



# **Cloud Computing**

## **MapReduce Programming Model**

Seyyed Ahmad Javadi

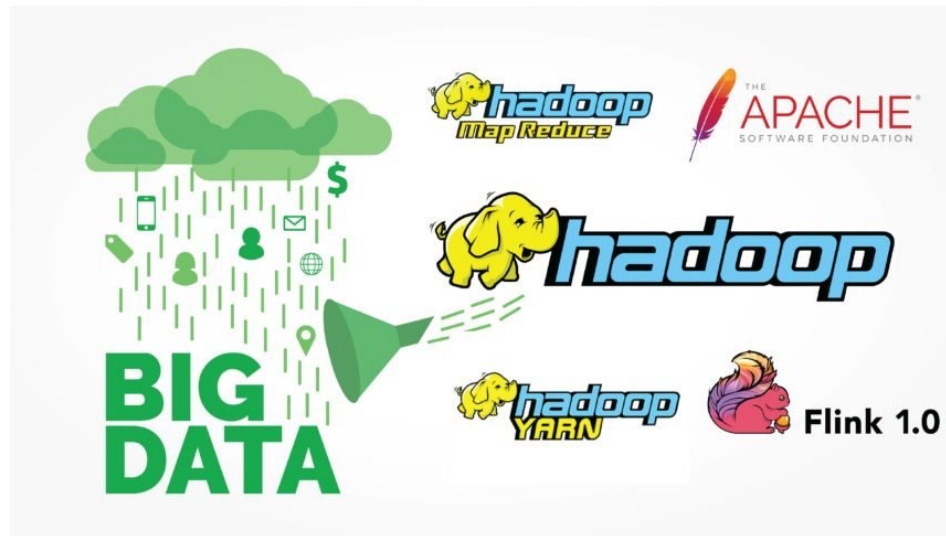
[sajavadi@aut.ac.ir](mailto:sajavadi@aut.ac.ir)

Fall 2022

# 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

---

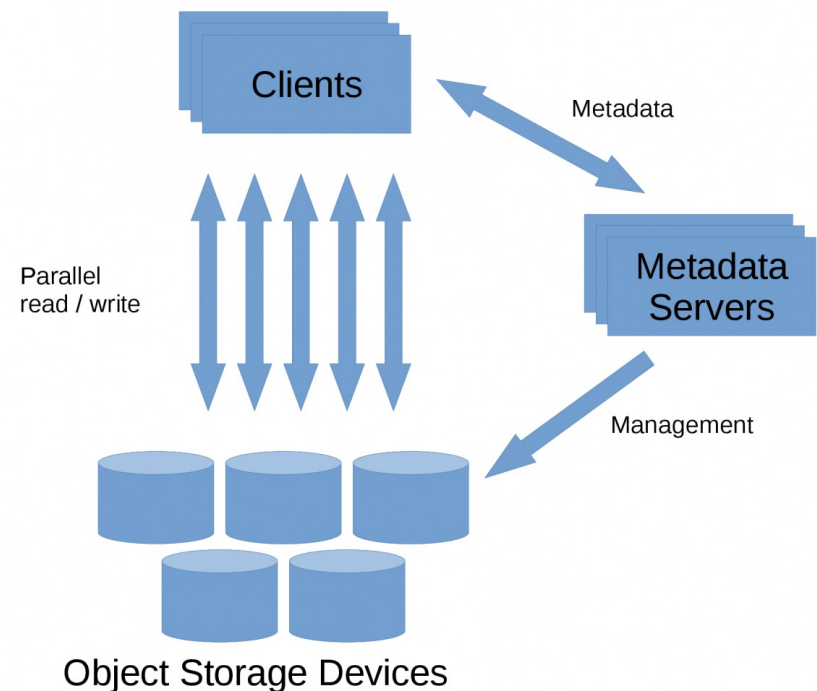
- Introduced by [Google](#) for processing large quantities of data.
- It expresses the computational logic of an application in two simple functions:

- **map**
- **reduce**



# The MapReduce Programming Model

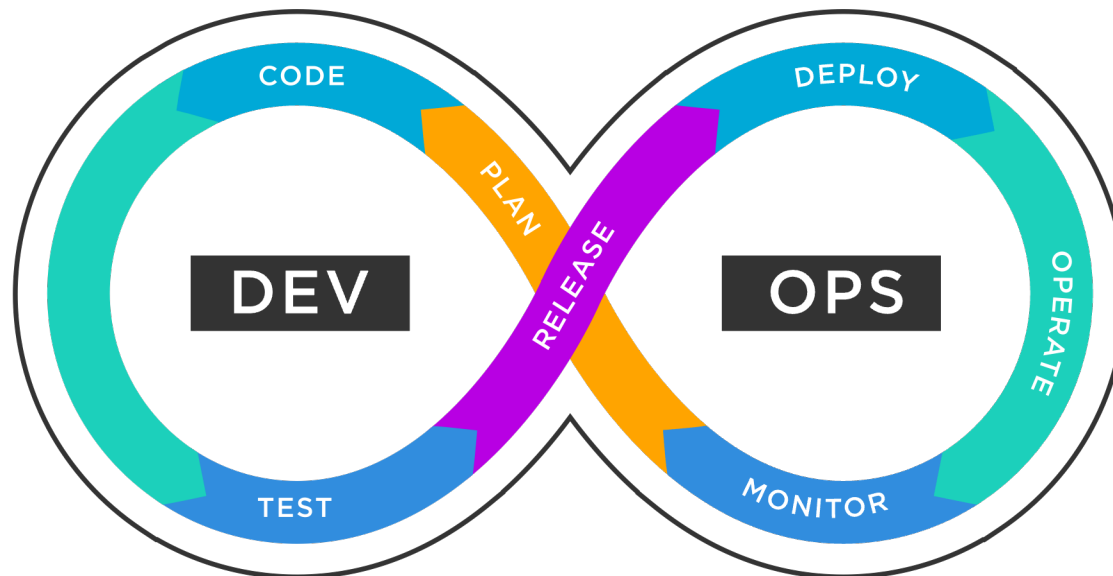
- Data transfer and management are completely handled by the Distributed Storage Infrastructure.
  - E.g., the Google File System
- Distributed Storage is in charge of providing access to data, replicating files, and eventually moving them where needed.



# The MapReduce Programming Model

---

- Developers no longer have to handle storage issues.
- Developers 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.

# The MapReduce Programming Model

---

- MapReduce model is expressed in the form of the two functions:
- $\text{map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$
  - $\text{Reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_3)$

# 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** / and passes them to the Reduce function.



# The MapReduce Programming Model

---

- The Reduce function accepts an intermediate key / 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:  $\text{list}(k1, v1) \rightarrow \text{list}(k2, v2)$

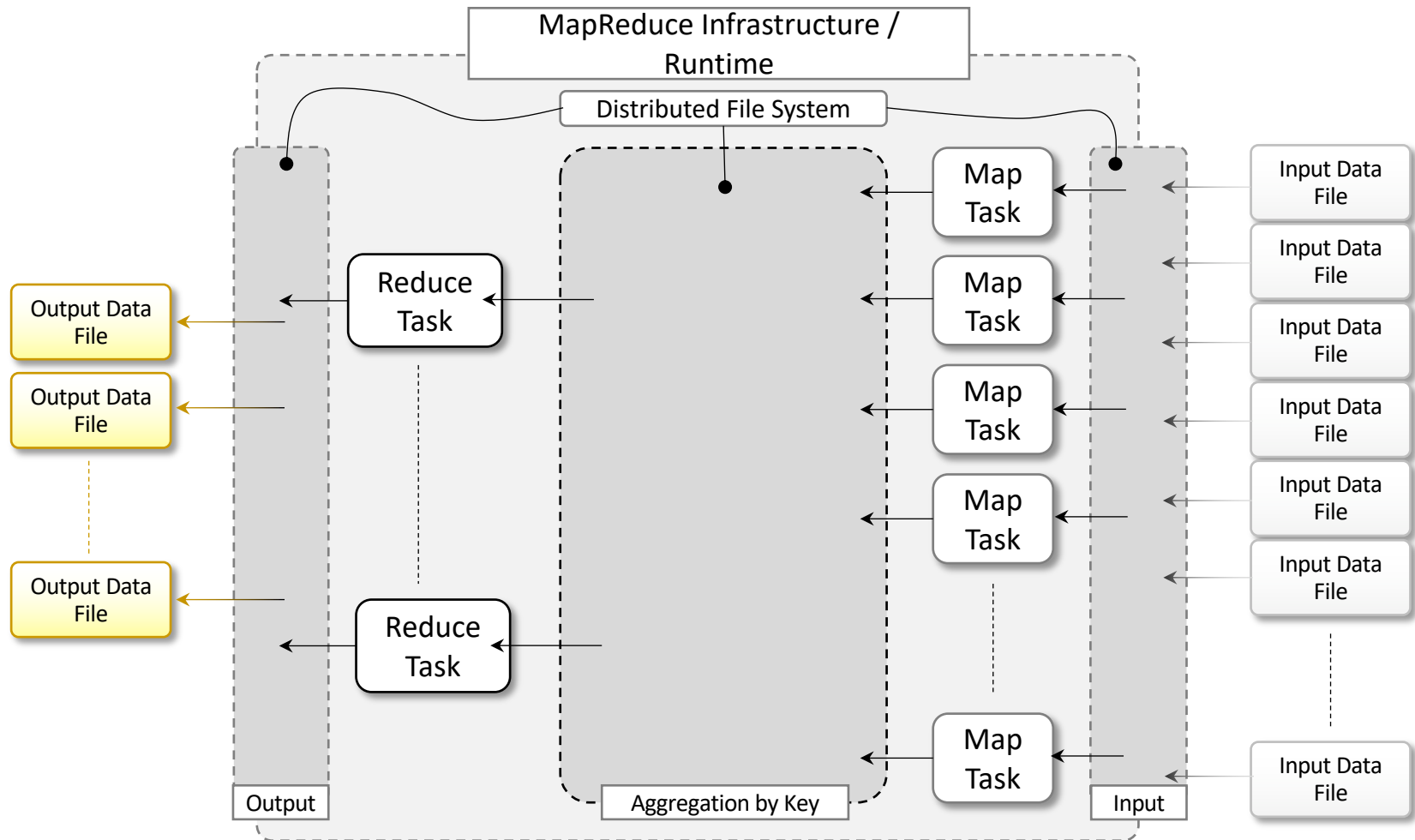
## ➤ Map Phase

- MapPhase:  $\text{list}(k1, v1) \rightarrow \text{list}(k2, v2)$

## ➤ Reduce Phase

- ReducePhase = preparation; reduction
- Preparation:  $\text{list}(k2, v2) \rightarrow \text{list}(k2, \text{list}(v2))$
- Reduction:  $\text{list}(k2, \text{list}(v2)) \rightarrow \text{list}(k2, v3)$

# 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 (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 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  $\langle \text{URL}, 1 \rangle$ .
- The reduce function adds together all values for the same URL and emits a  $\langle \text{URL}, \text{total count} \rangle$  pair.

# 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**
- **Aggregation**



# Analysis

---

- **Analysis** 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

---

- **Aggregation** 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.