Outline

- MapReduce Overview
- Hadoop–WordCount Demo
- Hadoop–Blast Demo
- Q&A
MapReduce in Simple Terms

- Mapreduce – The Story of Tom
- By Saliya Ekanayake
Map
- Apply the same function \( f \) to all the elements of a list

Fold
- Moves across a list, applying function \( g \) to each element plus an *accumulator*.
MapReduce Programming Model

- influenced by Functional Programming constructs
- Users implement interface of two functions:
  - `map (in_key, in_value) -> (out_key, intermediate_value) list`
  - `reduce (out_key, intermediate_value list) -> out_value list`

Source: http://code.google.com/edu/parallel/mapreduce-tutorial.html
A list of data elements are passed, one at a time, to map() functions which transform each data element to an individual output data element.

A map() produces one or more intermediate <key, values> pair(s) from the input list.
After map phase finish, those intermediate values with same output key are reduced into one or more final values.

**Intermediate map output**

| k’1 | V’1 | k’2 | V’2 | k’3 | V’3 | k’4 | V’4 | k’5 | V’5 | k’6 | V’6 | …… |

- Reduce
- Reduce
- Reduce

**Final Result**

| F1 | R1 | F2 | R2 | F3 | R3 | …… |
map() functions run in parallel, creating different intermediate values from different input data elements
reduce() functions also run in parallel, working with assigned output key
All values are processed independently
Reduce phase can’t start until map phase is completely finished.

Source: http://code.google.com/edu/parallel/mapreduce-tutorial.html
Apache Hadoop

- Open Source MapReduce framework implementation
- Widely used by
  - Amazon Elastic MapReduce
  - EBay
  - FaceBook
  - TeraSort in Yahoo!
  - ...
- Active community
- Many sub projects
  - HDFS
  - Pig Latin
  - Hbase
  - ...

Source: http://wiki.apache.org/hadoop/PoweredBy
Execution Overview

Source: http://code.google.com/edu/parallel/mapreduce-tutorial.html
MapReduce in Apache Hadoop

- Autoparallelization
- data distribution
- Fault-tolerant
- Status monitoring
- Simple programming constructs
Demo 1 – WordCount

- “Hello World” in MapReduce programming style
- Fits well with the MapReduce programming model
  - count occurrence of each word in given files
  - each file (may be split) run in parallel
  - need reduce phase to collect final result
- Three main parts:
  - Mapper
  - Reducer
  - Driver
Word Count

Input
foo car bar
foo bar foo
car car car

Mapping
foo, 1
car, 1
bar, 1

Shuffling
foo, 1
foo, 1
foo, 1

Reducing
foo, 3
bar, 2
car, 4
void Map (key, value) {
    for each word x in value:
        output.collect(x, 1);
}

WordCount Map
public static class Map
    extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
void Reduce (keyword, <list of value>){
    for each x in <list of value>:
        sum+=x;
        output.collect(keyword, sum);
}
public static class Reduce
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, 
                        Context context) throws .....{
        int sum = 0; // initialize the sum for each keyword
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);

        context.write(key, result);
    }
}
Log in to FutureGrid

- `qsub -l`
  - Get a FutureGrid Work node
- `hostname`
  - e.g. i51
  - Public hostname name should be
    - i51r.idp.iu.futuregrid.org
    - http://salsahpc.indiana.edu/tutorial/futuregrid_access.html
Building WordCount program

- cd Hadoop-WordCount
- cat WordCount.java
- cat build.sh
- ./build.sh
- ls -l
Running WordCount using Hadoop standalone

- cd ~/hadoop-0.20.2-standalone/bin
- cp ~/Hadoop-WordCount/wordcount.jar ~/hadoop-0.20.2-standalone/bin
- ./hadoop jar wordcount.jar WordCount ~/Hadoop-WordCount/input ~/Hadoop-WordCount/output
  - Simulate HDFS using the local directories
- cd ~/Hadoop-WordCount/
- cat output/part-r-00000
Optimizations

- Reduce phase can only be activated until all map tasks finish
  - It will be a waste if there is an extremely long map task
- “Combiner” functions can run on the same machine as a mapper
- Causes a mini-reduce phase to occur before the real reduce phase, to save bandwidth
Word Count with Combiner

Input: foo, car, bar
Mapping: foo, 1, car, 1, bar, 1
Combiner: foo, 1, car, 1, bar, 1
Shuffling: foo, 1, foo, 2, bar, 1, bar, 1, car, 4
Reducing: foo, 3, bar, 2, car, 4
Terminology

- Job
- Jobtracker
- Task
- TaskTracker
- Shard
HDFS

- Based on Google File System
- Run on commodity low-cost hardware
- Fault tolerance
  - Redundant storage
- High throughput access
- Suitable for large data sets
- Large capacity
- Integration with MapReduce
Architecture

- Single NameNode
- Multiple Data Nodes

Configuring HDFS

- dfs.name.dir
  - Dir in the namenode to store metadata
- dfs.data.dir
  - Dir in data nodes to store the data blocks
  - Must be in a local disk partition

```xml
<property>
  <name>dfs.name.dir</name>
  <value>/tmp/hadoop-test/name</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/tmp/hadoop-test/data</value>
</property>
```
Configuring HDFS

- **fs.default.name**
  - URI of the namenode

  ```xml
  <property>
    <name>fs.default.name</name>
    <value>hdfs://149.79.89.113:9000/</value>
  </property>
  ```

- **conf/masters**
  - ip address of the master node

- **conf/slaves**
  - ip address of the slave nodes

- **Should have password-less SSH access to all the nodes**
HDFS Cluster

- **Starting HDFS**
  - `cd ~/hadoop-0.20.2/bin`
  - `./hadoop namenode -format`
  - `./start-dfs.sh`

- **NameNode logs**
  - `cd ~/hadoop-0.20.2`
  - `logs/hadoop-<username>-namenode-<nodeid>.log`

- **Monitoring web interface**
  - `http://<public_ip>:50070`
HDFS Commands

`./bin/hadoop fs -[command]`
- `put`
- `get`
- `ls`
- `cp`
- `mkdir`
- `rm`

[http://hadoop.apache.org/common/docs/current/hdfs_shell.html](http://hadoop.apache.org/common/docs/current/hdfs_shell.html)

Programmatic API
Configuring Hadoop MapReduce

- mapred.job.tracker
- mapred.local.dir
- mapred.tasktracker.map.tasks.maximum

```xml
<property>
  <name>mapred.job.tracker</name>
  <value>149.79.89.113:9001</value>
</property>
<property>
  <name>mapred.local.dir</name>
  <value>/tmp/hadoop-test/local</value>
</property>
<property>
  <name>mapred.tasktracker.map.tasks.maximum</name>
  <value>8</value>
</property>
```
Starting Hadoop MapReduce Cluster

- Starting Hadoop MapReduce
  - cd ~/hadoop-0.20.2/bin
  - ./start-mapred.sh

- JobTracker logs
  - cd ~/hadoop-0.20.2/logs/
  - logs/hadoop-<username>-tasktracker<nodeid>.log

- Monitoring web interface
  - http://<public_ip>:50030
Running WordCount on Distributed Hadoop Cluster

- Upload input to HDFS
  - cd ~/hadoop-0.20.2/bin
  - ./hadoop fs -put ~/Hadoop-WordCount/input/input
  - ./hadoop fs -ls input

- Run word count
  - ./hadoop jar wordcount.jar WordCount input output
  - ./hadoop fs -ls output
  - ./hadoop fs -cat output/*
Demo 2: Hadoop–Blast

- An advance MapReduce implementation
- Use Blast (*Basic Local Alignment Search Tool*), a well-known bioinformatics application written in C/C++
- Utilize the Computing Capability of Hadoop
- Use Distributed Cache for the Blast Program and Database
- No Reducer in practice
public void map(String key, String value, Context context) throws IOException, InterruptedException {
    ...
    // download the file from HDFS
    ...
    fs.copyToLocalFile(inputFilePath, new Path(localInputFile));
    // Prepare the arguments to the executable
    ...
    execCommand = this.localBlastProgram + File.separator + execName + " " + execCommand + " -db " + this.localDB;

    // Create the external process
    Process p = Runtime.getRuntime().exec(execCommand);
    ...
    p.waitFor();
    // Upload the results to HDFS
    ...
    fs.copyFromLocalFile(new Path(outFile),outputFileName);
...
} // end of overriding the map
// using distributed cache
DistributedCache.addCacheArchive(new URI(BlastProgramAndDB), jc);

// other parts

// input and output format
job.setInputFormatClass(DataFileInputFormat.class);
job.setOutputFormatClass(SequenceFileOutputFormat.class);

...
Preparation

- Make sure the HDFS and Map–Reduce daemon start correctly
- Put Blast queries on HDFS
  - cd ~/hadoop-0.20.2/bin
  - ./hadoop fs -put ~/hadoop-0.20.2/apps/Hadoop-Blast/input HDFS_blast_input
  - ../hadoop fs -ls HDFS_blast_input
- Copy Blast Program and DB to HDFS
  - ../hadoop fs -copyFromLocal $BLAST_HOME/BlastProgramAndDB.tar.gz
    BlastProgramAndDB.tar.gz
  - ../hadoop fs -ls BlastProgramAndDB.tar.gz
Run the Hadoop–Blast program
- cd ~/hadoop-0.20.2/bin
- ./hadoop jar ~/hadoop-0.20.2/apps/Hadoop-Blast/executable/blast-hadoop.jar BlastProgramAndDB.tar.gz bin/blastx /tmp/hadoop-test/ db nr HDFS_blast_input HDFS_blast_output '-query #_INPUTFILE_# -outfmt 6 -seg no -out #_OUTPUTFILE_#'

Check the Result
- cd ~/hadoop-0.20.2/bin
- ./hadoop fs -ls HDFS_blast_output
- ./hadoop fs -cat HDFS_blast_output/pre_1.fa
Performance Tips at the end

- More Map tasks
  - Much more than the Map task capacity of the cluster
- Number of reduce tasks
  - 95% of the Reduce task capacity
  - Interleave computation with communication
- Data locality
- Map/Reduce task capacity of a worker
- Logs
  - Don’t store in NFS. They grow pretty fast.
References

- Big Data for Science Tutorial
- Yahoo! Hadoop Tutorial
- Hadoop wiki
- Apache Hadoop Map/Reduce Tutorial
- Google: Cluster Computing and MapReduce
Questions?
Thank You!!
WordCount

Input
- foo
- car
- bar

Mapping
- foo, 1
- car, 1
- bar, 1

Shuffling
- foo, 1
- car, 1
- bar, 1
- foo, 1
- car, 1
- bar, 1

Sorting
- bar, <1,1>
- car, <1,1,1,1>
- foo, <1,1,1>

Reducing
- bar, 2
- car, 4
- foo, 3

Execute 3 times

Framework level
MapReduce Fault Tolerance

- MasterNode handles failures
  - detects worker failures
  - rerun completed & in-progress map() tasks
  - rerun in-progress reduce() tasks
  - If particular input key/values cause crashes in map phase, and skips those values on re-execution.
    - Partly output

Source: http://code.google.com/edu/parallel/mapreduce-tutorial.html
MapReduce Locality

- Master program tries to have map() tasks on same machine as physical file data, or at least on same rack
- map task inputs are spilt into blocks with default sizes as 64MB.

Source: http://code.google.com/edu/parallel/mapreduce-tutorial.html
HDFS Assumptions

- High component failure rates
  - Inexpensive commodity components fail all the time
- “Modest” number of HUGE files
  - Just a few million
  - Each is 100MB or larger; multi-GB files typical
- Files are write–once, read many times
- Large streaming reads
- High sustained throughput favored over low latency
HDFS Design Decisions

- Files stored as blocks
  - Blocks stored across cluster
  - Default block size 64MB
- Reliability through replication
  - Default is 3
- Single master (NameNode) to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Focused on distributed applications
Advanced Hadoop

- Input formats
- Output formats
- Data types
- Distributed Cache
Input Formats

- Defines how to break the inputs
- Provides RecordReader objects to read the files
  - Read data and converts to <key, value> pairs
- It’s possible to write custom input formats
- Hadoop Built-in input formats
  - TextInputFormat
    - Treats each line as value
  - KeyValueTextInputFormat
    - Each line as key-value (tab delimited)
  - SequenceFileInputFormat
    - Hadoop specific binary representation
Output Formats

- How the result `<key,value>` pairs are written in to files
- Possible to write custom output formats
- Hadoop default OutputFormats
  - TextOutputFormat
    - `<key TAB value>`
  - SequenceFileOutputFormat
    - Hadoop specific binary representation
Data Types

- Implement Writable interface
  - Text, IntWritable, LongWritable, FloatWritable, BooleanWritable, etc…
- We can write custom types which Hadoop will serialize/deserialize as long as following is implemented.

```java
public interface Writable {
    void readFields(DataInput in);
    void write(DataOutput out);
}
```
Distributed Cache *

- Distributes a given set of data to all the mappers
  - Common data files, libraries, etc.
  - Similar to a broadcast
  - Available in the local disk
  - Automatically extract compressed archives

- First upload the files to HDFS
- DistributedCache.addCacheFile()
- DistributedCache.getLocalCacheFiles()