Amirkabir University of Technology
(Tehran Polytechnic)

Cloud Computing

Big data and analytics in cloud

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Spring 2020

Based on slide deck of Dr. Abrishami (Ferdowsi Uni. of Mashhad)

Main source: Mastering Cloud Computing By Rajkumar Buyya.

Introduction

- Our world is awash in a rising ocean of data
- Examples of big data
  - Modern modes of transportation – planes, cars, and ships – contain thousands of sensors that constantly generate data.
  - Medical researchers sort through thousands of genes in millions of patients, attempting to find genes that lead to diseases.
  - Online businesses such as Amazon and Netflix track their customers' online behavior and use that information to suggest books or movies for them to purchase.
  - Smartphones record audio, video, and images and their owners post them to social media sites.
- The age of big data has begun, and exploiting big data is changing our world.

Introduction (Cont.)

- Data-intensive computing is about **production, manipulation, and analysis of large-scale data** in the range of hundreds of megabytes (MB) to petabytes (PB) & beyond.
- There are many application domains.
- Exemplar: Computational science
  - Hundreds of GBs of data are produced by telescopes.
  - Bioinformatics applications mine terabytes of data.
  - Earthquake simulators process a massive amount of data.

Introduction- IT industry sectors

- Customer data for any telecom company would easily be in the range of 10-100 terabytes.

- This volume of information is not only processed to generate billing statements, but it is also mined to **identify scenarios, trends, and patterns** to provide better service.

- Google is reported to process about 24 petabytes of information per day and to sort petabytes of data in hours.

Types of Data

- **Structured data** is data that is organized in a structure
  - Fixed fields inside a record (e.g., relational database)
  - Well-formed format (XML or JSON).

- **Semi-structured data** has some structure, but the data isn't expressed in terms of rows and columns.
  - Example: an HTML page

- **Unstructured data** does **not** have fields in fixed locations, **nor** does it follow a standard format such as XML or JSON.
  - Examples: raw text files such as a server log, a Microsoft Word document, a Portable Document Format (PDF) file.

Sources of Data

- Sources of big data include:
  - Business operational data
  - Scientific data
  - Social networking
  - Web logs
  - Video streaming
  - Sensor data
  - Smartphone data
  - Many more …

Streams

- Some data is produced in streams.
- A data stream is “a sequence of digitally encoded signals used to represent information in transmission”.
- **Examples**: click streams, packet streams, sensor data, satellite data, a video stream produced by an online video camera, and financial data such as stock-market data.

Big Data Definition

- **First Definition Butler** (2013)
  - Data that has grown to a size that requires new techniques to store, organize, and analyze the data.
- The three "Vs" definition: **volume**, **velocity**, and **variety**.
- Big data has one or more of three key features: **a large volume of data**, **a high velocity** with which the data is created, or a **high degree of variety in the data**.

Big Data Definition (Cont.)

- **Volume** simply means the **amount of data**.
- **Velocity** means that **the data is being created rapidly** (e.g., hundreds of messages per second)
  - Velocity is typically associated with streams.
- **High variety** is usually associated with **unstructured data such as server logs.**
- How Big? generally the term “big data” is used to indicate data above the 100 GB range. Often it deals with hundreds of terabytes (TBs), and possibly a PB or more.

Preconditions

- There are three main preconditions for the rise of big data:
  - **The ability to store large volumes of data in a form that is accessible** (i.e., on hard drives or solid-state drives, not tape drives).
  - **The ability to process big data rapidly at a reasonable cost.**
    - This means inexpensive computing power, particularly in the form of cloud computing.
  - **The existence of producers of big data.**

Big Data Analytics

- The ability to store and process that data into something that is **understandable by humans in a reasonable amount of time** that has allowed the exploitation of that data.

- Successful big data analytics must be able to process the data into something **smaller than the raw data**
  - Allowing an application to present the results in a way that makes sense to a human.

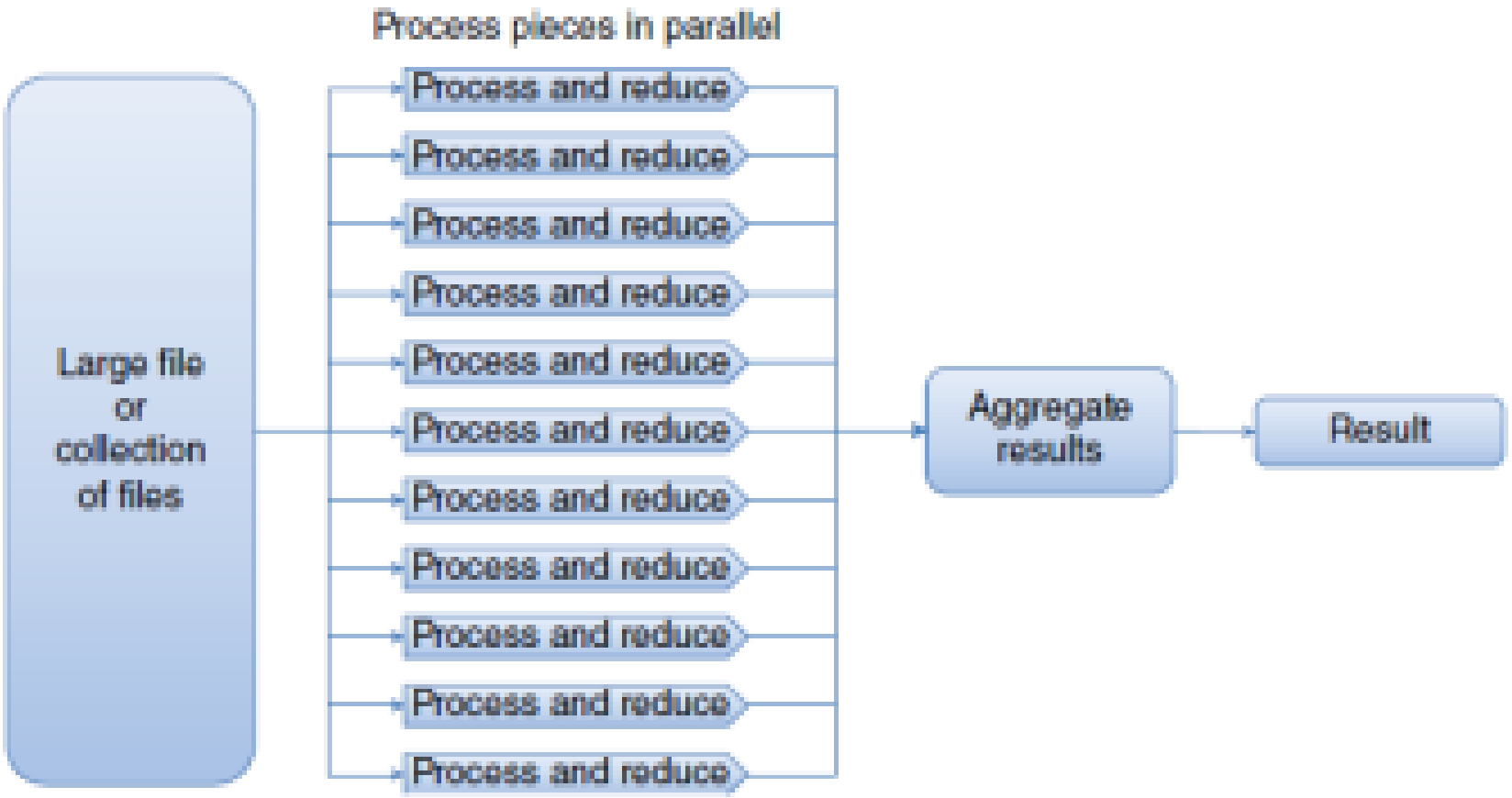
Big Data Analytics (Cont.)

- Because big data is so large, **it normally cannot be processed sequentially in a reasonable amount of time.**

- So the **data is broken up into chunks**, which are analyzed by a set of processes running in parallel.

- The results of the parallel analysis are then **joined** together to create the result.

High-level Big Data Analytics



Technologies for Big Data

- **Storage Systems**
  - Distributed file systems and storage clouds
  - NoSQL Databases

- **Programming Platforms**
  - Map-Reduce: Apache Hadoop, Aneka
  - Stream Processing: Heron, Apache Storm, Apache Spark
  - Graph Processing: Pregel, Apache Giraph

Storage Systems

- Traditionally, **database management systems** constituted the de facto storage support for several types of applications.

- Due to the explosion of unstructured data (e.g., blogs, Web pages, ...), the relational model in its original formulation **does not seem to be the preferred solution for supporting data analytics on a large scale.**

High-performance distributed file systems and storage clouds

- **Distributed file systems** constitute the primary support for data management.
- They provide an interface whereby to store information in the form of files and later access them for read and write.
- Mostly these file systems constitute the data storage support for large computing clusters, supercomputers, massively parallel architectures, and lately, storage/computing clouds.

High-performance distributed file systems and storage clouds

- Lustre
- Google File System (GFS)
- **Hadoop Distributed File System (HDFS)**
- Amazon Simple Storage Service (S3)
- And many more

Hadoop Distributed File System (HDFS)

- HDFS stores **very large files** (many terabytes and petabytes).
- It could store **tens of millions of files**.
- It can run on **hundreds or thousands of commodity servers**.
- A general assumption about HDFS is that **hardware failure is a norm – not an exception**.
- It is suitable for **big data analytics** and has been optimized to write **very big files** once and then to read them many times but it is not suitable for random reads/writes.

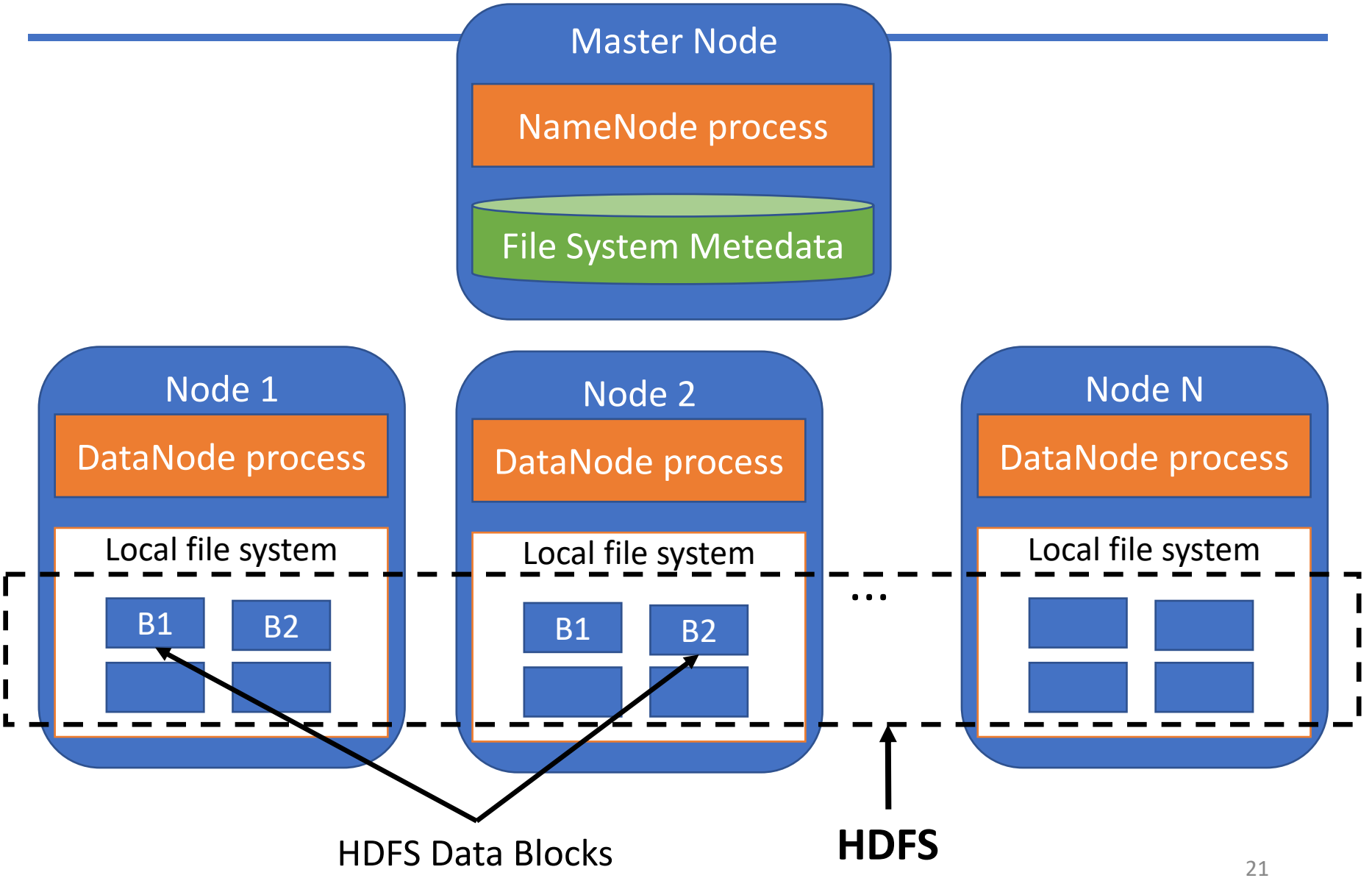
Hadoop Distributed File System (HDFS)

- An HDFS file is a sequence of **blocks stored in a cluster of multiple servers**.
- **Fault tolerance is at block level**.
- HDFS blocks are big ones – 64 MB by default.
- **HDFC is designed for applications that access (streaming) data sets successively**, It is not suitable for small files or for direct reads and writes.
- Hadoop moves the computations to the storage nodes
  - It is the best approach for cases **when the computing programs are relatively small**, and the stored data are big enough.

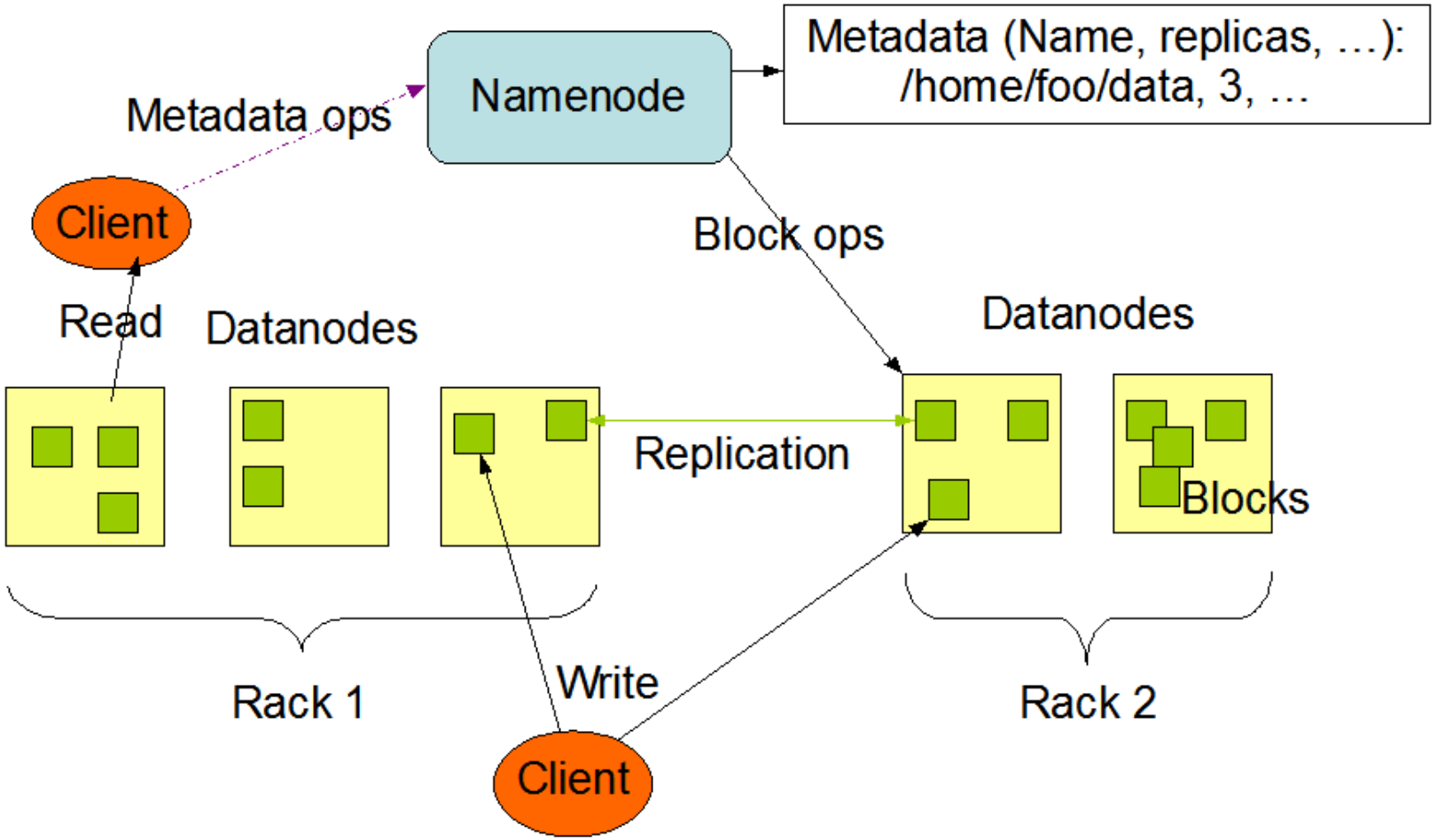
HDFS Structure

- There are *namenode* and *datanode* nodes (see next slide)
  - The *namenode* contains metadata for files, directories, file block locations, and so forth.
  - The *datanode* stores data blocks.
- The client opens files or directories using the metadata from a namenode, after that, the file datanodes execute the operations.
- The read operations directly access datanodes in a sequential read access mode.
- If a read fails, then the datanode uses a block replica.
- When HDFS has read all the data from a given block, it chooses the next block among all next block replicas.

HDFS Architecture



HDFS Architecture (Cont.)



Source: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Programming Platforms for Big Data

- Map-Reduce: **Apache Hadoop**, Aneka
- Stream Processing: Heron, Apache Storm, **Apache Spark**
- Graph Processing: Pregel, Apache Giraph

The MapReduce Programming Model

- MapReduce is a programming platform Google introduced for processing large quantities of data.
- It expresses the computational logic of an application in two simple functions: **map** and **reduce**.
- Data transfer and management are completely handled by the distributed storage infrastructure (i.e., the Google File System), which is in charge of providing access to data, replicating files, and eventually moving them where needed.
- Therefore, developers no longer have to handle these issues and are provided with an interface that presents data at a higher level: **as a collection of key-value pairs.**

The MapReduce Programming Model

- The computation of MapReduce applications is then organized into a workflow of **map** and **reduce** operations that is entirely controlled by the runtime system.
  - **Developers need only specify how the map and reduce functions operate on the key-value pairs.**
- MapReduce model is expressed in the form of the two functions:
  - map(k_1, v_1) → list(k_2, v_2)
  - Reduce(k_2, list(v_2)) → list(v_3)

The MapReduce Programming Model

- Map takes an input pair and produces a set of intermediate key/value pairs.
- The MapReduce library **groups together all intermediate values associated with the same intermediate key *I*** and passes them to the Reduce function.
- The Reduce function accepts an intermediate key ***I*** and a set of values for that key.
- It **merges** together these values to form a possibly smaller set of values.
  - Typically just zero or one output value is produced per Reduce invocation.

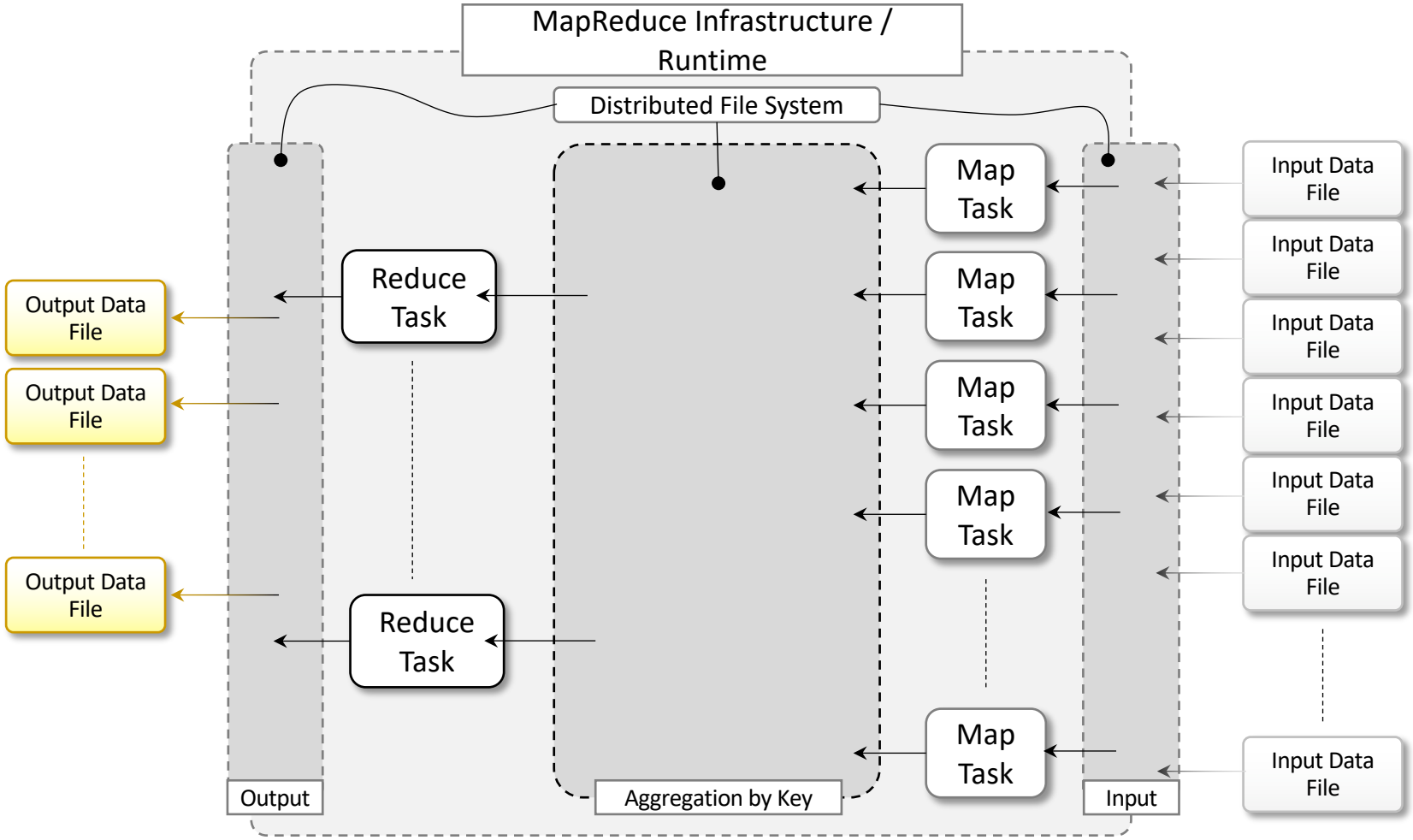
The MapReduce Programming Model

- **The framework signature is**:
  - computation: list(k1, v1) → list(k2, v2)

- **Map Phase**
  - MapPhase: list(k1, v1) → list(k2, v2)

- **Reduce Phase**
  - ReducePhase = preparation; reduction
  - Preparation: list(k2, v2) → list(k2, list(v2))
  - Reduction: list(k2, list(v2)) → list(k2, v3)

MapReduce Framework



MapReduce Example

Problem: counting the number of occurrences of each word in a large collection of documents

```
map(String key, String value):
   // key: document name
   // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1")
```

```
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

More MapReduce Examples

- **Distributed Grep:**
  - Looking for a string pattern in several files
  - Print lines within several files that matches the given pattern
- What is map function?
- What is reduce function?

More MapReduce Examples (Cont.)

- **Distributed Grep:**
  - Looking for a string in several files
  - Print name of files that contains a string
- **Distributed Grep (Solution)**
  - The map function emits a line if it matches a supplied pattern.
  - The reduce function is an identity function that just copies the supplied intermediate data to the output.

More MapReduce Examples (Cont.)

- **Count of URL Access Frequency:**
  - **How many times a URL is accessed given many web logs?**
- What is map function?
- What is reduce function?

More MapReduce Examples (Cont.)

- **Count of URL Access Frequency:**
  - **How many time a URL is accessed given many web logs?**

- **Count of URL Access Frequency (solution)**
  - The map function processes logs of web page requests and outputs <URL,1>.
  - The reduce function adds together all values for the same URL and emits a <URL, total count> pair.

More MapReduce Examples (Cont.)

- **Inverted index:**
  - **Given a set of documents, create the set of {word → list of documents containing the word}.**
- What is map function?
- What is reduce function?

More MapReduce Examples (Cont.)

- **Inverted index:**
  - **Given a set of documents, create the set of {word → list of documents containing the word}.**
- **Inverted Index (Solution)**
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.
  - The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.
  - The set of all output pairs forms a simple inverted index.

The MapReduce Programming Model

- In general, any computation that can be expressed in the form of two major stages can be represented in the terms of MapReduce computation.
  - **Analysis**: This phase operates directly on the data input file and corresponds to the operation performed by the map task.
    - The computation at this stage is expected to be embarrassingly parallel, since map tasks are executed without any sequencing or ordering.
  - **Aggregation**: This phase operates on the intermediate results and is characterized by operations that are aimed at aggregating, summing, and/or elaborating the data obtained at the previous stage to present the data in their final form.
    - This is the task performed by the reduce function.

Apache Hadoop

- Apache Hadoop is the driving force behind the big data industry.
- It is an open-source, Java-based framework.
- It stores data (Hadoop Distributed File System – HDFS) and executes jobs (MapReduce) on large clusters of commodity servers.
- Hadoop is highly fault tolerant and it is scalable from a single server to thousands of servers.

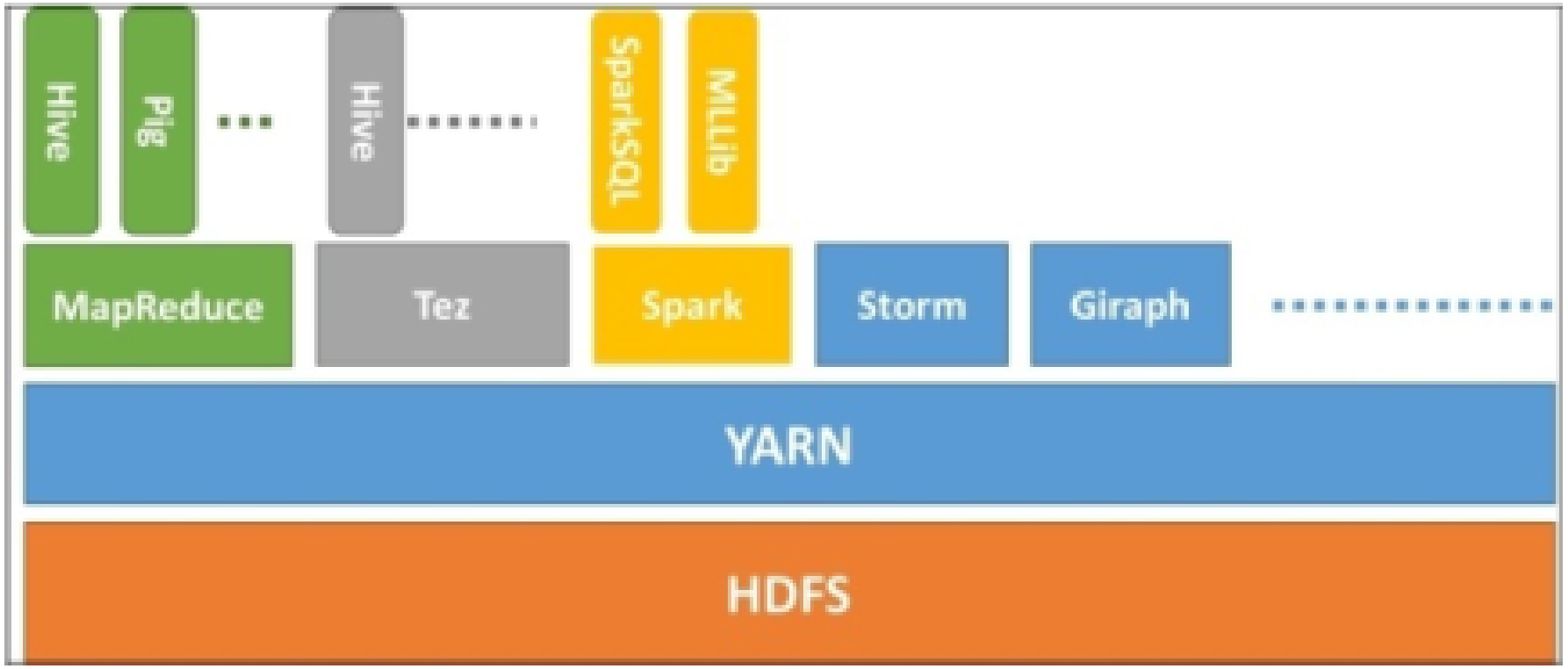
Hadoop Components

- *Common:* utilities supporting the other Hadoop modules, components and interfaces.
- ***MapReduce (YARN):*** a framework for job scheduling and cluster resource management. It is a programming model and an execution engine running on clusters of commodity servers.
- ***Hadoop Distributed File System****:* a distributed file system running on clusters of commodity computers.
- *Hbase:* a distributed, column-oriented database.
- *Sqoop:* a tool for transfer of data between structured data and HDFS.
- *ZooKeeper:* a distributed, highly available coordination service.
- *Pig:* a data-flow language and an execution engine for big data.
- *Ambari:* a Web-based tool for provisioning, managing, and monitoring Hadoop clusters.
- *Hive:* a distributed data warehouse.
- *Cassandra:* a scalable multimaster database with no single points of failure.

Hadoop MapReduce v.2

- **YARN** (**Yet Another Resource Negotiator**) is the major new improvement introduced in Hadoop v2.
- YARN is a resource management system that allows ***multiple distributed processing frameworks*** to effectively share the compute resources of a Hadoop cluster and to utilize the data stored in HDFS.
- YARN is a central component in the Hadoop v2 ecosystem and provides a common platform for many different types of distributed applications.
- Some examples of the current YARN applications include the **MapReduce** framework, **Spark** processing engine, and the Storm real-time stream processing framework, Giraph.

Hadoop YARN


Hive
Pig
Hive
SparkSQL
MLLib
MapReduce
Tez
Spark
Storm
Giraph
YARN
HDFS

Hadoop YARN (Cont.)

- The **YARN ResourceManager** process is the central resource scheduler that manages and allocates resources to the different applications (also known as jobs) submitted to the cluster .

- **YARN NodeManager is a per node process** that manages the resources of a single compute node .

- Scheduler component of the ResourceManager allocates resources in response to the resource requests made by the applications, taking into consideration the cluster capacity and the other scheduling policies that can be specified through the YARN policy plugin framework.

Hadoop YARN (Cont.)

- YARN has a concept called **containers**, which is the unit of resource allocation.
  - Each allocated container has the rights to a certain amount of CPU and memory in a particular compute node.
  - Applications can request resources from YARN by specifying the required number of containers and the CPU and memory required by each container.

- ApplicationMaster is a per-application process that coordinates the computations for a single application.

- The first step of executing a YARN application is to deploy the ApplicationMaster.

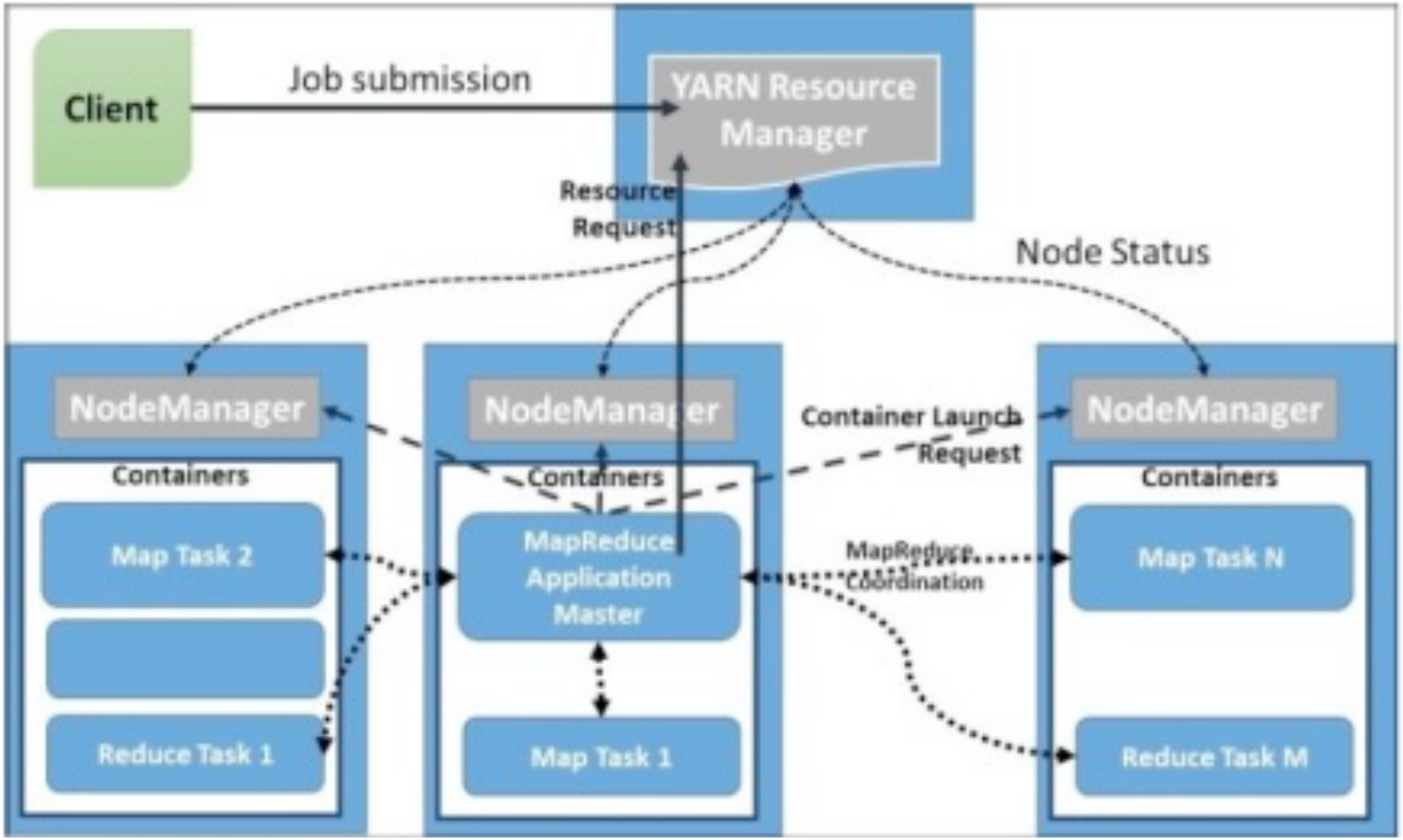
Hadoop YARN (Cont.)

- **After an application is submitted by a YARN client, the ResourceManager allocates a container and deploys the ApplicationMaster for that application.**

- Once deployed, the ApplicationMaster is responsible for requesting and negotiating the necessary resource containers from the ResourceManager.

- Once the resources are allocated by the ResourceManager, ApplicationMaster coordinates with the NodeManagers to launch and monitor the application containers in the allocated resources.

Hadoop MapReduce (Cont.)



Hadoop MapReduce (Cont.)

- A MapReduce **job** is a unit of work that the client wants to be performed.
  - It consists of the input data, the MapReduce program, and configuration information.

- Hadoop runs the job by dividing it into tasks, of which there are two types: **map tasks** and **reduce tasks.**

- Hadoop divides the input to a MapReduce job into fixed-size pieces called **input splits**, or just **splits**.

- Hadoop creates **one map task for each split**, which runs the user defined map function for each record in the split.

Hadoop MapReduce (Cont.)

- For most jobs, a good split size tends to be the size of an HDFS block, 64 MB by default.

- Hadoop does its best **to run the map task on a node where the input data resides in HDFS**.

- **Why does Hadoop do this?**

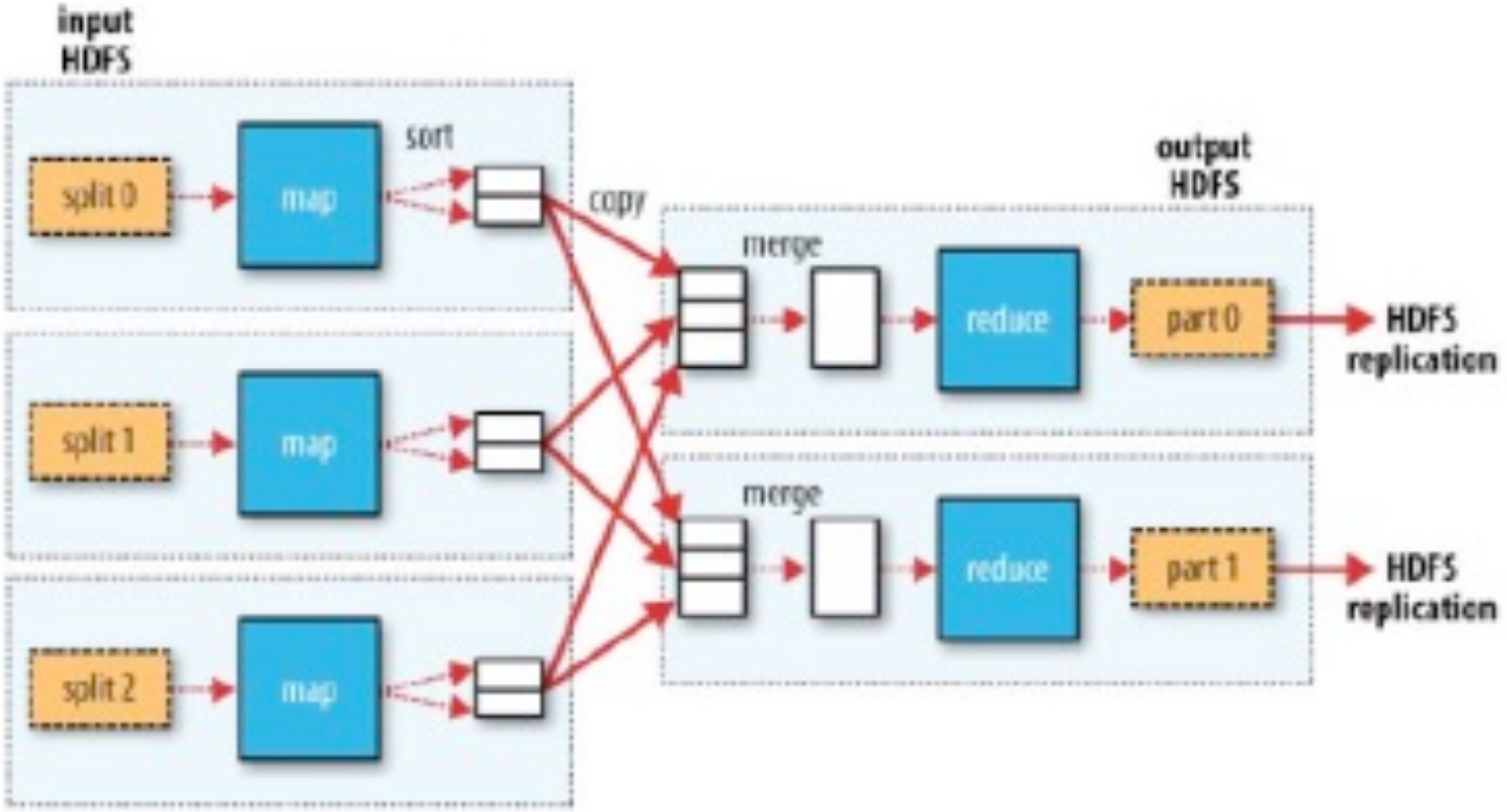
Hadoop MapReduce (Cont.)

- Hadoop does its best **to run the map task on a node where the input data resides in HDFS**.
- This is called the data locality optimization
  - It should now be clear why the optimal split size is the same as the block size.
- **Map tasks write their output to the local disk, not to HDFS**
  - Because the Map output is intermediate output, not the final output.

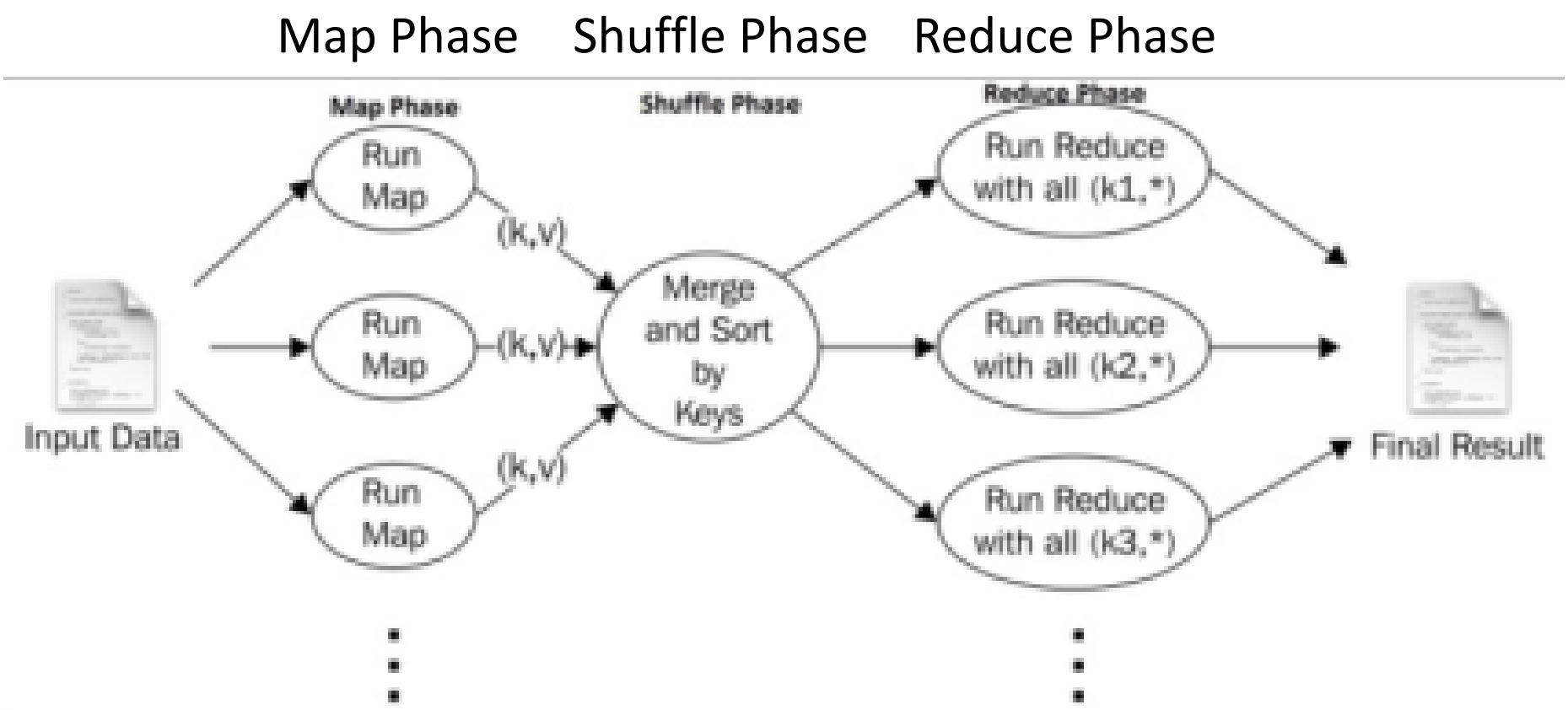
Hadoop MapReduce

- Reduce tasks don't have the advantage of data locality.
- When there are multiple reducers, the map tasks **partition** their output, each creating one partition for each reduce task.
- There can be many keys (and their associated values) in each partition, but the records for **any given key are all in a single partition.**
- The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner works very well.
  - Default partitioner buckets keys using a **hash function**.
- The data flow between map and reduce tasks is colloquially known as "**the shuffle**", as each reduce task is fed by many map tasks.

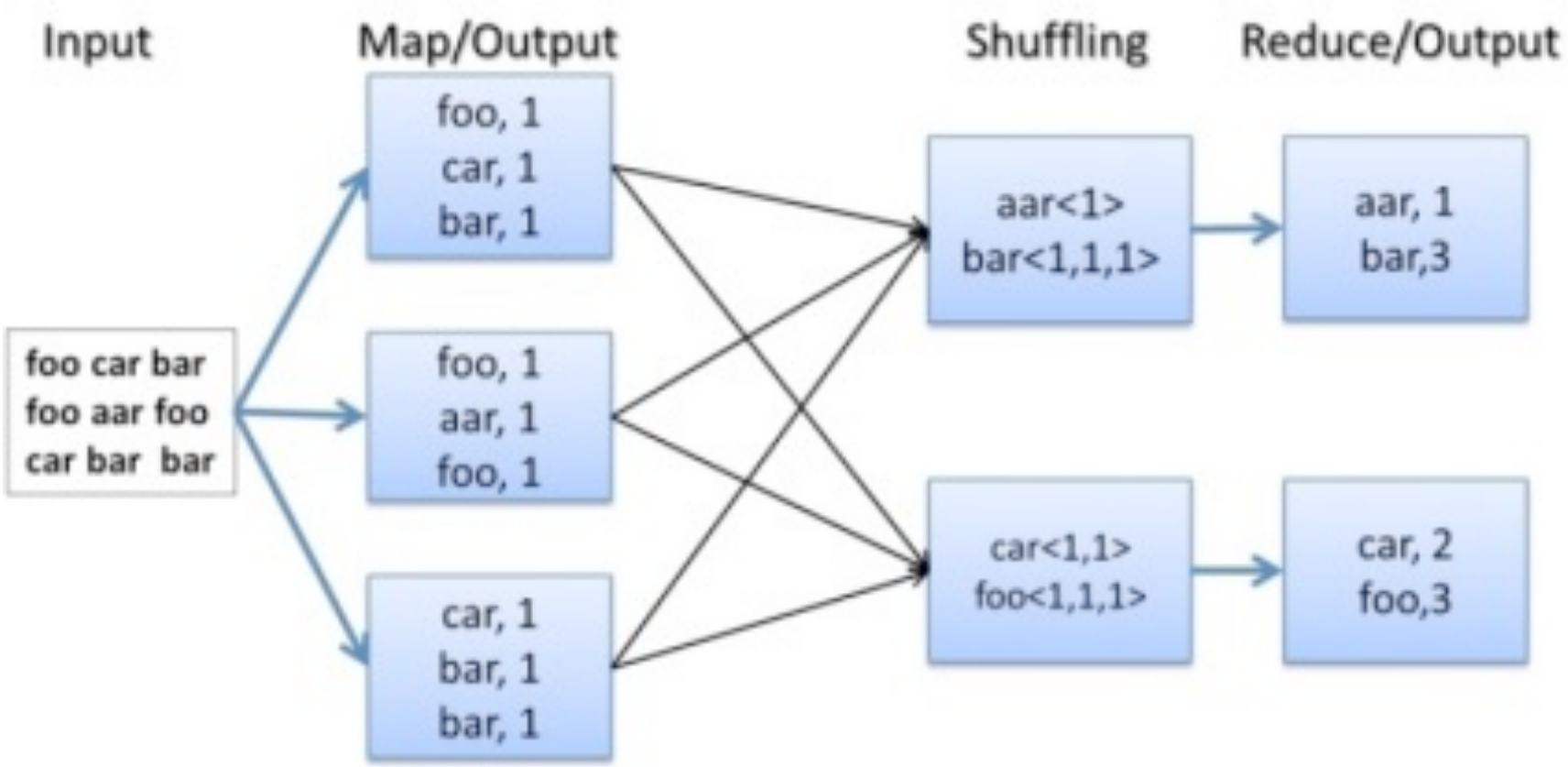
Hadoop MapReduce



Hadoop MapReduce



MapReduce Example



MapReduce on Utility Clouds

- Amazon offers an analytic cloud service called Amazon Elastic MapReduce (EMR) which provides a managed Hadoop MapReduce on Amazon EC2 instances.

- Microsoft also offers a big data service, which is called Windows Azure HDInsight Service, which deploys and provisions ApacheHadoop clusters in the Azure cloud.

- AppEngine-MapReduce is an open-source library for doing MapReduce-style computations on the Google App Engine platform with pricing that is completive with Amazon EMR.