### FAQs

- Discussion board is ready in Canvas
  - Are you looking for teammate(s)? Post your message after the class!
  - Your name
  - Strengths
  - Rough idea about your topic

- Programming Assignment 1
  - Due: Feb. 6
  - Creating Uni-gram, and Bi-gram profile for a Corpus

- Help session: Jan. 21 Friday 11:00 AM ~ noon CSB130
  - The link to the video clip will be posted on the class web
  - Please try to attend and ask questions!
  - This is not mandatory

### What is MapReduce?

This material is developed based on,


- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013
**MapReduce (1/2)**

- MapReduce is inspired by the concepts of map and reduce in Lisp.
- Developed within Google as a mechanism for processing large amounts of raw data.
  - Crawled documents or web request logs
  - Distributes these data across thousands of machines
  - Same computations are performed on each CPU with different dataset

**MapReduce (2/2)**

- MapReduce provides an abstraction that allows engineers to perform simple computations while hiding the details of parallelization, data distribution, load balancing and fault tolerance

**Mapper**

- Mapper maps input key/value pairs to a set of intermediate key/value pairs
  - Maps are the individual tasks that transform input records into intermediate records
  - The transformed intermediate records do not need to be of the same type as the input records
  - A given input pair may map to zero or many output pairs
  - The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job

**Reducer**

- Reducer reduces a set of intermediate values which share a key to a smaller set of values
  - Reducer has 3 primary phases
    - Shuffle, sort and reduce
    - Shuffle
      - Input to the reducer is the sorted output of the mappers
      - The framework fetches the relevant partition of the output of all the mappers via HTTP
    - Sort
      - The framework groups input to the reducer by keys

**Example 1: WordCount**

- For text files stored under `usr/joe/wordcount/input`, count the number of occurrences of each word
- How do files and directory look?

```bash
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/
/user/joe/wordcount/input/file01
/user/joe/wordcount/input/file02
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World, Bye World!
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop, Goodbye to hadoop.
```
Example 1: WordCount [2/5]

- Run the MapReduce application

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

```shell
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
```

Bye 1
Goodbye 1
Hello 1
World, 1
World, 1
hadoop 1
to 1

Example 1: WordCount [3/5]

Mappers
1. Read a line
2. Tokenize the string
3. Pass the `<key, value>` output to the reducer

Reducers
1. Collect `<key, value>` pairs sharing same key
2. Aggregate total number of occurrences

What do you have to pass from the Mappers?

Example 1: WordCount [4/5]

```java
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 {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

Example 1: WordCount [5/5]

```java
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

NCDC data example

- A national climate data center record

```text
0057
332130 # USAF weather station identifier
99999 # WMO weather station identifier
19500101 # observation date
0300 # observation time 4
+ 51317 # latitude (degrees x 1000)
+ 028783 # longitude (degrees x 1000)
FM-12
+ 0171 # elevation (meters)
99999
V020
320 # wind direction (degrees)
1 # quality code
```
The first entries for 1990

% ls raw/ 1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz

Analyzing the data with Unix Tools (1/2)

• A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $ year .gz`
  gunzip -c $ year |
  awk '{ temp = substr( $0, 88, 5) + 0; q = substr( $0, 93, 1); if (temp != 9999 && q ~ /[01459]/ && temp > max) max = temp } END { print max }'
  done
```

Results?

• The complete run for the century took 42 minutes
• To speed up the processing
  • We need to run parts of the program in parallel
  • Process different years in different processes
  • What will be the problems?

Challenges

• Dividing the work into equal-size pieces
• Data size per year?
• Combining the results from independent processes
  • Combining results and sorting by year?
• You are still limited by the processing capacity of a single machine (the worst one)!

Map and Reduce

• MapReduce works by breaking the processing into two phases
  • The map phase
  • The reduce phase
• Each phase has key-value pairs as input and output
• Programmers should specify
  • Types of input/output key-values
  • The map function
  • The reduce function
Visualizing the way the MapReduce works (1/3)

Sample lines of input data

00670119099999150051507004... 9999999+00001 +99999999999... 00430119099999150051512004... 99999999+00221 +99999999999... 00430119099999150051518004... 99999999999-00111 +99999999999... 0043012650999991949032412004... 0500001N9 + 01111 +99999999999... 0043012650999991949032418004... 0500001N9 + 00781 +99999999999...

These lines are presented to the map function as the key-value pairs

(0, 00670119099999150051507004...9999999N9 + 00001+99999999999...) (106, 00430119099999150051512004...99999999999+00221+99999999999...) (212, 00430119099999150051518004...99999999999-00111+99999999999...) These keys are the line offsets within the file

Visualizing the way the MapReduce works (2/3)

The map function extracts the year and the air temperature and emit them as its output

(1950, 0) (1950, 22) (1950, -11)

This output key-value pairs will be sorted and grouped by key. Our reduce function will see the following input:

[(1949, [111, 78]) (1950, [0, 22, -11])]

Visualizing the way the MapReduce works (3/3)

Reduce function iterates through the list and pick up the maximum reading

(1949, 111) (1950, 22)

This is the final output

MapReduce Example 3

Comparison with other systems

- MPI vs. MapReduce
  - MapReduce tries to collocate the data with the compute node
  - Data access is fast
    - Data is local!
- Volunteer computing vs. MapReduce
  - SETI@home
    - Using donated CPU time

Dataset

- Link information in \( n \) log files
  - Source
    - 220.4.20
    - 160.33.1.3
  - Destination
    - 160.33.1.3
    - 160.33.1.3
  - There are two attributes
    - From and To
    - IP1, IP2: There was a link from IP1 to IP2
Relational-Algebra style queries [1/5]
- Large relation can be stored as a file in a DFS (Distributed File System)
  - We will discuss DFS next class
- The data queries (e.g. SQL style) can be processed without storing dataset in a SQL database using MapReduce
  - Selection
    - Apply a condition \( C \) to each tuple in the relation and provide as output only those tuples that satisfy \( C \)

```
220.4.5.20    160.33.1.3
220.4.5.20    100.33.1.5
160.33.1.3    79.45.66.9
100.33.1.5    79.45.66.9
100.33.1.5    160.33.1.3
```
- Union/Intersection and Difference
- Set operations

Relational-Algebra style queries [2/5]
- Projection
  - For some subset \( S \) of the attributes of the relation, produce from each tuple only the components for the attributes in \( S \).
- Union/Intersection and Difference
  - Set operations

```
220.4.5.20    160.33.1.3
220.4.5.20    100.33.1.5
160.33.1.3    79.45.66.9
100.33.1.5    79.45.66.9
100.33.1.5    160.33.1.3
```

Relational-Algebra style queries [3/5]
- Union/Intersection and Difference
- Set operations

```
Data File 1
121.42.15.2    160.33.1.3
220.4.5.21    100.33.1.5
160.33.1.3    79.45.66.9
100.33.1.5    79.45.66.9
```
```
Data File 2
220.4.5.20    160.33.1.3
220.4.5.20    100.33.1.5
160.33.1.3    79.45.66.9
100.33.1.5    79.45.66.9
```
- Union
  - DataFile 1
  - DataFile 2

Relational-Algebra style queries [4/5]
- Natural Join
  - Given two relations, compare each pair of tuples, one from each relation
  - If the tuples agree on all the attributes that are common to the two schemas, then produce a tuple that has components for each of the attributes in either schema and agrees with the two tuples on each attribute

```
Data File 1
Feb.02.2016-12:00:01 121.42.15.2 160.33.1.3
Feb.02.2016-12:00:19 220.4.5.21 100.33.1.5
Feb.02.2016-12:01:22 160.33.1.3 79.45.66.9
Feb.02.2016-12:02:11 100.33.1.5 79.45.66.9
```
```
Data File 2
Feb.02.2016-12:00:01 121.42.15.2 160.33.1.3
Feb.02.2016-12:00:19 220.4.5.21 100.33.1.5
Feb.02.2016-12:01:22 160.33.1.3 79.45.66.9
Feb.02.2016-12:02:11 100.33.1.5 79.45.66.9
```

Relational-Algebra style queries [5/5]
- Grouping and Aggregation
  - Grouping
    - Given a relation \( R \), partition its tuples according to their values in one set of attributes \( G \)
  - Aggregation
    - Then for each group, aggregate the values
    - SUM, COUNT, AVG, MIN and MAX

Computing Selections by MapReduce
- For each tuple, check the condition and produce the key-value pair \((k,v)\)
- For the shuffled pairs \((k,v)\) returns \(\{v\}\) as the result of eliminating duplications
Computing Selections by MapReduce -continued

- The Map function
  - For each tuple \( t \) in \( R \), test if it satisfies \( C \). If so, produce the key-value pair \((t,t)\)
  - Both of the key and values are \( t \)

- The Reduce function
  - It simply passes each key-value pair to the output

Computing Projections by MapReduce

- The Map function
  - For each tuple \( t \) in \( R \), construct a tuple \( t' \) by eliminating from \( t \) those components whose attributes are not in \( S \)
  - Output the key-value pair \((t', t)\)

- The Reduce function
  - For each key \( t' \) produced by any of the Map tasks, there will be one or more key-value pairs \((t', t')\)
  - Eliminate duplicates
    - \((t', t', t', t', \ldots, t') \rightarrow (t', t')\)

Computing Union by MapReduce

- The Map function
  - Turn each input tuple \( t \) into a key-value pair \((t, t)\)

- The Reduce function
  - Associated with each key \( t \) there will be either one or two values.
### Computing Intersection by MapReduce

- **The Map function**
  - Turn each tuple `t` into a key-value pair `(t, t)`

- **The Reduce function**
  - If key `t` has another value list `[t, t]`, then produce `(t, t)`
  - Otherwise nothing

### Computing aggregation

- Get the total count of connections per source IP address
- Use your worksheet
- Design your map and reduce