High-level Design of Spark

Get introduced to the primary building blocks and programming model of Spark.

We'll cover the following

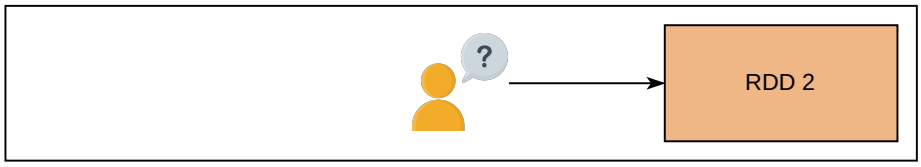
- Building blocks
  - Resilient distributed datasets (RDDs)
  - Driver
  - Worker nodes
- Programming model
  - Spark programming interface
    - RDD abstraction
    - Driver and worker
    - Operations
    - Persistence

Building blocks

Building blocks of Spark include resilient distributed datasets, driver, and worker nodes. The details of these components have been described briefly in this lesson.

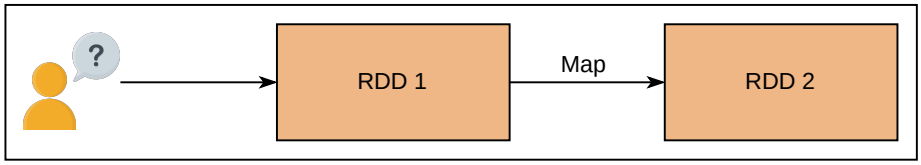
Resilient distributed datasets (RDDs)

- They are an abstraction, a read-only collection of resilient objects stored across a cluster of machines.
- RDDs can be created in two ways—by applying transformation on an existing RDD or by reading data from a distributed file system.
  - Whenever an RDD is created, it has partitions of data in it.
  - Those partitions are saved on a cluster of machines.
- For example, let's say an RDD is initially created from a file, then a subsequent RDD is created from that RDD, and so on.
- Spark will keep a graph that records the sources of all the RDDs called a **lineage graph**.



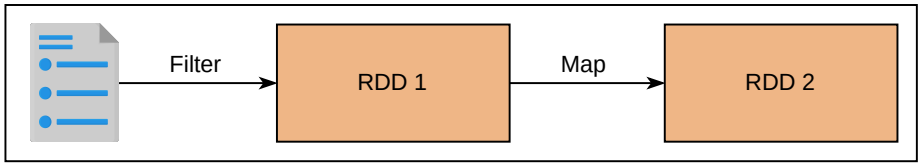
Source of RDD 2?

1 of 3



Source of RDD 1?

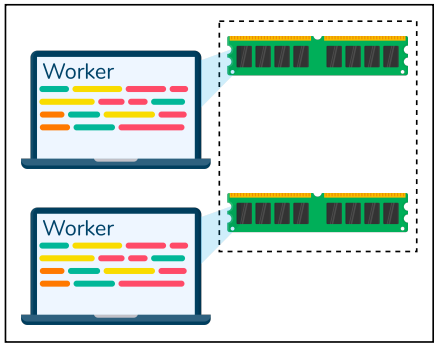
2 of 3



Complete lineage graph

3 of 3

- RDDs implement an interface that keeps the following details:
  - A list of partition objects that contains their own sets of data
  - An iterator that traverses the data in a partition
  - A list of worker nodes where the partition can be quickly accessed ensures task scheduling on the appropriate workers



RDD consisiting of memory of two workers

The abstraction provided by RDD resembles distributed shared memory where only a part of data is present on a node. To get good performance from this model of computation, we run processing on local nodes (instead of bringing required data to a computation on a specific node), mostly as **single program multiple data** paradigm (for example, running a `Map` function on all the data on all the cluster nodes present on the part of data held by the node).

Driver

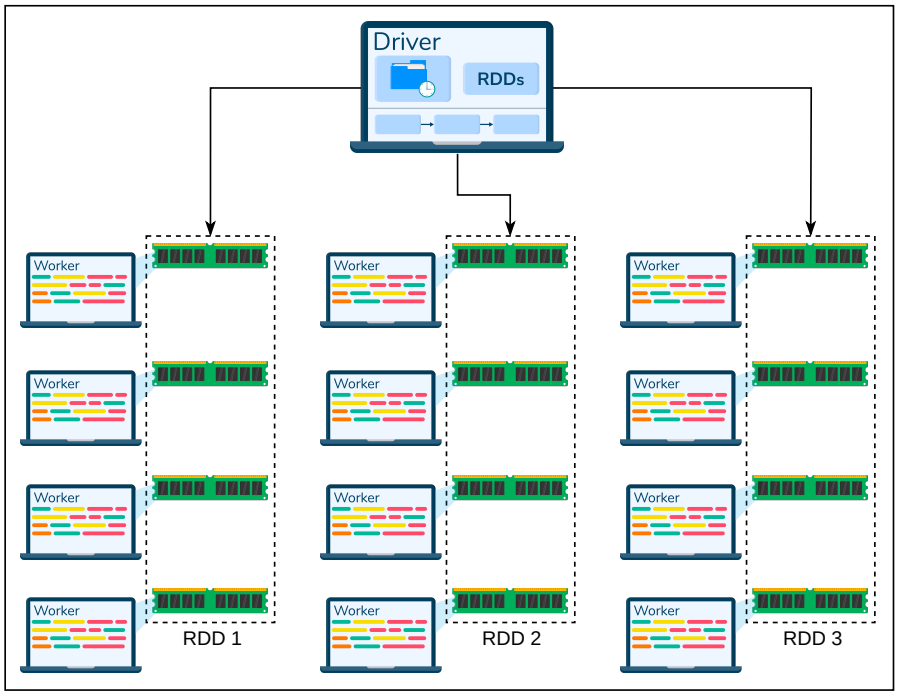
A **driver** is like a manager that orchestrates data sharding among the workers, actual data processing, and reporting the results back to the client.

- It launches a cluster of workers.
- It defines RDDs.
- It keeps a lineage graph of RDDs.
- It creates stages of execution of a program.
- It creates further tasks in each stage and sends them to the worker nodes working in parallel.
- It schedules the tasks.
- It shares some variables with all the worker nodes called shared variables and keeps track of these variables.

Worker nodes

Worker nodes are the workhorse of our system, where actual processing happens on the part of RDD data they hold. Workers are connected with each other on a high-speed network.

- They receive tasks from the driver.
- They perform the assigned tasks.
- They can store partitions of distributed memory abstraction computation in their RAM/drive.



Spark architecture

> **Note:** The network topology inside a data center (the way workers are connected with each other) has evolved over the years. Usually asymmetric bandwidths are available in a data center (more bandwidth for communication with machines that are connected via the same top-of-the-

> rack switch than communication between two workers who are on different racks). Usually, scheduling algotithms can take the actual network topology into account. Cluster with over-subscribed netwroks might exhibit peculiar performance artifacts on certain operations.

Programming model

The programming model of Spark consists of the following:

- **Resilient distributed datasets (RDDs):** They are memory abstractions of a cluster of worker nodes.
- **Parallel operations:** They can be applied to the RDDs through a function.

Spark programming interface

Spark uses the driver to perform all the tasks. Its API design is explained as follows:

RDD abstraction

Spark makes RDDs through a language-integrated API, where an object represents each RDD, and transformations are applied using various methods. Programmers can define one or more RDDs with the help of transformations on disk or some other RDDs (i.e., map and join). These RDDs can then be used to perform actions that return a value or data (e.g., count, collect, and save).

Read: Spark can read data from a distributed file system like HDFS. The `read` function loads the data from the disks into the memories of all the worker nodes.

```
read("hdfs://...")
```

Driver and worker

Programmers must write a driver program to use Spark and utilizing multiple worker nodes. The driver defines RDDs and invokes actions on them to get desired values or a dataset. The code also keeps a tab on the lineage graph of RDDs. Workers can keep the RDD partitions in their memory for future operations.

Operations

Programmers can perform multiple operations on RDD by providing arguments like `filter()`.

```
filter(data)
```

Persistence

If programmers want some partitions to be kept in the memory for future operations, they can call a `persist()` method. By default, Spark stores persistent RDDs in memory, but it can also store them in

storage if we don't have enough space in memory.

Persist: Spark can save intermediary results in memory for future operations.

```
persist(data/results)
```

Save: Spark can save intermediary results on the disk.

```
save(results)
```

> **Note:** The word **persistence** is usually used for data that can tolerate power cycles (for example data on disks). However, Spark's use of the word persistence is different because RDD data is primarily in the RAM of the worker machines only (until it is explicitly saved by the programmer, or implicitly spills to the disk due to shortage of RAM).

← Back
Requirements of Spark

Next →
Resilient Distributed D...

Mark as Completed