Hadoop/MapReduce: Optimizations

How it looks like in Java

File Edit Options Buffers Tools Java Help

```
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);
    }}}

  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));
    }}

  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);
  }}
```

--:--- **mapreduce.java** All L9 (Java/l Abbrev)----------

Wrote /home/shivnath/Desktop/mapreduce.java

Provide implementation for Hadoop's Mapper abstract class

Map function

Provide implementation for Hadoop's Reducer abstract class

Reduce function

Job configuration

Optimizations

- **Four optimizations for map-reduce processing**

Optimization 1

- **In Color Count example, what if number of colors is small → then can we optimize the map-side ?**

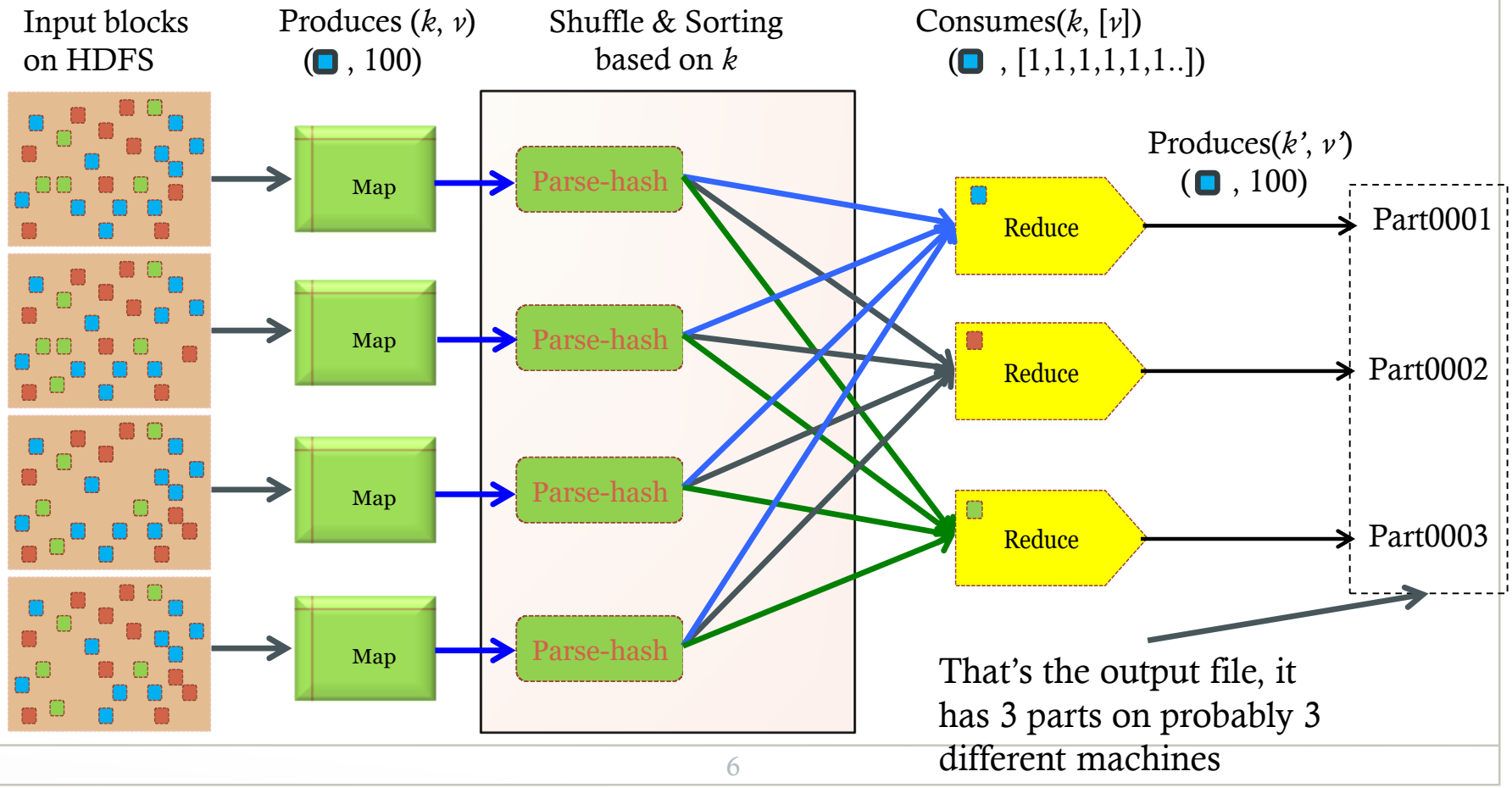
- Each map function can have a small main-memory hash table (color, count)
- With each line, update the hash table and produce nothing
- When done, report each color and its local count

■	10
■	5
■	7
■	20

Gain: Reduce amount of shuffled/sorted data over network

Optimization 1: Takes Place inside Mappers

Saves network messages (Typically very expensive phase)



Optimization 1

- Small main-memory hash table (color, count)
- Update hash table
- Report result using hash table when done.

■	10
■	5
■	7
■	20

Q1: Where to build the hash table?
Q2: How to know when done?
Q3: What do, when done?

Inside the Mapper Class

Called once after all records done
(Here you can produce the output)

Method Summary

protected void	**cleanup**(Mapper.Context context) Called once at the end of the task.
protected void	**map**(KEYIN key, VALUEIN value, Mapper.Context context) Called once for each key/value pair in the input split.
void	**run**(Mapper.Context context) Expert users can override this method for more complete control over the execution of the Mapper.
protected void	**setup**(Mapper.Context context) Called once at the beginning of the task.

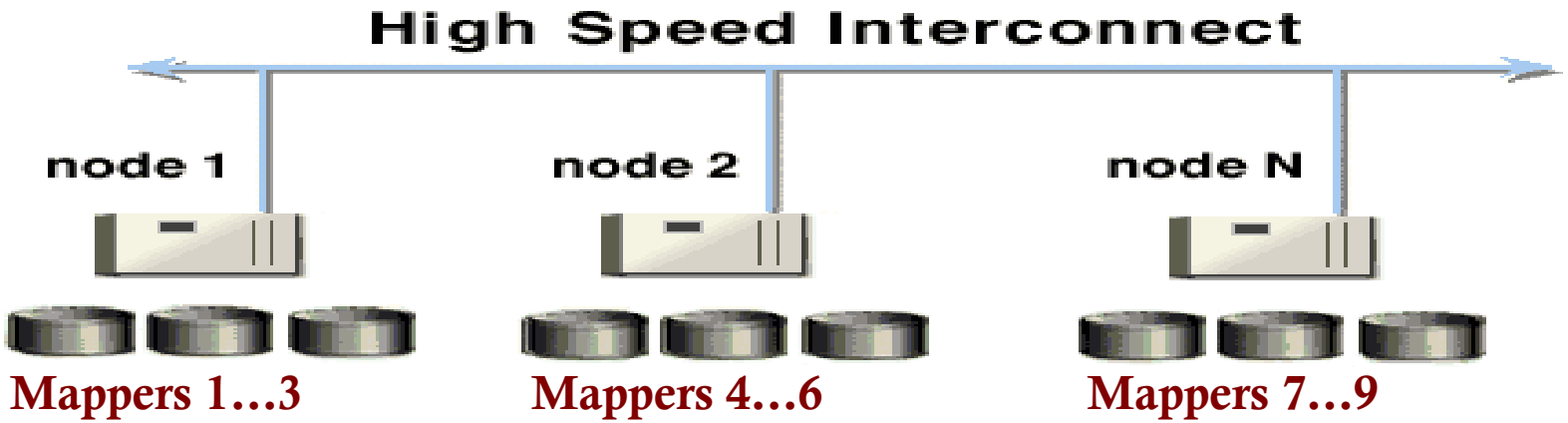
Called for each record

Called once before any record processed
(Here you can build the hash table)

Reducer has similar functions…

Opt. 2: Map-Combine-Reduce

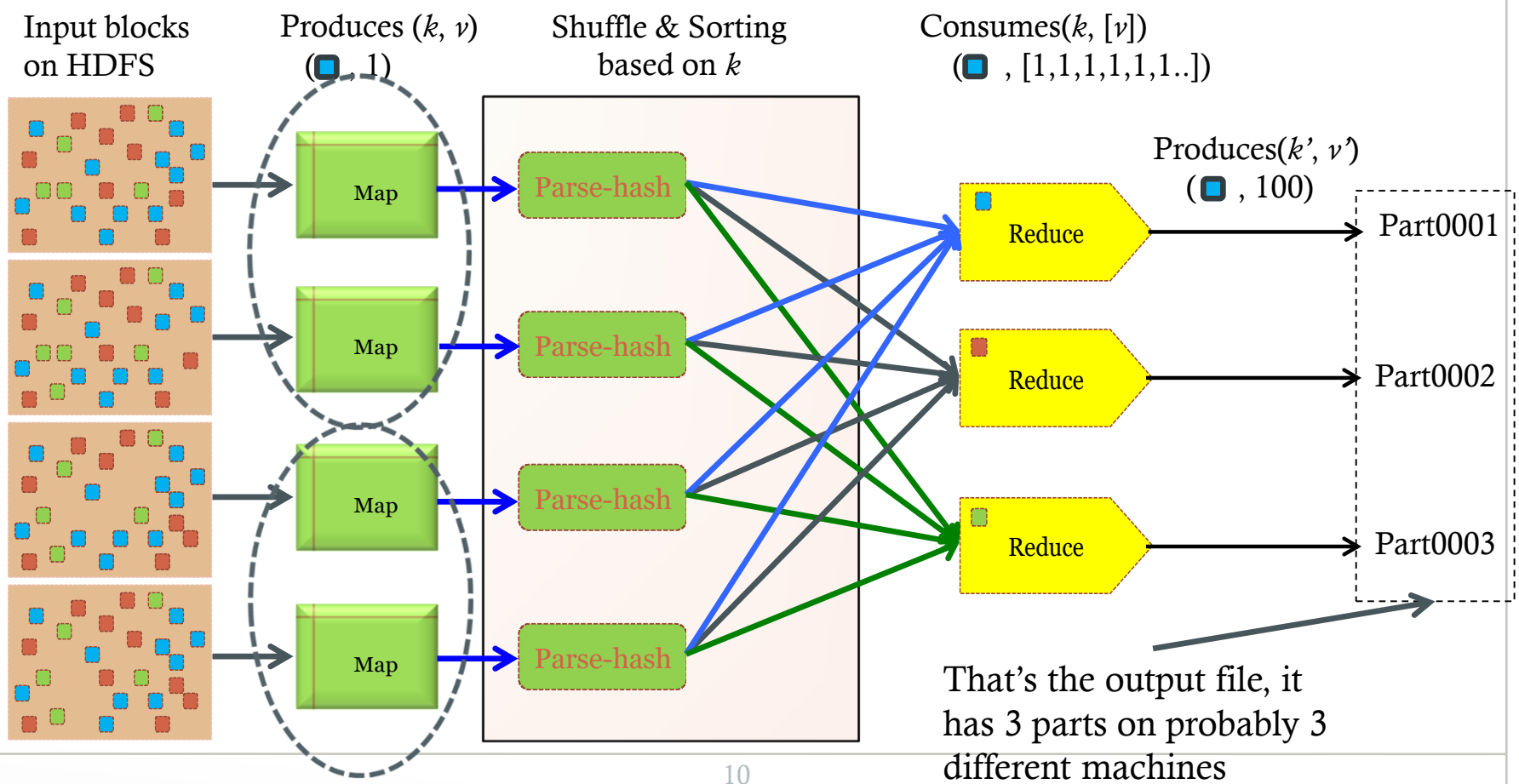
- On each machine, we partially aggregate results from mappers



- A ***combiner*** is a ***reducer*** that runs on each machine to locally aggregate (*via user code*) mappers' outputs from this machine
- Combiners' output is shuffled and sorted for 'real' reducers

Optimization 2: Outside Mappers, But on Each Machine

Combiner runs on each node to partially aggregate local mappers' output



Tell Hadoop to use a Combiner

File Edit Options Buffers Tools Java Help

```
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);
  }}}

  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));
  }}

  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);
  }}
```

--:--- **mapreduce.java** All L9 (Java/l Abbrev)----------

Wrote /home/shivnath/Desktop/mapreduce.java

Not all jobs can use a combiner

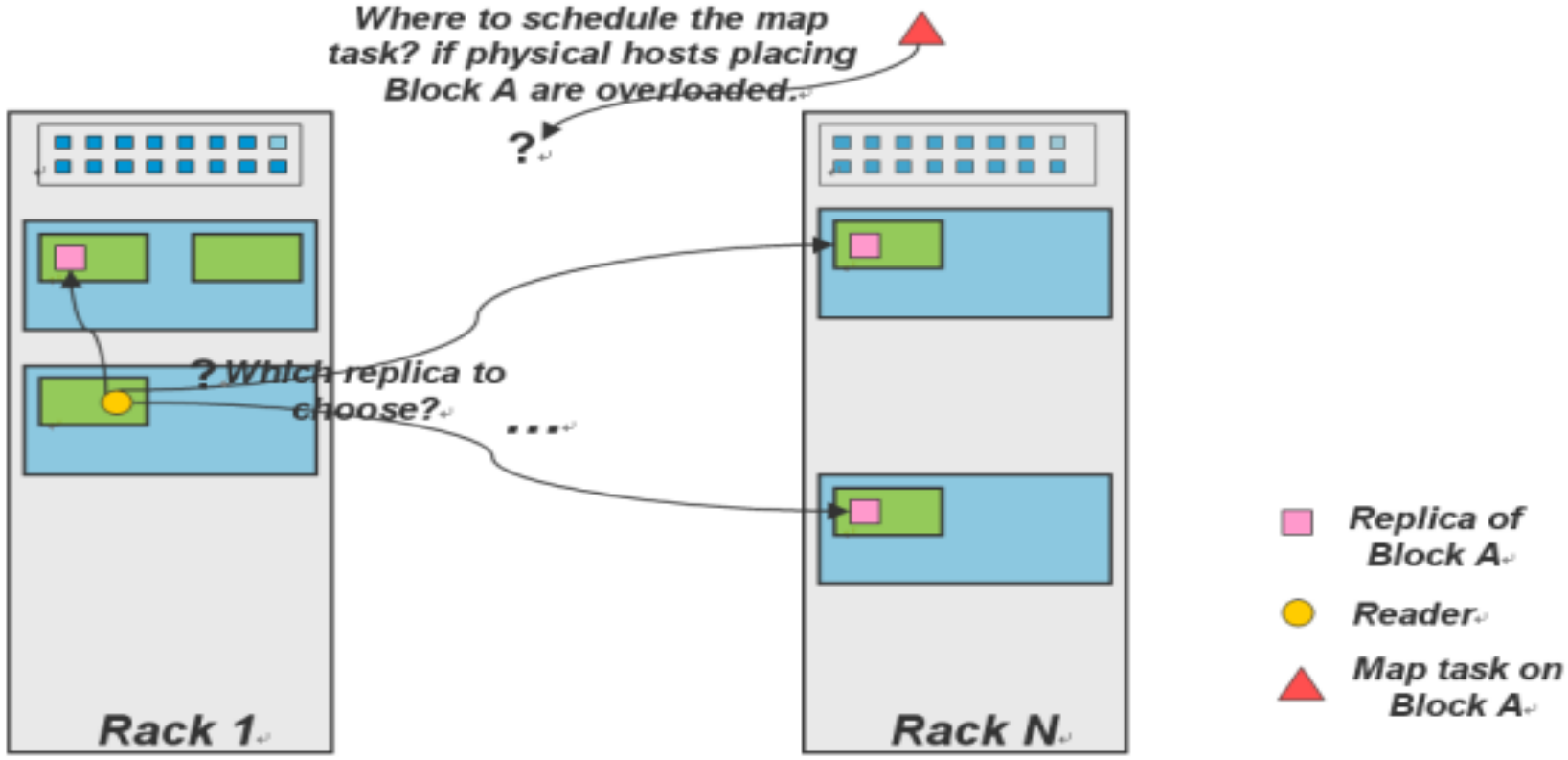
Use a combiner

Optimizations 3: Speculative Execution

- **Problem:** If one node is slow, it slows the entire job
- **Solution: Speculative Execution**
  - Hadoop automatically runs each task multiple times in parallel on different nodes
  - First one finishes, its result is used
  - Others will be killed

Optimizations 4: Locality

- **Locality:** Run map code on same machine with relevant data
  - If not possible, then machine in the same rack
  - Best effort, as no guarantees could be given



Optimizations

- **Optimizations for speed-up or scale-up of map-reduce processing**