TDDD43 Advanced Data Models and Databases

Hadoop, HDFS, MapReduce

Huanyu Li
huanyu.li@liu.se

LINKÖPING
UNIVERSITY

Outline

- Hadoop and HDFS
- MapReduce
- Lab 5 overview

Hadoop

- Originally designed for computer clusters built based on commodity hardware
- Hadoop can be viewed as a distributed system for data storage and analysis
- The base Hadoop framework contains some modules:
  - Hadoop Common: libraries and utilizes needed by other modules
  - HDFS: Hadoop Distributed File System storing data on commodity machines
  - YARN: resource management and application scheduling platform
  - Hadoop MapReduce: MapReduce programming model for data processing in Hadoop
  - …
- The Hadoop framework itself is mostly written in Java

Focuses for today

- Originally designed for computer clusters built based on commodity hardware
- Hadoop can be viewed as a distributed system for data storage and analysis
- The base Hadoop framework contains some modules:
  - Hadoop Common: libraries and utilizes needed by other modules
  - HDFS: Hadoop Distributed File System storing data on commodity machines
  - YARN: resource management and application scheduling platform
  - Hadoop MapReduce: MapReduce programming model for data processing in Hadoop
  - …
- The Hadoop framework itself is mostly written in Java

HDFS overview

- A file system designed for storing very large files on clusters
- Runs on top of the native file system
  - Files are divided into blocks/shards (128MB by default)
  - 3 replicas per block for fault tolerance

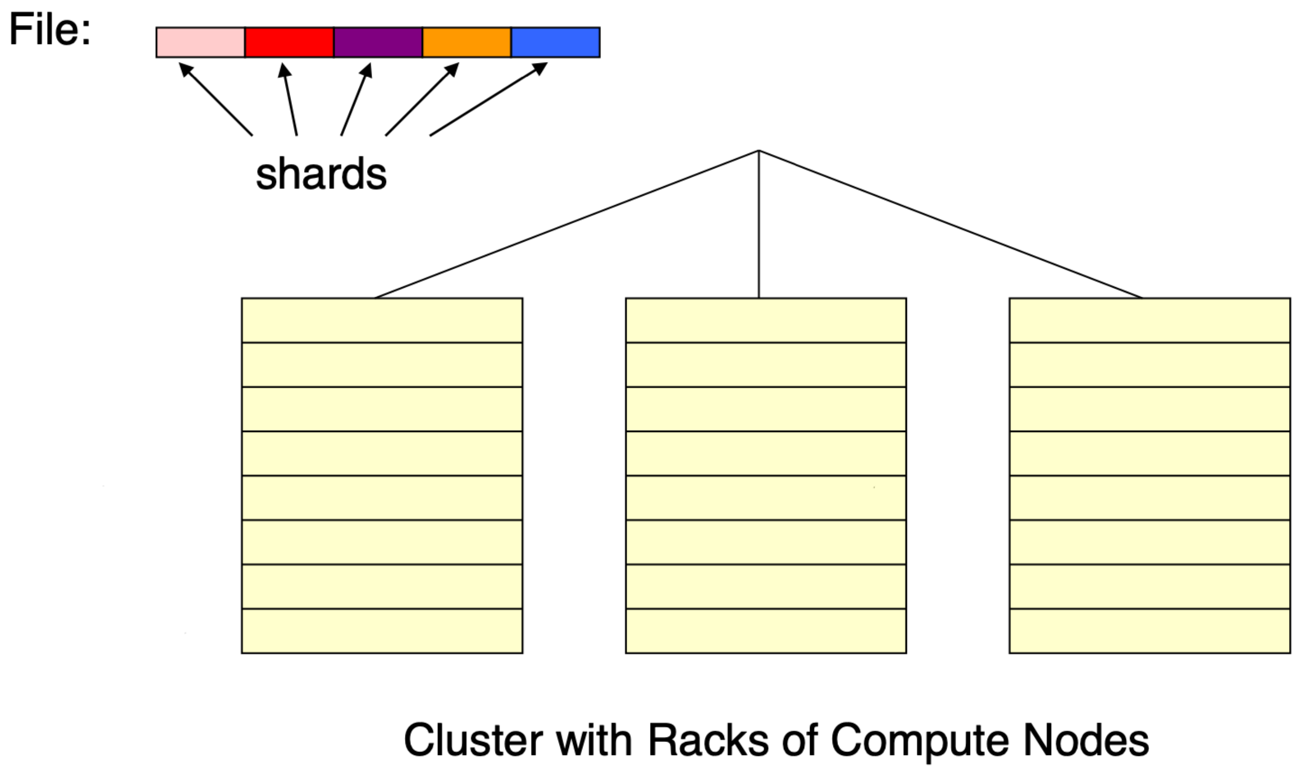
- HDFS files: write once, read multiple times, (single writer, multiple readers)
  - Caching blocks is possible
  - Exposes the locations of file blocks via API

HDFS node organization

- Two types of nodes operating in a master-worker pattern
- Namenode (master)
  - Manages the file system namespace, maintains the file system tree and metadata
  - Stores in memory the locations of all copies of all blocks for each HDFS file
- Datanodes (worker)
  - Workhorses of the file system
  - Store and retrieve blocks when they are told by clients or the namenode
  - Report back to the namenode periodically with lists of blocks that they are storing

HDFS distributing files example

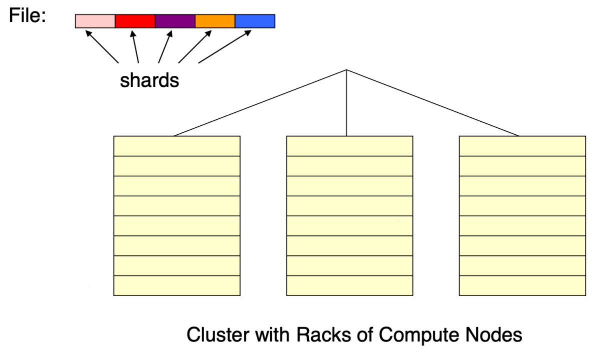
- How to distribute a HDFS file (different blocks) with replicas?



Source: J. D. Ullman invited talk EDBT 2011

HDFS Block Placement and Replication

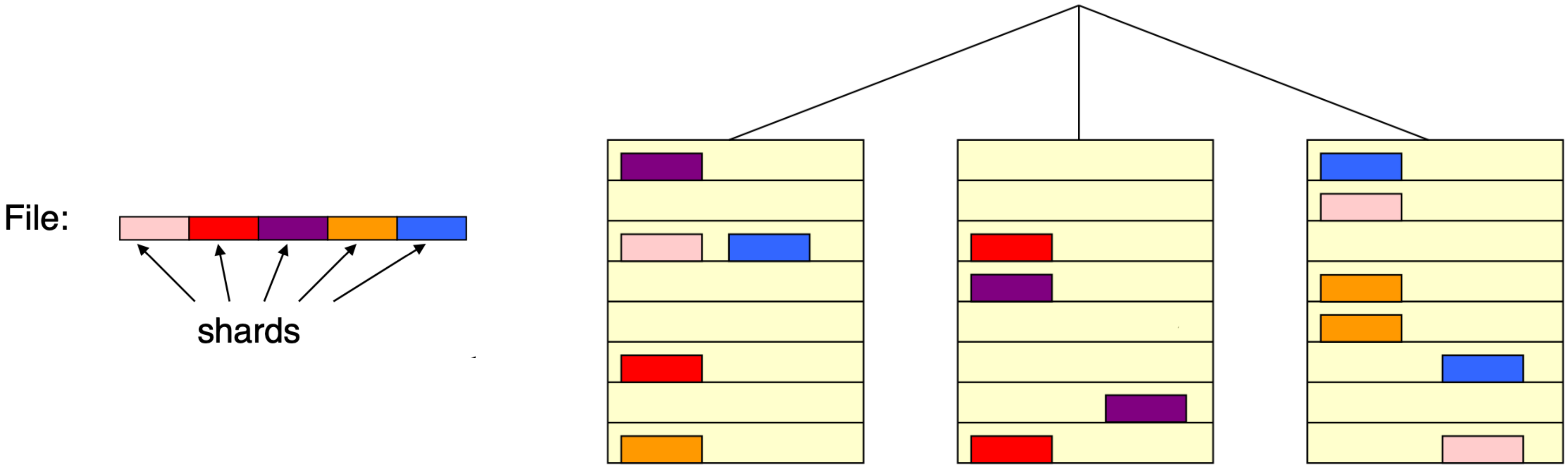
- Aims: to improve data reliability, availability, and network bandwidth utilization
- Default policy (3 replicas):
  - No datanode contains more than one replica
  - No rack contains more than two replicas of the same block
- Name-node ensures that the number of replicas is reached
- Balancer tool – balances the disk space usage
- Block scanner – periodically verifies checksums



Source: J. D. Ullman invited talk EDBT 2011

HDFS Block Placement and Replication

- The first replica is located on the writer node
- The second and third replica are on different nodes in a different rack
- Any other replicas (if any, more than 3) are located on random nodes



HDFS Pros and Cons

- Pros
  - Storing very large files, GBs/TBs
  - High-throughput parallel I/O
    - Write-once, read many times
    - Time to read the entire dataset is more important than the latency in reading the first record
  - Commodity hardware
    - Clusters are built from commonly available hardware
    - Designed to continue working without a noticeable interruption in case of failure

HDFS Pros and Cons

- Cons
  - Not suitable for low-latency data access
    - HDFS is optimized for delivering high throughput of data
  - Lots of small files
    - The amount of files is limited by the memory of the namenode
  - Not suitable for re-writing the same HDFS, and arbitrary file modifications
    - HDFS files are append only, write is only allowed at the end of the file

HDFS commands

- hadoop fs -mkdir <FOLDER_NAME>
  - -make a folder on HDFS

- hadoop fs -mkdir -p <FOLDER_NAME> <FOLDER_NAME>
  - -make multiple folders

- hadoop fs -test -d <FOLDER_NAME>
  - -if the path is a directory, return 0

- hadoop fs -rm -r <FOLDER_NAME>
  - -deletes the directory and any content under it recursively

HDFS commands

- hadoop fs -cat <FOLDER_ON_HDFS> [local]
  - -copy HDFS path to stdout

- hadoop fs -copyFromLocal <localsrc> ... <dst>
  - -copy single src, or multiple srcs from local Sigma to HDFS
  -
- hadoop fs -copyToLocal <dst> ... <localsrc> -
  - copy single src, or multiple srcs from HDFS to local Sigma

Next question…

- How to perform computations over data stored on HDFS?
- How to write distributed programs

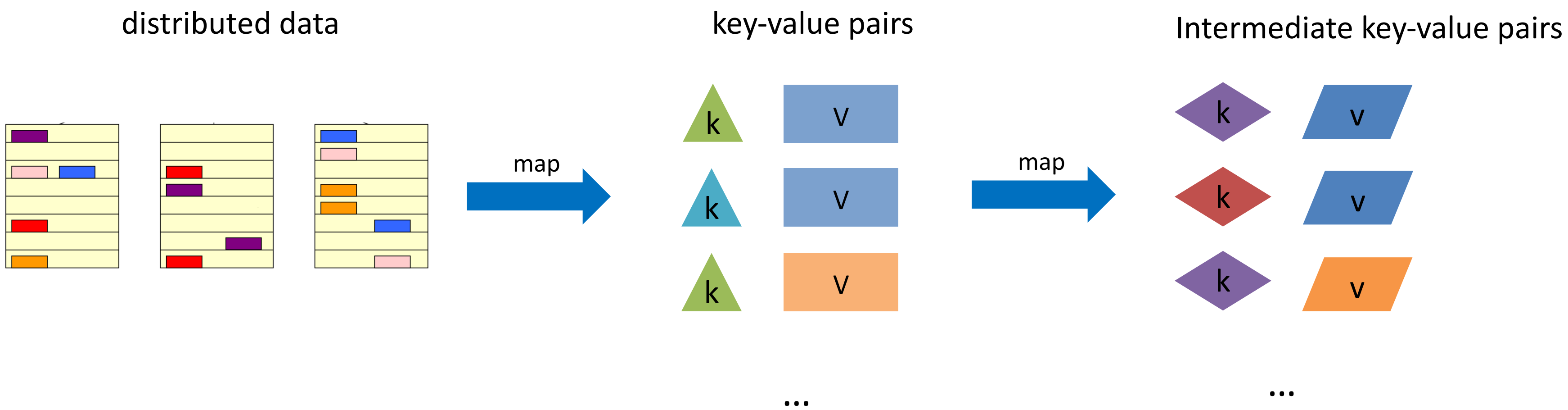
- MapReduce programming model

MapReduce

- Operate on large distributed input data sets, e.g., HDFS
- Implemented in Hadoop and other frameworks
- A high-level parallel programming construct
- Algorithmic design pattern:
  - Map, Shuffle (group by key), Reduce

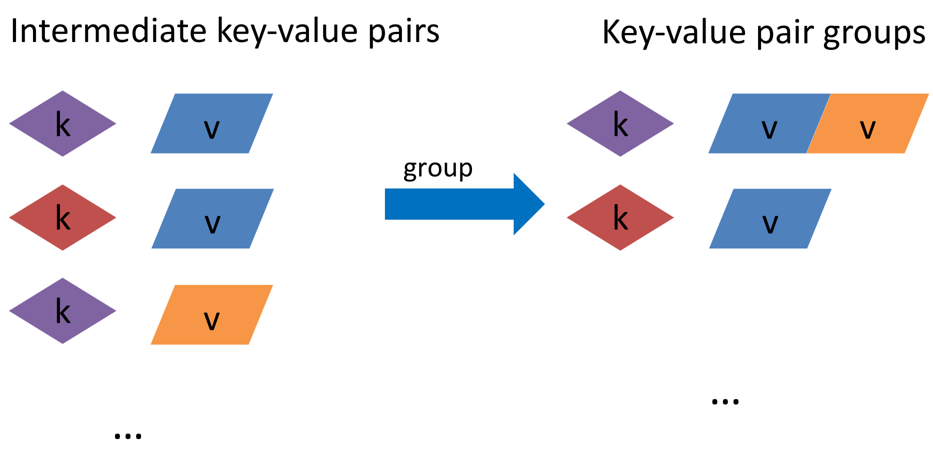
MapReduce

- Map Phase (extract data you care about)
  - Parses an input file into key-value pairs (Record reader)
  - Performs a user-defined function to each element, then produce new key-value pairs as intermediate elements (Mapper)



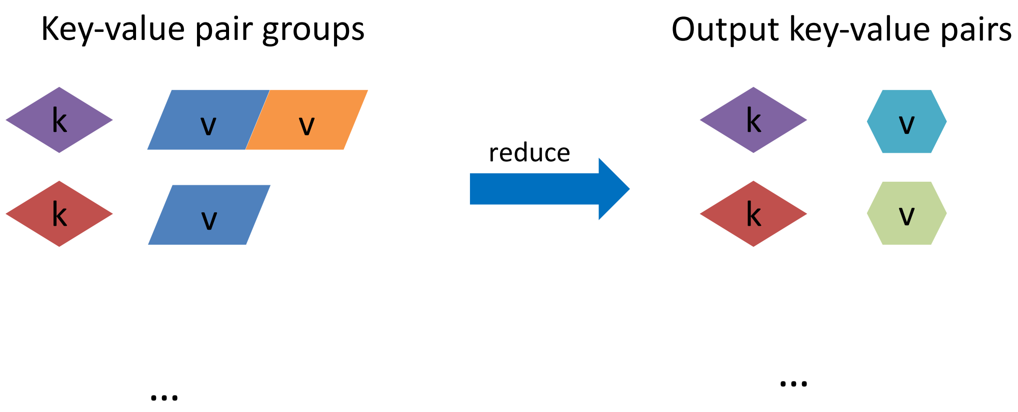
MapReduce

- Shuffle phase
  - Downloads the needed files to the node where the reducer is running
  - Pairs with the same keys will be grouped



MapReduce

- Reduce phase
  - Performs a user-defined reduce function once for each key grouping
  - Outputs key-value pairs
  - E.g., aggregation (maximum, minimum, sum), filter

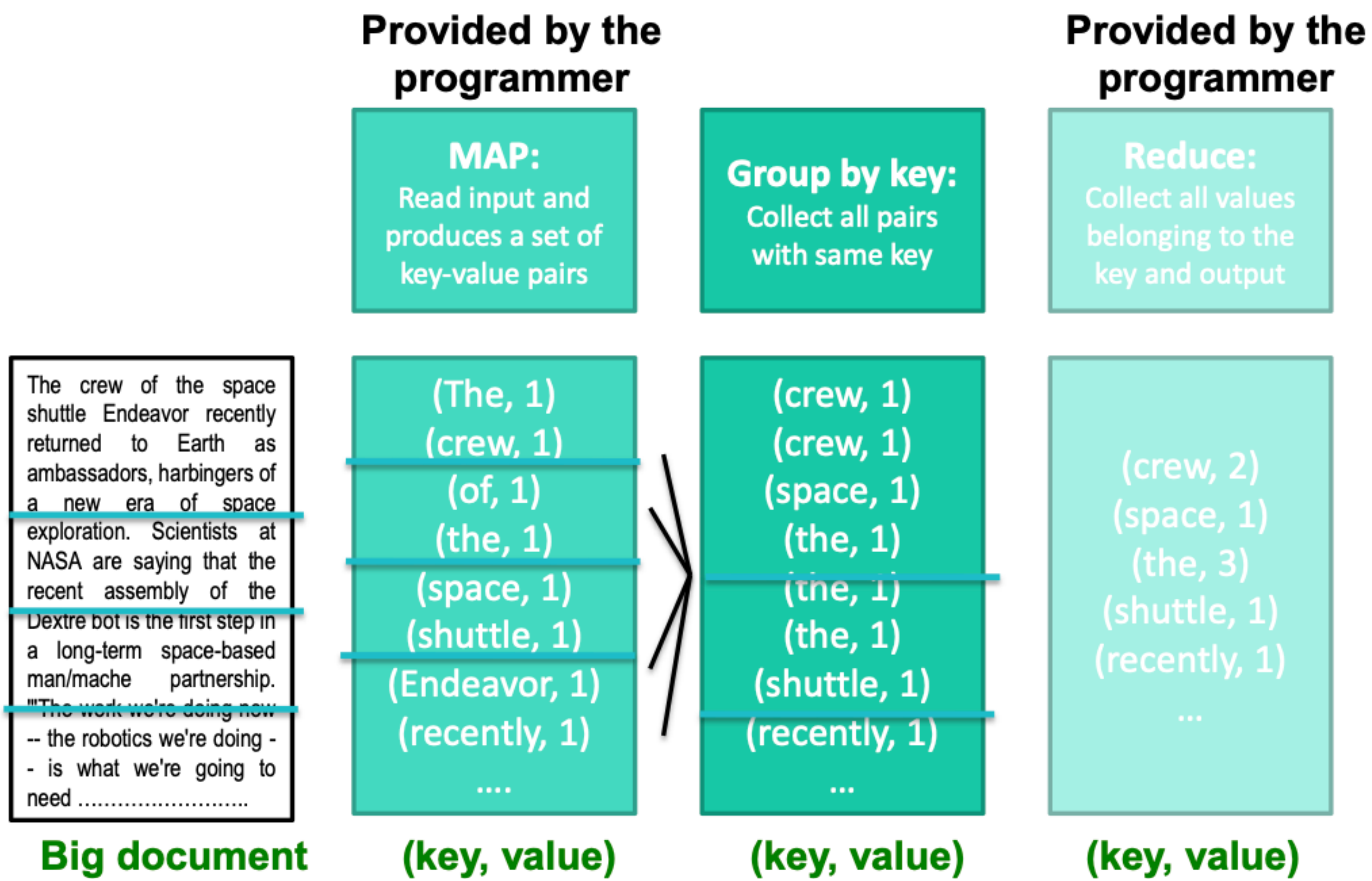


Word Count Example

- We have a huge text document
- Count the number of times each distinct word appears in the file

- Example application: analyze web server logs to find popular URLs

Word Count Example



Word Count Example

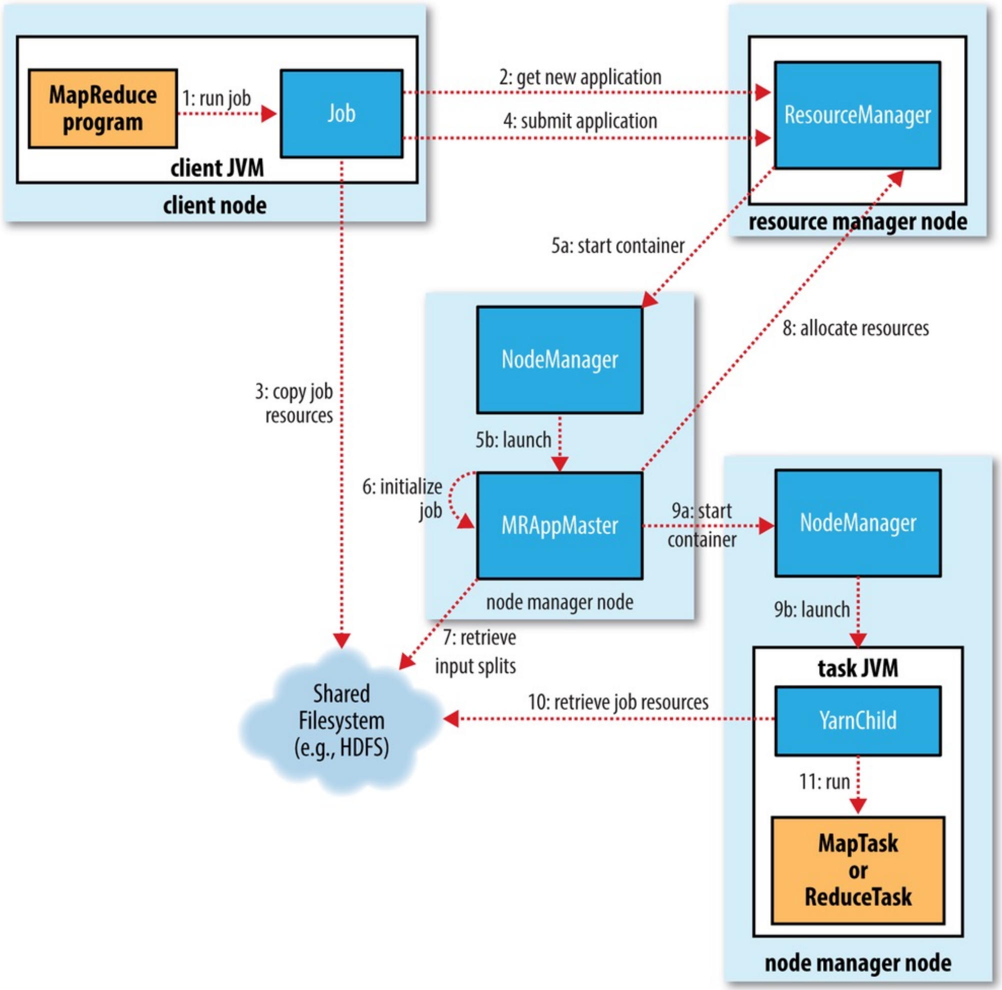
```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
      emit(w, 1)
reduce(key, values):
// key: a word; value: an iterator over counts
   result = 0
   for each count v in values:
      result += v
   emit(key, result)
```

MapReduce environment in Hadoop

- MapReduce environment takes care of:
  - Partitioning the input data
  - Scheduling the program's execution across a set of machines
  - Performing the shuffle (group by key)
  - Handling machine failures
  - Managing required inter-machine communication

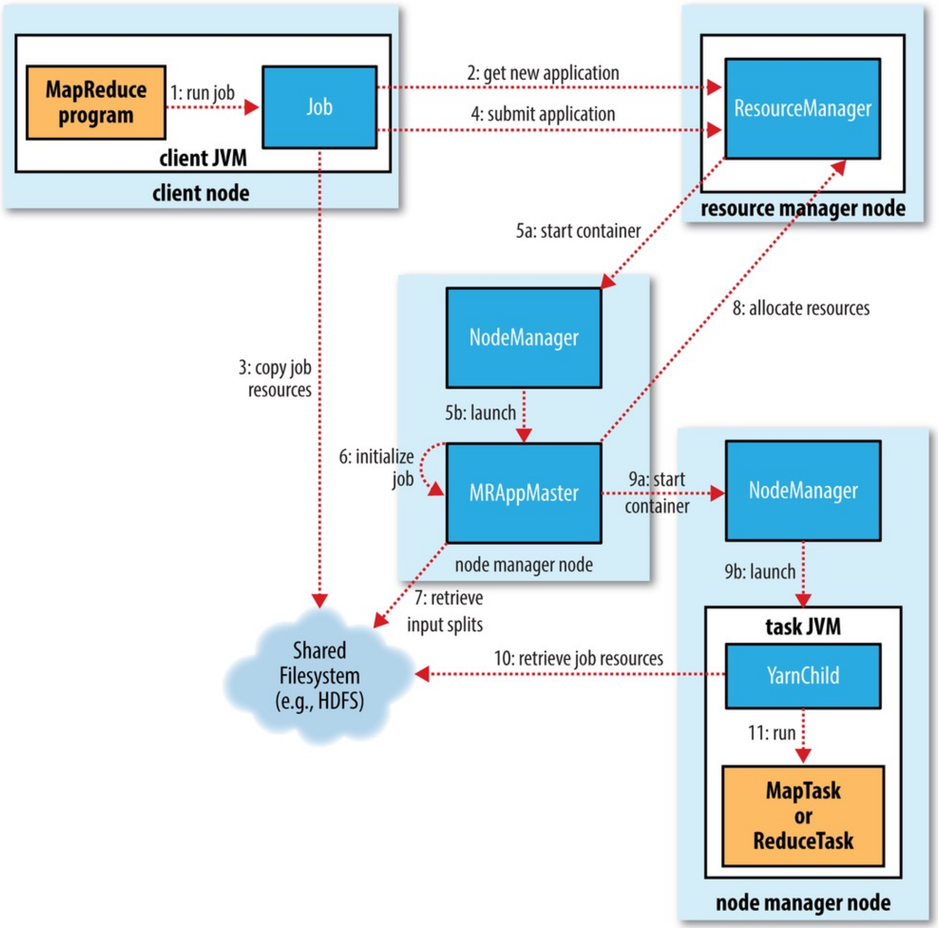
Hadoop MapReduce Execution Flow

- Job Submission
  - Step 1-4
- Job Initialization
  - Steps 5a, 5b, 6, 7
- Task Assignment
  - Step 8
- Task Execution
  - Steps 9a, 9b, 10, 11



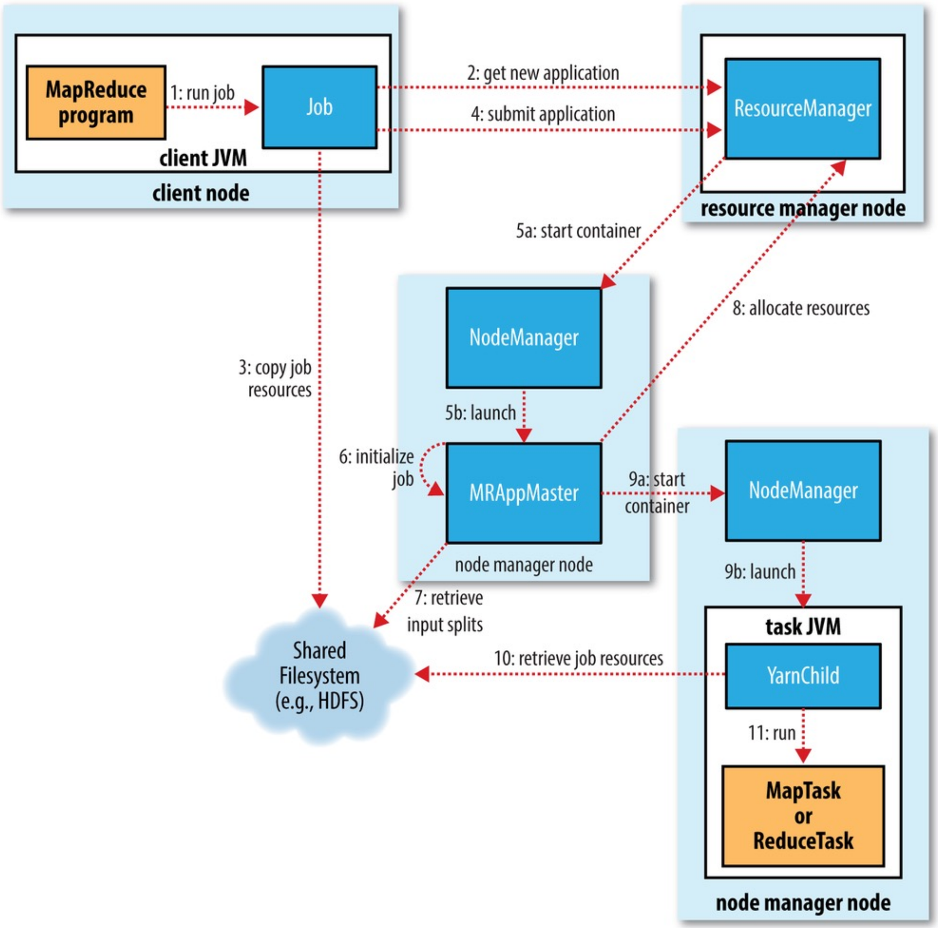
Hadoop MapReduce Execution Flow

- Job Submission
  - Step 1
    - Run job on the client using “hadoop jar PARAMETERS”, and give the jar file name, class name, and other parameters such as input files
  - Step 2
    - Ask the resource manager an application ID
  - Step 3
    - Copy jar file, and other resources to HDFS
  - Step 4
    - Then the client actually submit the application to resource manager



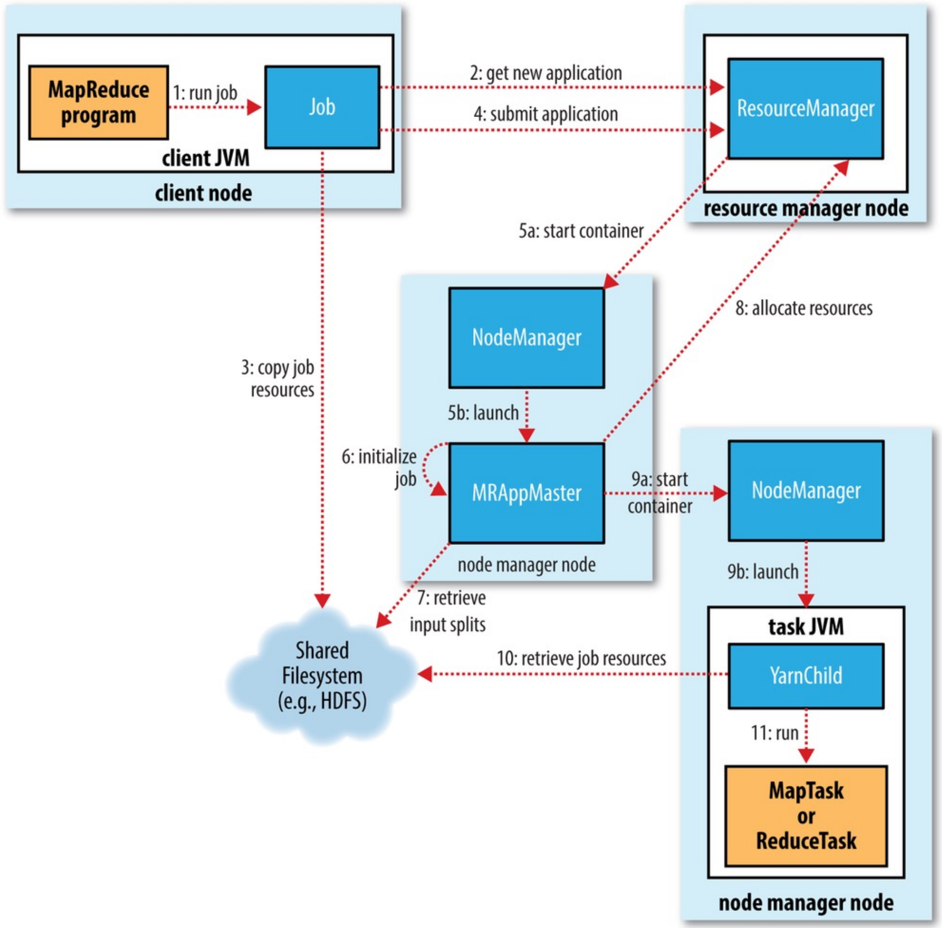
Hadoop MapReduce Execution Flow

- Job Initialization
  - Step 5a, 5b
    - The YARN scheduler allocates a container, and then the resource manager launches the application mater’s process
  - Step 6
    - Create bookkeeping objects to keep job’s progress
  - Step 7
    - Retrieve each input split and create a map task for each split, as well as create a number of reduce tasks



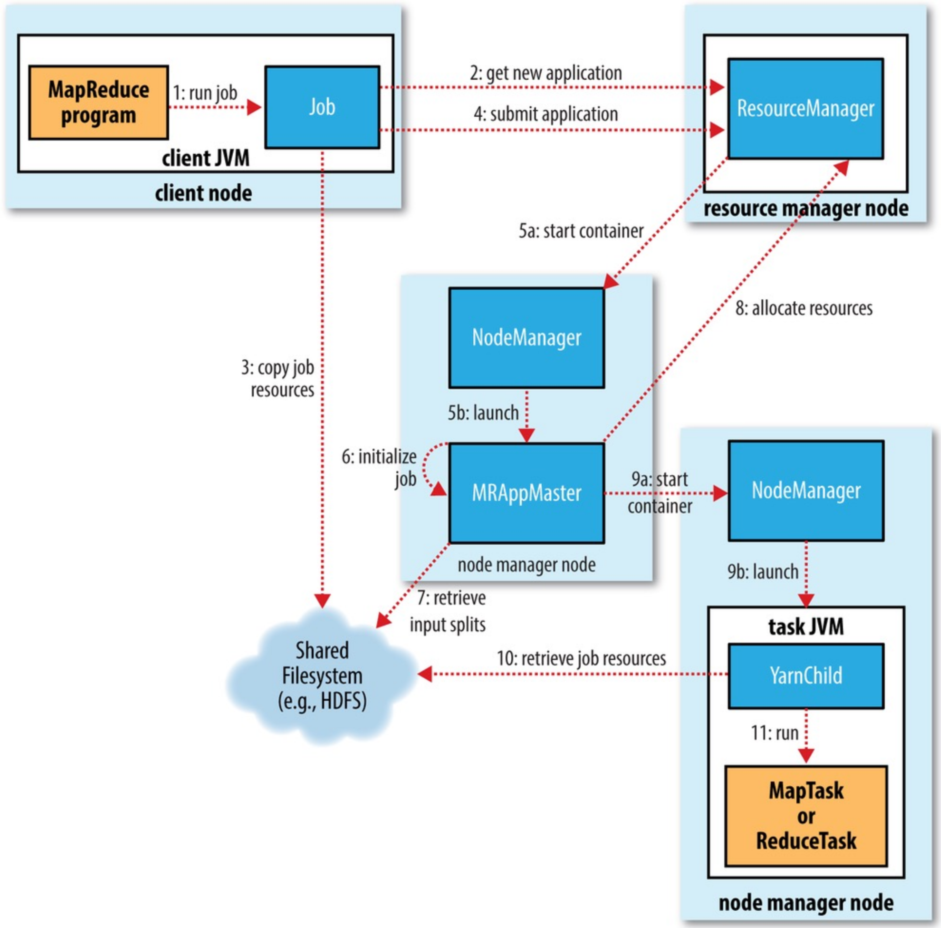
Hadoop MapReduce Execution Flow

- Task Assignment
  - Step 8
    - Application requests resources for map and reduce tasks



Hadoop MapReduce Execution Flow

- Task Execution
  - Step 9a, 9b
    - Application master starts containers by contacting node managers
  - Step 10
    - Retrieve needed resources and localize them
  - Step 11
    - Run tasks



MapReduce Application

- Matrix-Vector Multiplication
  - Suppose we have an ***n*n*** matrix ***M***, whose element in row ***i*** and column ***j*** is denoted m_{ij}
  - Suppose we have a vector ***v*** of length ***n***, whose ***j***th element is v_j
  - Then the matrix-vector product is the the vector ***x*** of length ***n***, whose ***i***th element x_i is given by

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} AP + BQ + CR \\ DP + EQ + FR \\ GP + HQ + IR \end{bmatrix}$$

MapReduce Application

- MapReduce for Matrix-Vector Multiplication
  - The matrix ***M*** and the vector ***v*** each will be stored in a file on HDFS. We assume that the row-column coordinates of each matrix element will be discoverable, either from its position in the file or stored with explicit coordinates, e.g., ($\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{m_{ij}}$)
  - We also assume the position of elements in the vector is discoverable in the similar way

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} AP + BQ + CR \\ DP + EQ + FR \\ GP + HQ + IR \end{bmatrix}$$

MapReduce Application

- Matrix-Vector Multiplication
  - The Map function
    - Each map task takes the entire vector ***v*** and a chunk of the matrix
    - For each matrix element m_{ij}, it produces the key-value pair (i, $m_{ij}v_j$), Thus, all terms of the sum making up the component x_i will get the same key
  - The Reduce function
    - A reduce task simply sum up all the values values associated with a given key i. The result will be a pair (i, x_i)

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

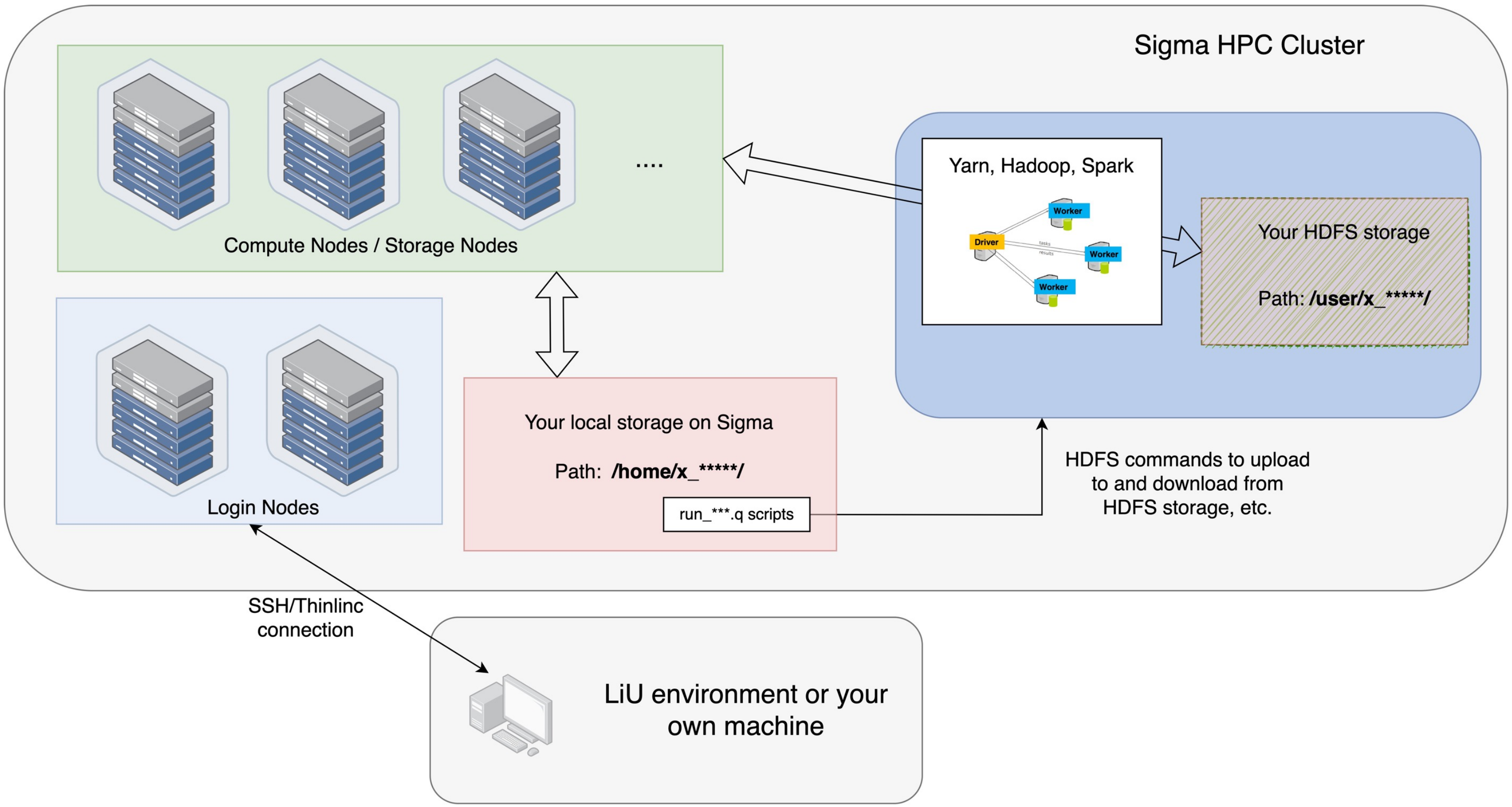
$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} AP + BQ + CR \\ DP + EQ + FR \\ GP + HQ + IR \end{bmatrix}$$

Outline

- ✓ Hadoop
- ✓ HDFS
- ✓ MapReduce
- ➢ Lab 5 overview

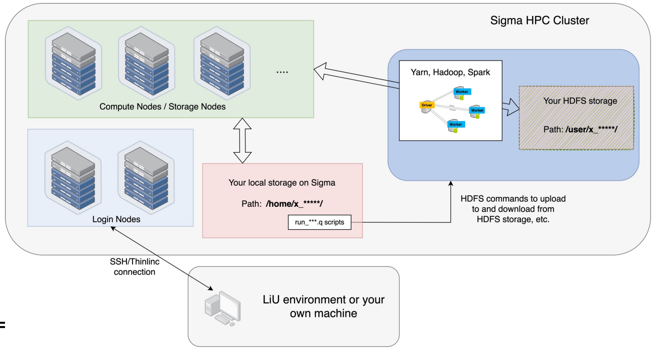
Lab 5 overview

- How to work on Sigma



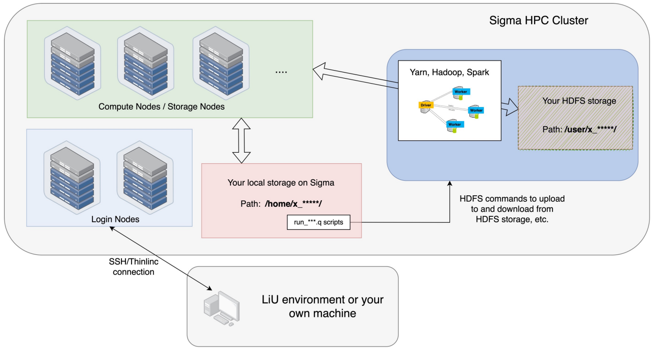
Lab 5 overview

- Connection
  - Option 1: ssh –X username@sigma.nsc.liu.se
    - 1.1: Connect the University environment first. You can use thinlinc to connect ‘thinlinc.edu.liu.se’, then use ‘ssh -X’ to connect sigma.
    - 1.2: With your own laptop, you need a X forwarding configuration.
      - An X server software installed on your computer.
        - If you run Linux, this is already taken care of.
        - If you run MacOS, you might need to install and start X11.app (XQuartz: https://www.xquartz.org) which is included in MacOS but not always installed.
        - If you run Windows, you need to find a third-party X server software (e.g Xming https://sourceforge.net/projects/xming/), as this is not normally included in Windows.



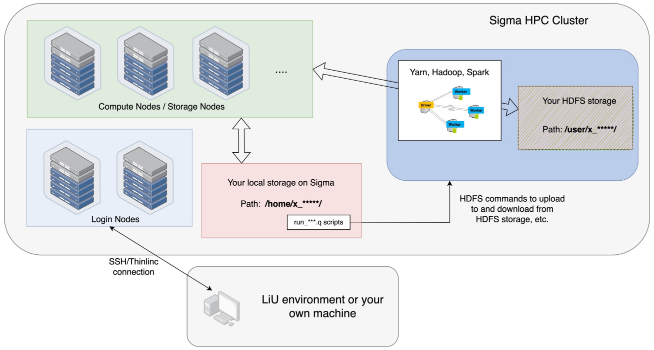
Lab 5 overview

- Connection
  - Option 2: Thinlinc connection to Sigma directly (sigma.nsc.liu.se)
    - If you are at a computer in an SU room at the university, to use thinlinc, you need to run following two commands first in a terminal:
      - ***module load course/TDDD43***
      - ***tlclient***
    - If you use your own computers, you just need to download thinlinc and then connect Sigma
    - Notice: During the lab sessions, for each group, please just use at most one thinlinc connection.



Lab 5 overview

- Submit, monitor, cancel jobs at Sigma
  sbatch, squeue, scancel commands
- Demo on sigma
  /software/sse/manual/spark/examples/java_mapreduce_on_hdfs/2_java_wordcount_1.0/
- A script for compiling java code using Hadoop
  complile.sh
- A script for interacting with HDFS and running code
  run.q



How to work on Sigma (run word count example)

- Step 1: Login Sigma
- Step 2: Copy the code to your home folder on Sigma

```
huali50 — x_huali@sigma:~/lab-test-2023/2_java_wordcount_1.0 — ssh x_huali@sigma.nsc.liu.se — 90×71
(base) huali50@mac01048 ~ % ssh x_huali@sigma.nsc.liu.se
(x_huali@sigma.nsc.liu.se) Password:
(x_huali@sigma.nsc.liu.se) Verification code:
Last login: Thu May 11 10:09:18 2023 from 2001:6b0:17:fc08:b48d:1605:1e5c:b90b
Welcome to NSC and Sigma!

**** Project storage directories available to you:
/proj/liu-compute-2023-24/users/x_huali
/proj/theophys/users/x_huali

**** Documentation and getting help:
https://www.nsc.liu.se/support/systems/sigma-getting-started/
https://www.nsc.liu.se/support

**** Useful commands
To see your active projects and CPU time usage: projinfo
To see available disk storage and usage: snicquota
To see your last jobs: lastjobs
Login to compute node to check running job: jobsh

To tweak job priorities, extend timelimits and reserve nodes: see
https://www.nsc.liu.se/support/batch-jobs/boost-tools/

(Run "nsc-mute-login" to not show this information)

[x_huali@sigma ~]$ mkdir lab-test-2023
[x_huali@sigma ~]$ cp -r /software/sse/manual/spark/
BDA_demo/                       spark-2.4.3-bin-hadoop2.7.tgz
examples/                       spark-2.4.3-hadoop2.7-nsc1/
hadoop-3.1.2.tar.gz             tools/
old/
[x_huali@sigma ~]$ cp -r /software/sse/manual/spark/examples/java_mapreduce_on_hdfs/2_java
_wordcount_1.0/ ./lab-test-2023/
[x_huali@sigma ~]$ cd lab-test-2023/
[x_huali@sigma lab-test-2023]$ ls
2_java_wordcount_1.0
[x_huali@sigma lab-test-2023]$ cd 2_java_wordcount_1.0/
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount.java  complile.sh  input  run.q
```

Step 1

Step 2

How to work on Sigma (run word count example)

- Step 3: Compile java program
- Step 4: Submit a job on Sigma
- Step 5: Check the output slurm file

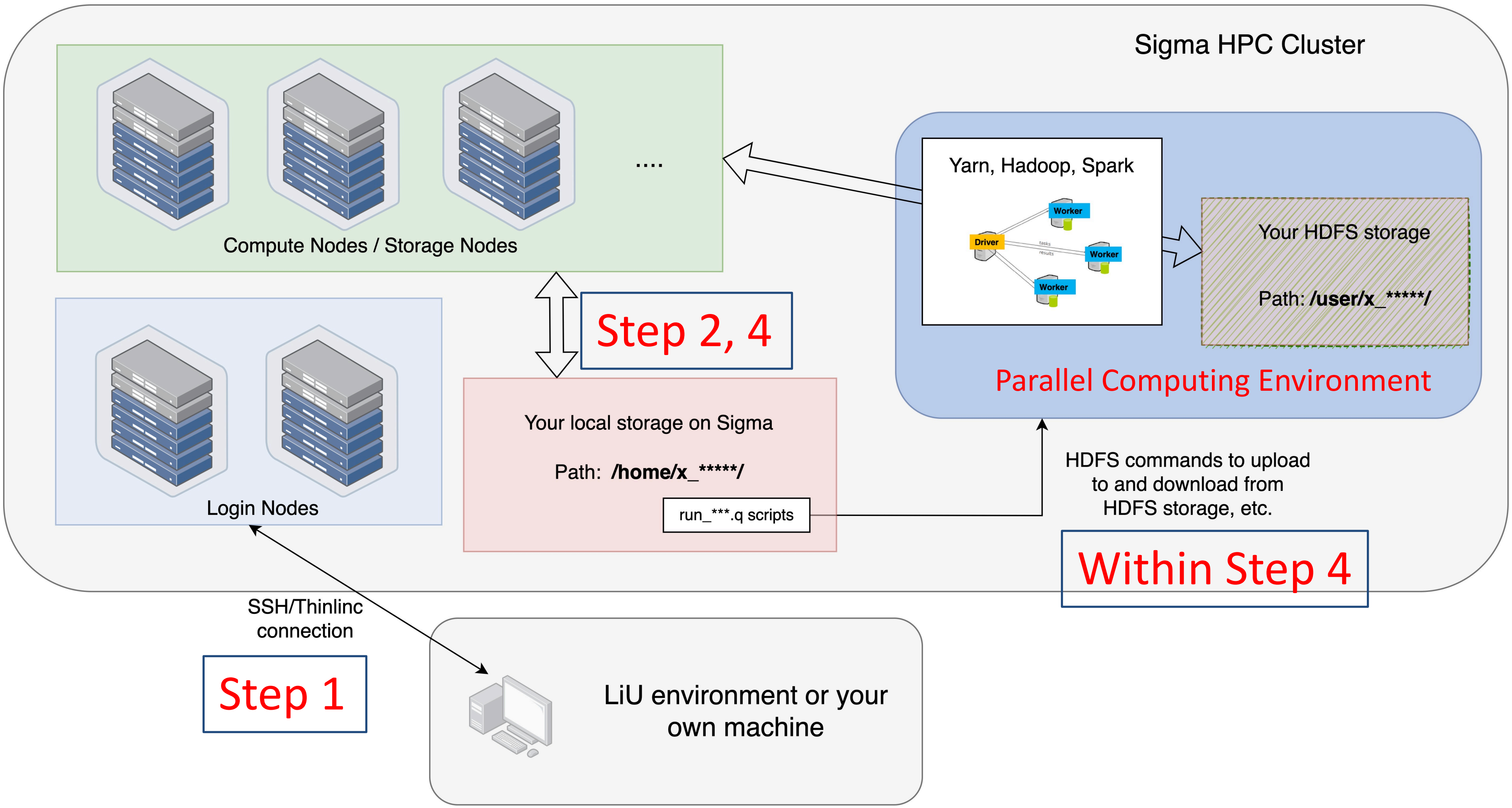
```
[x_huali@sigma ~]$ cd lab-test-2023/
[x_huali@sigma lab-test-2023]$ ls
2_java_wordcount_1.0
[x_huali@sigma lab-test-2023]$ cd 2_java_wordcount_1.0/
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount.java  compile.sh  input  run.q
[x_huali@sigma 2_java_wordcount_1.0]$ ./compile.sh
Compiling Wordcount Program...
WARNING: log4j.properties is not found. HADOOP_CONF_DIR may be incomplete.
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount$IntSumReducer.class    WordCount.class  compile.sh  run.q
WordCount$TokenizerMapper.class  WordCount.java   input       wordcount.jar
[x_huali@sigma 2_java_wordcount_1.0]$ listreservations
Reservations available to user:x_huali / project(s):liu-compute-2023-24,liu-2019-26
 devel  from NOW to INF  (everyone)

Note: set one of the above as default by running:
 usereservation RESERVATIONNAME
Or without the usereservation alias:
 source /software/tools/bin/usereservation.sh RESERVATIONNAME
[x_huali@sigma 2_java_wordcount_1.0]$ sbatch -A liu-compute-2023-24 --reservation devel ru
n.q
Submitted batch job 3709518
[x_huali@sigma 2_java_wordcount_1.0]$ squeue -u x_huali
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           3709518     sigma    run.q  x_huali  R       0:07      2 n[1165,1169]
[x_huali@sigma 2_java_wordcount_1.0]$ squeue -u x_huali
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           3709518     sigma    run.q  x_huali  R       0:46      2 n[1165,1169]
[x_huali@sigma 2_java_wordcount_1.0]$ squeue -u x_huali
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount$IntSumReducer.class    WordCount.java  run.q               wordcount.jar
WordCount$TokenizerMapper.class  compile.sh      slurm-3709518.out
WordCount.class                  input           spark
[x_huali@sigma 2_java_wordcount_1.0]$ vi slurm-3709518.out
```

Step 3

Step 4

Step 5

How to work on Sigma



Word count example code

- Main function – Job configuration

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Word count example code

- Mapper

Class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object 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);
        }
    }
}
```

Specifilize output format of key-value pairs of a mapper

protected void map(KEYIN key, VALUEIN value, Context context)

Mapper/Reducer interacts with Hadoop, e.g., emit output

void write(KEYOUT key, VALUEOUT value)

Word count example code

- Reducer

Class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>

protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)

```
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                         ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

www.liu.se

LINKÖPING UNIVERSITY