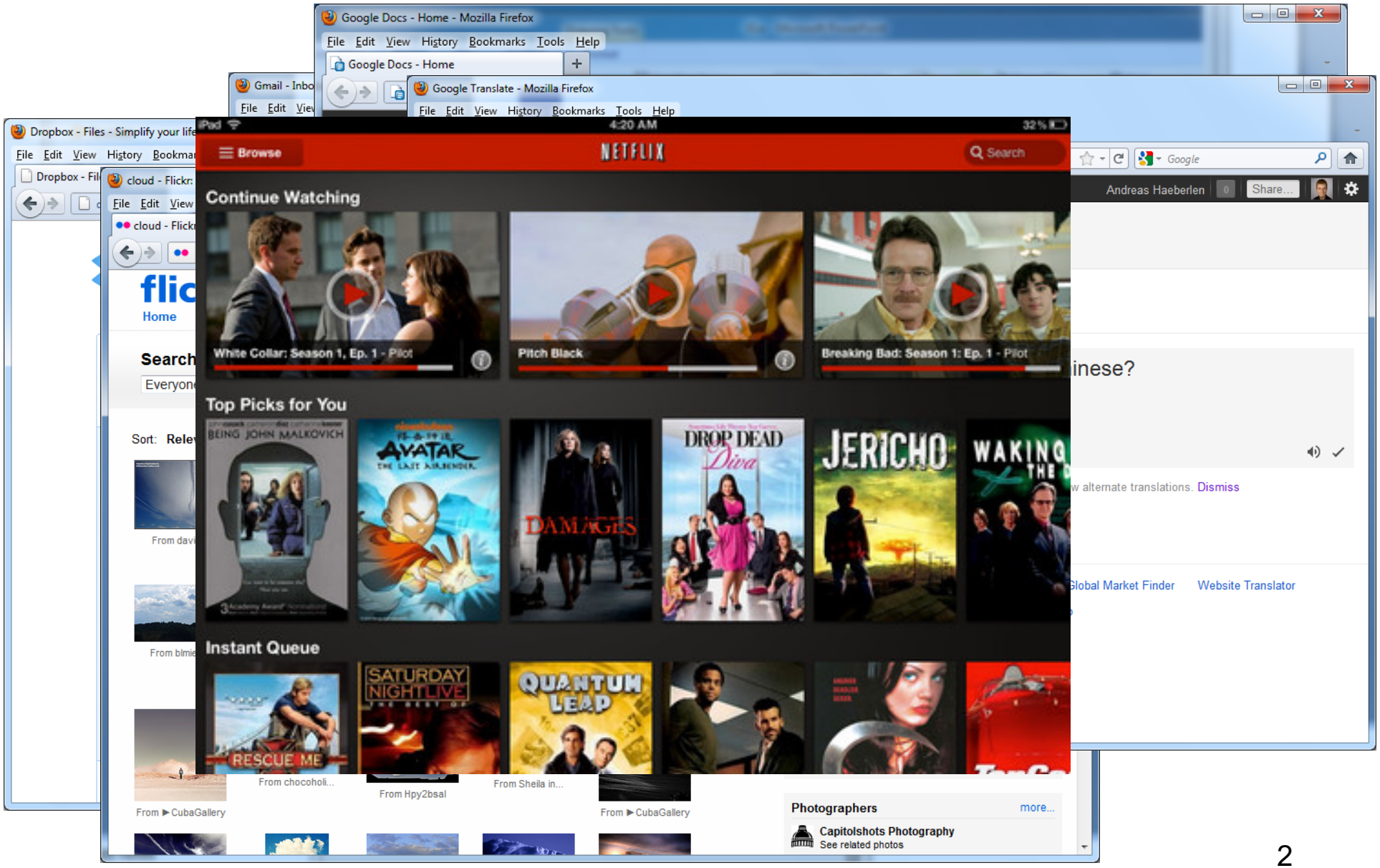
CSC 3331
Analysis of Algorithms

Instructor: Debzani Deb

Parallel Programming & MapReduce

Have you used these before?



What is so special about them?

- The key challenge is scale!
  - Lots of data, lots of users everywhere on the planet
  - Hosted on massive shared infrastructure (data centers – think computers the size of a football field!)
- Scale brings new challenges
  - Many algorithms do not work at these scales
  - Need special solutions for security, performance, ...
  - "Big Data": Data analysis at scale

How many users and objects?

- Flickr has >6 billion photos
- Facebook has 1.7 billion active users
- Google is serving >1.2 billion queries/day on more than 27 billion items
- >2 billion videos/day watched on YouTube

How much data?

- Modern applications use massive data:
  - Rendering 'Avatar' movie required >1 petabyte of storage
  - eBay has >6.5 petabytes of user data
  - CERN's LHC will produce about 15 petabytes of data per year
  - In 2008, Google processed 20 petabytes per day
  - German Climate computing center dimensioned for 60 petabytes of climate data
  - Google now designing for 1 exabyte of storage
  - NSA Utah Data Center is said to have 5 zettabyte (!)
- How much is a zettabyte?
  - 1,000,000,000,000,000,000,000 bytes
  - A stack of 1TB hard disks that is 25,400 km high

How much computation?

- No single computer can process that much data
  - Need many computers!
  - Need parallel processing!
- How many computers do modern services need?
  - Facebook is thought to have more than 60,000 servers
  - 1&1 Internet has over 70,000 servers
  - Akamai has 95,000 servers in 71 countries
  - Intel has ~100,000 servers in 97 data centers
  - Microsoft reportedly had at least 200,000 servers in 2008
  - Google is thought to have more than 1 million servers, is planning for 10 million (according to Jeff Dean)

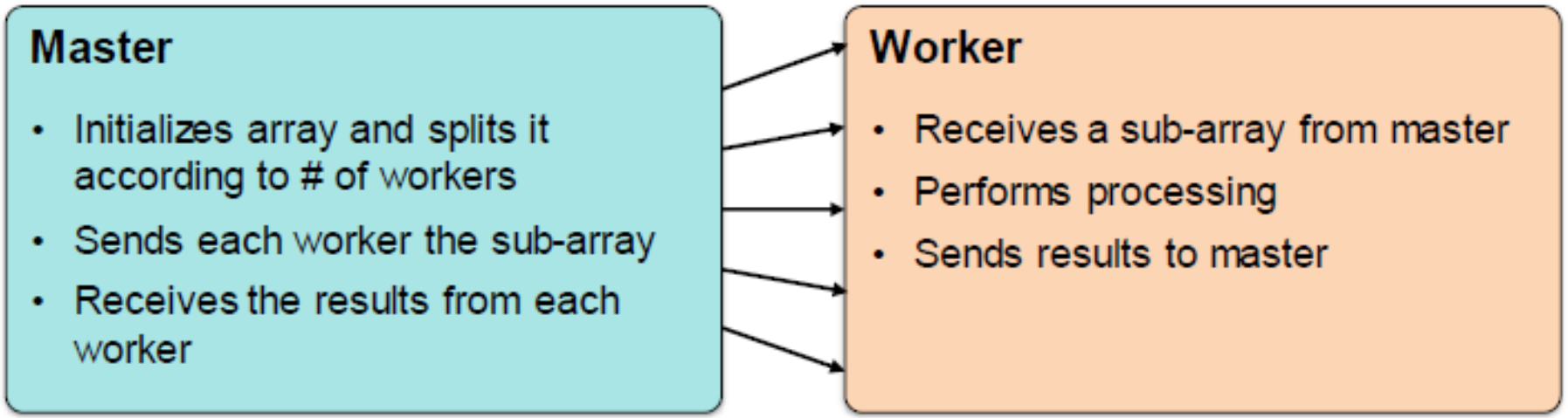
Parallel Programming

- Traditional programming is serial
  - Program consisted of a sequence of instructions, where each instruction executed one after the other. The program ran from start to finish on a single processor.
- Parallel programming
  - Break processing into parts that can be executed concurrently on multiple processors.
- Challenge
  - Identify tasks that can run concurrently and/or groups of data that can be processed concurrently
  - Not all problems can be parallelized

Simplest environment for parallel processing

No dependencies in the computations, and no communication required between tasks.

- Data can be split into equal-size chunks
- Each process can work on a chunk.
- Master/worker approach.



MapReduce

- Created by Google in 2004
  - Jeffrey Dean and Sanjay Ghemawat
- Inspired by LISP
  - Map(function, set of values)
  - Applies function to each value in the set

  (map 'length' (() (a) (a b) (a b c))) ⇒ (0 1 2 3)
  - Reduce(function, set of values)
  - Combines all the values using a binary function (e.g., +)

  (reduce #'+ '(1 2 3 4 5)) ⇒ 15

MapReduce

- MapReduce
  - Framework for parallel computing
  - Programmers get simple Java-based API
  - Don’t have to worry about handling
    - parallelization
    - data distribution
    - load balancing
    - fault tolerance
- Allows one to process huge amounts of data (terabytes and petabytes) on thousands of processors

How to Use and Who uses it?

- Google – Original proprietary implementation
- Apache Hadoop MapReduce http://hadoop.apache.org/
  - Most common (open-source) implementation
  - Built to specs defined by Google
- Backbone technology for Amazon Web Service (AWS) and other Cloud service provider.
- Who uses Hadoop and MapReduce?
  - http://wiki.apache.org/hadoop/PoweredBy

MapReduce

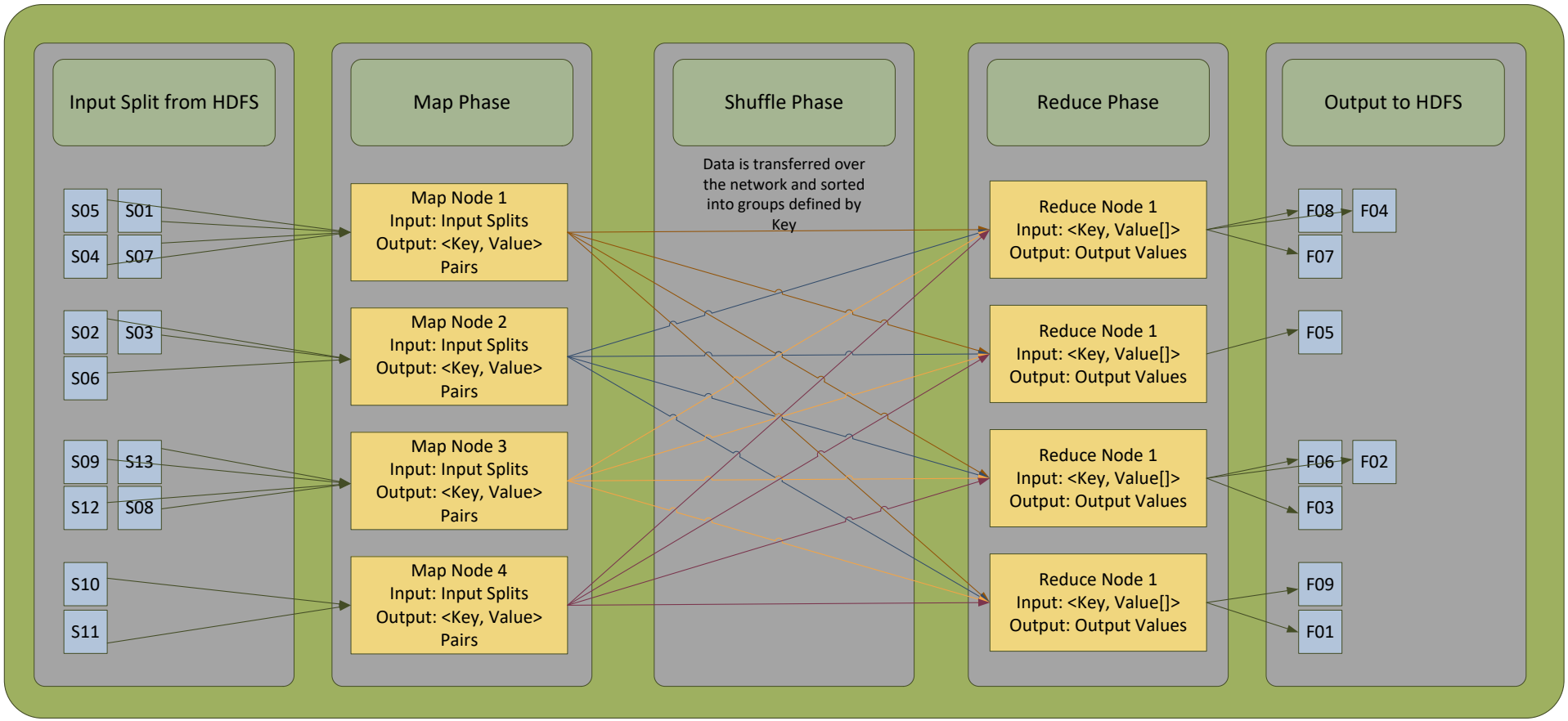
- How does MapReduce work?
  - Three primary steps are used to run a MapReduce job
    - Map
    - Shuffle
    - Reduce
  - Data is read in a parallel fashion across many different nodes in a cluster (Map)
    - Groups are identified for processing the input data, then output
  - The data is then shuffled into these groups (Shuffle)
    - All data with a common group identifier (Key) is then sent to a single location
  - Each group is then processed atomically across the nodes in parallel. (Reduce)
    - Each node will receive one or more groups to process
    - Each group is then processed and the results of the union of each group is the result of the entire job.

MapReduce

- A MapReduce (MR) program consists of generally two user defined stages.
  - Map:
    - Map(Key_p, $Value_p$) → list(Key_i, $Value_i$)
      - Takes in a data element
      - Outputs Key-Value pairs
  - Reduce:
    - Reduce(Key_i, list($Value_i$)) → list($Value_j$)
      - Takes in a Key and a collection of Values
      - Outputs results

MapReduce

Diagram of a MapReduce Job Flow



MapReduce

- Map Phase:
  - The map phase of MapReduce takes in a data element and divides it into zero or more output Key-Value pairs to be processed.
  - Each group to be processed in the Reduce is identified by the Key_i generated by the call to Map(Key_p, $Value_p$)
- Many different Map Tasks will be created across the different nodes in the distributed cluster
  - Each task will receive a block of the data to be processed
  - For each element in that block, it will make a call to Map(Key_p, $Value_p$)
  - For each call of the Map(Key_p, $Value_p$) , it will emit a list of Key-Value pairs derived from the input element in the form list(Key_i, $Value_i$)
- All of the list(Key_i, $Value_i$) from the different Map Tasks and calls to Map will then be shuffled across the network to the different Reduce Tasks
  - This shuffling produces the <Key_i, list($Value_i$)> data collection that are the input to the Reduce Phase

MapReduce

- Reduce Phase:
  - The reduce phase of MapReduce takes in a group defined by a unique key (Key_i) and a list of values (list($Value_i$))
- Many different reduce tasks will be created across the different nodes in the distributed cluster
  - Each task will make a call to Reduce(Key_i, list($Value_i$)) for every reduce group that it receives
  - Each call will process that reduce group atomically

MapReduce: Word Count

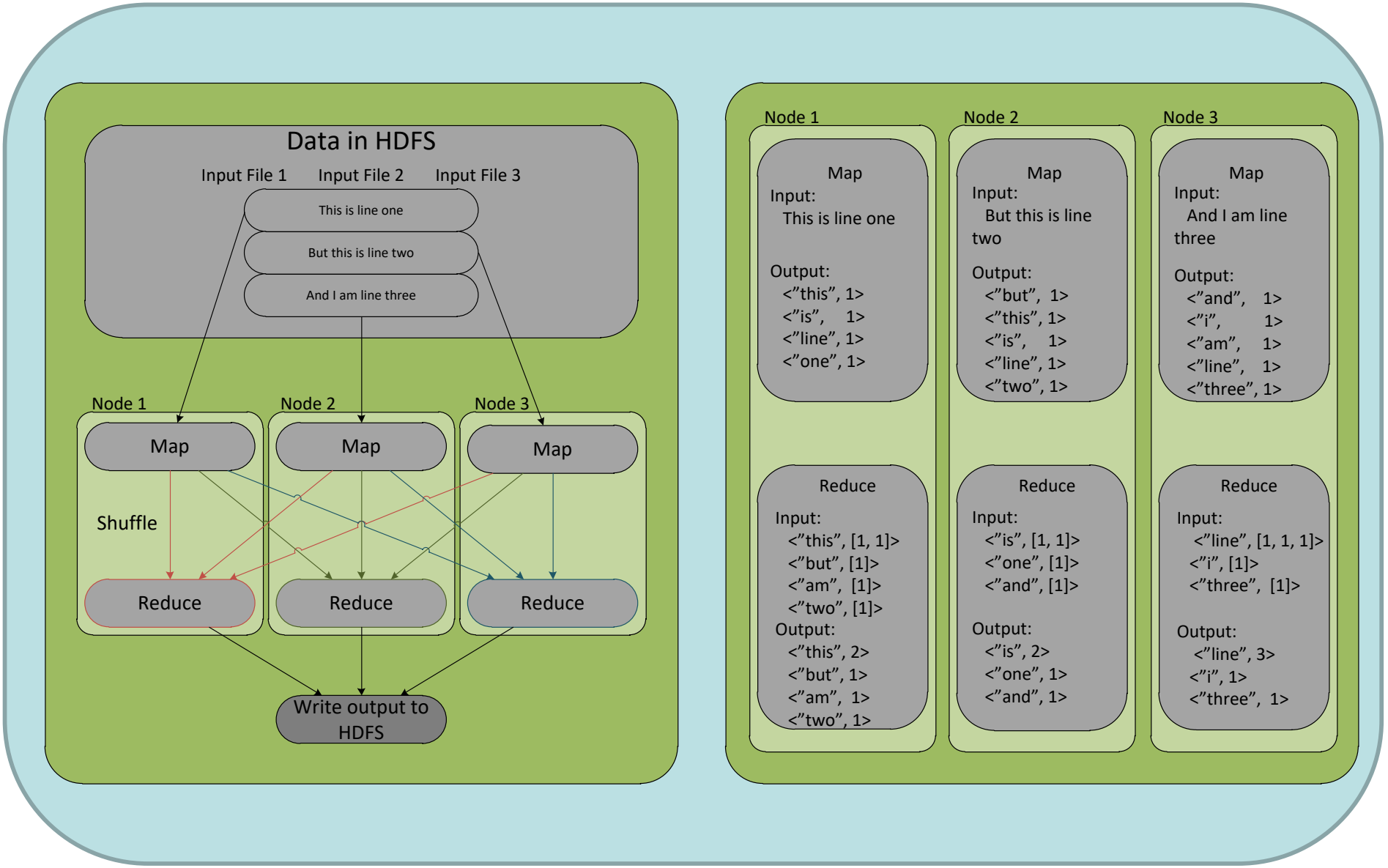
- MapReduce Example Program: Word Count
- Reads in files from the Hadoop Distributed File System (HDFS)
- Returns the counts of all words located in the files
- Map:

```
Mapper (line_number, line_contents)
    for each word in line_contents
        emit(word, 1)
```

- Reduce:

```
Reducer (word, values[])
    sum = 0
    for each value in values
        sum = sum + value
    emit(word, sum)
```

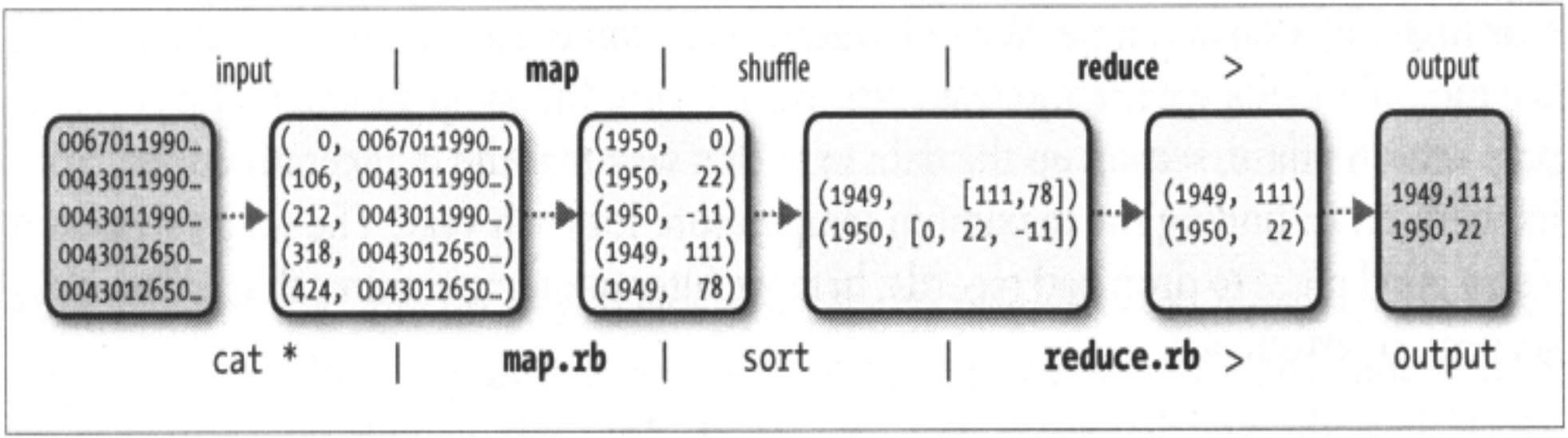
MapReduce: Word Count



MapReduce: MaxTemperature

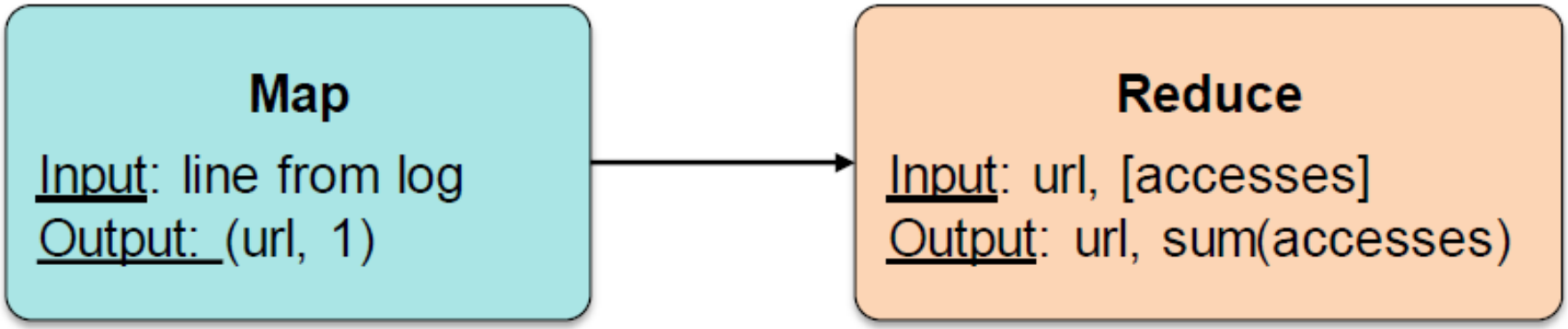
- Determine the maximum temperature for each year from National Climatic Data Center weather data
- A large quantity of raw data
  - YYYY = Year
  - TTTTT = Temperature in units of 0.1 C; 9999 means missing
  - Q = Quality code

MapReduce: MaxTemperature



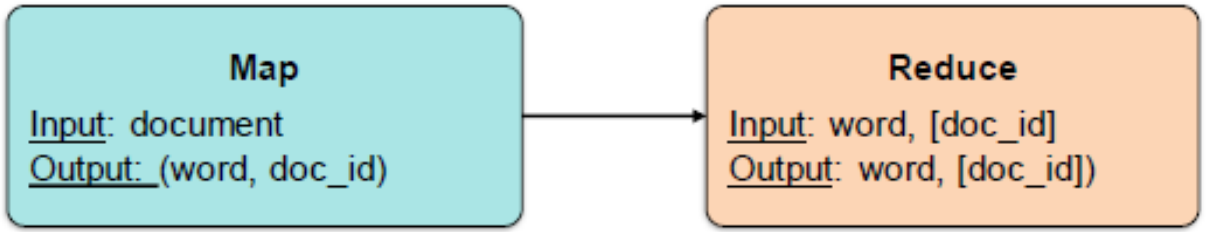
Other MapReduce Examples

Count URL access frequency
– *Find the frequency of each URL in web logs*
– Map: process logs of web page access; output <URL, 1>
– Reduce: add all values for the same URL



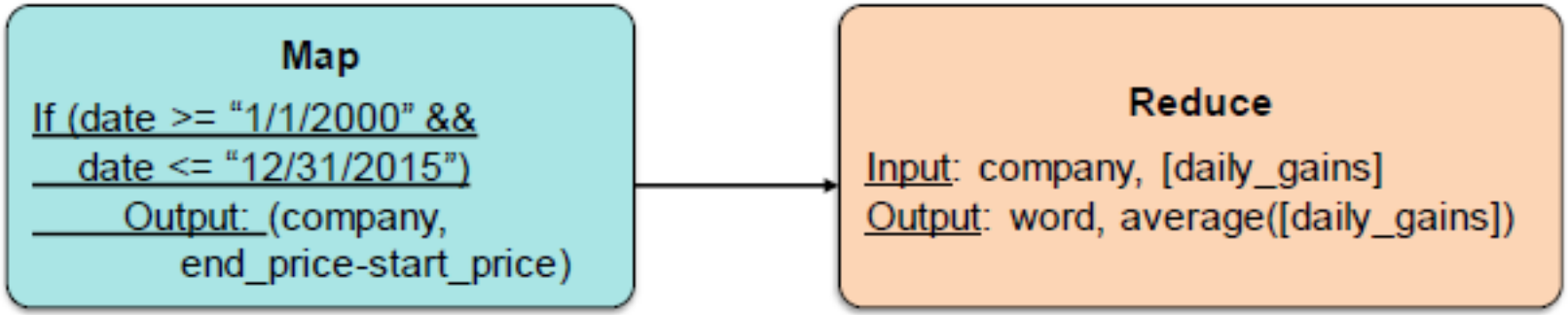
Other MapReduce Examples

- Inverted index
  - *Find what documents contain a specific word*
  - Map: parse document, emit <word, document-ID> pairs
  - Reduce: for each word, sort the corresponding document IDs
  - Emit a <word, list(document-ID)> pair
  - The set of all output pairs is an inverted index



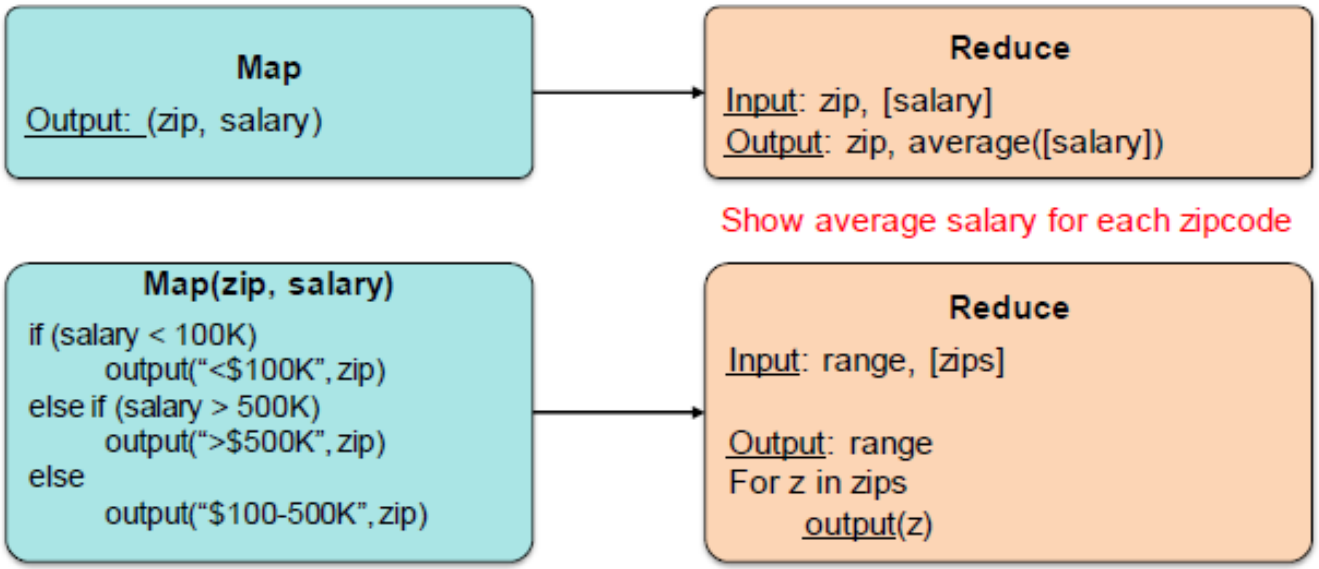
Other MapReduce Examples

- Stock summary
  - *Find average daily gain of each company from 1/1/2000 – 12/31/2015*
  - Input data is a set of lines: {date, company, start_price, end_price}



Other MapReduce Examples

- Average salaries in regions
- Show zip codes where average salaries are in the ranges: (1) < $100K (2) $100K … $500K (3) > $500K
- Input data is a set of lines: {name, age, address, zip, salary}



Hadoop

- What is Hadoop?
  - Hadoop Distributed File System (HDFS)
    - The file system is dynamically distributed across multiple computers
    - Allows for nodes to be added or removed easily
    - Highly scalable in a horizontal fashion
  - Hadoop Development Platform
    - Uses a MapReduce model for working with data
    - Users can program in Java, C++, and other languages

Hadoop

- Some of the Key Characteristics of Hadoop:
  - On-demand Services
  - Rapid Elasticity
    - Need more capacity, just assign some more nodes
  - Scalable
    - Can add or remove nodes with little effort or reconfiguration
  - Resistant to Failure
    - Individual node failure does not disrupt the system
  - Uses off the shelf hardware

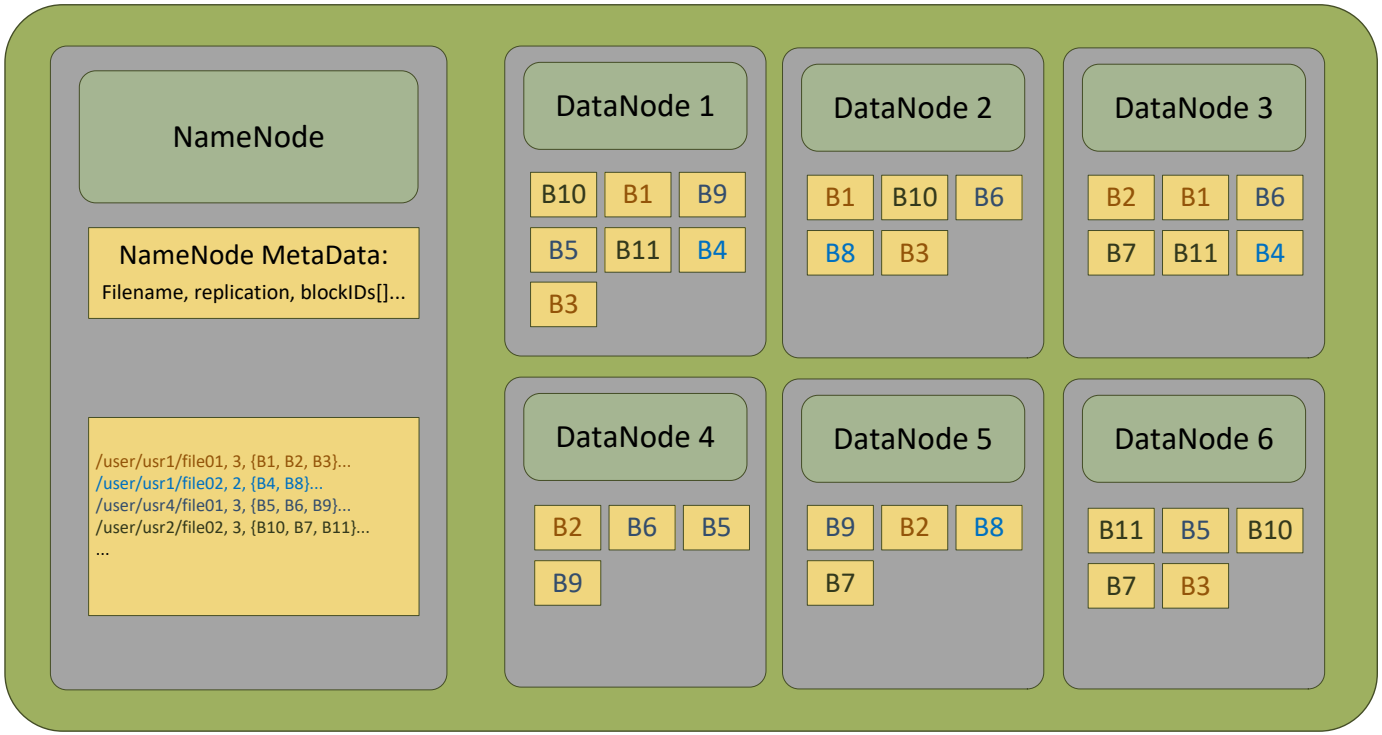
Hadoop

- How does Hadoop work?
  - Runs on top of multiple commodity systems
  - A Hadoop cluster is composed of nodes
    - One Master Node
    - Many Worker/Client Nodes
  - Multiple nodes are used for storing data & processing data
  - System abstracts the underlying hardware to users/software

Hadoop: HDFS

- HDFS is a multi-node system
  - Name Node (Master)
    - Single point of failure
  - Data Node (Client)
    - Failure tolerant (Data replication)

- HDFS Consists of data blocks
  - Files are divided into data blocks
    - Default size of 64MB
    - Default replication of blocks is 3
  - Blocks are spread out over Data Nodes



Hadoop MapReduce

Data Types

- Writable/Comparable
  - Text vs String
  - LongWritable vs long
  - IntWritable vs int
  - DoubleWritable vs double

Structure of a Hadoop Mapper (WordCount)

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

    public void map(LongWritable inputKey, Text inputValue, Context context) throws IOException, InterruptedException
    {
        //The inputValue is going to be a line of text for our program.
        //We will be manipulating this line of text to get every word out of it and then output it.
        //A call of the map method will be invoked for every line of text in the input files.

        //Create a string tokenizer from the input file so that we can
        //easily output all of the "word" in this input
        StringTokenizer line = new StringTokenizer(inputValue.toString());

        /*
         * for each "word" or token in the input string, output that word with a count of "1".
         * We will add these words up later in the reduce.
         * We can also write a combiner class if we want to help combine some of these values
         * before we transmit the information over the network. This will not however be done
         * for simplicities sake.
         */
        while(line.hasMoreTokens())
        {
            //this writes the Key Value pair to the system so that it can be shuffled
            //around and then sent to the appropriate reduce group.
            //The key in this program is going to be the word, this will cause all counts
            //of a word to be sent to the same place. The value will be '1' in this case.
            context.write(new Text(line.nextToken()), new IntWritable(1));
        }


    }//end map

}//end MyMapper
```

Structure of a Hadoop Reducer (WordCount)

```
public static class MyReducer extends Reducer<Text, IntWritable, Text, Text>
{

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
    {
        //create a sum that we can add up values into.
        long sum = 0;

        //create the interator to iterate over the Iterable list values
        Iterator<IntWritable> valItr = values.iterator();

        //While there are more values to process, get them and add them to our sum
        while(valItr.hasNext())
        {
            //Get the next IntWritable and then, from that get
            //it's value so that we can add it to the sum
            sum += valItr.next().get();

        }

        //Since we now have the sum of all the counts of a specific word, we can output this as our result.

        //create a new text object with the sum as it's value
        //output the key and the sum as a result.
        context.write(key, new Text("" + sum));


    }//end reduce

}//end MyReducer
```

MapReduce: Limitations

- Batch-oriented
- Not suited for near-real-time processes
  - Can't handle analyzing streaming data
- Cannot start a new phase until the previous has completed
    - Reduce cannot start until all Map workers have completed
- Suffers from "stragglers" – workers that take too long (or fail)

Current State of MR

- Apache Hadoop now dominates use of the MapReduce framework
- Often, Hadoop map and reduce functions are no longer written directly
- Instead, a user writes a query in a very high level language and uses another tool to compile the query into map/reduce functions!
  - Hive (another Apache project) compiles SQL queries into map/reduce
  - Pig (yet another Apache project) compiles direct relational algebra into map/reduce

Apache Spark: The Future

- Eventually, users started realizing that a much larger class of algorithms could be expressed as an iterative sequence of map/reduce operations
  - Many machine learning algorithms fall into this category
- Tools started to emerge to enable easy expression of multiple map/ reduce operations, along with smart scheduling
- Apache Spark: General purpose functional programming over a cluster
  - Caches results of map/reduce operations in memory so they can be used on subsequent iterations without accessing disk each time
  - Tends to be 10-100 times faster than Hadoop for many applications

Acknowledgement

- A. Haeberlen, Upenn NETS212
- Paul Krzyzanowski, Rutgers University