CS242 PROJECT

Presented by Moloud Shahbazi
Spring 2015
AGENDA

- Project Overview
- Data Collection
- Indexing
- Big Data Processing
PROJECT- PART I

1.1 Data Collection:
- 5G < data size < 10G
- Deliverables:
  - Document
  - partner's contribution
  - Design
  - Obstacles
  - Limitations
- Implementation
  - Runnable + usage instruction
  - Code

1.2 Indexing:
- Deliverables:
  - Document:
    - Index fields
    - Text analyzer choices
    - Runtime of indexing
    - Limitations
  - Implementation
    - Runnable + usage instruction
    - code
PROJECT- PART2

• 2.1 Hadoop:
  • Description of Hadoop jobs
  • Indexes built by Hadoop
  • Run time of Index construction
  • Ranking with Hadoop
  • Obstacles
  • Limitations
  • Code + usage instruction

• 2.2 Web Interface:
  • Minimal is enough
  • Option for Lucene/Hadoop
  • Integration with Hadoop Index
  • Lucene QueryParser integration
  • Limitations
  • Code + usage instructions
  • Demo Screenshots
DATA COLLECTION

RESOURCES

• Shiwen’s slides for CS172 http://www.cs.ucr.edu/~schen064/cs172/
  Covers: robots.txt, jsoup, threading, XML, JSP and Servlet examples

• Twitter Streaming API: https://dev.twitter.com/docs/streaming-apis
  Horsebird Client: https://github.com/twitter/hbc

• Twitter4j: http://twitter4j.org/en/index.html
  Examples: http://twitter4j.org/en/code-examples.html#streami
APACHE LUCENE

http://lucene.apache.org/core/
http://lucene.apache.org/pylucene/
http://www.lucenetutorial.com/

Version 5.0.0: https://lucene.apache.org/core/5_0_0/demo/src-html/org/apache/lucene/demo/IndexFiles.html

Note, only use the Lucene Core or PyLucene, Solr is not acceptable
LUCENE

• **Definition**: Full Text Index java library

• **Concepts**:
  - **Documents**
    - ex. web page, tweet, image.
  - **Filed**

• **Functionality**:
  - **Indexing**
    - Create Document from data
    - Call IndexWriter
  - **Searching**
    - Create Query
    - Call IndexSearcher
CONSTRUCTING LUCENE INDEX

http://lucene.apache.org/core/5_0_0/core/org/apache/lucene/index/IndexWriter.html

```java
Directory dir = FSDirectory.open(Paths.get(indexPath));
Analyzer analyzer = new StandardAnalyzer();
IndexWriterConfig iwc = new IndexWriterConfig(analyzer);

if (create)
   iwc setOpenMode(OpenMode.CREATE);
else
   iwc setOpenMode(OpenMode.CREATE_OR_APPEND);

IndexWriter writer = new IndexWriter(dir, iwc);
indexDocs(writer, docDir);

writer.close();
```
/** Indexes a single document */
static void indexDoc(IndexWriter writer, Path file, long lastModified) throws IOException {
  try (InputStream stream = Files.newInputStream(file)) {

    Document doc = new Document();

    Field pathField = new StringField("path", file.toString(), Field.Store.YES);
    doc.add(pathField);

    doc.add(new LongField("modified", lastModified, Field.Store.NO));
    doc.add(new TextField("contents", new BufferedReader(new InputStreamReader(stream, StandardCharsets.UTF_8))));

    if (writer.getConfig().getOpenMode() == OpenMode.CREATE) {
      writer.addDocument(doc);
    } else {
      writer.updateDocument(new Term("path", file.toString()), doc);
    }
  }
}
SEARCH LUCENE INDEX

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/IndexSearcher.html

```java
String index = "indexDirectory";
String field = "contents";

IndexReader reader = DirectoryReader.open(FSDirectory.open(Paths.get(index)));
IndexSearcher searcher = new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();

BufferedReader in = null;
if (queries != null) {
    in = Files.newBufferedReader(Paths.get(queries), StandardCharsets.UTF_8);
} else {
    in = new BufferedReader(new InputStreamReader(System.in, StandardCharsets.UTF_8));
}

QueryParser parser = new QueryParser(field, analyzer);
Query query = parser.parse("search term");

TopDocs results = searcher.search(query, 100);
ScoreDoc[] hits = results.scoreDocs;

System.out.println(results.totalHits + " total matching documents");
for (int i = 0; i < hits.length; i++) {
    Document doc = searcher.doc(hits[i].doc);
    String path = doc.get("path");
    String title = doc.get("title");
}
doPagingSearch(in, searcher, query, hitsPerPage, raw, queries == null && queryString == null);
reader.close();
```
QUERY SEMANTICS

http://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/Query.html

- Keyword matching, Wildcard matching, Proximity matching, Range search, Boosts
WHAT IS IT?
https://hadoop.apache.org/#What+Is+Apache+Hadoop%3F

Apartment Hadoop Ecosystem

- **Ambari**
  - Provisioning, Managing and Monitoring Hadoop Clusters

- **Apache Hadoop**
  - Distributed Processing Framework

- **HDFS**
  - Hadoop Distributed File System

- **YARN**
  - Map Reduce v2

- **Oozie**
  - Workflow

- **Pig**
  - Scripting

- **Mahout**
  - Machine Learning

- **R Connectors**
  - Statistics

- **Hive**
  - SQL Query

- **Sapce HBase**
  - Columnar Store
BASE COMPONENTS

• Fault tolerant **storage** for big data
  • HDFS
  • Replication on multiple machines
  • Parallel I/O
  • I/O balancing

• **Distributed** system for **processing** large quantities of data
  • Parallel processing - Map Reduce
  • Data-Aware job scheduling
  • Failure handling (Tracking/Restarting)
HOW?

- Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.
HDFS

STORE

Large amount of input data...

Data loading step

Node 1
Slice of input

Node 2
Slice of input

Node 3
Slice of input

TRACK

NameNode:
Stores metadata only

METADATA:
/user/aaron/foo → 1, 2, 4
/user/aaron/bar → 3, 5

DataNodes: Store blocks from files

2 4 5 2
1 5 3
4 1 3
HDFS COMMUNICATIONS

HDFS clients talk to the NameNode for metadata-related activities, and to DataNodes to read and write files.

The HDFS NameNode keeps in memory the metadata about the filesystem, such as which DataNodes manage the blocks for each file.

DataNodes communicate with each other for pipeline file reads and writes.

Files are made up of blocks, and each file can be replicated multiple times, meaning there are many identical copies of each block for the file (by default 3).
HDFS COMMAND LINE


hadoop fs [-fs <local | file system URI>] [-conf <configuration file>]
[-D <property=value>] [-ls <path>] [-lsr <path>] [-du <path>]
[-dus <path>] [-mv <src> <dst>] [-cp <src> <dst>] [-rm <src>]
-rmr <src>] [-put <localsrc> ... <dst>] [-copyFromLocal <localsrc> ... <dst>]
[-moveFromLocal <localsrc> ... <dst>] [-get [-ignoreCrc] [-crc] <src> <localdst>
[-getmerge <src> <localdst> [addnl]] [-cat <src>]
[-copyToLocal [-ignoreCrc] [-crc] <src> <localdst>] [-moveToLocal <src> <localdst>]
[-touchz <path>] [-test [-ezd] <path>] [-stat [format] <path>]
[-tail [-f] <path>] [-text <path>]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][:[GROUP]] PATH...]
[-chgrp [-R] GROUP PATH...]
[-count[-q] <path>]
[-help [cmd]]
Working With HDFS

- **Start Here**
  - hadoop dfs -ls /users/eruiz009

- **Copy to HDFS**
  - hadoop dfs -copyFromLocal tweets.txt /users/eruiz009

- **Check File**
  - hadoop dfs -ls /users/eruiz009/tweets.txt

- **Tail**
  - hadoop dfs -tail /users/eruiz009/tweets.txt

- **Move to Local**
  - hadoop dfs -copyToLocal /users/eruiz009/tweets.txt

- **Remove**
  - hadoop dfs -rmr /users/eruiz009/tweets.txt
HDFS JAVA API

// Get default file system instance
fs = Filesystem.get(new Configuration());

// Or Get file system instance from URI
fs = Filesystem.get(URI.create(uri), new Configuration());

// Create, open, list, ...
OutputStream out = fs.create(path, ...);

InputStream in = fs.open(path, ...);

boolean isDone = fs.delete(path, recursive);

FileStatus[] fstat = fs.listStatus(path);
HADOOP MAP-REDUCE

The JobTracker coordinates activities across the slave TaskTracker processes. It accepts MapReduce job requests from clients and schedules map and reduce tasks on TaskTrackers to perform the work.

MapReduce clients talk to the JobTracker to launch and manage jobs.

The TaskTracker is a daemon process that spawns child processes to perform the actual map or reduce work. Map tasks typically read their input from HDFS, and write their output to the local disk. Reduce tasks read the map outputs over the network and write their outputs back to HDFS.
MAPREDUCE EXAMPLE

- **Map Phase**: Words are split into smaller pieces (e.g., "Mary had a little lamb", "its fleece was white as snow", "and everywhere that Mary went", "the lamb was sure to go"). Each word or phrase is then processed by a function (e.g., a count function) to produce a key-value pair (e.g., "Mary: 1", "lamb: 1", "white: 1").

- **Reduce Phase**: Key-value pairs are grouped by key, and the values are aggregated (e.g., summing the counts of the word "lamb"). The output is a list of key-value pairs with the aggregated values (e.g., "lamb: 3", "white: 2").
CREATING MAP-REDUCE JOB

• Input and Output types of a MapReduce job:

\[(\text{input}) \quad \langle k_1, v_1 \rangle \rightarrow \text{map} \rightarrow \langle k_2, v_2 \rangle \rightarrow \text{combine} \rightarrow \langle k_2, v_2 \rangle \rightarrow \text{reduce} \rightarrow \langle k_3, v_3 \rangle \quad (\text{output})\]

• The right level of parallelism for maps seems to be around 10-100 maps per-node
• each map takes a block
• if file_size = 5G & block size = 128M ——> #blocks > 5G/128M=40
• **Recommended file size on HDFS** => slightly less than block size.
The overall MapReduce word count process

Input: Deer Bear River Car Car River Deer Car Bear

Splitting:
- Deer Bear River
- Car Car River
- Deer Car Bear

Mapping:
- Deer, 1
  - Bear, 1
  - River, 1
- Car, 1
  - Car, 1
  - River, 1
- Deer, 1
  - Deer, 1
  - Bear, 1

Shuffling:
- Bear, 1
- Bear, 1
- Car, 1
- Car, 1
- Car, 1
- Deer, 1
- Deer, 1
- River, 1

Reducing:
- Bear, 2
- Car, 3
- Deer, 2
- River, 2

Final result:
- Bear, 2
- Car, 3
- Deer, 2
- River, 2
OUR HADOOP CLUSTER

Server 1:
- Task Tracker
- NameNode
- Secondary NameNode

Servers 2 to 6:
- JobTracker
- Data Node
import org.apache.hadoop.*

public class WordCount {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
    }

    public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    }

    public static void main(String[] args) throws Exception {
    }
}
CUSTOMIZING MAP-REDUCE JOB

• Data types for key and values
• Input format
• Output format
• Partitioning of mapper output
• Combiners process mapper output in memory
public static class Map extends MapReduceBase implements 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, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
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);
}
local data storage (!using HDFS):

```
~ $ Java -cp $CLASSPATH WordCount.java tweets.txt tweets_counts
```

Using HDFS:

```
~ $ hadoop jar WordCount.jar /users/mshah008/tweets.txt /users/mshah008/tweets_counts
```
DEBUGGING A MAP-REDUCE JOB

• Using log files created by Hadoop
  • /var/log/hadoop/job_*.xml -> job configuration XML logs
  • /var/log/hadoop/userlogs/at...%
    • /stderr -> standard error logs
    • /stdout -> standard out logs
    • /syslog -> log4j logs

• referring to stderr for uncaught exceptions, pad file path, et

• Note that exact path may vary after cluster setup
MAP-REDUCE FRIENDLY PATTERNS

• Counting, Summing, Sorting
  • Log analysis, Data querying, ETL

• Collating
  • Inverted Index, Facebook friends in common, ETL

• Filtering, Parsing and Validation
  • Log analysis, ETL, Data validation, Data querying

• Distributed Task Execution
  • Numerical Analysis, Performance Testing

• Not so basic:
  • Graph propagation, Distinct values. Cross validation
ANTI-MAP-REDUCE PATTERNS

- Too much caching
  - Problem: out of memory error
  - Solution: chain jobs, e.g., M1->R1->M2-> R2->...
- Large Input Records
  - Problem: record cannot fit into memory
  - Solution: partition the record / catch the error
- Too many input files
  - Problem: thousands of files strain the NameNode
  - Solution: combine files into one large file
- Overwhelming external data sources
  - Possible to kill an external SQL server using a large cluster
MAP-REDUCE TIPS

• Work with a small subset of data for development!

• Custom FileWriters must write to the job’s PWD

• Malformed Input / Parsing Errors

• Be particular about Hadoop’s version and configuration when porting code across clusters

• Take care with dynamic memory allocation
Questions?