Lab Course: distributed data analytics

03. Distributed Environments - Hadoop

Nghia Duong-Trung, Mohsan Jameel

Information Systems and Machine Learning Lab (ISMLL)
University of Hildesheim, Germany

International Master's Program in Data Analytics
Summer Semester 2017

Outline

1. Hadoop Architecture
2. HDFS and YARN
3. MapReduce
4. Hadoop Streaming

Outline

Apache Hadoop

- Apache Hadoop is an open source software framework that can be installed on a cluster of machines such that the machines can communicate and work together to store and process large amounts of data in a highly distributed manner.
- The core of Apache Hadoop consists of a storage part, Hadoop Distributed File System (HDFS), and a processing part, MapReduce programming model.
  - Hadoop Common: contains libraries and utilities needed by other Hadoop modules.
  - From Hadoop 2.x, Hadoop YARN: a resource-management platform responsible for managing computing resources.

Apache Hadoop

- Hadoop consists of two main components:
  - Hadoop Distributed File System (HDFS): responsible for managing data stored on disks across a cluster.
  - Yet Another Resource Negotiator (YARN): responsible for allocating computational assets to applications that wish to perform a distributed computation.
- HDFS and YARN are implemented by several daemon process.
  - Processes that run in the background and do not require user input.
  - Hadoop processes run all the time on a cluster node and accept input and deliver output through the network.

Hadoop Architecture Overview

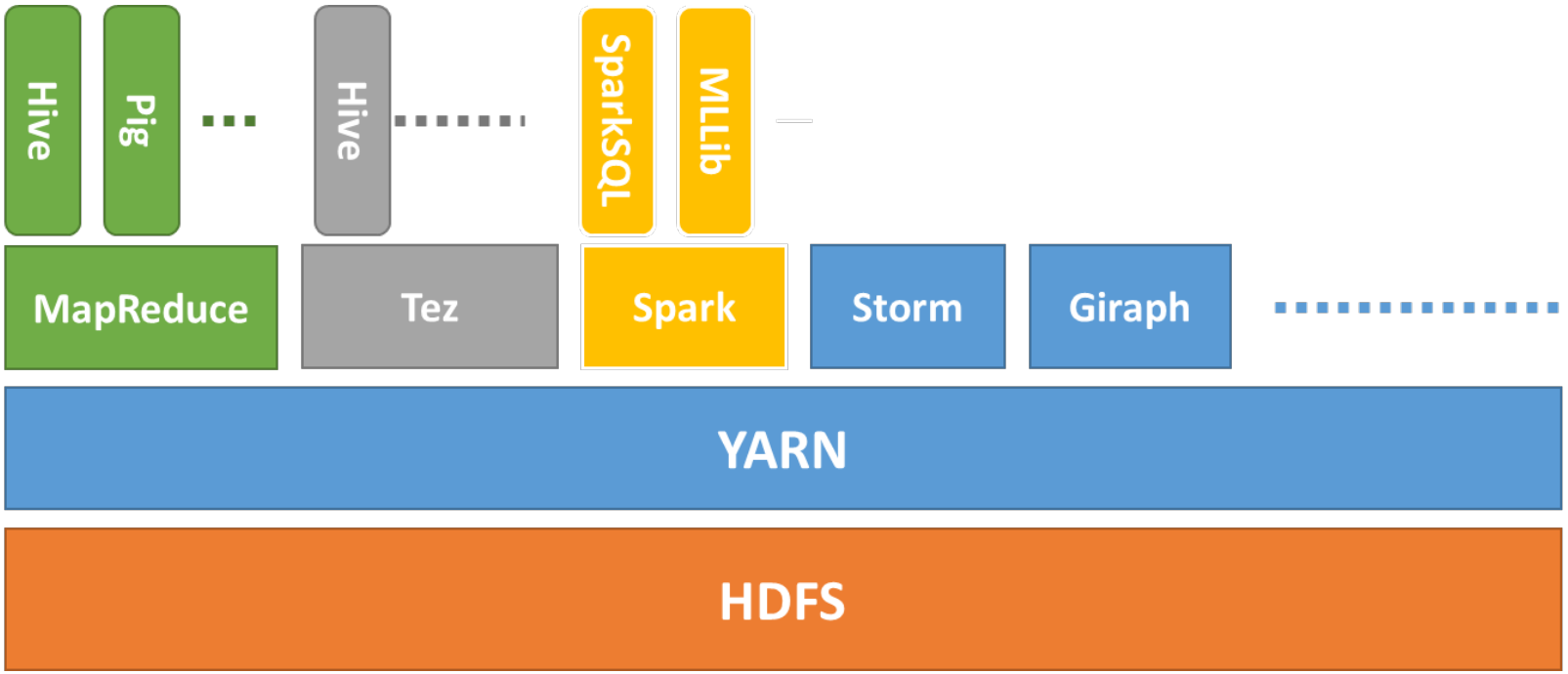


Figure: Hadoop architecture [14]

A Hadoop Cluster

- A Hadoop cluster is a set of machines that runs HDFS and YARN.
- Each individual machine is called a node.
- Nodes may be partitioned in racks. This is the hardware part of the infrastructure.
- A cluster can have a single node, or many thousands of nodes.
- Nodes are scale horizontally.
- Adding more nodes results in the total cluster increase in both capacity and performance in a linear manner.

A Hadoop Cluster

- Each node in the cluster is identified by the type of process or processes that it runs:
  - Master nodes
    - one or a few master nodes in the cluster.
    - run coordinating services for worker nodes.
    - entry points for user access to the cluster.
  - Worker nodes
    - majority of the computers in the cluster.
    - run services that accept tasks from master nodes.
    - a distributed computation is run by parallelizing across worker nodes.

A Hadoop Cluster

- For HDFS, the master and worker nodes are:
  - NameNode (Master)
    - stores the directory tree of the file system, file metadata, and the locations of each file across the cluster.
  - Secondary NameNode (Master)
    - performs housekeeping tasks and checkpoint schedule.
  - DataNode (Worker)
    - stores data in HDFS blocks on the local disk.

A Hadoop Cluster

- For YARN, the master and worker nodes ares:
  - ResourceManager (Master)
    - monitors and allocates available cluster resources to needed applications.
    - schedules jobs on the cluster.
  - ApplicationMaster (Master)
    - tracks the execution of jobs scheduled by the ResourceManager.
  - NodeManager (Worker)
    - runs processing tasks on local node.

Outline

HDFS

- HDFS is a distributed file system designed to run on commodity hardware.
- HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
- HDFS provides high throughput access to application data and is suitable for applications that have large data sets.
- Key HDFS goals:
  - Quick and automatic recovery from hardware failure.
  - Batch processing and streaming data.
  - Support huge data sets from gigabytes to terabytes in size.

HDFS

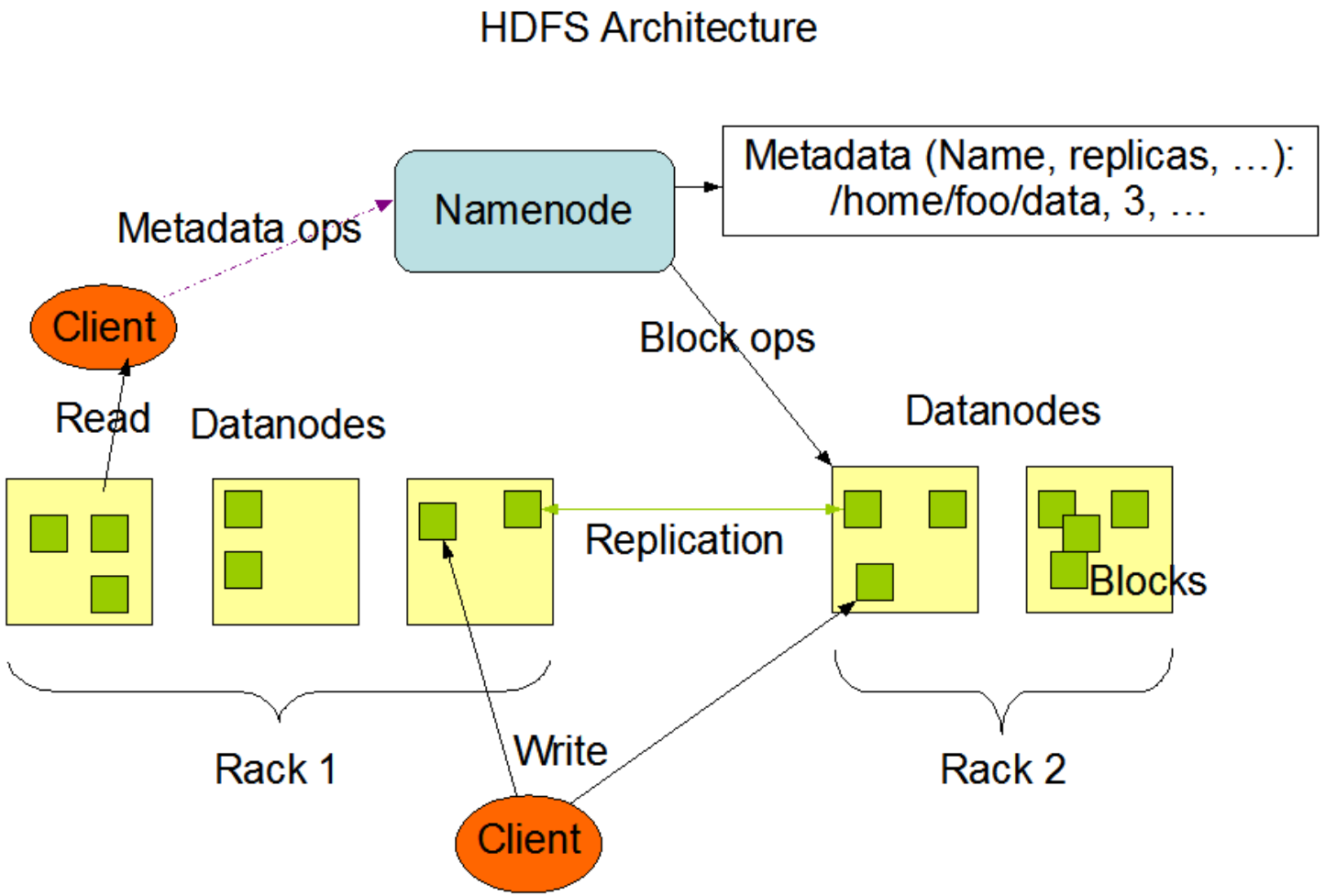


Figure: HDFS architecture [1]

YARN

- YARN is responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions.
- The application startup process is the following:
  - a client submits an application to the Resource Manager.
  - the Resource Manager allocates a container.
  - the Resource Manager contacts the related Node Manager.
  - the Node Manager launches the container.
  - the Container executes the Application Master.

YARN

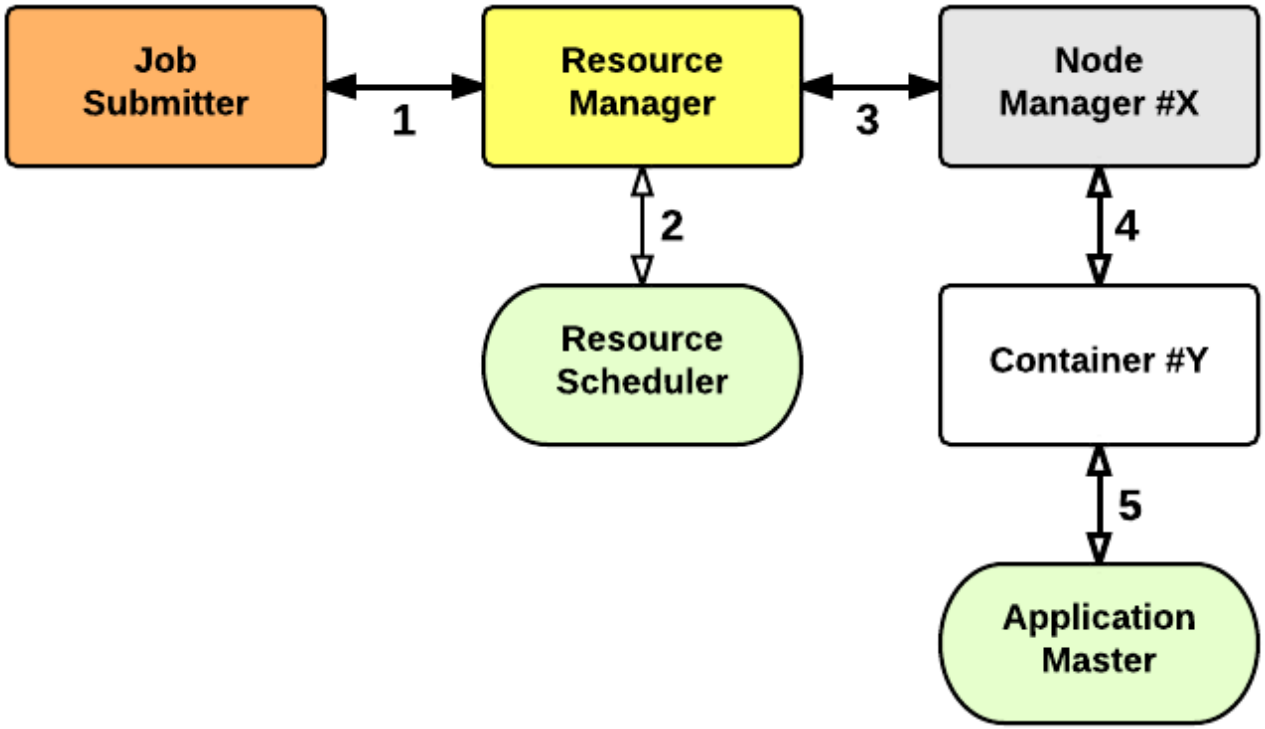


Figure: Application startup process in YARN [1]

Outline

MapReduce

- MapReduce is a functional programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.
  - Stateless; independent functions; depend solely on input.
  - Work on small chunks of datasets.
- The model is a specialization of the split-apply-combine strategy for data analysis.
- MapReduce is specifically designed to enable fault-tolerant distributed computation across a cluster.
- MapReduce provides analytical capabilities for analyzing huge volumes of complex data.

MapReduce

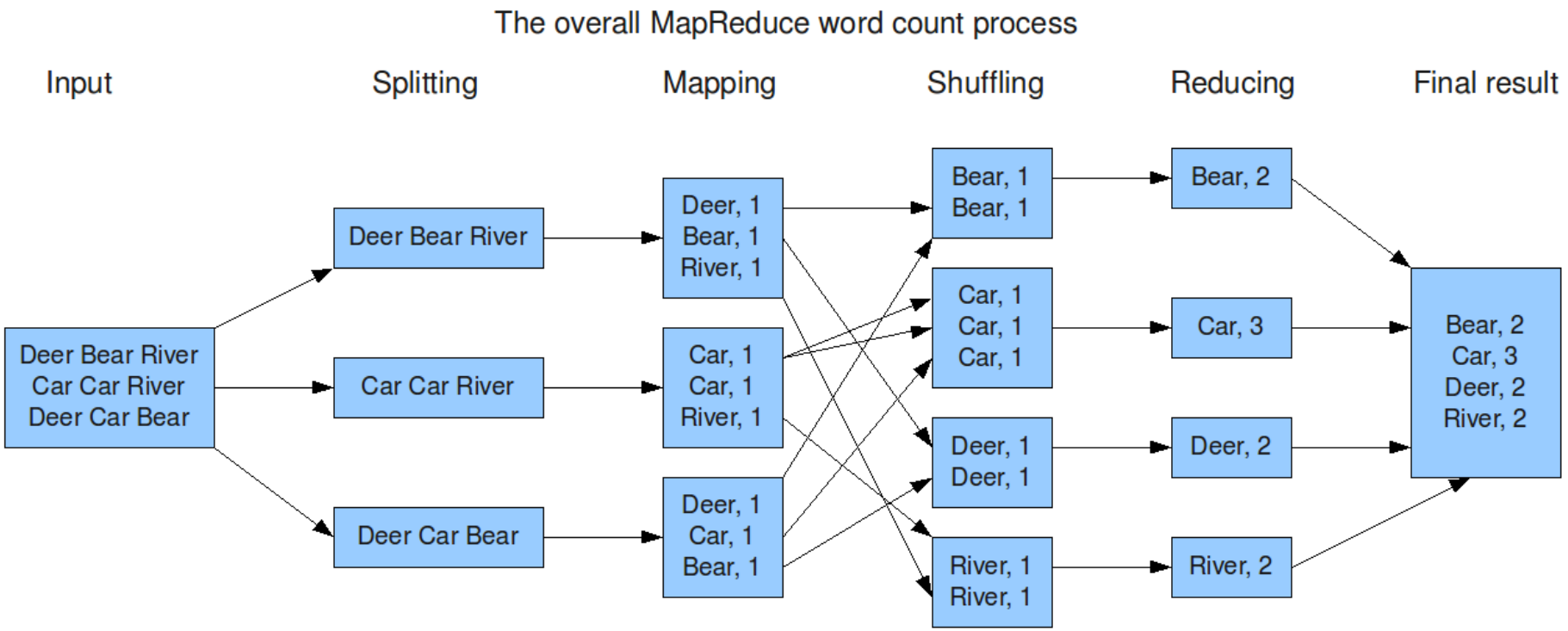
- MapReduce consists of two functions that distribute work and aggregate results: `map` and `reduce`.
- MapReduce utilizes `key-value` pairs to coordinate computation.
- The Pseudo-code for `map` and `reduce` looks as follows:

```
1 def map(key, value):
2   # perform processing
3   return (inter_key, inter_value)
4
5 def reduce(inter_key, inter_value):
6   # perform processing
7   return (key, value)
```

How MapReduce works?

- MapReduce works in two main phases, namely `map` and `reduce`:
  - In the `map` phase, a `map` function takes as input a series of key-value pairs and operates on each individual pair.
  - After the map phase, any `emitted` key-value pairs will then be grouped by key and those key-value groups are applied as input to `reduce` functions on a per-key basis.
  - In the `reduce` phase, a `reduce` function takes the output from `map` as an input and combines key-value pairs into a smaller set of tuples.
  - The `reduce` function is always performed after the `map` function.

How MapReduce works?



How MapReduce works?

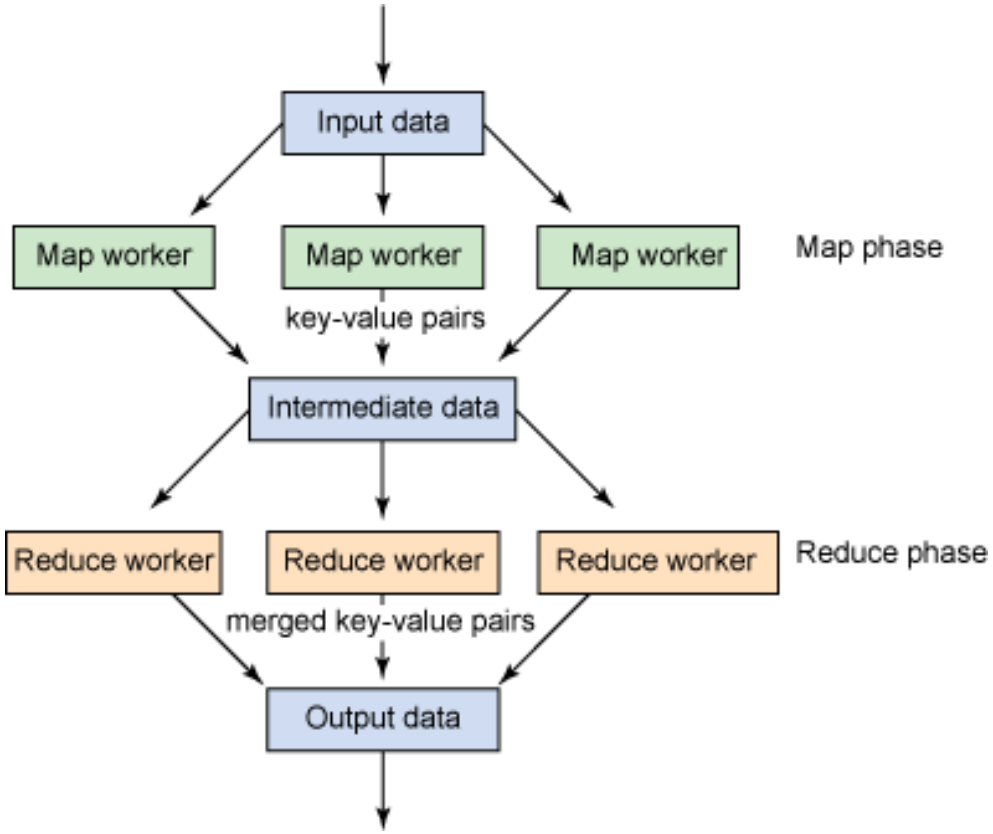


Figure: Simplified view of MapReduce processing [15]

Implementation of MapReduce on a cluster

- Because `mappers` and `reducers` apply the same function to each element independently, they are suitable for distribution across nodes on a cluster.
- There can be any number of `mappers` and `reducers` working on as much data as possible.
- Either network communication between `mappers` or network communication between `reducers` is required.
- Carefully consider how the key-value pairs are defined.

Implementation of MapReduce on a cluster

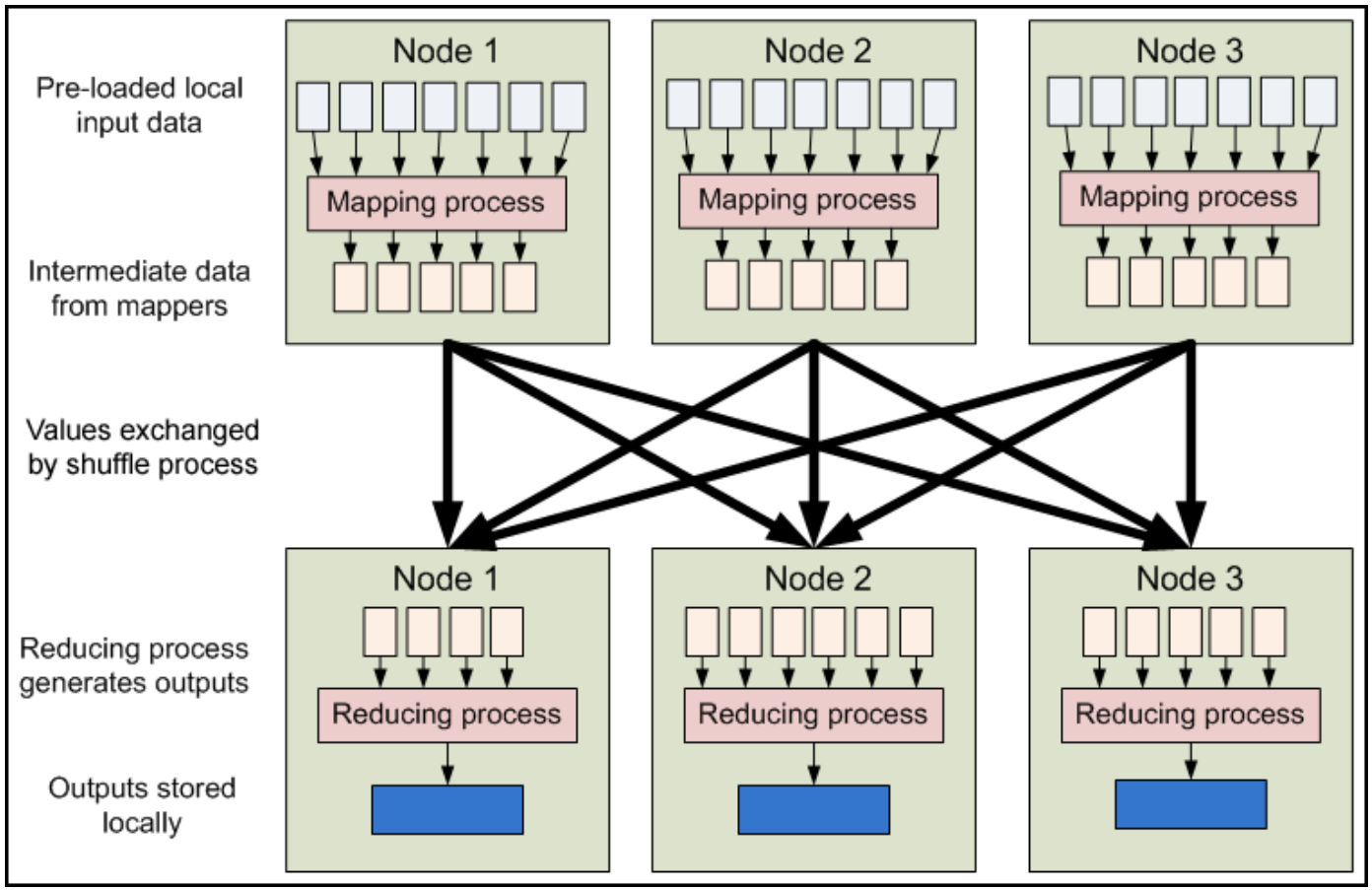


Figure: Data flow of a MapReduce job being executed on a cluster [10]

Implementation of MapReduce on a cluster

- Local storage vs. HDFS storage.
- `mapper.py` and `reducer.py` are stored in local storage.
- Required datasets are copied into HDFS.
- Output of `reduce` jobs are stored on HDFS.
- `-getmerge` or copy the output from HDFS to local storage.

Outline

Hadoop Streaming

- Hadoop Streaming is a utility, packaged as a `JAR` file that comes with the Hadoop distribution.
- Its location in the Hadoop directory is: `%HADOOP_HOME%/share/hadoop/tools/lib/hadoop-streaming-*.ja` e.g. * is 2.7.3.
- Hadoop Streaming utilizes the standard Unix streams for input and output [12].
- Input to both `mapper` and `reducer` is read from `stdin`, which can be accessed via standard `import sys` module in Python.
- Similarly, output from both `mapper` and `reducer` is exported via `stdout`.

Hadoop Streaming

- `$ cat input.txt | ./mapper.py | sort | ./reducer.py > output.txt`

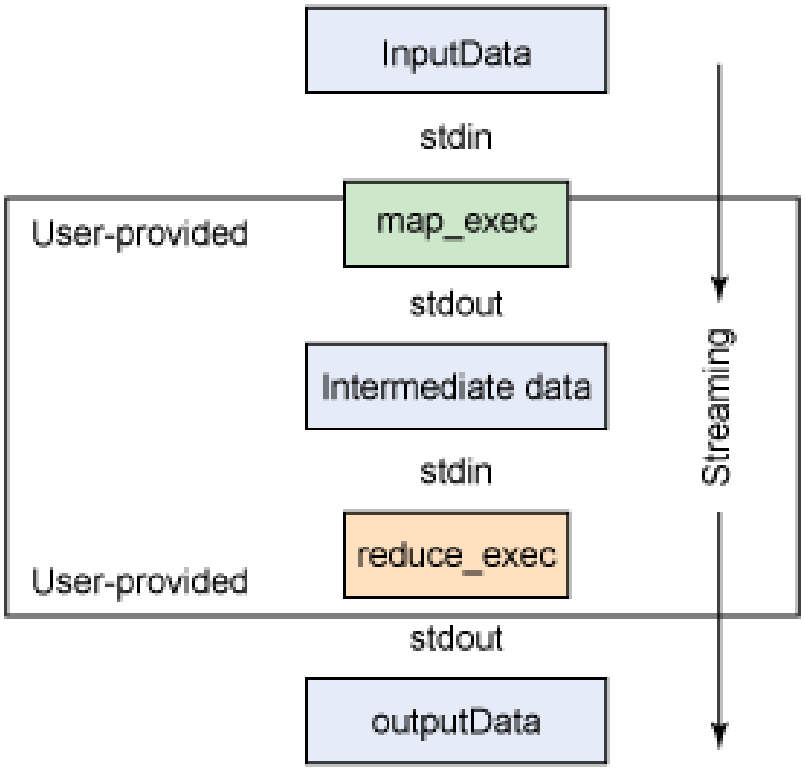


Figure: Graphical streaming example [15]

How Hadoop Streaming works?

- We need to create two Python files, a `mapper.py` and a `reducer.py`, that we need to import the `sys` module to get access to `stdin` and `stdout`.
- The `mapper.py` converts the input data it gets from `stdin`, operates some operations while simultaneously passes results to `stdout`.
- The `mapper` expects output to be in a string key-value format, where key and value are separated by some separator, e.g. tab (\t).
- The key-value strings from the `mapper` are streamed into `reducer` as input via `stdin`.
- The data is then be grouped by key and emitted to `stdout`.

A pipeline for Hadoop Streaming jobs

1. Define what operations one should do in the `map` phase: `mapper.py`

```
import sys
for line in sys.stdin:
  # do somthing to define keys
  for key in keys:
    value = 1
    print('{0}\t{1}'.format(key, value))
```

2. Define what operations one should do in the `reduce` phase: `reducer.py`

```
import sys
for line in sys.stdin:
  # do somthing to calcuate values
  print( "{0}\t{1}".format(key, total_values))
```

3. Copy required datasets from local storage to HDFS, e.g hdfs_path/data_files
4. Set a desired path for output directory on HDFS, e.g. hdfs_path/output_folder. The output directory is automatically generated.

A pipeline for Hadoop Streaming jobs

5. Execute the streaming job (* is the Hadoop version); assume that `mapper.py` and `reducer.py` are at the folder where you execute the `hadoop` command:
   - Ubuntu:
```
hadoop jar
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar
-input /hdfs_path/data_files
-output /hdfs_path/output_folder
-mapper mapper.py
-reducer reducer.py
-file ./mapper.py -file ./reducer.py
```
   - Windows:
```
hadoop jar
%HADOOP_HOME%/share/hadoop/tools/lib/hadoop-streaming-*.jar
-input /hdfs_path/data_files
-output /hdfs_path/output_folder
-mapper "python mapper.py"
-reducer "python reducer.py"
-file mapper.py -file reducer.py
```

A pipeline for Hadoop Streaming jobs

6. Check and debug the processes by scanning the output texts in the terminal during the job is executed and/or browsing Hadoop User Interface.

7. The results are now in hdfs_path/output_folder

   - One can choose to download the results using Hadoop User Interface, or
   - run the command line: `hdfs dfs -getmerge /hdfs_path/output_folder output.txt`

Further Reading

1. Apache Hadoop official wiki `https://wiki.apache.org/hadoop`
2. White, T. (2012). Hadoop: The definitive guide. " O'Reilly Media, Inc.".
3. Schutt, R., & O'Neil, C. (2013). Doing data science: Straight talk from the frontline. " O'Reilly Media, Inc.".
4. Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... & Saha, B. (2013, October). Apache hadoop yarn: Yet another resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (p. 5). ACM.
5. Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
6. Dean, J., & Ghemawat, S. (2010). MapReduce: a flexible data processing tool. Communications of the ACM.
7. Intro to Hadoop and MapReduce `https://www.udacity.com/course/intro-to-hadoop-and-mapreduce--ud617`
8. Introduction to YARN `https://www.ibm.com/developerworks/library/bd-yarn-intro/`

Further Reading

9. Hadoop internals `http://ercoppa.github.io/HadoopInternals/`
10. Yahoo! Developer Network: MapReduce
    `https://developer.yahoo.com/hadoop/tutorial/module4.html`
11. Udemy: Programming Hadoop with Python
    `https://www.udemy.com/programming-hadoop-with-python/`
12. Overview of STREAMS
    `http://www.shrubbery.net/solaris9ab/SUNWdev/STREAMS/p4.html`
13. Hadoop streaming `https://hadoop.apache.org/docs/r2.7.3/hadoop-streaming/HadoopStreaming.html`
14. Perera, S. (2013). Hadoop MapReduce Cookbook. Packt Publishing Ltd.
15. Distributed data processing with Hadoop
    `https://www.ibm.com/developerworks/library/l-hadoop-3/`