Map and Reduce functions for WordCount
  - Map tokenizes each line and sets <word, 1> into the pairs
  - Reduce sums for all word

```
14 public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
15    private final static IntWritable one = new IntWritable(1);
16    private Text word = new Text();
17
18    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
19        String line = value.toString();
20        StringTokenizer tokenizer = new StringTokenizer(line);
21        while (tokenizer.hasMoreTokens()) {
22            word.set(tokenizer.nextToken());
23            output.collect(word, one);
24        }
25    }
26 }
27
28 public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
29
30    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
31        int sum = 0;
32        while (values.hasNext()) {
33            sum += values.next().get();
34        }
35        output.collect(key, new IntWritable(sum));
36    }
37 }
```

Hadoop

- Map and Reduce functions are tied into a Job class, which makes them resuable.

```
public static void main(String[] args) throws Exception {
   JobConf conf = new JobConf(WordCount.class);
   conf.setJobName("wordcount");

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

   conf.setMapperClass(Map.class);
   conf.setCombinerClass(Reduce.class);
   conf.setReducerClass(Reduce.class);

   conf.setInputFormat(TextInputFormat.class);
   conf.setOutputFormat(TextOutputFormat.class);

   FileInputFormat.setInputPaths(conf, new Path(args[0]));
   FileOutputFormat.setOutputPath(conf, new Path(args[1]));

   JobClient.runJob(conf);
}
```

Hadoop Programming

- The previous example showed Java, but other options are supported
  - Python – via Jython to produce a jar
  - C and C++
    - Build a binary executable
    - Load the binary onto a location in HDFS
    - Execute the binary be using the built-in pipes algorithm and an XML configuration file.
- Using things like streaming hadoop, it's possible to use things like shell and perl scripts to execute code under hadoop.
- Additional tools allow non-java access to the HDFS-API, so applications can fetch and store data.

HOD

- Hadoop is designed primarily to run as the only thing running on a cluster, not an application executed on a general-purpose system.
- Hadoop on Demand (HOD) is a tool for starting Hadoop on a subset of nodes inside of a general-purpose HPC system. HDFS and MapReduce daemons are brought up as part of a normal job, which results in a temporary Hadoop system which jobs can be run on.
- We have created a patch to run HOD on Ranger and can work with individual clients to help them get this running for their needs (initial configuration help is probably needed as the patch has not yet been formally accepted into Hadoop core).

Hadoop on Demand

- ***Allocation*** – provisions a Hadoop system on a subset of the system and creates a needed config file to run jobs.

```
$ export HOD_PYTHON_HOME=/usr/local/bin/python2.5
$ export HOD_CONF_DIR=${HOD_HOME}/conf
$ cd ${HOD_HOME}/bin
$ ./hod allocate -d ~/hadoop/cluster -n 5 -t ~/hadoop/hadoop-0.18.3.tar.gz -A acctName -l 3600
INFO - Cluster Id 2938.scheduler.v4linux
INFO - HDFS UI at http://compute-3-42.v4linux:54072
INFO - Mapred UI at http://compute-3-43.v4linux:52010
INFO - hadoop-site.xml at /home/gfs01/naw47/hadoop/cluster
$ ./hod list -d ~/hadoop/cluster
INFO - alive 2938.scheduler.v4linux /home/gfs01/naw47/hadoop/cluster
```

- ***Data is a problem*** – Using HOD this way means that data must be reloaded onto HDFS for every use, which means that a significant part of job time may be loading data.

```
$ export HADOOP_CONF_DIR=~/hadoop/cluster
$ cd hadoop-0.18.3/bin
$ ./hadoop fs -mkdir input
$ ./hadoop fs -mkdir output
$ ./hadoop fs -put ~/somefile input/words
$ ./hadooop jar ~/path/to/hadoop-0.18.3-examples.jar wordcount input/words output/wordcount
$ ./hadoop fs -get output/wordcount ~/wc_results
```