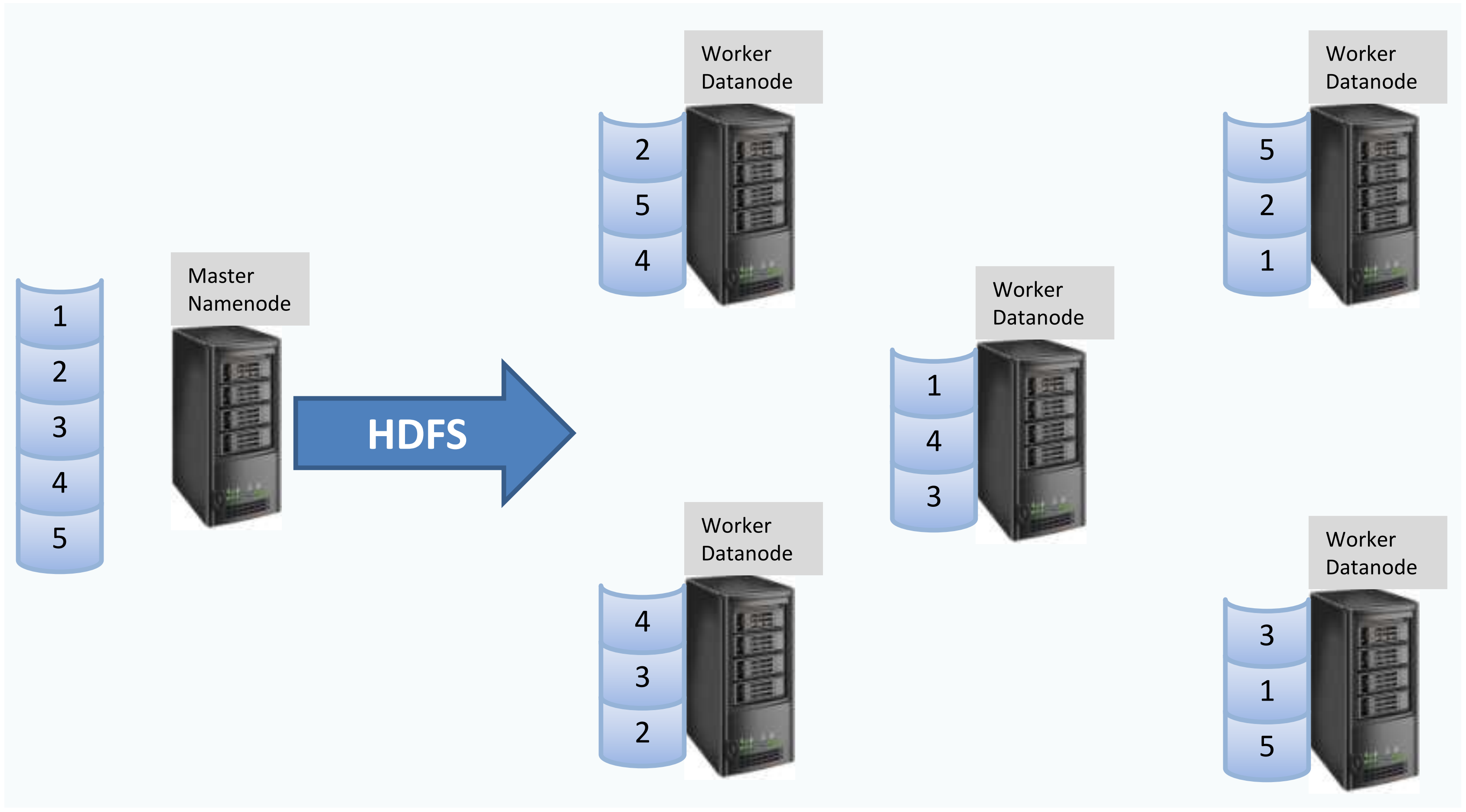
Hadoop
MapReduce

Training Session

Contents

Hadoop Distributed File System (HDFS)



MapReduce Architecture

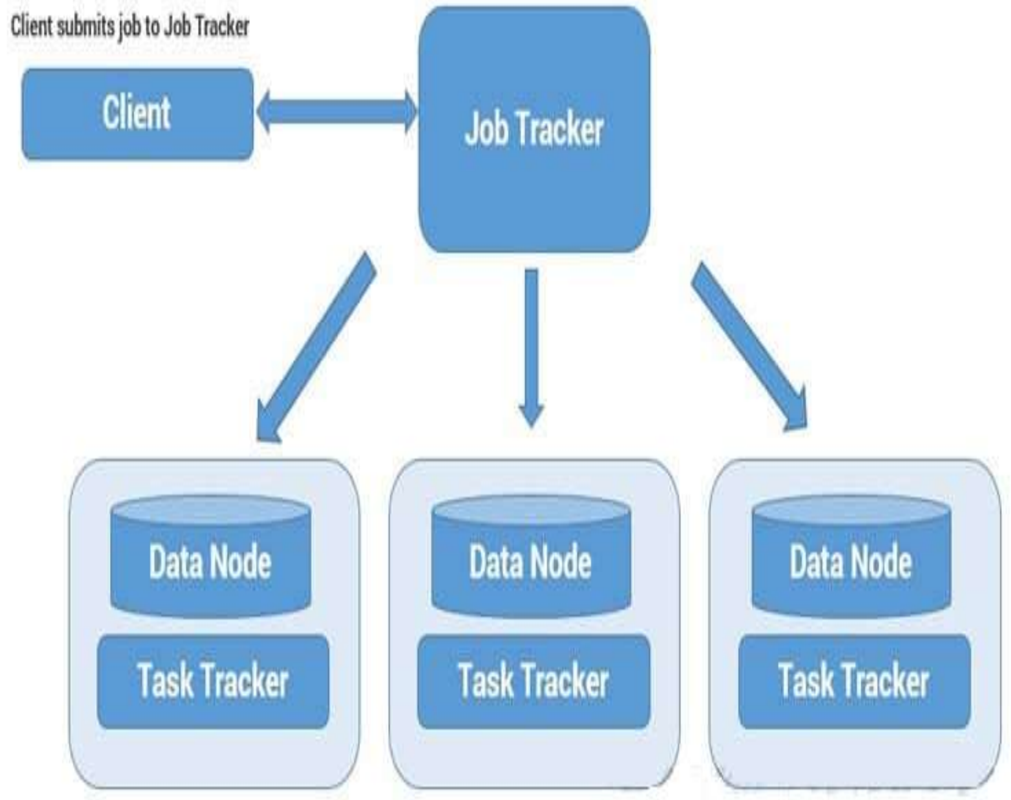
- Hadoop *MapReduce* is a software framework for developing applications that process Big Data on HDFS in a reliable, fault-tolerant manner.
- A MapReduce job is divided into 2 phases – *Map phase* and *Reduce phase.*
- MapReduce *job* splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner.
- The framework sorts the outputs of the maps, which are then input to the *reduce tasks*.
- The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.
- In Hadoop 1 The framework consists of a single master *JobTracker* and slave *TaskTracker* per each cluster-node.
- *JobTracker* is responsible for scheduling the tasks on the slaves, monitoring them and re-executing the failed tasks.
- The slaves execute the tasks as directed by the master and report the progress.

MapReduce Architecture

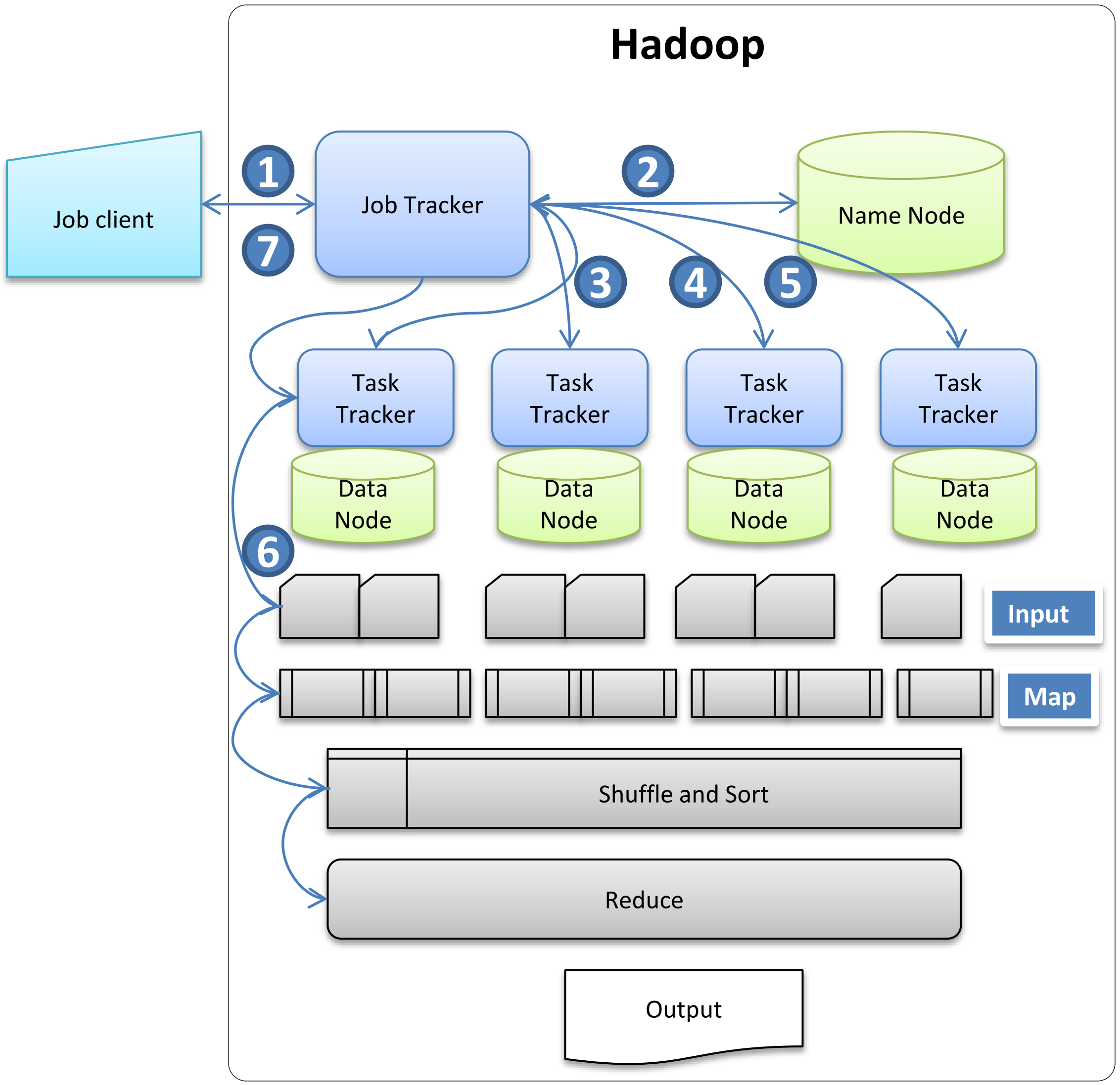
- MapReduce breaks the processing into two phases: the *map* phase and the *reduce* phase
- Each of these phases are divided into *tasks*
- The input data is divided into fixed-size chunks called *input splits* or just *splits*
- MapReduce creates one *map* task for each split, all of which are run in parallel
- Typically *data nodes* and the nodes on which the tasks are executed are the same
- This configuration allows the framework to schedule tasks where data resides i.e. ensures *data locality*
  - The framework tries its best to run map tasks on same node where the corresponding data resides
  - If map slots are not available on the node then the task will be scheduled on a node in the same rack
  - Only when this is also not possible then the map task will be scheduled on any other node

ARCHITECTURAL DESCRIPTION

- Clients (one or more) submit their work to Hadoop System.
- When Hadoop System receives a Client Request, first it is received by a Master Node.
- Master Node's MapReduce component "Job Tracker" is responsible for receiving Client Work and divides into manageable independent Tasks and assign them to Task Trackers.
- Slave Node's MapReduce component "Task Tracker" receives those Tasks from "Job Tracker" and perform those tasks by using MapReduce components.
- Once all Task Trackers finished their job, Job Tracker takes those results and combines them into final result.



MapReduce Architecture



1. Client Submits the job to JobTracker
2. JobTracker talks to NameNode and gets DataNode locations
3. JobTracker creates execution plan
4. JobTracker submits the work to TaskTrackers
5. TaskTrackers report the progress via heartbeats
6. JobTracker manages phases
7. JobTracker updates the status

MapReduce Architecture

- The split size normally is equal to the block size
  - If the split size is too small then managing the splits and map task creation will be an overhead
  - If the split size is larger than the block size then part of the split may reside in a different node which needs to be transferred across network which decreases the performance
- The number of mappers is usually driven by the total size of the input, that is, the total number of blocks of the input file/files
- Map task consists of a user-defined map function which is executed for each record in the split
- The outputs of the *map tasks* are shuffled and sorted by the framework
- These are then input into the *reduce tasks* which comprise of user-defined reduce function
- The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks

MapReduce Architecture

- Applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These and other job parameters, comprise the *job configuration*
- The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the *JobTracker* which takes the responsibility of:
  - Distributing the software/configuration to the slaves
  - Scheduling tasks and monitoring them
  - Providing status and diagnostic information to the job-client
- *AppMaster* working with *ResourceManager* in the case of MR2 takes up the above responsibilities

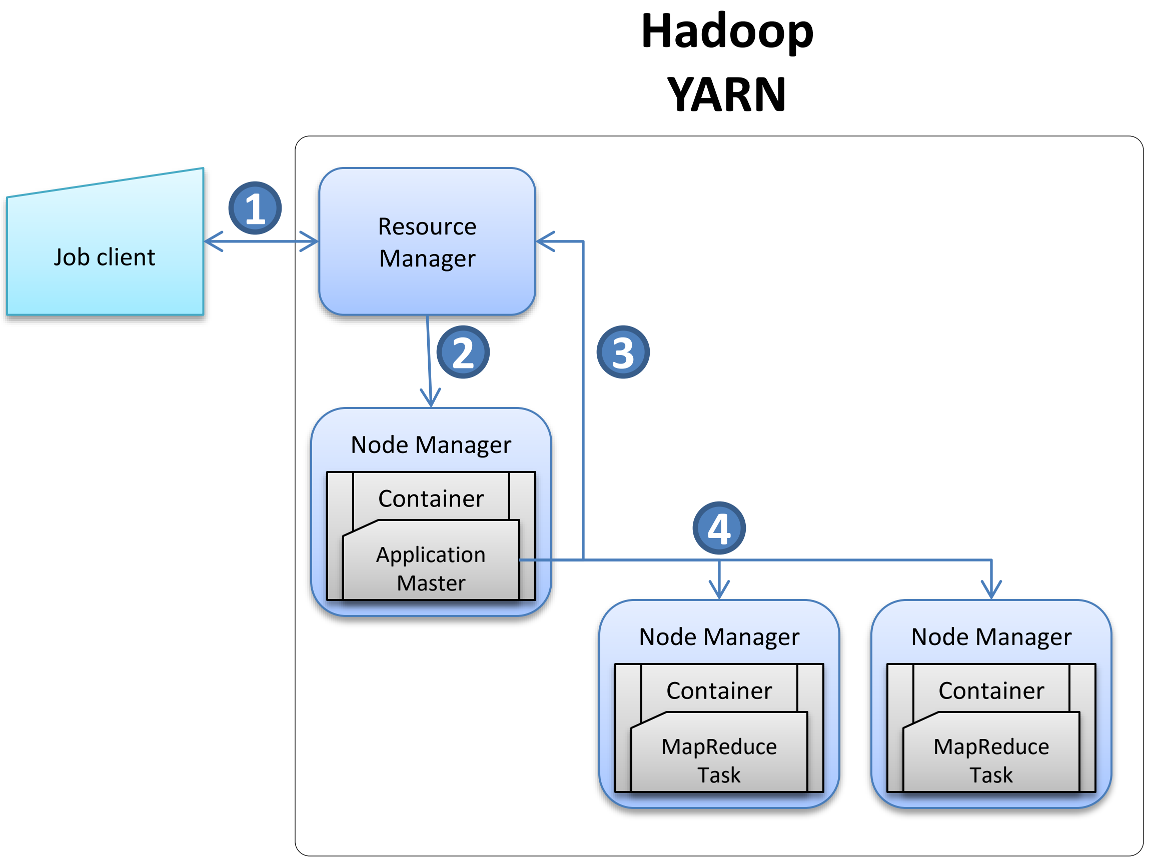
MapReduce Architecture – YARN

- YARN (Yet Another Resource Negotiator) was introduced in Hadoop 2 to improve the MapReduce implementation.
- It is general enough to support other distributed computing paradigms as well.
- ResourceManager and NodeManager form the data-computation framework in Hadoop 2.
  - *Resource manager (one per cluster) to manage the use of resources across the cluster and*
  - *node managers running on each node of the cluster to launch and monitor containers and reporting their resource usage to Resource Manager*
- In YARN the functionalities of resource management and job scheduling/ monitoring are separated.
- The ResourceManager has two main components: Scheduler and ApplicationsManager.
- The Scheduler is responsible for allocating resources to the various running applications subject to constraints like capacities, queues.
- Scheduler performs its scheduling function based on the concept of a *Container* which incorporates resources such as memory, cpu, disk etc.

MapReduce Architecture – YARN

- The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific *ApplicationMaster*.
- The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.
- ApplicationMaster thus works with the NodeManagers to execute and monitor the tasks.
- ApplicationManager also provides the service for restarting the ApplicationMaster container on failure.

MapReduce Architecture on YARN



1. Client submits the job to Resource Manager
2. RM through its Application Manager finds a Node Manager and launches Application Master in a Container
3. Application Master negotiates for Containers with Scheduler of RM on other cluster-nodes to run the tasks of the job
4. Application Master starts the Containers under respective Node Managers to run the tasks of the job

MapReduce Internals

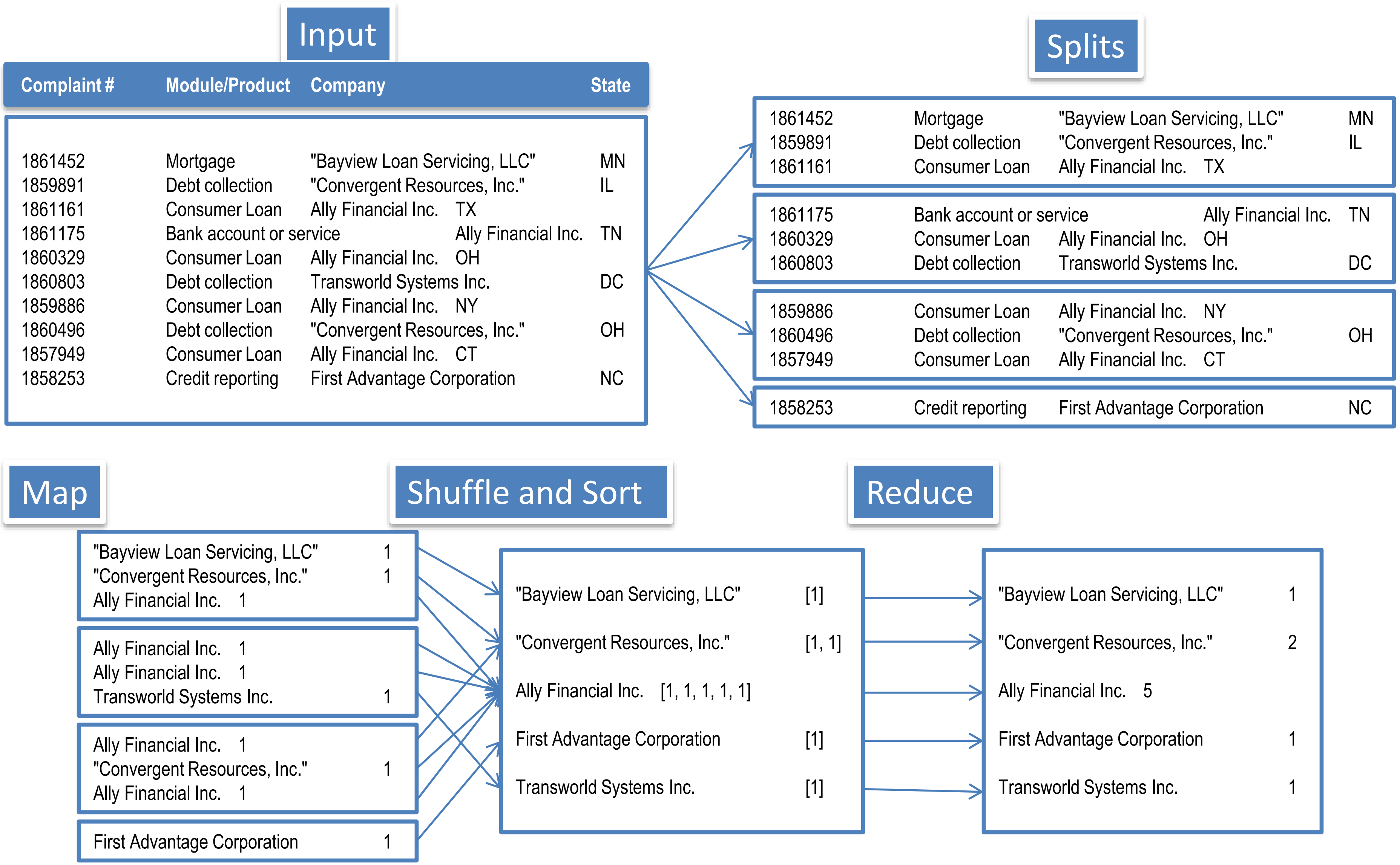
- Following the overview of MapReduce framework, let us get an insight into the internals and the code.
- We need three things: a map function, a reduce function, and the code which runs the job – Mapper, Reducer and Driver as they are referred to.
- Data is divided into blocks or splits as we know. However most processing jobs require data to be brought together in some way.
- MapReduce provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values.
- MapReduce framework operates exclusively on <key, value> pairs. It views the input to the job as a set of <key, value> pairs.
- It produces the output as a set of <key, value> pairs. The input and output could be of different types.
- So, each phase has key-value pairs as input and output, the types of which may be chosen by the programmer as per requirements.
- Input and Output types of a MapReduce job:

```
(input) <k1, v1> → map → <k2, v2> →
reduce → <k3, v3> (output)
```

MapReduce Internals

- Hadoop can process many different types of data formats, from flat text files to databases.
- TextInputFormat is the default InputFormat and each record is a line of the input file.
- The key is the byte offset within the file from the beginning of the line and is of type LongWritable. The value is the contents of the line.
- Other InputFormats include:
  - **KeyValueTextInputFormat** in which the first field is the key and rest of the line is the value
  - **SequenceFileInputFormat** stores sequences of binary key-value pairs
  - **SequenceFileAsTextInputFormat** which is a variant of SequenceFileInputFormat
  - **SequenceFileAsBinaryInputFormat** is another variant that retrieves the keys and values as opaque binary objects
  - **FixedLengthInputFormat** is used for reading fixed-width binary records from a file
- RecordReader, a pre-defined class generates the record key-value pairs and passes to the map function.
- In our example, we will use a test data-set with 10 lines (records) each having 4 tab separated fields
- These are trouble tickets or complaints raised for a software system by different companies that use the system
- Our objective is to develop a MapReduce program that outputs the list of companies and the number of complaints raised by each company

MapReduce Internals



Input

Complaint #	Module/Product	Company	State
1861452	Mortgage	"Bayview Loan Servicing, LLC"	MN
1859891	Debt collection	"Convergent Resources, Inc."	IL
1861161	Consumer Loan	Ally Financial Inc.	TX
1861175	Bank account or service	Ally Financial Inc.	TN
1860329	Consumer Loan	Ally Financial Inc.	OH
1860803	Debt collection	Transworld Systems Inc.	DC
1859886	Consumer Loan	Ally Financial Inc.	NY
1860496	Debt collection	"Convergent Resources, Inc."	OH
1857949	Consumer Loan	Ally Financial Inc.	CT
1858253	Credit reporting	First Advantage Corporation	NC

Splits

1861452	Mortgage	"Bayview Loan Servicing, LLC"	MN
1859891	Debt collection	"Convergent Resources, Inc."	IL
1861161	Consumer Loan	Ally Financial Inc.	TX

1861175	Bank account or service	Ally Financial Inc.	TN
1860329	Consumer Loan	Ally Financial Inc.	OH
1860803	Debt collection	Transworld Systems Inc.	DC

1859886	Consumer Loan	Ally Financial Inc.	NY
1860496	Debt collection	"Convergent Resources, Inc."	OH
1857949	Consumer Loan	Ally Financial Inc.	CT

1858253	Credit reporting	First Advantage Corporation	NC

Map

"Bayview Loan Servicing, LLC"	1
"Convergent Resources, Inc."	1
Ally Financial Inc.	1

Ally Financial Inc.	1
Ally Financial Inc.	1
Transworld Systems Inc.	1

Ally Financial Inc.	1
"Convergent Resources, Inc."	1
Ally Financial Inc.	1

First Advantage Corporation	1

Shuffle and Sort

"Bayview Loan Servicing, LLC"	[1]
"Convergent Resources, Inc."	[1, 1]
Ally Financial Inc.	[1, 1, 1, 1, 1]
First Advantage Corporation	[1]
Transworld Systems Inc.	[1]

Reduce

"Bayview Loan Servicing, LLC"	1
"Convergent Resources, Inc."	2
Ally Financial Inc.	5
First Advantage Corporation	1
Transworld Systems Inc.	1

MapReduce Internals

Mapper:

- The map function is represented by the Mapper class, which declares an abstract map() method.
- The Mapper class is a generic type, with four type parameters that specify the input key, input value, output key, and output value types of the map function.
- Hadoop provides its own set of basic data types that are optimized for network serialization rather than using built-in Java data types.
- In our example we use LongWritable, which corresponds to a Java Long, Text (like Java String), and IntWritable (like Java Integer)
- In our code the input key is a long integer offset, the input value is a line of text, the output key is company name (text string) and the output value is the count (integer).
- In Mapper we convert the input value into a Java String, then use its split() method to extract the column we are interested in.
- The map() method also provides an instance of OutputCollector to collect the output. In this case, we write the company name as a Text object (since we are just using it as a key) and we hard-code 1 an IntWritable.

MapReduce Internals

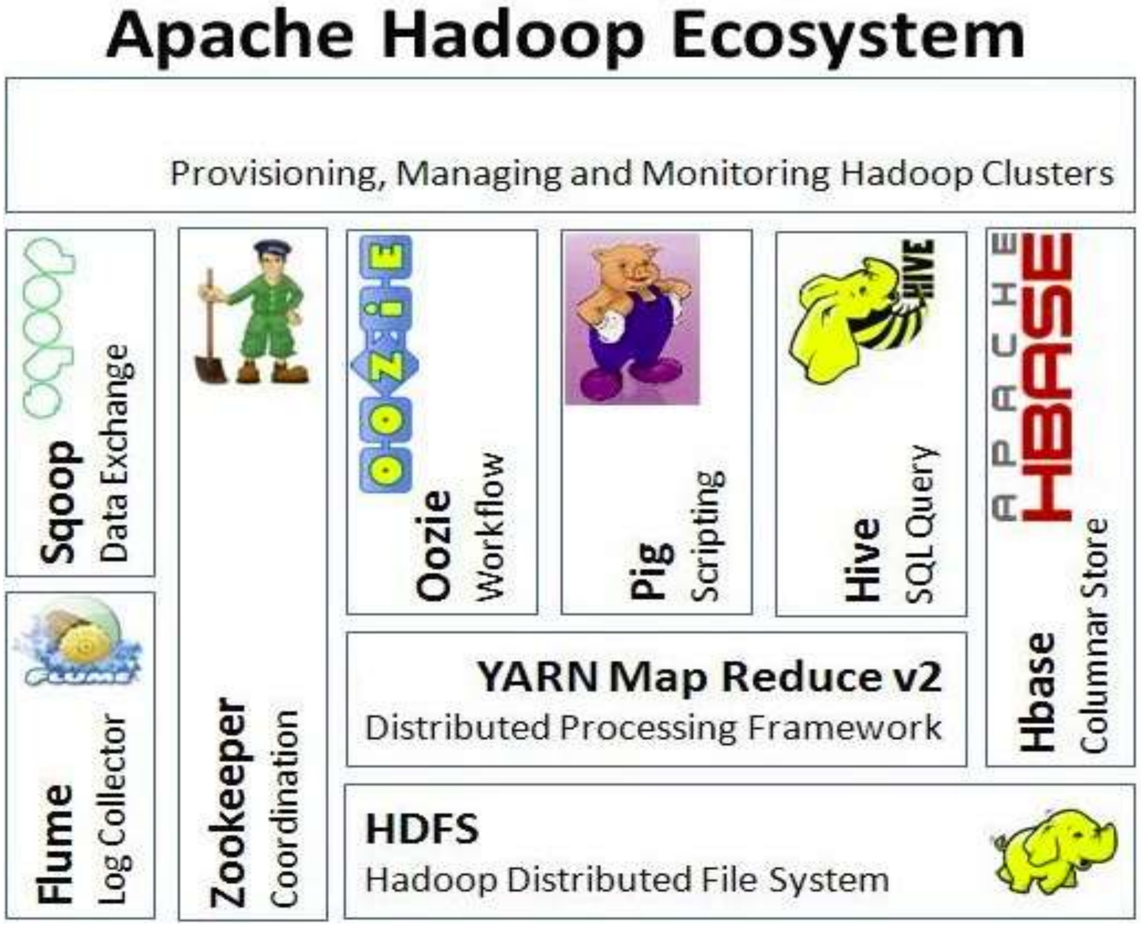
Reducer:

- *Reduce* function is similarly defined using *Reducer* with four type parameters to specify the input and output types
- Input types of *reduce* function should match the output types of *map* function: Text and IntWritable
- Output types of *reduce* function are also Text and IntWritable, for company name and the total complaints raised by them which we find by iterating through the *values* (hard-coded 1s) and summing them up

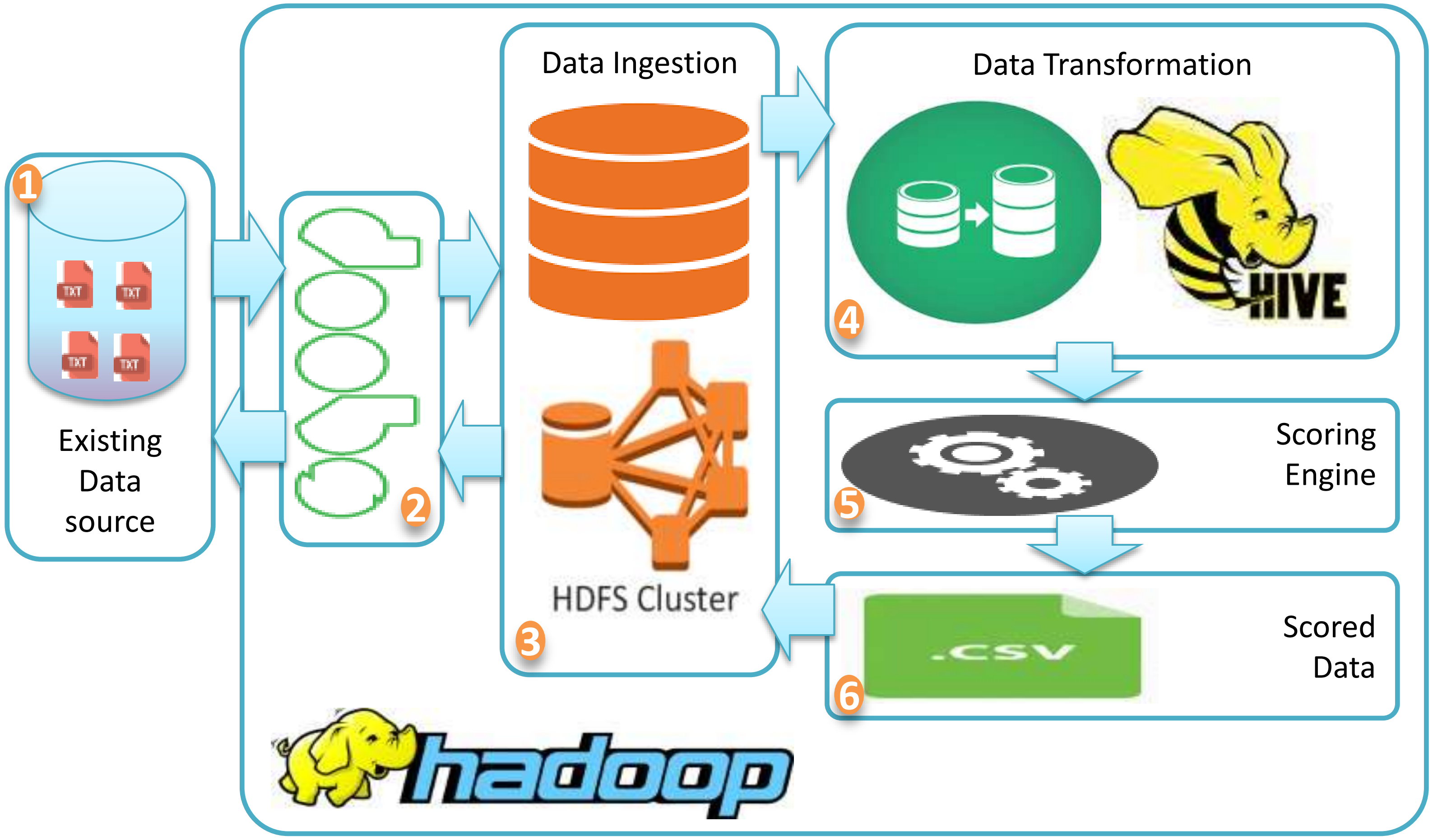
Driver:

- The third piece of code – Driver, runs the MapReduce job.
- A JobConf object forms job configuration and allows the job to be run.
- The job which is to be run on Hadoop cluster, will be packaged as a JAR file . Hadoop will distribute around the cluster.
- Then the input and output paths are set. Please note that for output, Hadoop will create a directory by the name provided. The output directory should not be existing else Hadoop will give an error message and abort the job. This is to ensure that any previous outputs are not overwritten.

Hadoop Ecosystem Components



A Hadoop-based Analytics Project


1
TXT
TXT
TXT
TXT
Existing Data source
2
Data Ingestion
HDFS Cluster
3
Data Transformation
HIVE
4
5
Scoring Engine
.CSV
6
Scored Data
hadoop