Introduction to Hadoop (programming)

LC Hsieh
NTU, 2011

Some slides are inspired by the slides of Philip Zeyliger at http://www.slideshare.net/cloudera/apache-hadoop-talk-at-qcon

Data is everywhere.

Data is important.

Data is exploding.

The Trend of Data Growth

- In 2009, the digital data grew 62% over 2008 to 800 billion GB (0.8 ZB)
- The digital data created in 2010 (1.2 ZB) equals
  - Every man, woman and child on Earth "Tweeting" continuously for 100 years
  - 75 billion fully-loaded 16 GB Apple iPads, which could fill the Taipei 101 Tower 23 times
- The number of files, images, records and other digital information containers will grow by a factor of 67
  - Despite this growth, the number of IT professionals globally will grow only by a factor of 1.4

"I keep saying that the sexy job in the next 10 years will be statisticians, and I'm not kidding."

Hal Varian
Google chief economist

What Is (Apache) Hadoop?

The Apache™ Hadoop™ project develops open-source software for reliable, scalable, distributed computing.

-- http://hadoop.apache.org/

Two Main Components of hadoop

HDFS

hadoop HDFS

High-bandwidth clustered storage

MapReduce

hadoop MapReduce

Fault-tolerant distributed computing

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown

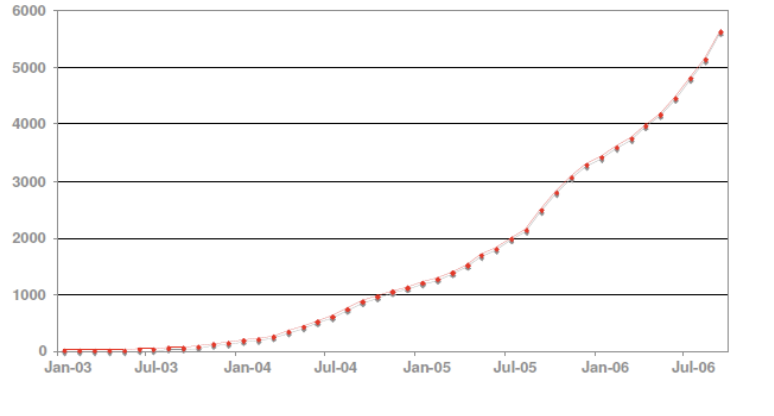
given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

- A simple programming model that applies to many large-scale computing problems [Dean, PACT'06 Keynote]
- MapReduce is a patented software framework introduced by Google to support distributed computing on large data sets on clusters of computer [Wikipedia]
- MapReduce = Distributed computation + distributed storage + scheduling / fault tolerance [Spiros Papadimitriou, KDD Tutorials'2010]

Growth of MapReduce Programs in Google

[Dean, PACT'06 Keynote]

- Growth of MapReduce programs in Google source tree (2003~2006)
  - Distributed grep, distributed sort, web access log stats, web link-graph reversal, inverted index construction, statistical machine translation, machine learning, document clustering, etc.



The M/R Programming Model

Defining your map() and reduce() functions.

The framework takes care other tasks (fault-tolerance, parallelization, etc.) for you.

- map function: the worker processes smaller problem
  - map: K1, V1 -> list(K2, V2)

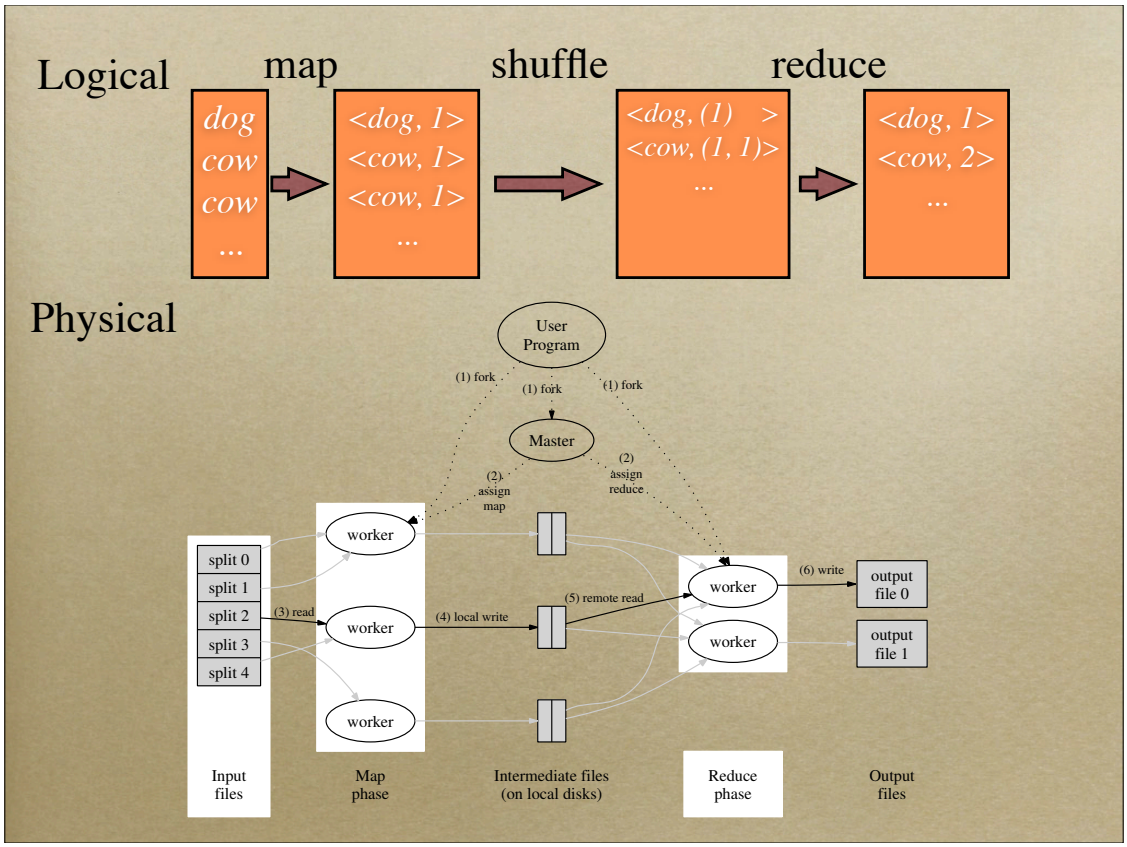
- (shuffle) supported by MapReduce framework
  - map output is assigned to reducer
  - map output is sorted by key

- reduce function: the worker that aggregately processes the intermediate output from mappers
  - reduce: K2, list(V2) -> list(K3, V3)

Word Counting by MapReduce

```
void map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    EmitIntermediate(w, "1");

void reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  int result = 0;
  for each pc in partialCounts:
    result += ParseInt(pc);
  Emit(AsString(result));
```



Hadoop Virtual Image

- You can experiment with Hadoop platform right now without setting a Hadoop cluster by using pre-configured virtual machine images
  - Google Hadoop virtual image [http://code.google.com/edu/parallel/tools/hadoopvm/index.html]
  - Yahoo! Hadoop virtual machine [http://developer.yahoo.com/hadoop/tutorial/module3.html]
- Virtualization software
  - VMware Player as mentioned in previous web pages
  - VirtualBox works with Google Hadoop virtual image

Hadoop Java API

- Map class

```
public class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

    public void map(LongWritable key, Text value, Context context) throws
      IOException, InterruptedException {

    }

}
```

- Reduce class

```
public class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
      Context context) throws IOException, InterruptedException {

    }

}
```

Inverted List Construction

- MapReduce "killer app" [Lin & Dryer, Tutorial at NAACL/HLT 2009]
- From document-term model to term-document model
  - map function tokenizes text document and emits (term, doc id) as key/value pair
  - reduce function aggregates doc ids for every term and outputs (term, <doc id>*) as inverted list

Inverted List Construction: map

```
public class Map extends Mapper<LongWritable, Text, Text, Text> {
        private Text word = new Text();
        private Text doc_id = new Text();

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

            if (tokenizer.hasMoreTokens())
                doc_id.set(tokenizer.nextToken());
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, doc_id);
            }
        }
}
```

Inverted List Construction: reduce

```
public class Reduce extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {

            StringBuffer strbuf = new StringBuffer();
            for (Text val : values) {
                strbuf.append(val.toString() + " ");
            }
            context.write(key, new Text(strbuf.toString()));
        }
}
```

OK. I have written a Hadoop program in Java. How do I run it?

All hadoop commands are invoked by the bin/hadoop script. Running the hadoop script without any arguments prints the description for all commands.

```
Usage: hadoop [--config confdir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

Using command 'jar':

jar

Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command.

```
Usage: hadoop jar <jar> [mainClass] args...
```

For example:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount /usr/joe/
    wordcount/input /usr/joe/wordcount/output
```

Put your data on HDFS:

fs

`Usage: hadoop fs [GENERIC_OPTIONS] [COMMAND_OPTIONS]`

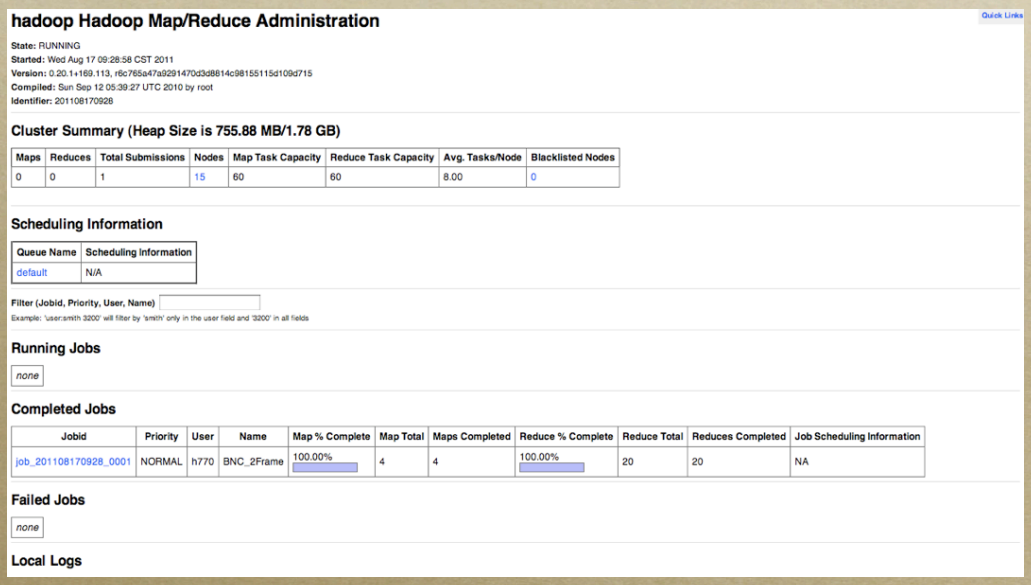
Runs a generic filesystem user client.

For example:

```
$ bin/hadoop fs  -ls /usr/joe/wordcount/output
```

Most of the commands have similar behaviors like corresponding Unix commands.

Examine the status of your submitted jobs



hadoop Hadoop Map/Reduce Administration

Quick Links

State: RUNNING
Started: Wed Aug 17 09:28:58 CST 2011
Version: 0.20.1+169.113, r6c765a47a929147 0d3d8814c98155115d109d715
Compiled: Sun Sep 12 05:39:27 UTC 2010 by root
Identifier: 201108170928

Cluster Summary (Heap Size is 755.88 MB/1.78 GB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	1	15	60	60	8.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Completed Jobs

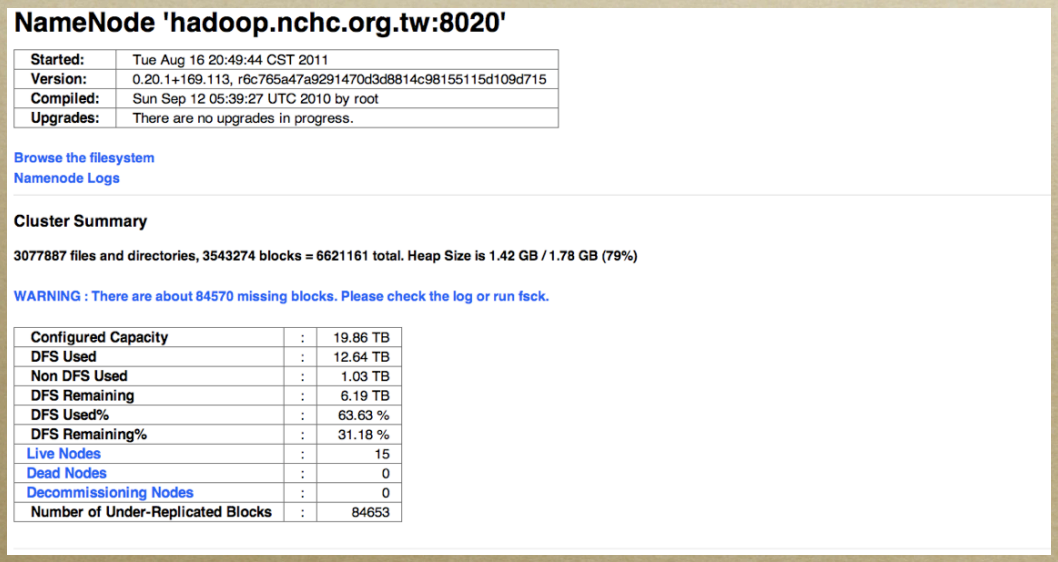
Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201108170928_0001	NORMAL	h770	BNC_2Frame	100.00%	4	4	100.00%	20	20	NA

Failed Jobs

none

Local Logs

Examine the status of HDFS



NameNode 'hadoop.nchc.org.tw:8020'

Started:	Tue Aug 16 20:49:44 CST 2011
Version:	0.20.1+169.113, r6c765a47a9291470d3d8814c98155115d109d715
Compiled:	Sun Sep 12 05:39:27 UTC 2010 by root
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

3077887 files and directories, 3543274 blocks = 6621161 total. Heap Size is 1.42 GB / 1.78 GB (79%)

WARNING : There are about 84570 missing blocks. Please check the log or run fsck.

Configured Capacity	:	19.86 TB
DFS Used	:	12.64 TB
Non DFS Used	:	1.03 TB
DFS Remaining	:	6.19 TB
DFS Used%	:	63.63 %
DFS Remaining%	:	31.18 %
Live Nodes	:	15
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	84653

What if you cannot or do not want to write Hadoop programs in Java?

There is more than the Java API

- Streaming
  - Perl/Python/Ruby/Shell Script.....
- Pipes (C/C++)
- Higher level interface
  - Pig, Hive

- On the research clusters of Yahoo!, most research jobs are not Java based [Owen O'Malley, ApacheCon US 2008]
  - 42% Streaming
  - 28% Pig
  - 28% Java
  - 2% C++

Hadoop Streaming

Hadoop streaming is a utility that comes with the Hadoop distribution.

The utility allows you to create and run MapReduce jobs with any executable or script as the mapper and/or the reducer.

Word Counting with Streaming

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /bin/wc
```

- Using your program as mapper/reducer
- Package files in your job with -file option

```
  -mapper myScript \
  -reducer /bin/wc
  -file myScript
```

Feature Extraction Using Hadoop Streaming

- Many images stored on Hadoop HDFS that wait for feature extraction
- Existing binary programs for feature extraction that take image filename as command parameter

- The steps to integrate the existing program with Hadoop streaming
  - Write a text file listing those image filenames
  - Write a wrapper script to process the text file using Hadoop Streaming
  - In the script
    - Move image from HDFS to local disk
    - Call external program to extract features for each image
    - Move extracted feature back to HDFS

A Ruby Script for Extract Feature

```
#!/usr/bin/env ruby

STDIN.each_line do |line|
   # obtain filename from file list
   filename = line.chop!
   if /\A.*\/(.*?)\z/i.match(filename)
      localfilename = $1
   else
      exit
   end
   # copy file from HDFS to local disk
   system("hadoop dfs -get /user/username/" + filename + " " + localfilename)

   #begin to process local file
   system("chmod a+x ./extract_features_64bit.ln")  # the parallelized existing program
   system("./extract_features_64bit.ln -hesaff -sift -i ./" + localfilename + " -o1 ./" + localfilename +
".hes")
   system("hadoop dfs -put " + localfilename + ".hes" + " " + "/user/username/output/features/.")
   puts "0\t#{localfilename}"
   system("rm " + localfilename)
   system("rm " + localfilename + ".hes")
end
```

- The image list file on HDFS

.....

input/images/1316255637_t.pgm
input/images/1918648013_t.pgm
input/images/2134555347_t.pgm

.....

Hadoop Streaming Usage

- Using the script as mapper/reducer

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
   -input input/images_list \
   -output output/features \
   -mapper extract_local_feature.rb \
   -file extract_local_feature.rb \
   -file extract_features_64bit.ln \
   -jobconf mapred.map.tasks=10
   -jobconf mapred.reduce.tasks=0
```

Problem: Large Number of (Small) Files

- Large number of small files stored on HDFS is inefficient due to the design of HDFS
  - A file is small when it is less than the HDFS block size (128MB by default)
  - Files and blocks are name objects in HDFS and occupy namespace
  - Namespace is limited by physical memory size in HDFS NameNode
- Try more complicated pre-processing
  - Archive many files together and process the archives in script
  - Archive processed results before moving them back to HDFS

Modified Script

```
#!/usr/bin/env ruby
STDIN.each_line do |line|
  ....
  system("mkdir images")
  system("mv " + localfilename + " images/.")
  system("tar xfv images/" + localfilename + " -C images")
  system("chmod a+x ./extract_features_64bit.ln")

  files = Dir.glob("images/*")
  files.each do |image_file|
    #begin to process local file
    system("/usr/bin/convert " + image_file + " " + image_file + ".pgm")
    system("./extract_features_64bit.ln -hesaff -sift -i " + image_file + ".pgm -o1 " + image_file + ".hes")
  end

  system("mkdir features")
  system("mv images/*.hes features/.")
  system("tar cf " + localfilename + " -C features .")
  system("/opt/hadoop/bin/hadoop dfs -put " + localfilename + " " + "/user/username/output/features/.")
  ....
end
```

- The image list file on HDFS

```
.....
input/images/images_1.tar
input/images/images_2.tar
input/images/images_3.tar
.....
```

Simple Experiment

- Extracting SIFT features for 917 images
  - 5 tar files: 269, 355, 144, 80 and 69 images, respectively
- Feature extraction in 4.5 minutes using 5 mapper
  - 203.7778 image/per minute in average

Question?