PARLab Parallel Boot Camp

Cloud Computing with MapReduce and Hadoop

Matei Zaharia
Electrical Engineering and Computer Sciences
University of California, Berkeley
What is Cloud Computing?

• “Cloud” refers to large Internet services like Google, Yahoo, etc that run on 10,000’s of machines

• More recently, “cloud computing” refers to services by these companies that let external customers rent computing cycles on their clusters
  - Amazon EC2: virtual machines at 10¢/hour, billed hourly
  - Amazon S3: storage at 15¢/GB/month

• Attractive features:
  - Scale: up to 100’s of nodes
  - Fine-grained billing: pay only for what you use
  - Ease of use: sign up with credit card, get root access
What is MapReduce?

• Simple data-parallel programming model designed for scalability and fault-tolerance

• Pioneered by Google
  – Processes 20 petabytes of data per day

• Popularized by open-source Hadoop project
  – Used at Yahoo!, Facebook, Amazon, ...
What is MapReduce used for?

• **At Google:**
  - Index construction for Google Search
  - Article clustering for Google News
  - Statistical machine translation

• **At Yahoo!**:
  - “Web map” powering Yahoo! Search
  - Spam detection for Yahoo! Mail

• **At Facebook:**
  - Data mining
  - Ad optimization
  - Spam detection
Example: Facebook Lexicon

www.facebook.com/lexicon
Example: Facebook Lexicon

Search: hola, salut, ciao

Suggestions: vacation | xoxo, xoxoxo | midterm, final | party tonight, hangover

✓ hola ✓ salut ✓ ciao

www.facebook.com/lexicon
What is MapReduce used for?

• In research:
  - Astronomical image analysis (Washington)
  - Bioinformatics (Maryland)
  - Analyzing Wikipedia conflicts (PARC)
  - Natural language processing (CMU)
  - Particle physics (Nebraska)
  - Ocean climate simulation (Washington)
  - <Your application here>
Outline

- MapReduce architecture
- Example applications
- Getting started with Hadoop
- Higher-level languages over Hadoop: Pig and Hive
- Amazon Elastic MapReduce
1. **Scalability to large data volumes:**
   - 1000's of machines, 10,000's of disks

2. **Cost-efficiency:**
   - Commodity machines (cheap, but unreliable)
   - Commodity network
   - Automatic fault-tolerance (fewer administrators)
   - Easy to use (fewer programmers)
Typical Hadoop Cluster

- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- Node specs (Yahoo terasort): 8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)
Typical Hadoop Cluster
1. **Cheap nodes fail, especially if you have many**
   - Mean time between failures for 1 node = 3 years
   - Mean time between failures for 1000 nodes = 1 day
   - **Solution:** Build fault-tolerance into system

2. **Commodity network = low bandwidth**
   - **Solution:** Push computation to the data

3. **Programming distributed systems is hard**
   - **Solution:** Data-parallel programming model: users write “map” & “reduce” functions, system distributes work and handles faults
Hadoop Components

• Distributed file system (HDFS)
  - Single namespace for entire cluster
  - Replicates data 3x for fault-tolerance

• MapReduce framework
  - Executes user jobs specified as “map” and “reduce” functions
  - Manages work distribution & fault-tolerance
Hadoop Distributed File System

- Files split into 128MB blocks
- Blocks replicated across several datanodes (usually 3)
- Single namenode stores metadata (file names, block locations, etc)
- Optimized for large files, sequential reads
- Files are append-only
MapReduce Programming Model

- Data type: key-value records

- Map function:
  \[(K_{\text{in}}, V_{\text{in}}) \rightarrow \text{list}(K_{\text{inter}}, V_{\text{inter}})\]

- Reduce function:
  \[(K_{\text{inter}}, \text{list}(V_{\text{inter}})) \rightarrow \text{list}(K_{\text{out}}, V_{\text{out}})\]
Example: Word Count

```python
def mapper(line):
    for word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```
Word Count Execution

Input

the quick brown fox
the fox ate the mouse
how now brown cow

Map

the, 1
brown, 1
fox, 1

Map

how, 1
now, 1
brown, 1

Map

the, 1
fox, 1
the, 1

Reduce

the, 1
brown, 2
fox, 2
how, 1
now, 1
the, 3

Reduce

ate, 1
cow, 1
mouse, 1
quick, 1

Output
MapReduce Execution Details

- Single *master* controls job execution on multiple *slaves*

- Mappers preferentially placed on same node or same rack as their input block
  - Minimizes network usage

- Mappers save outputs to local disk before serving them to reducers
  - Allows recovery if a reducer crashes
  - Allows having more reducers than nodes
An Optimization: The Combiner

• A combiner is a local aggregation function for repeated keys produced by same map
• Works for associative functions like sum, count, max
• Decreases size of intermediate data
• Example: map-side aggregation for Word Count:

```python
def combiner(key, values):
    output(key, sum(values))
```
Word Count with Combiner

Input: the quick brown fox, the fox ate the mouse, how now brown cow

Map & Combine:
- Map: the, 1; brown, 1; fox, 1
- Map: the, 2; fox, 1
- Map: how, 1; now, 1; brown, 1

Shuffle & Sort:

Reduce:
- Reduce: the, 3
- Reduce: brown, 2; fox, 2; how, 1; now, 1
- Reduce: ate, 1; brown, 1; mouse, 1
- Reduce: quick, 1; cow, 1

Output:
brown, 2; fox, 2; how, 1; now, 1; the, 3; ate, 1; brown, 1; mouse, 1; quick, 1
Fault Tolerance in MapReduce

1. If a task crashes:
   - Retry on another node
     » OK for a map because it has no dependencies
     » OK for reduce because map outputs are on disk
   - If the same task fails repeatedly, fail the job or ignore that input block (user-controlled)

➢ Note: For these fault tolerance features to work, your map and reduce tasks must be side-effect-free
2. If a node crashes:
   - Re-launch its current tasks on other nodes
   - Re-run any maps the node previously ran
     » Necessary because their output files were lost along with the crashed node
3. If a task is going slowly (straggler):
   - Launch second copy of task on another node 
     ("speculative execution")
   - Take the output of whichever copy finishes first, and 
     kill the other

➢ Surprisingly important in large clusters
   - Stragglers occur frequently due to failing hardware, software bugs, misconfiguration, etc
   - Single straggler may noticeably slow down a job
Takeaways

• By providing a data-parallel programming model, MapReduce can control job execution in useful ways:
  - Automatic division of job into tasks
  - Automatic placement of computation near data
  - Automatic load balancing
  - Recovery from failures & stragglers

• User focuses on application, not on complexities of distributed computing
Outline

- MapReduce architecture
- Example applications
  - Getting started with Hadoop
  - Higher-level languages over Hadoop: Pig and Hive
  - Amazon Elastic MapReduce
1. Search

- **Input**: (lineNumber, line) records
- **Output**: lines matching a given pattern

- **Map**: 

  ```python
  if(line matches pattern):
    output(line)
  ```

- **Reduce**: identify function
  - Alternative: no reducer (map-only job)
2. Sort

- **Input**: (key, value) records
- **Output**: same records, sorted by key

- **Map**: identity function
- **Reduce**: identify function

- **Trick**: Pick partitioning function $h$ such that $k_1 < k_2 \Rightarrow h(k_1) < h(k_2)$
3. Inverted Index

- **Input:** (filename, text) records
- **Output:** list of files containing each word

- **Map:**
  
  ```python
  foreach word in text.split():
    output(word, filename)
  ```

- **Combine:** uniquify filenames for each word

- **Reduce:**
  
  ```python
  def reduce(word, filenames):
    output(word, sort(filenames))
  ```
Inverted Index Example

- **hamlet.txt**
  - to be or not to be
    - to, hamlet.txt
    - be, hamlet.txt
    - or, hamlet.txt
    - not, hamlet.txt

- **12th.txt**
  - be not afraid of greatness
    - be, 12th.txt
    - not, 12th.txt
    - afraid, 12th.txt
    - of, 12th.txt
    - greatness, 12th.txt

- **afraid, (12th.txt)**
  - be, (12th.txt, hamlet.txt)
  - greatness, (12th.txt)
  - not, (12th.txt, hamlet.txt)
  - of, (12th.txt)
  - or, (hamlet.txt)
  - to, (hamlet.txt)
4. Most Popular Words

- **Input:** (filename, text) records
- **Output:** top 100 words occurring in the most files

- **Two-stage solution:**
  - **Job 1:**
    » Create inverted index, giving (word, list(file)) records
  - **Job 2:**
    » Map each (word, list(file)) to (count, word)
    » Sort these records by count as in sort job

- **Optimizations:**
  - Map to (word, 1) instead of (word, file) in Job 1
  - Count files in job 1’s reducer rather than job 2’s mapper
  - Estimate count distribution in advance and drop rare words
5. Numerical Integration

- **Input:** (start, end) records for sub-ranges to integrate
  - Easy using custom InputFormat
- **Output:** integral of \( f(x) \, dx \) over entire range

- **Map:**
  ```python
def map(start, end):
    sum = 0
    for x = start; x < end; x += step):
      sum += f(x) * step
    output("", sum)
  ```

- **Reduce:**
  ```python
def reduce(key, values):
    output(key, sum(values))
  ```
• MapReduce architecture
• Example applications
• Getting started with Hadoop
  • Higher-level languages over Hadoop: Pig and Hive
  • Amazon Elastic MapReduce
• Download from hadoop.apache.org
• To install locally, unzip and set JAVA_HOME
• Details: hadoop.apache.org/core/docs/current/quickstart.html

• Three ways to write jobs:
  - Java API
  - Hadoop Streaming (for Python, Perl, etc)
  - Pipes API (C++)
public class MapClass extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

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

    public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> out,
            Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            out.collect(new Text(itr.nextToken()), ONE);
        }
    }
}
public class ReduceClass extends MapReduceBase
implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
                        OutputCollector<Text, IntWritable> out,
                        Reporter reporter) throws IOException {

        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }

        out.collect(key, new IntWritable(sum));
    }
}
public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setMapperClass(MapClass.class);
    conf.setCombinerClass(ReduceClass.class);
    conf.setReducerClass(ReduceClass.class);
    FileInputFormat.setInputPaths(conf, args[0]);
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));
    conf.setOutputKeyClass(Text.class); // out keys are words (strings)
    conf.setOutputValueClass(IntWritable.class); // values are counts
    JobClient.runJob(conf);
}
Mapper.py:

```python
import sys

for line in sys.stdin:
    for word in line.split():
        print(word.lower() + "\t" + 1)
```

Reducer.py:

```python
import sys

counts = {}

for line in sys.stdin:
    word, count = line.split("\t")
    dict[word] = dict.get(word, 0) + int(count)

for word, count in counts:
    print(word.lower() + "\t" + 1)
```
Outline

• MapReduce architecture
• Example applications
• Getting started with Hadoop
  • Higher-level languages over Hadoop: Pig and Hive
• Amazon Elastic MapReduce
Motivation

- Many parallel algorithms can be expressed by a series of MapReduce jobs

- But MapReduce is fairly low-level: must think about keys, values, partitioning, etc

- Can we capture common “job building blocks”?
Pig

- Started at Yahoo! Research
- Runs about 30% of Yahoo!'s jobs
- Features:
  - Expresses sequences of MapReduce jobs
  - Data model: nested “bags” of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc)
  - Easy to plug in Java functions
  - Pig Pen development environment for Eclipse
An Example Problem

Suppose you have user data in one file, page view data in another, and you need to find the top 5 most visited pages by users aged 18 - 25.

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt
The code snippet demonstrates a MapReduce example from Apache Pig. It includes java imports, class definitions, and methods for processing data. The code snippet showcases the use of MapReduce in a Pig script, with examples of how to define Map and Reduce functions, iterate over data, and handle various operations such as string manipulation and conditional logic.
Users = load 'users' as (name, age);
Filtered = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
        count(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;

store Top5 into 'top5sites';

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt
Ease of Translation

Notice how naturally the components of the job translate into Pig Latin.

- Users = load ...  
- Filtered = filter ... 
- Pages = load ... 
- Joined = join ... 
- Grouped = group ... 
- Summed = ... count()... 
- Sorted = order ... 
- Top5 = limit ...

Example from http://wiki.apache.org/pig-data/attachments/Pig Talks Papers/attachments/ApacheConEurope09.ppt
Ease of Translation

Notice how naturally the components of the job translate into Pig Latin.

- Load Users
- Load Pages
- Filter by age
- Join on name
- Group on url
- Count clicks
- Order by clicks
- Take top 5

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt
Hive

- Developed at Facebook
- Used for majority of Facebook jobs
- "Relational database" built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HQL)
  - Can call Hadoop Streaming scripts from HQL
  - Supports table partitioning, clustering, complex data types, some optimizations
Sample Hive Queries

- Find top 5 pages visited by users aged 18-25:
  
  SELECT p.url, COUNT(1) as clicks  
  FROM users u JOIN page_views p ON (u.name = p.user)  
  WHERE u.age >= 18 AND u.age <= 25  
  GROUP BY p.url  
  ORDER BY clicks  
  LIMIT 5;

- Filter page views through Python script:
  
  SELECT TRANSFORM(p.user, p.date)  
  USING 'map_script.py'  
  AS dt, uid CLUSTER BY dt  
  FROM page_views p;
Outline

• MapReduce architecture
• Example applications
• Getting started with Hadoop
• Higher-level languages over Hadoop: Pig and Hive
• Amazon Elastic MapReduce
Amazon Elastic MapReduce

- Provides a web-based interface and command-line tools for running Hadoop jobs on Amazon EC2
- Data stored in Amazon S3
- Monitors job and shuts down machines after use
- Small extra charge on top of EC2 pricing

- If you want more control over how you Hadoop runs, you can launch a Hadoop cluster on EC2 manually using the scripts in src/contrib/ec2
Elastic MapReduce Workflow

Create a New Job Flow

Creating a job flow to process your data using Amazon Elastic MapReduce is simple and quick. Let's begin by giving your job flow a name and selecting its type. If you don't already have an application you'd like to run on Amazon Elastic MapReduce, samples are available to help you get started.

Job Flow Name*: My Job Flow

Type*: Streaming

A Streaming job flow allows you to write single-step mapper and reducer functions in a language other than Java.

Custom Jar (advanced)

A custom Jar on the other hand gives you more complete control over the function of Hadoop but must be a compiled Java program. Amazon Elastic MapReduce supports custom Jars developed for Hadoop 0.18.3.

Pig Program

Pig is a SQL-like language built on top of Hadoop. This option allows you to define a job flow that runs a Pig script, or set up a job flow that can be used interactively via SSH to run Pig commands.

Sample Applications

Select a sample application and click Continue. Subsequent forms will be filled with the necessary data to create a sample Job Flow.

Word Count (Streaming)  Word count is a Python application that counts occurrences of each word in provided documents. Learn more and view license

Continue
Elastic MapReduce Workflow

Create a New Job Flow

Define Job Flow  Specify Parameters  Configure EC2 Instances  Review

Specify Mapper and Reducer functions to run within the Job Flow. The mapper and reducers may be either (i) class names referring to a mapper or reducer class in Hadoop or (ii) locations in Amazon S3. (Click Here for a list of available tools to help you upload and download files from Amazon S3.) The format for specifying a location in Amazon S3 is bucket_name/path_name. The location should point to an executable program, for example a python program. Extra arguments are passed to the Hadoop streaming program and can specify things such as additional files to be loaded into the distributed cache.

Input Location*: elasticmapreduce/samples/wordcount/input
The URL of the Amazon S3 Bucket that contains the input files.

Output Location*: <yourbucket>/wordcount/output/2009-08-19
The URL of the Amazon S3 Bucket to store output files. Should be unique.

Mapper*: elasticmapreduce/samples/wordcount/wordSplitter.py
The mapper Amazon s3 location or streaming command to execute.

Reducer*: aggregatereg
The reducer Amazon s3 location or streaming command to execute.

Extra Args:

* Required field
Elastic MapReduce Workflow

Create a New Job Flow

Enter the number and type of EC2 instances you'd like to run your job flow on.

**Number of Instances**: 4

The number of EC2 instances to run in your Hadoop cluster. If you wish to run more than 20 instances, please complete the limit request form.

**Type of Instance**: Small (m1.small)

The type of EC2 instances to run in your Hadoop cluster (learn more about instance types).

Show advanced options

* Required field
Elastic MapReduce Workflow

### Your Elastic MapReduce Job Flows

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Creation Date</th>
<th>Elapsed Time</th>
<th>Normalized Instance Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>My Job Flow</td>
<td>STARTING</td>
<td>2009-08-19 14:50 PDT</td>
<td>0 hours 0 minutes</td>
<td>0</td>
</tr>
</tbody>
</table>

1 Job Flow selected

- **Id:** j-463L0YQ7ZH1
- **Name:** My Job Flow
- **State:** STARTING
- **Last State Change Reason:** Starting instances
- **Availability Zone:** us-east-1b
- **Instance Count:** 4

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All rights reserved.
Conclusions

• MapReduce programming model hides the complexity of work distribution and fault tolerance

• Principal design philosophies:
  - *Make it scalable*, so you can throw hardware at problems
  - *Make it cheap*, lowering hardware, programming and admin costs

• MapReduce is not suitable for all problems, but when it works, it may save you quite a bit of time

• Cloud computing makes it straightforward to start using Hadoop (or other parallel software) at scale
Resources

• Hadoop: http://hadoop.apache.org/core/
• Pig: http://hadoop.apache.org/pig
• Hive: http://hadoop.apache.org/hive
• Video tutorials: http://www.cloudera.com/hadoop-training

• Amazon Web Services: http://aws.amazon.com/
• Amazon Elastic MapReduce guide: http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/

• My email: matei@berkeley.edu