Introduction to MapReduce and Hadoop

Jie Tao
Karlsruhe Institute of Technology
jie.tao@kit.edu

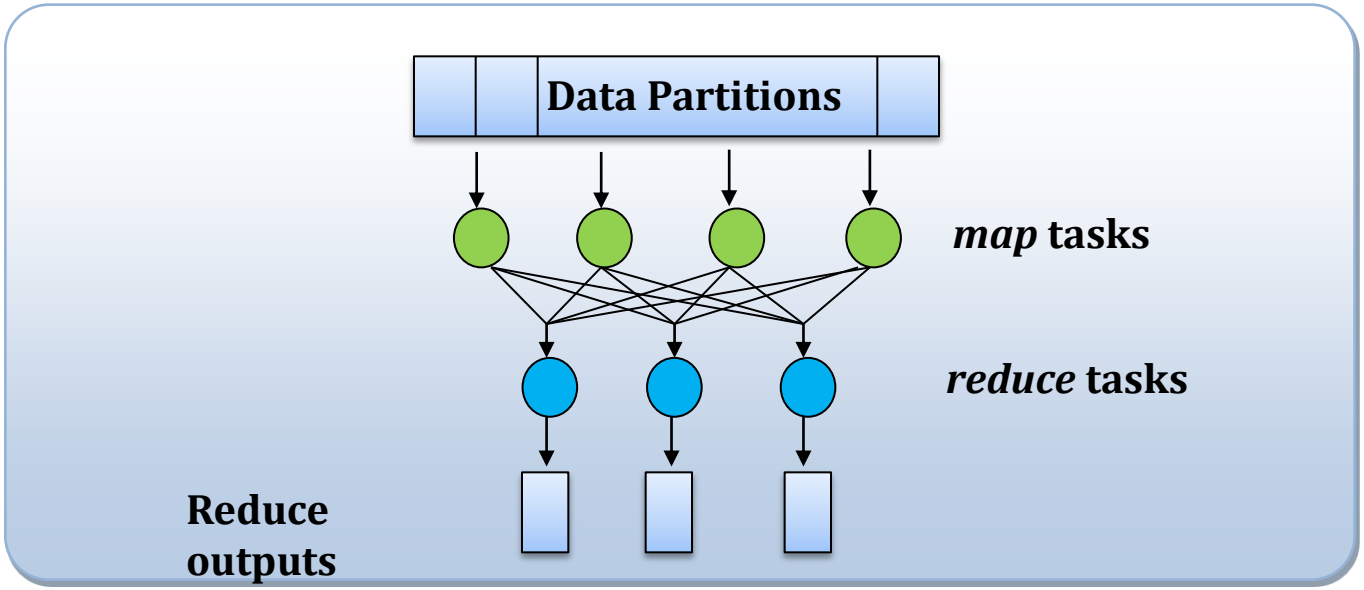
Why Map/Reduce?

- Massive data
  - Can not be stored on a single machine
  - Takes too long to process in serial
- Application developers
  - Never dealt with large amount (petabytes) of data
  - Less experience with parallel programming
- Google proposed a simple programming model – MapReduce – to process large scale data
  - The first MapReduce library on March 2003
- Hadoop: Map/Reduce implementation for clusters
  - Open source
  - Widely adopted by large Web companies

Introduction to MapReduce

- A programming model for processing large scale data
  - Automatic parallelization
  - Friendly to procedural languages
- Data processing is based on a map step and a reduce step
  - Apply a map operation to each logical “record” in the input to generate intermediate values
  - Followed by the reduce operation to merge the same intermediate values
- Sample use cases
  - Web data processing
  - Data mining and machine learning
  - Volume rendering
  - Biological applications (DNA sequence alignments)

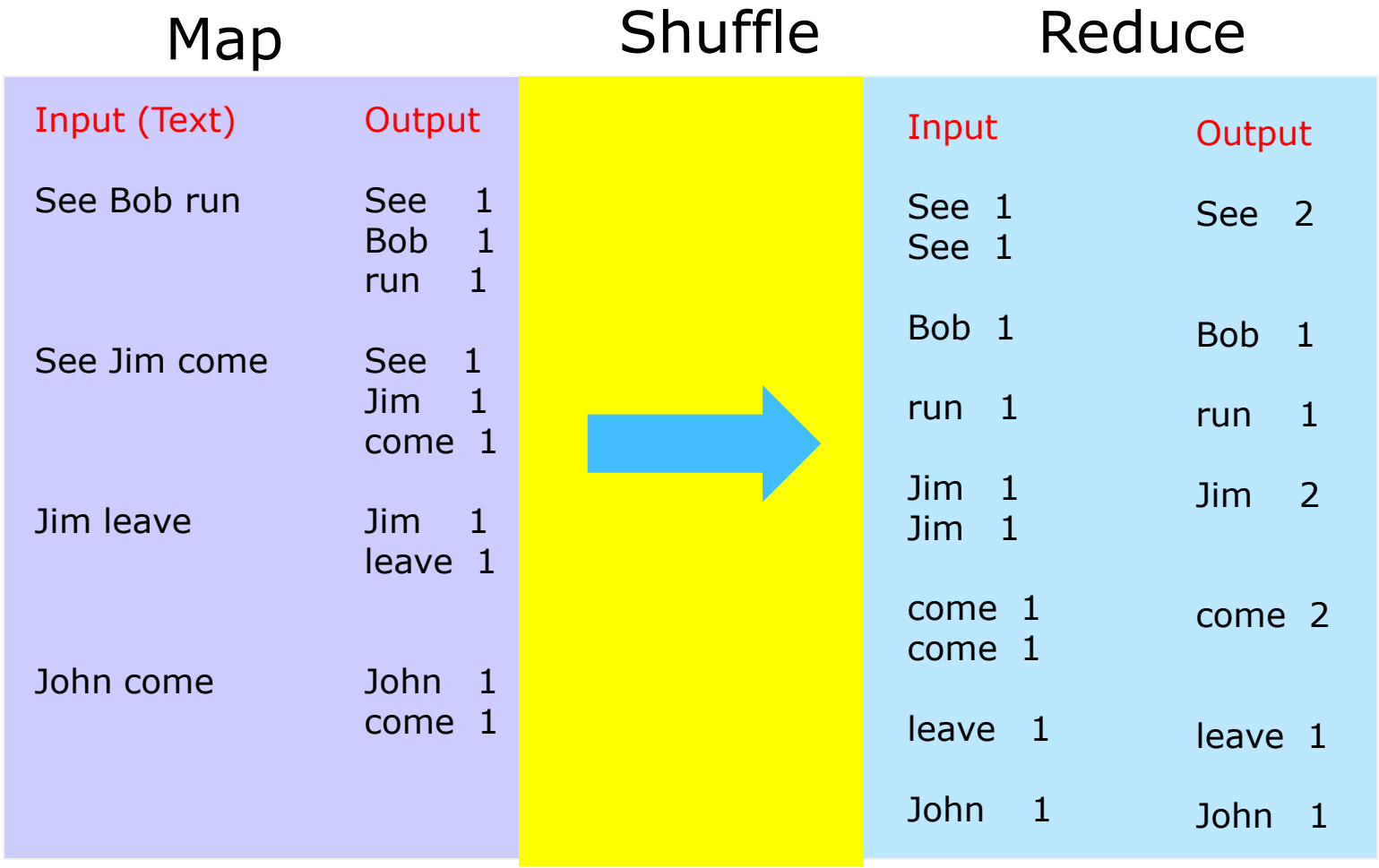
The MapReduce Execution Model



- Splitting the input data
- Sorting the output of the map functions
- Passing the output of map functions to reduce functions

Simple Example: WordCount

- Count the occurrence of each word in a text



Programming WordCount

- Write *map* and *reduce* methods
- Input and output: <*key, value*>

  map: <*k1, v1*> ⟶ <*k2, v2*> (input of WordCount: no *key*)

  reduce: <*k2, v2*> ⟶ <*k3, v3*>
- *maps* transform input records into intermediate records
- *reduces* reduce a set of intermediate values which share a key to a smaller set of values
- Implement *Mapper and Reducer* interfaces
- MapReduce user interfaces
  - Most important: *Mapper, Reducer*
  - Other core interfaces: *JobConf, JobClient, Partitioner, OutputCollector, Reporter, InputFormat, OutputFormat, OutputCommitter*

The Map Class

```
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);
    }
  }
}
```

Job Configuration

- How much data a *map* processes?
  - Use the *InputFormat* in the job configuration

```
public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");
  conf.setMapperClass(Map.class);
  conf.setReducerClass(Reduce.class);
  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);
   ……..
  JobClient.runJob(conf);
}
```

- How many *maps*?
  - Driven by the total size of the inputs and the number of processor nodes
  - Suggestion: a map takes at least a minute to execute

The Reduce Class

All intermediate values associated with an output key are grouped and passed to the Reducer(s)

```
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));
  }
}
```

Control which keys go to which Reducer by implementing the *Partitioner* interface

Run WordCount on Hadoop

- Create the executable

```
$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-core.jar
          -d wordcount_classes WordCount.java
$ jar -cvf /user/xxx/wordcount.jar -C wordcount_classes/ .
```

- Sample input files

```
$ bin/hadoop dfs -ls /user/xxx/wordcount/input/
/user/xxx/wordcount/input/file01
/user/xxx/wordcount/input/file02
```

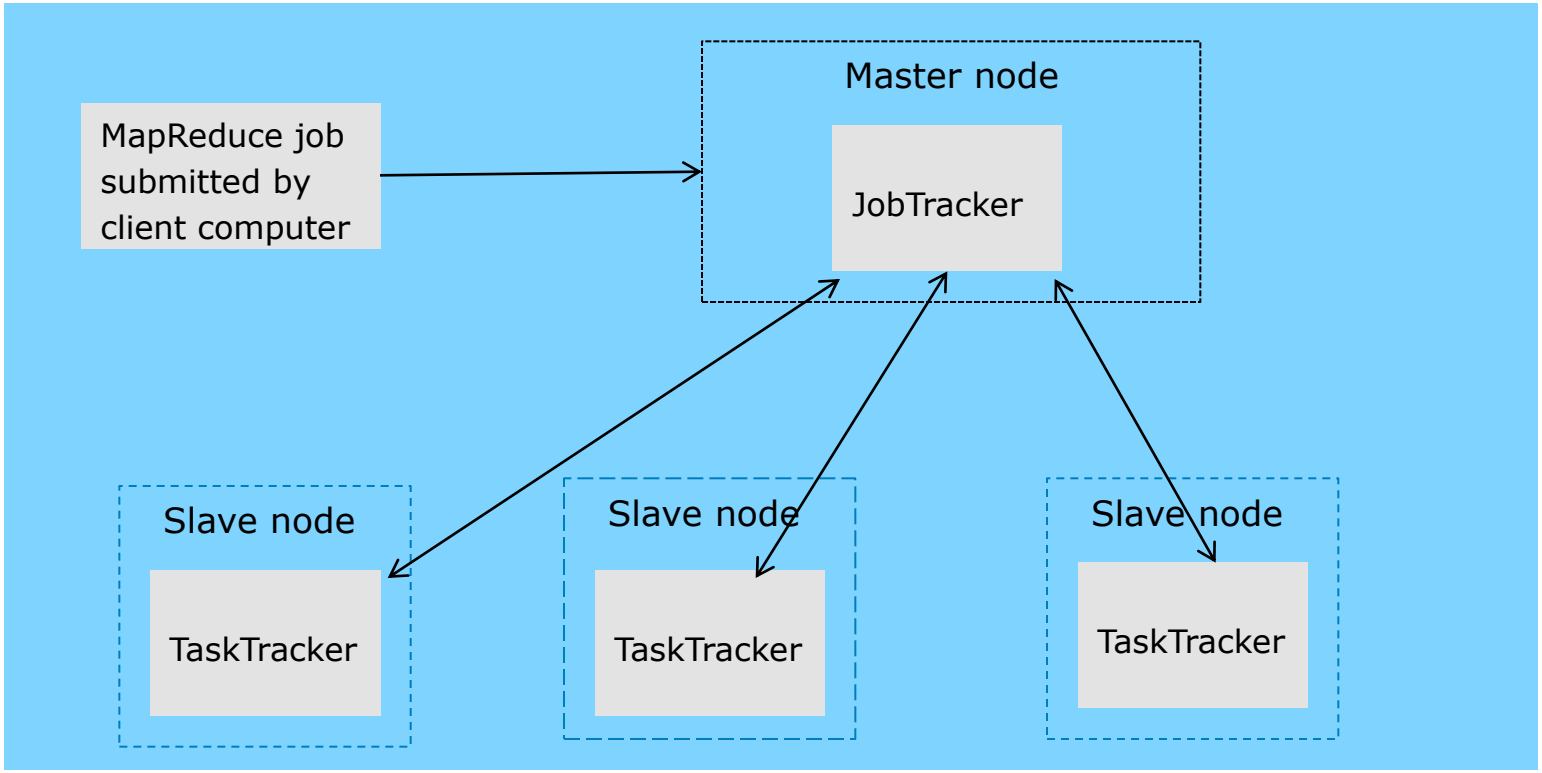
- Run the application

```
$bin/hadoop  jar  /user/xxx/wordcount.jar
          /user/xxx/wordcount/input  /user/xxx/wordcount/output
Output:
$ bin/hadoop dfs -cat /user/xxx/wordcount/output/part-00000
Bob 1
come 2
Jim 2
……
```

Hadoop MapReduce

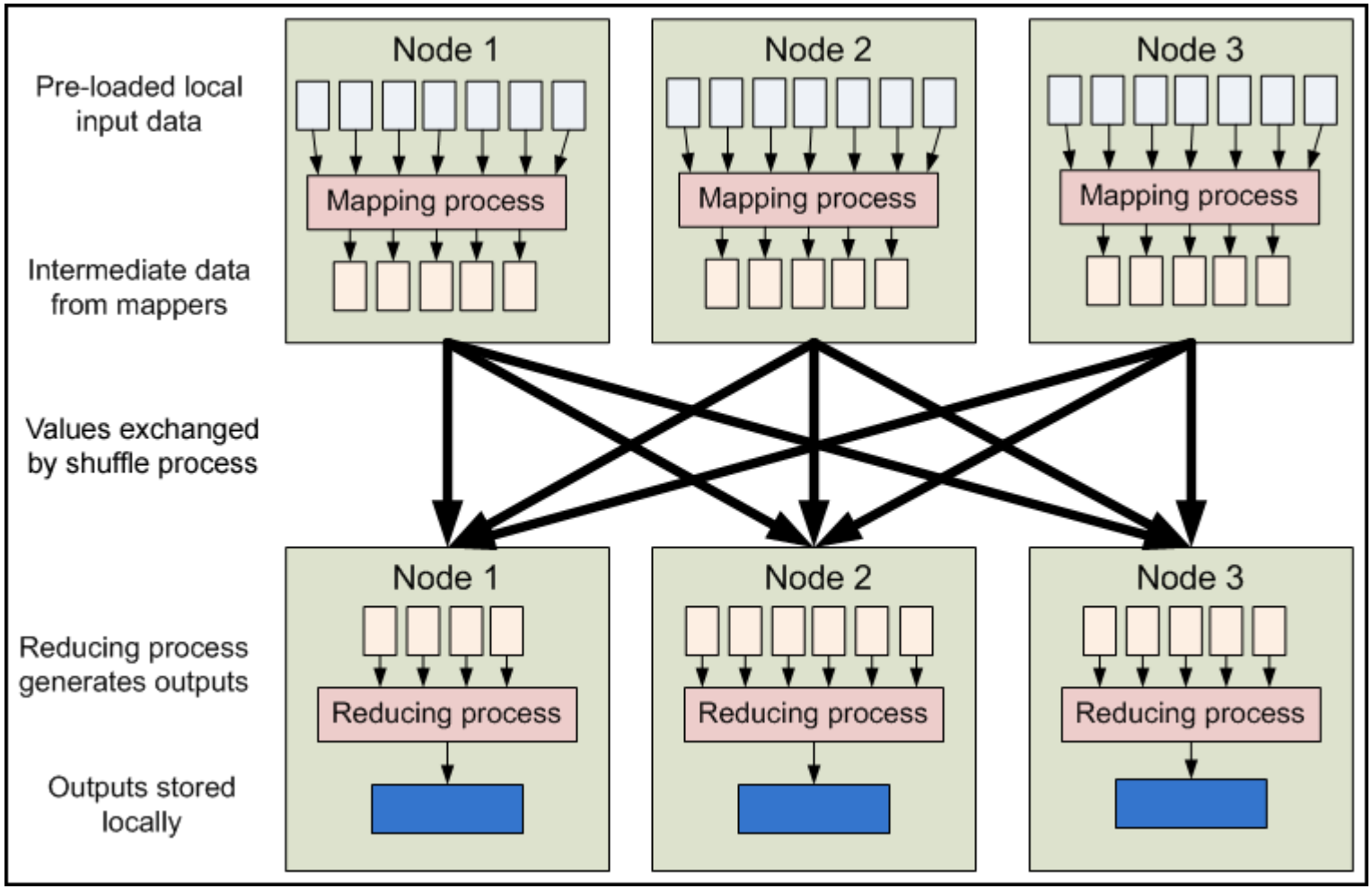
- Data to process do not fit on one node
- Move computation to data
- Distribute filesystem, built in replication, automatic failover in case of failure
- Apache Hadoop provides
  - Automatic parallelization and distribution
  - Fault tolerance
  - Monitoring and status reporting
  - Clean abstraction interface for developers
- Based on the Hadoop Distributed File System (HDFS)

Hadoop MapReduce Architecture



- Single master running a JobTracker instance
- Multiple cluster-nodes running each a TaskTracker instance
- JobTracker distributes Map/Reduce tasks to TaskTrackers

Hadoop MapReduce Data Flow



Hadoop Distributed File System

- A scalable, fault tolerant distributed file system able to run on a cluster of commodity hardware
- A master/slave architecture
  - Single NameNode
  - Multiple DataNodes
- Files are broken into 64 MB or 128 MB Blocks
- Blocks replicated across several DataNodes to handle hardware failure (default replication number is 3)
- Single file system Namespace for the whole cluster
  - NameNode holds the file system metadata (files names, block locations, replication factor, etc)

Hadoop Distributed File System (cont.)

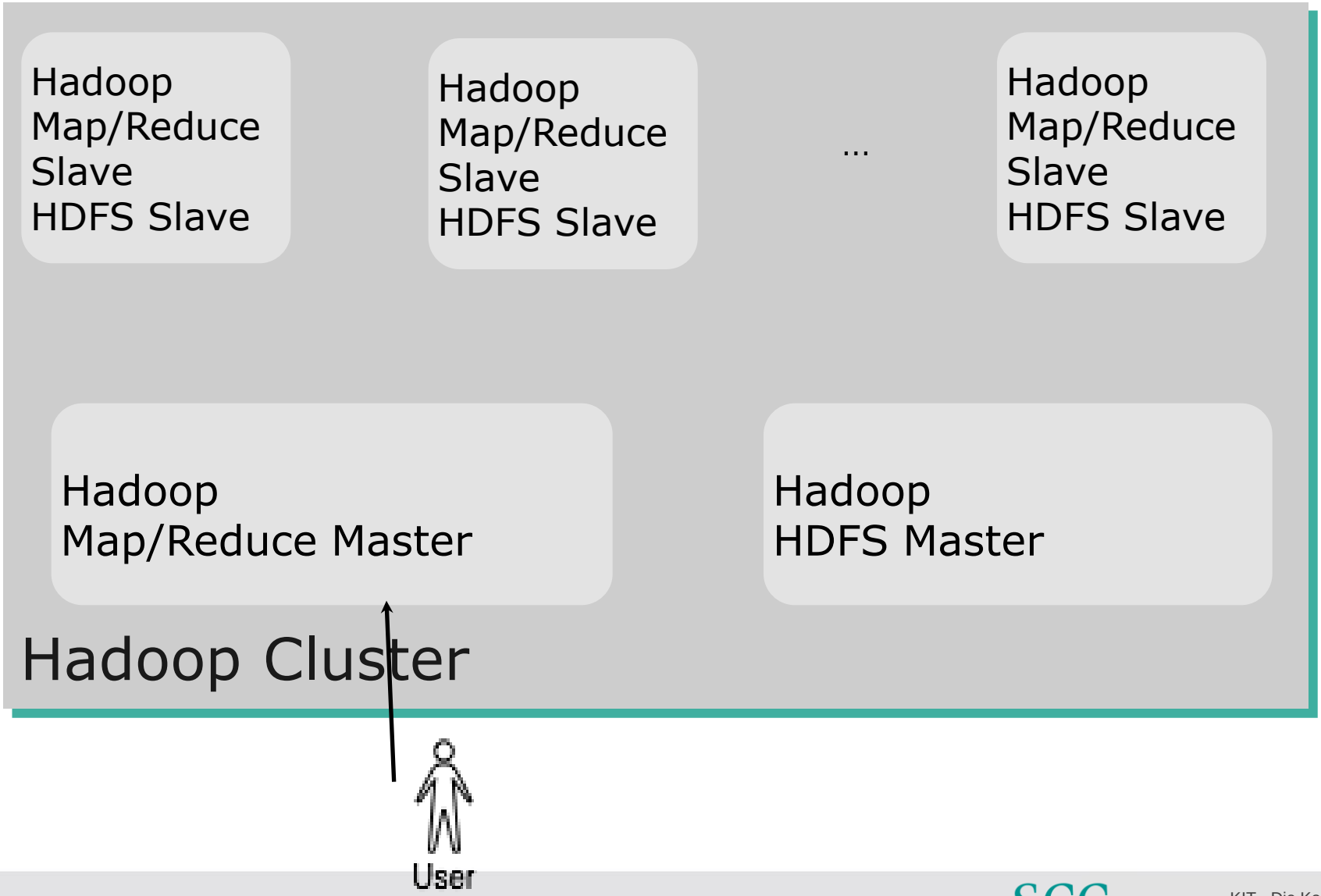
- Data operations via FS shell - the command line interface
  - bin/hadoop dfs -mkdir *dirname*
  - bin/hadoop dfs -rmr *dirname*
  - bin/hadoop dfs -ls *dirname*
  - bin/hadoop dfs -cat *filename*
- Example: copy the input of WordCount to HDFS
  - /bin/hadoop dfs –put file01 file02 /user/xxx/wordcount/input/
  - /bin/hadoop dfs -ls /user/xxx/wordcount/input/

    Found 2 items
    …… /user/xxx/wordcount/input/file01
    …… /user/xxx/wordcount/input/file02
  - /bin/hadoop jar ……. –input /user/xxx/wordcount/input/…..

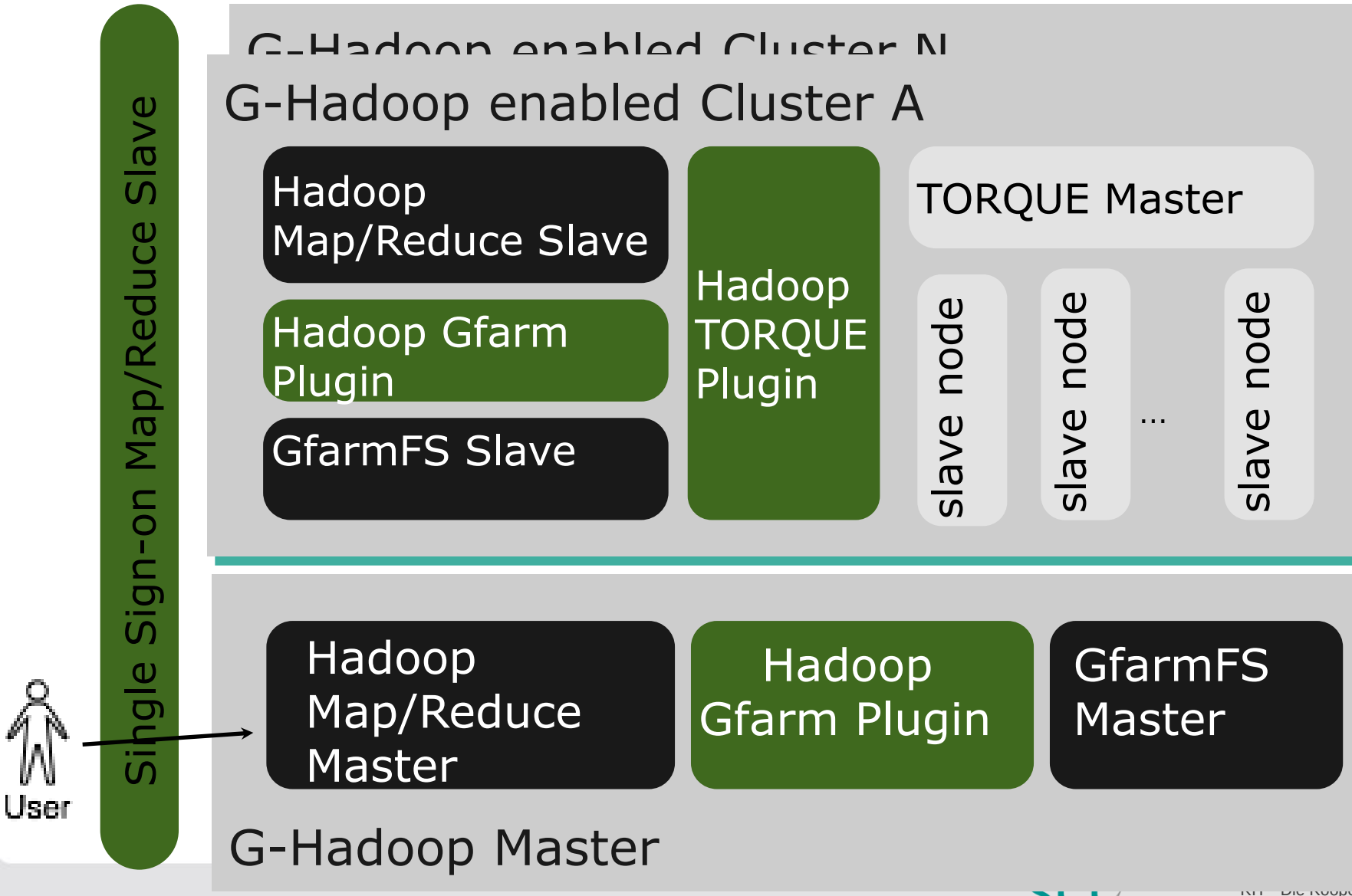
Hadoop on Distributed Clusters

- Motivation
  - Hadoop is limited to a single cluster
  - Existing distributed data centers owns solid cluster scheduling frameworks
  - Requirement on computational capacity from large applications
- Goal: g-Hadoop – runing MapReduce on multiple clusters
- An on-going work at SCC/KIT; collaboration with the Indiana University
- Tasks
  - Replacing HDFS with Gfarm – a Grid Datafarm
  - Integration with the batch systems

g-Hadoop vs. Hadoop: Hadoop Architecture View



g-Hadoop vs. Hadoop: g-Hadoop Architecture



Conclusion

- Map/Reduce is a framework for distributed computing on large data sets
- Hadoop is a widely accepted and used implementation of the Map/Reduce framework, but
  - No load-balancing mechanisms
  - Fault tolerance is based on a simple scheme
- Other implementations of MapReduce on clusters
  - Twister – iterative MapReduce
  - DryadLINQ MapReduce framework
- MapReduce on other architectures
  - Mars - Map/Reduce on NVIDIA Graphics processors
  - Phoenix - Map/Reduce for Shared memory

Links

- Hadoop (also MapReduce and HDFS)

  http://hadoop.apache.org/

- Twister Interactive MapReduce

  http://www.iterativemapreduce.org/

- 1st MapReduce Workshop (by HPDC 2010)

  http://graal.ens-lyon.fr/mapreduce/

- *Dean, Jeff and Ghemawat, Sanjay,* MapReduce: Simplified Data Processing on Large Clusters

  http://labs.google.com/papers/mapreduce-osdi04.pdf