15-440 Recitation 8: Hadoop Programming

Vijay Vasudevan

Outline

- Hadoop walkthrough

Hadoop

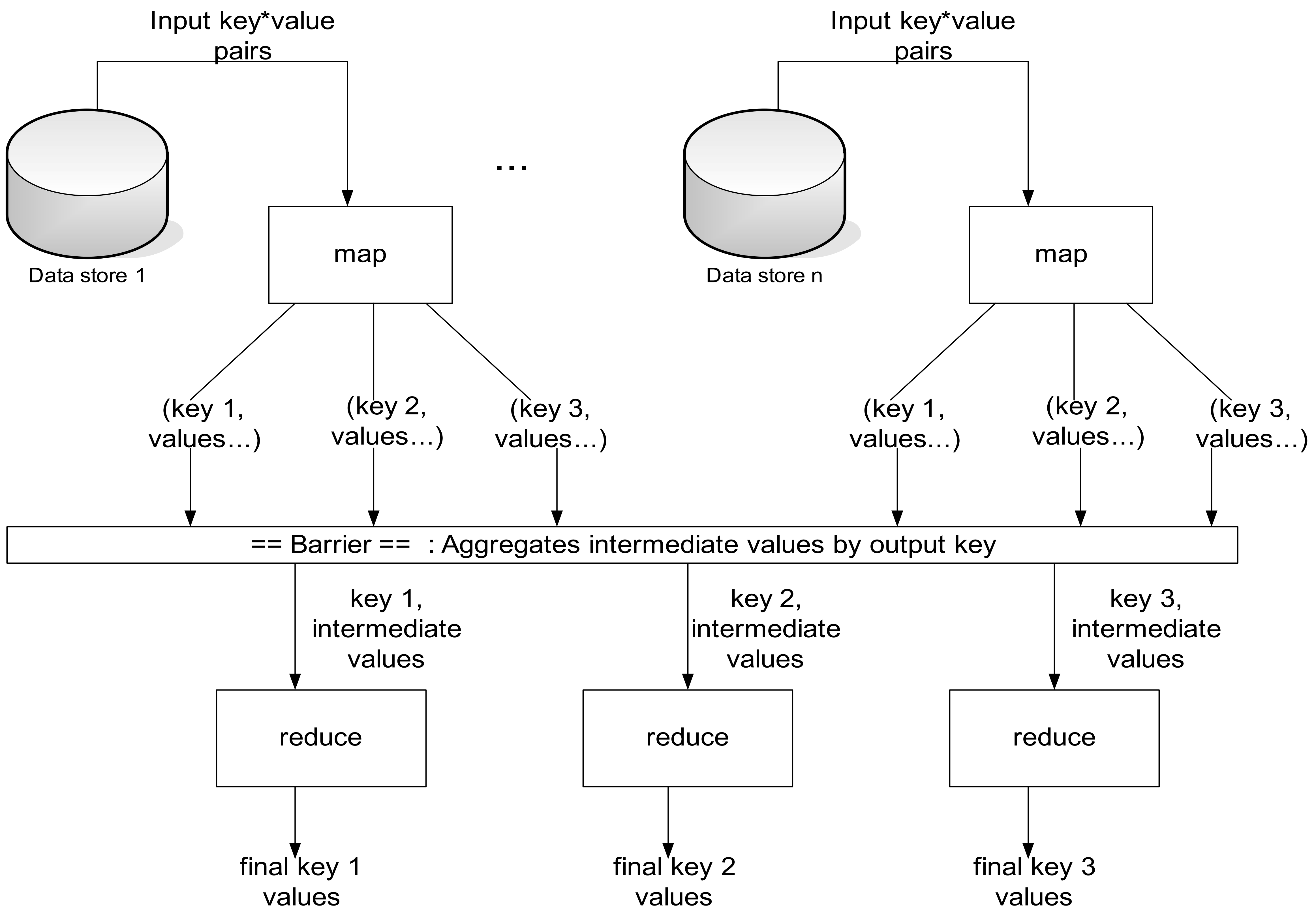
- Apache Hadoop is an open-source version of Google's MapReduce
- Who uses it?
  - Yahoo, Facebook, Physicists, Wall Street...
- Great for large-scale data processing

map

- Records from the data source (lines out of files, rows of a database, etc) are fed into the map function as key*value pairs: e.g., (filename, line).
- map() produces one or more *intermediate* values along with an output key from the input.

reduce

- After the map phase is over, all the intermediate values for a given output key are combined together into a list
- reduce() combines those intermediate values into one or more *final values* for that same output key
- (in practice, usually only one final value per key)


Input key*value pairs
Input key*value pairs
Data store 1
map
...
Data store n
map
(key 1, values...)
(key 2, values...)
(key 3, values...)
(key 1, values...)
(key 2, values...)
(key 3, values...)
== Barrier == : Aggregates intermediate values by output key
key 1, intermediate values
key 2, intermediate values
key 3, intermediate values
reduce
reduce
reduce
final key 1 values
final key 2 values
final key 3 values

Parallelism

- map() functions run in parallel, creating different intermediate values from different input data sets
- reduce() functions also run in parallel, each working on a different output key
- All values are processed *independently*
- Bottleneck: reduce phase can’t start until map phase is completely finished.

Optimizations

- “Combiner” functions can run on same machine as a mapper
- Causes a mini-reduce phase to occur before the real reduce phase, to save bandwidth

Word Count

```
// key = document name, value = document contents

map(String key, String value):
  for each word w in value:
    EmitIntermediate(w, “1”)


reduce(String key, list values):
  int count = 0
  for each v in values:
      count += parseInteger(v)
  Emit(string(count))
```

Map Function

```
  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);
        }
    }
}
```

Reduce Function

```
 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));
}
}
```

Writing Main()

```
  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);

// Multiple phases?  just set up a new JobConf!
}
```

Compiling Hadoop job

```
$ mkdir ~/wordcount_classes

$ javac -classpath /usr/local/hadoop-0.20.1-core.jar
    -d ~/wordcount_classes ~/WordCount.java

$ jar -cvf ~/wordcount.jar -C ~/wordcount_classes/ .
```

Running a Hadoop job

```
   $ bin/hadoop dfs -ls data/words/
data/words/file01
data/words/file02

$ bin/hadoop dfs -cat data/words/file01
Hello World Bye World

$ bin/hadoop dfs -cat data/words/file02
Hello Hadoop Goodbye Hadoop

$ bin/hadoop jar ~/wordcount.jar org.myorg.WordCount
      data/words username/output
$ bin/hadoop dfs -cat username/output/part-00000
  Bye 1
  Goodbye 1
  Hadoop 2
  Hello 2
  World 2
```

Alternative ways

- Higher-level languages developed on top of MapReduce
  - Hive/Pig for Hadoop
  - Sawzall for MapReduce (@ Google)
  - DryadLINQ for Dryad (@ Microsoft)

Project 3

- Learn how to write and run Hadoop jobs
- Do some large scale data processing
- Have fun!