MapReduce Framework

Rim Moussa

rim.moussa@gmail.com

MapReduce Framework

- Challenges of Divide & Conquer for solving a problem
  - How to divide a problem into units of work?
  - How to assign units of work to workers?
  - How works coordinate and send to each other intermediate results?
  - How to track the job processing? What if workers fail?
- Published by Jeffrey Dean and Sanjay Ghemawat (Google) in 2004

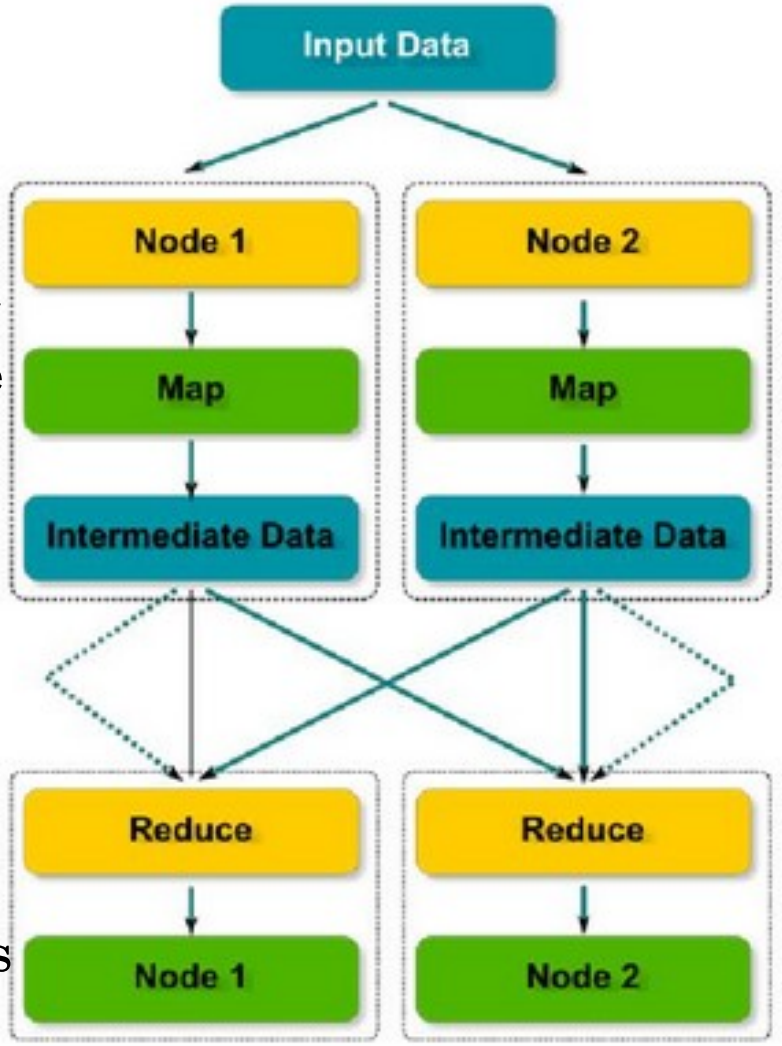
- Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner.

RDBMS compared to MapReduce by T. White

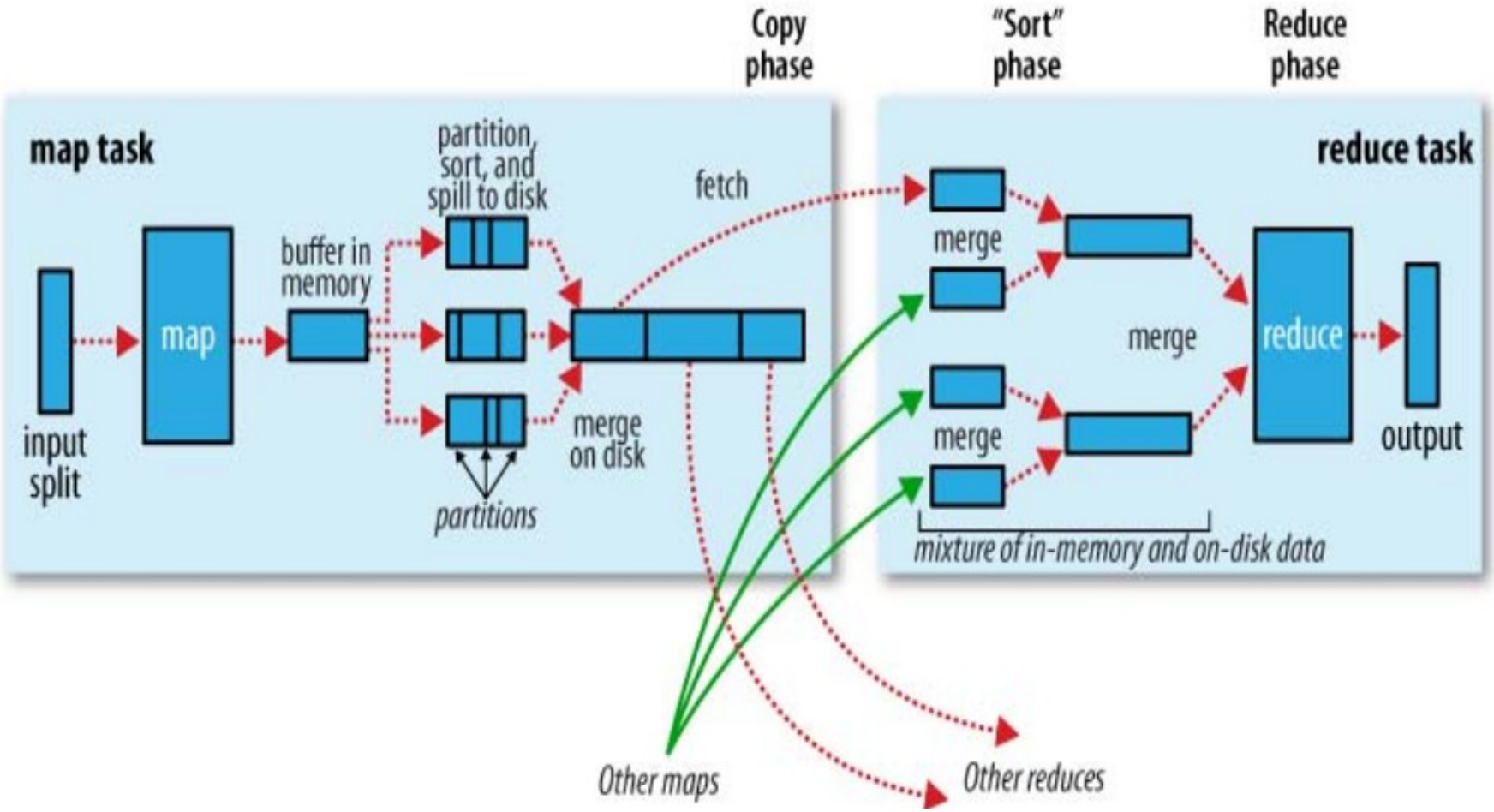
	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

MapReduce Flow

- Input data is one or multiple files partitioned over data nodes of a hadoop cluster
- By default, the input split is processed by each mapper is equal to the file block size
- Each mapper emits intermediate results in the form of (k,v) pairs
- Shuffling is the process of transferring data from the mappers to the reducers
  - Shuffling can start before the map phase has finished
- The output of shuffling is sent to reducers for processing (default partitioner is based on a hash function on k')
- Each reducer emits the final output in the form of key-value pairs



MapReduce Flow in details



Max Temperature Example

- Data from National Climatic Data Center
  - A large volume of log data collected by weather sensors
- Data Format
  - Line-oriented ASCII format
  - Each record has many elements
  - We focus on year and temperature data
- Query: What's the highest recorded global temperature for each in the dataset

```
(0, 006701199099999**1950**051507004...9999999N9+**0000**1+99999999999...)
(106, 004301199099999**1950**051512004...9999999N9+**0022**1+99999999999...)
(212, 004301199099999**1950**051518004...9999999N9-**0011**1+99999999999...)
(318, 004301265099999**1949**032412004...0500001N9+**0111**1+99999999999...)
(424, 004301265099999**1949**032418004...0500001N9+**0078**1+99999999999...)
```

Max Temperature Mapper (ctnd. 1)

- Input (k: offset of the line in the file, v: a line)
- Output (k': year, v': temperature)

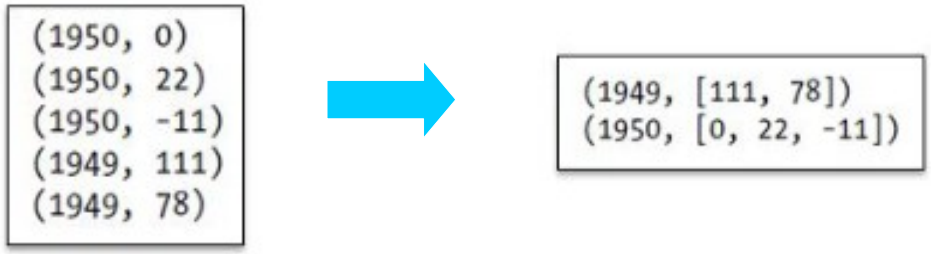
```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

```
public void map(LongWritable key, Text value,
                Context context) throws IOException,InterruptedException {


  String line = value.toString();
  String year = line.substring(15, 19);
  int airTemperature=0;
  if (line.length()>92 && line.charAt(87) == '+') {
    airTemperature = Integer.parseInt(line.substring(88, 92));
  } else {
    if (line.length()>92)airTemperature = Integer.parseInt(line.substring(87, 92))
  }
  if (line.length()>93){
      String quality = line.substring(92, 93);
      if (airTemperature != MISSING && quality.matches("[01459]")) {
        context.write(new Text(year), new IntWritable(airTemperature));
      }
  }
}
```

Max Temperature Reducer (ctnd. 2)

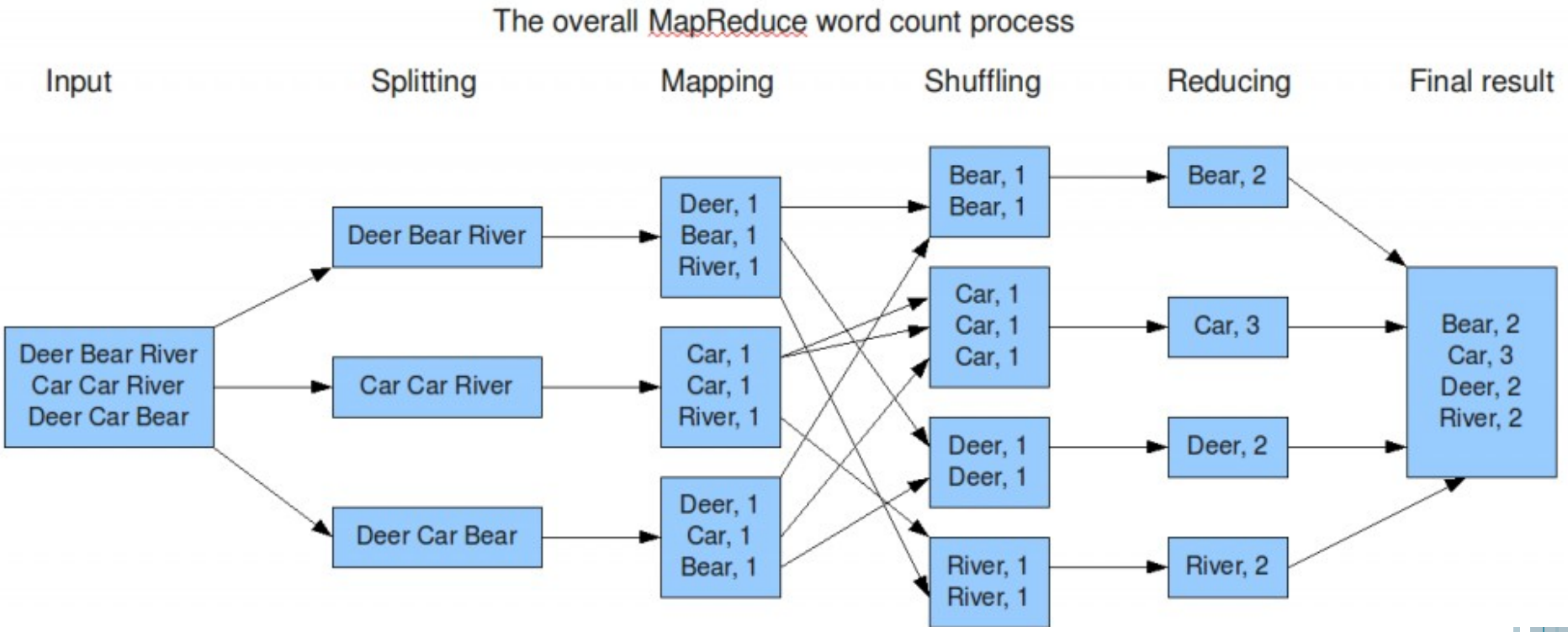
- The framework sorts and groups by *k'* all key-value pairs



- Reducer
  - Input (*k'*, *list of values*)
  - Output (*k'*, *max(list of values)*)


```
/**
 * reducer class
 */
public static class Reduce1 extends Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context) throws IOException,InterruptedException {
    int maxValue = Integer.MIN_VALUE;
    Iterator<IntWritable> it = values.iterator();
    while (it.hasNext()) {
      maxValue = Math.max(maxValue, it.next().get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

Word Count Example

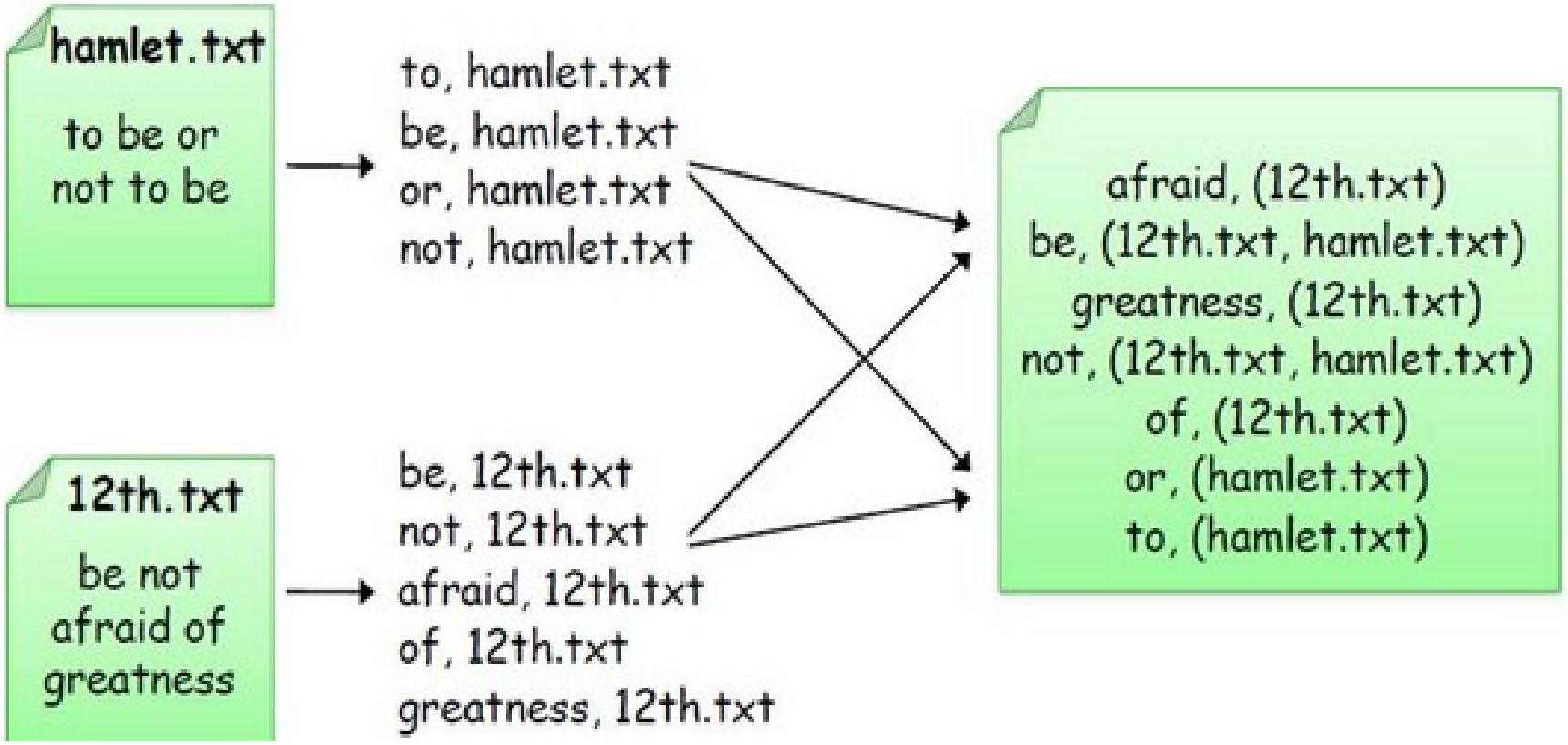


Word Count Example (ctnd. 1)

```
public static class Map1 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 {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

```
  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context) throws IOException,InterruptedException {

    int sum = 0;
    for (IntWritable value : values) {
      sum += value.get();
    }
    context.write(key, new IntWritable(sum));
  }

}
```

Inverted Index Example



Inverted Index Example (ctnd. 1)

```
public void map(LongWritable key, Text value,
                Context context) throws IOException,InterruptedException {
    StringTokenizer tokenizer = new StringTokenizer(value.toString());
    FileSplit fileSplit = (FileSplit)context.getInputSplit();
    Text filePath = new Text(fileSplit.getPath().getName());

    while (tokenizer.hasMoreTokens()){
        context.write(new Text(tokenizer.nextToken()), filePath);
    }
}
```

```
public void reduce(Text key, Iterable<Text> values,
                   Context context) throws IOException,InterruptedException {
    HashMap hm = new HashMap();
    int count=0;
    for(Text t: values){
        String str = t.toString();
        //Check if file name is present in the HashMap ,if File name is not present
        // then add the Filename to the HashMap and increment the counter by one
        if(hm != null && hm.get(str) != null){
            count = (int)hm.get(str);
            hm.put(str, ++count);
        }
        else{//if file name is already added then just increase the count for that file name
            //which is stored as key in the hash map
            hm.put(str, 1);
        }
    }
    //Emit word and [file1→count of the word1 in file1,file2→count of the word1 in file2 ……
    context.write(key, new Text(hm.toString()));
}
```

MapReduce Job Performance Tuning

- IO
  - Data block size can be set for each file
  - Input/Shuffle phase: data compression
  - Combiners (local reducers)
- Parallelism
  - set the Input split to a fraction of block size → This will increase the number of mappers
  - Number of reducers
- Communication
  - Data locality: run a Map on local data (no blocks' transfers in the net)
  - Data compression
  - Combiners (local reducers)
- Resource Management
  - Each node has a computing and a memory capacity
  - Mappers and Reducers have different allocated resources

Lab

- TPC-H bench
  - Customer file (SF*150K) and Orders file (SF*1.5M) are relatively big
  - Nation (25 lines) and Region (5 lines) files are small
- Propose MapReduce jobs for the following queries
  - Select customers of a given nation
  - Return names and phone numbers of customers
  - Join Customer and Nation
  - Join Customer and Orders
  - Count the number of customers by Region and Nation

References

- Jeff Markham, Joseph Niemiec, Vinod Kumar Vavilapalli, Arun C. Murthy, Doug Eadline. *Apache Hadoop YARN: moving beyond MapReduce and batch Processing with Apache Hadoop 2*. Addison Wesley prof. 2014.
- Chuck Lam. *Hadoop in Action*. Manning. 2011.
- Tom White. *Hadoop: The Definitive Guide*. O'Reilly, Yahoo! Press. 2009. https://github.com/tomwhite/hadoop-book/
- Mark C. Chu-Carroll. *Databases are hammers; MapReduce is a screwdriver.* 2008. http://scienceblogs.com/goodmath/2008/01/databases_are_hammers_mapreduc.php