COSC 416 NoSQL Databases

Hadoop and HDFS

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

MapReduce and Hadoop

MapReduce was invented by Google and has an open source implementation called Apache Hadoop (hadoop.apache.org/).

- Implemented in Java, Apache Top Level Project, most contributors from Yahoo

Hadoop is a software library designed for distributed processing of large data sets using computer clusters.

Key components:

- HDFS – Hadoop Distributed File System
- Hadoop MapReduce – Parallel processing of data
- Many other related projects and systems (we will talk about Hive and Pig later).

Hadoop/HDFS Architecture

Distributed architecture assumes inexpensive, unreliable commodity hardware. Replication allows reliability.

Two types of nodes:

- ***NameNode*** maintains file metadata
- ***DataNodes*** manage storage

File access API:

- Mostly sequential access
- Single writers and no locking

"Computation at the data":

- Servers are for both storage and computation

Hadoop/HDFS Architecture (2)

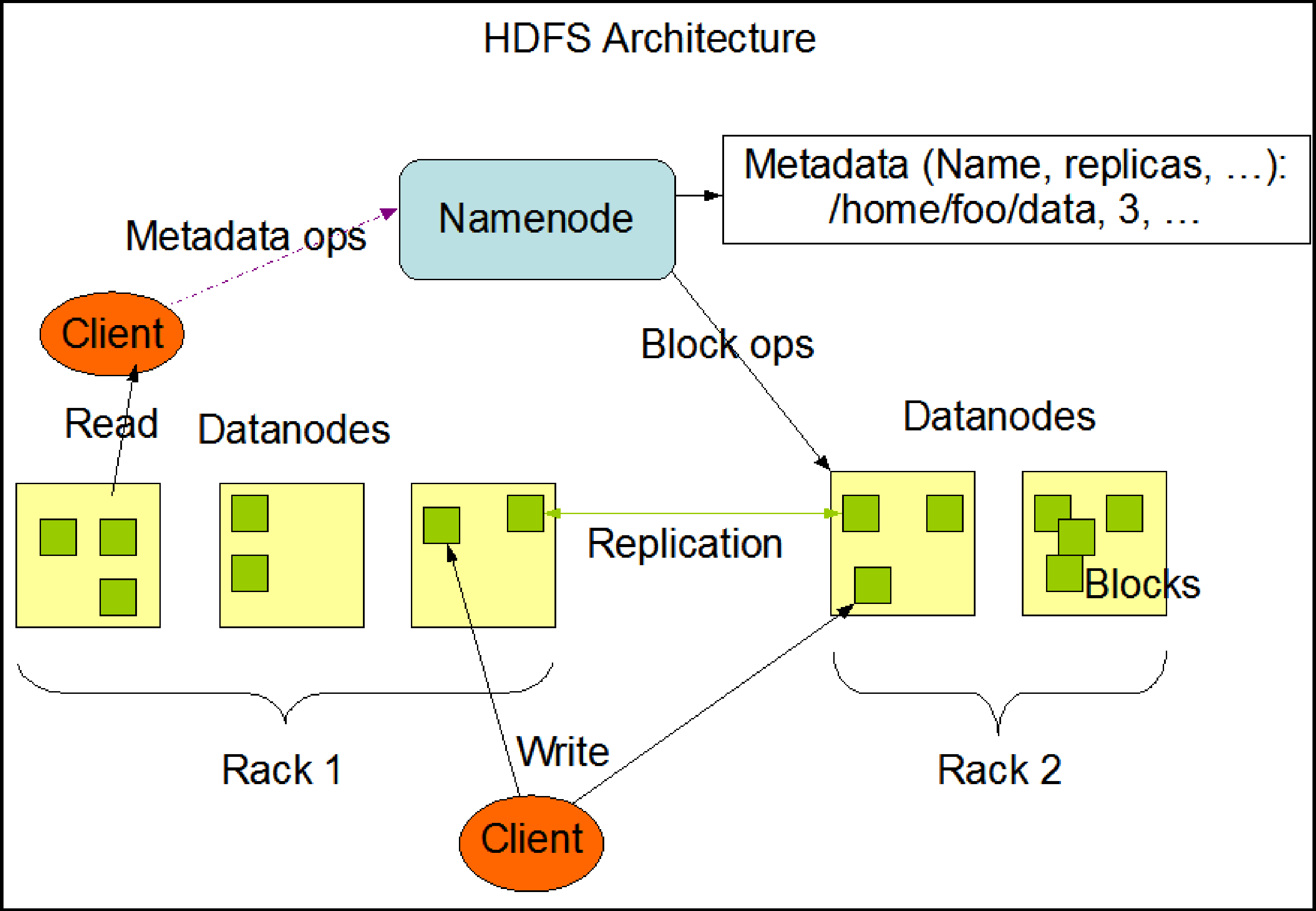
Data:

- Data is organized into files and directories.
- Files are divided into fixed sized blocks and replicated across cluster nodes. Replication factor on a per file basis (default 3).
- NameNode stores the block replication information.
- Writes are pipelined to replicas. Reads from nearest replica.

Master-Slave architecture:

- NameNode is master and manages file information, blocks, replication/mapping locations.
- DataNodes store blocks on underlying OS. Clients can access blocks directly on DataNodes without going through nameNode.

HDFS Architecture



Source: hadoop.apache.org

MapReduce

MapReduce/Hadoop parallel processing involves mapping and grouping file records by keys.

Data is stored in files. Users provide functions:

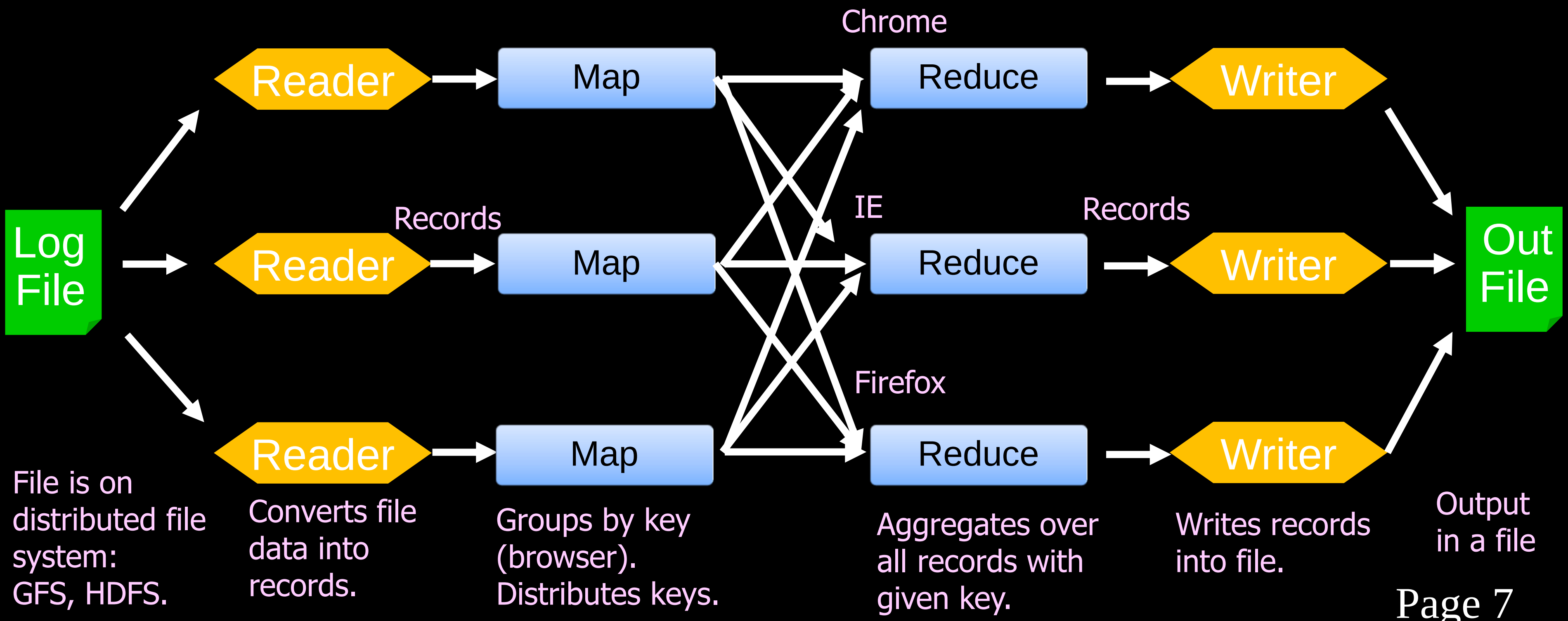
- reader(*file*) – converts file data into records
- map(*records*) – converts records into key-value pairs
- combine(*key, list of values*) – optional aggregation of pairs after map stage
- reduce(*key, list of values*) – summary on key values to produce output records
- write(file) – writes records to output file

MapReduce (Hadoop) provides infrastructure for tying everything together and distributing work across machines.

MapReduce Example Web Data Analysis

Data file records: URL, timestamp, browser

Goal: Determine the most popular browser used.

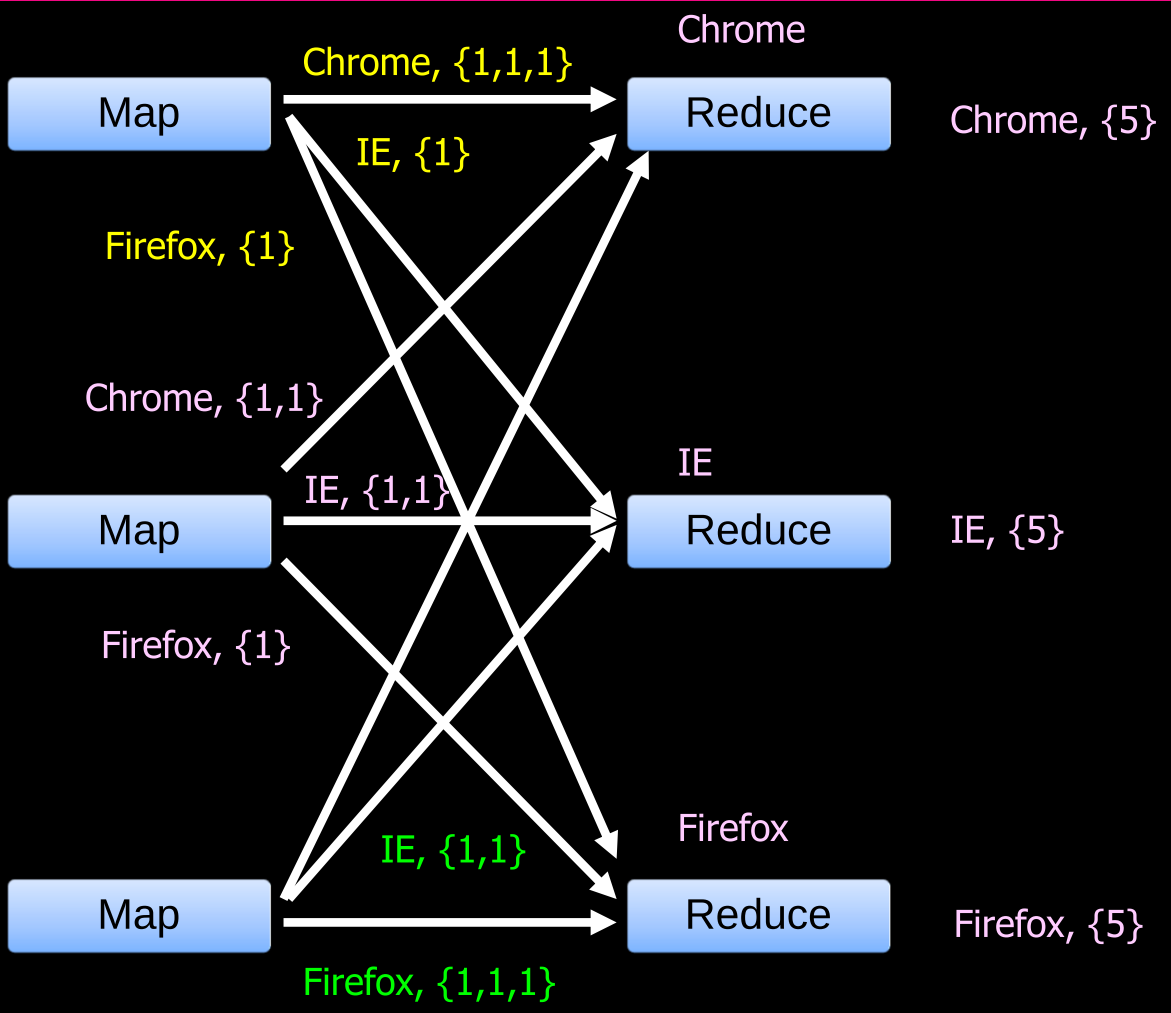

Log File
Reader
Reader
Reader
Records
Map
Map
Map
Chrome
IE
Firefox
Reduce
Reduce
Reduce
Records
Writer
Writer
Writer
Out File
File is on distributed file system: GFS, HDFS.
Converts file data into records.
Groups by key (browser). Distributes keys.
Aggregates over all records with given key.
Writes records into file.
Output in a file

MapReduce Example (2) Web Data Analysis

yahoo.com, Chrome
google.com, Firefox
google.com, Chrome
msdn.com, IE
yahoo.ca, Chrome

xyz.com, Chrome
linkedin.com, Chrome
google.ca, IE
msdn.ca, Firefox
msdn.com, IE

costco.ca, Firefox
walmart.com, Firefox
amazon.com, Firefox
msdn.ca, IE
ubc.ca, IE



WordCount Hadoop Example

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {
     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);
       }
     }
   }
```

← Import Hadoop APIs

Create Mapper

WordCount Hadoop Example (2)

```
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));
  }
}
```

Reducer

WordCount Hadoop Example (3)

```
    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);
    }
}
```

$\leftarrow$ Main program

Job Configuration Parameters

Job configuration parameters control the execution of Hadoop.

Examples:

- core-site.xml
  - `fs.default.name` – file system name node name
- hdfs-site.xml
  - `dfs.replication` - # of block replicas
  - `dfs.data.dir` – directory to store data blocks
- mapred-site.xml
  - `mapred.job.tracker` – URL of job tracker

Hadoop Versions

As a relatively new open source project, Hadoop is rapidly changing. There are many different versions, and incompatibilities and differences between them (including configuration files).

We are running 1.0.4 (latest stable release) that is consistent with 0.23.X releases. When using web sources as references, watch for the version being used.

Conclusion

Hadoop is an open source implementation of a highly scalable distributed file system and MapReduce job processing architecture.

It is designed for extremely large data processing on commodity hardware.

Using MapReduce requires writing code that defines a mapping and reducer and executing it on the cluster. Efficient algorithms should be highly parallelizable and minimize the amount of data transfer.

Objectives

Understand the Hadoop/HDFS architecture and file/block allocation strategy.

Explain how a MapReduce program is designed.

Be able to understand a simple MapReduce program and write small amounts of code.