Introduction to Hadoop

Owen O'Malley

Yahoo Inc!

omalley@apache.org

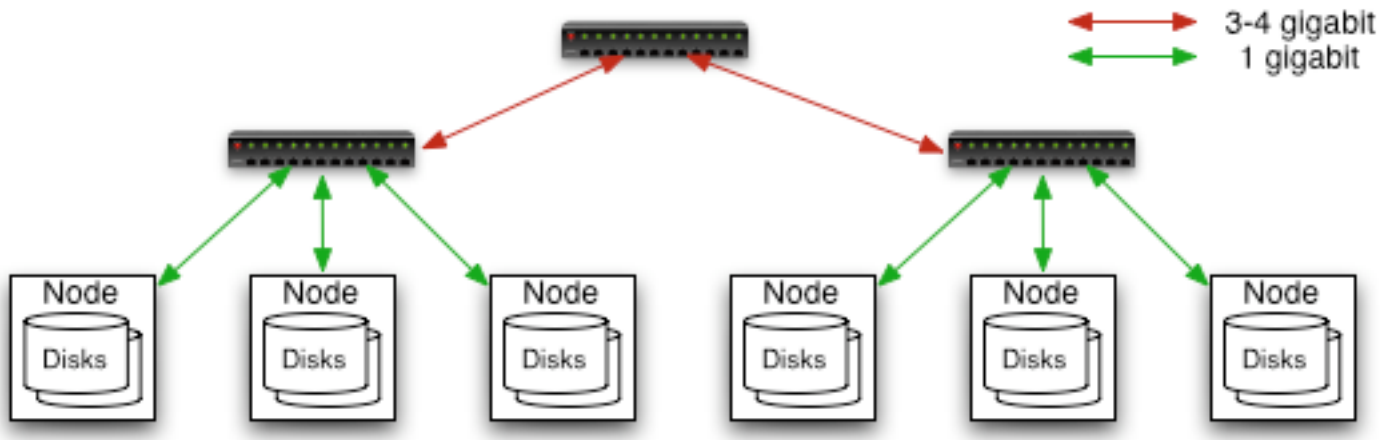
Hadoop: Why?

- Need to process 100TB datasets with multi-day jobs
- On 1 node:
  - scanning @ 50MB/s = 23 days
  - MTBF = 3 years
- On 1000 node cluster:
  - scanning @ 50MB/s = 33 min
  - MTBF = 1 day
- Need framework for distribution
  - Efficient, reliable, easy to use

Hadoop: How?

- Commodity Hardware Cluster
- Distributed File System
  - Modeled on GFS
- Distributed Processing Framework
  - Using Map/Reduce metaphor
- Open Source, Java
  - Apache Lucene subproject

Commodity Hardware Cluster



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 30-40 nodes/rack
  - Uplink from rack is 3-4 gigabit
  - Rack-internal is 1 gigabit

Distributed File System

- Single namespace for entire cluster
  - Managed by a single *namenode*.
  - Hierarchal directories
  - Optimized for streaming reads of large files.
- Files are broken in to large blocks.
  - Typically 64 or 128 MB
  - Replicated to several *datanodes*, for reliability
  - Clients can find location of blocks
- Client talks to both namenode and datanodes
  - Data is not sent through the namenode.

Distributed Processing

- User submits Map/Reduce *job*
- System:
  - Splits job into lots of *tasks*
  - Schedules tasks on nodes close to data
  - Monitors tasks
  - Kills and restarts if they fail/hang/disappear
- Pluggable file systems for input/output
  - Local file system for testing, debugging, etc…

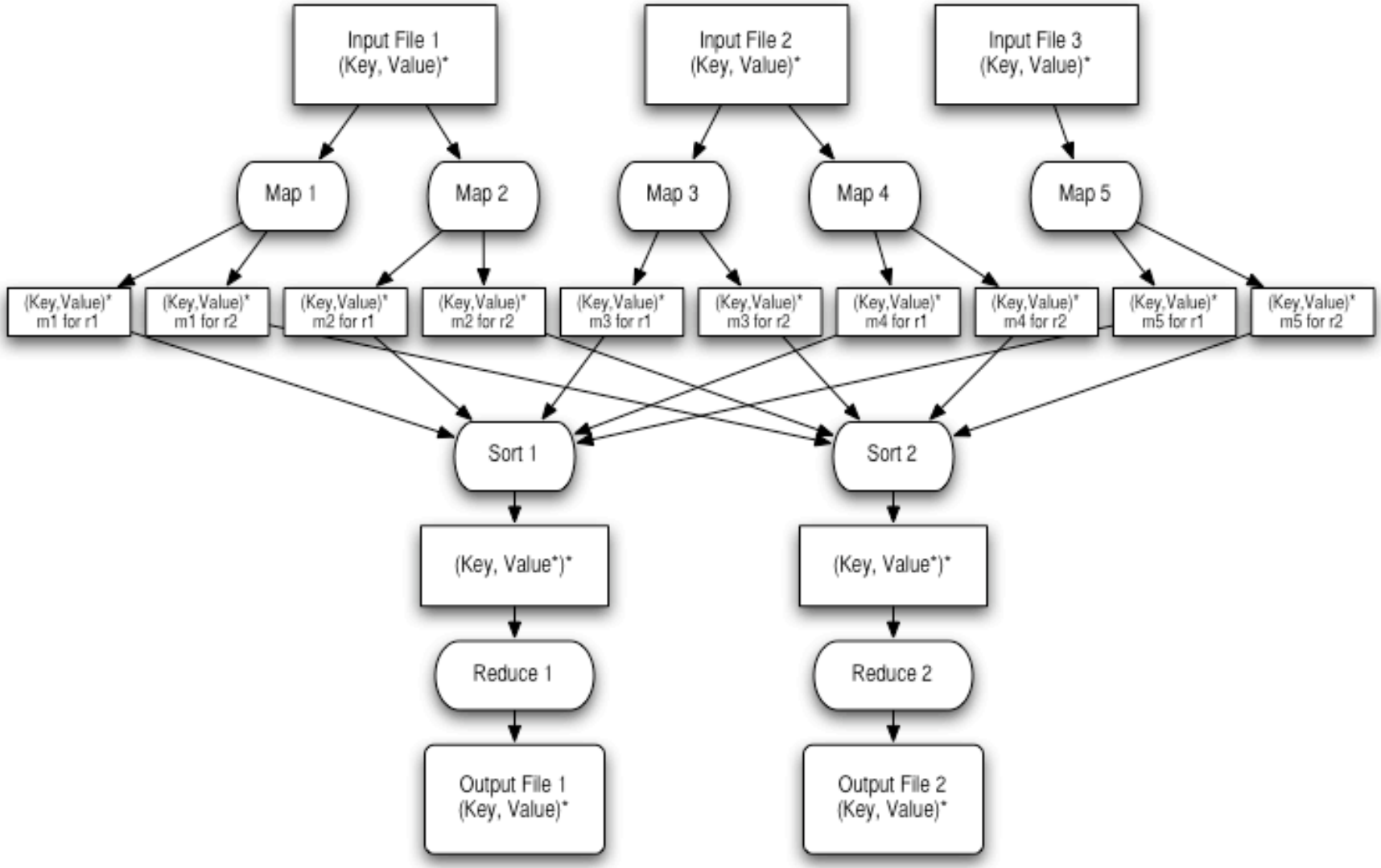
Map/Reduce Metaphor

- Reliable distributed processing of large datasets
- Abstracts a very common pattern (munge, regroup, munge)
- Natural for
  - Building or updating offline databases (eg. indexes)
  - Computing statistics (eg. query log analysis)
- Software framework
  - Frozen part: distributed sort, reliability, and re-execution
  - Hot parts: input, map, partition, compare, reduce, and output

Map/Reduce Metaphor

- Data is a stream of *keys* and *values*
- Mapper
  - Input: *key1,value1* pair
  - Output: *key2, value2* pairs
- Reducer
  - Called once per a key, in sorted order
  - Input: *key2*, **stream** of *value2*
  - Output: *key3, value3* pairs
- Launching Program
  - Creates a JobConf to define a job.
  - Submits JobConf and waits for completion.

Map/Reduce Dataflow



Map/Reduce Optimizations

- Mapper locality
  - Schedule mappers close to the data.
- Combiner
  - Mappers may generate duplicate keys
  - Side-effect free reducer run on mapper node
  - Minimize data size before transfer
  - Reducer is still run
- Speculative execution
  - Some nodes may be slower
  - Run duplicate task on another node

HOWTO: Setting up Cluster

- Modify **hadoop-site.xml** to set directories and master hostnames.
- Create a **slaves** file that lists the worker machines one per a line.
- Run **bin/start-dfs** on the namenode.
- Run **bin/start-mapred** on the jobtracker.

HOWTO: Write Application

- To write a distributed word count program:
  - Mapper: Given a line of text, break it into words and output the word and the count of 1:
    - “hi Apache bye Apache” ->
    - (“hi”, 1), (“Apache”, 1), (“bye”, 1), (“Apache”, 1)
  - Combiner/Reducer: Given a word and a set of counts, output the word and the sum
    - (“Apache”, [1, 1]) -> (“Apache”, 2)
  - Launcher: Builds the configuration and submits job

Word Count Mapper

```
public class WCMap extends MapReduceBase implements Mapper {

  private static final IntWritable ONE = new IntWritable(1);

  public void map(WritableComparable key, Writable value,
                  OutputCollector output,
                  Reporter reporter) throws IOException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      output.collect(new Text(itr.next()), ONE);
    }
  }
}
```

Word Count Reduce

```
public class WCReduce extends MapReduceBase implements Reducer {

  public void reduce(WritableComparable key, Iterator values,
                     OutputCollector output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += ((IntWritable) values.next()).get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Word Count Launcher

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

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

  conf.setMapperClass(WCMap.class);
  conf.setCombinerClass(WCReduce.class);
  conf.setReducerClass(WCReduce.class);

  conf.setInputPath(new Path(args[0]));
  conf.setOutputPath(new Path(args[1]));

  JobClient.runJob(conf);
}
```

Running on Amazon EC2/S3

- Amazon sells cluster services
  - EC2: $0.10/cpu hour
  - S3: $0.20/gigabyte month
- Hadoop supports:
  - EC2: cluster management scripts included
  - S3: file system implementation included
- Tested on 400 node cluster
- Combination used by several startups

Scalability

- Runs on 1000 nodes
- 5TB sort on 500 nodes takes 2.25 hours
- Distributed File System:
  - 150 TB
  - 3M files

Thank You

- Questions?
- For more information:
  - http://lucene.apache.org/hadoop/