Resilient Distributed Datasets

Tushar B. Kute,
http://tusharkute.com

Resilient Distributed Datasets

Tushar B. Kute,
http://tusharkute.com

Resilient Distributed Datasets

- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark.
- It is an immutable distributed collection of objects.
- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.
- RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Resilient Distributed Datasets

- Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs.
- RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- There are two ways to create RDDs – parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Resilient Distributed Datasets

- Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations.
- Let us first discuss how MapReduce operations take place and why they are not so efficient.

Data Sharing in MapReduce

- MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster.
- It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

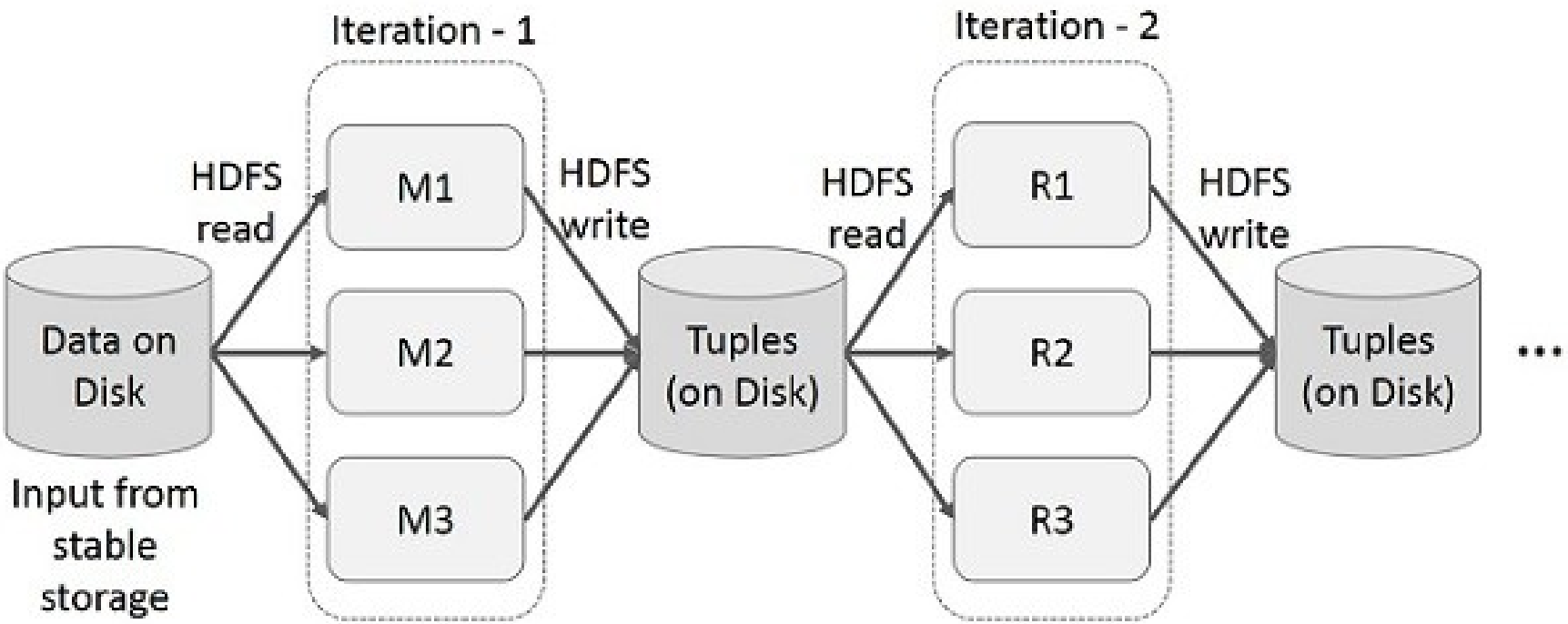
Data Sharing in MapReduce

- Unfortunately, in most current frameworks, the only way to reuse data between computations (Ex – between two MapReduce jobs) is to write it to an external stable storage system (Ex – HDFS).
- Although this framework provides numerous abstractions for accessing a cluster's computational resources, users still want more.
- Both Iterative and Interactive applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to replication, serialization, and disk IO.
- Regarding storage system, most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Iterative Operations on MapReduce

- Reuse intermediate results across multiple computations in multi-stage applications.
- The following illustration explains how the current framework works, while doing the iterative operations on MapReduce.
- This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.

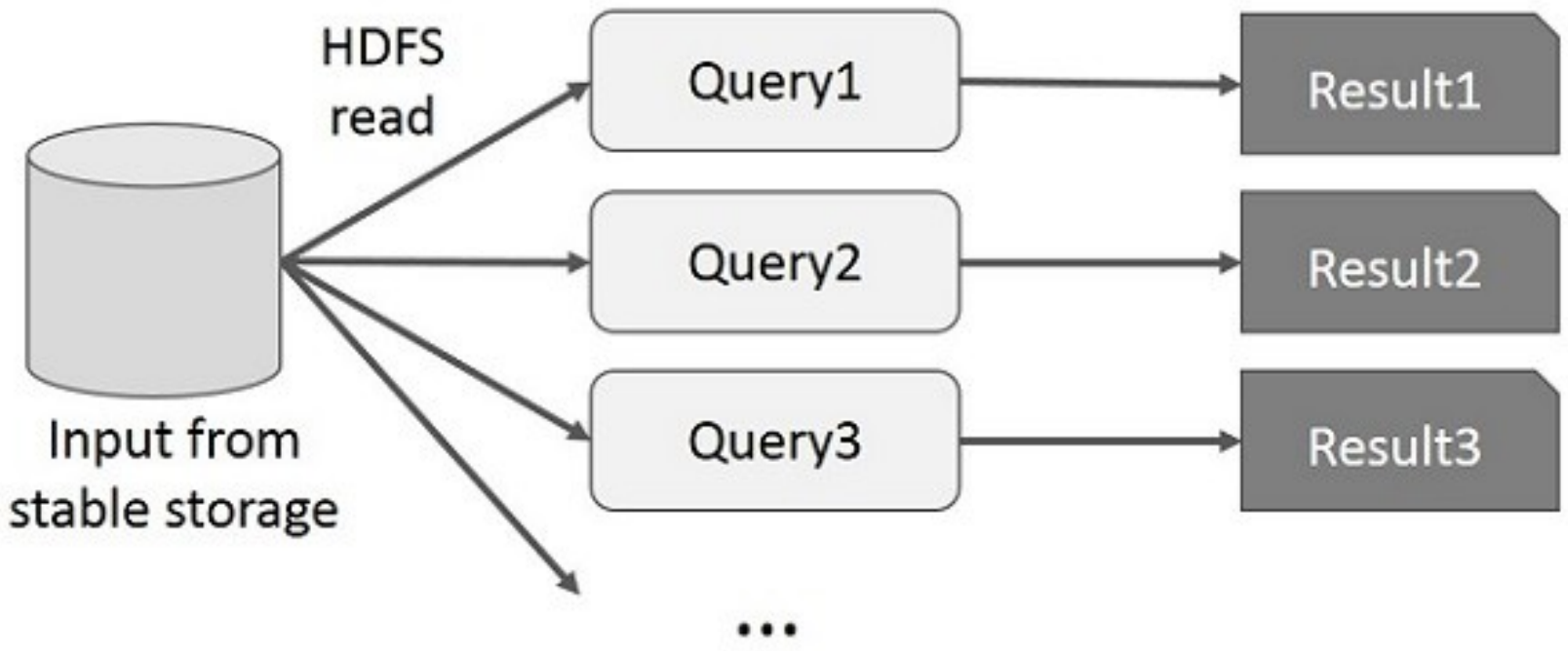
Iterative Operations on MapReduce



Interactive Operations on MapReduce

- User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominate application execution time.
- The following illustration explains how the current framework works while doing the interactive queries on MapReduce.

Interactive Operations on MapReduce



Data Sharing using Spark RDD

- Data sharing is slow in MapReduce due to replication, serialization, and disk IO.
- Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.
- Recognizing this problem, researchers developed a specialized framework called Apache Spark.

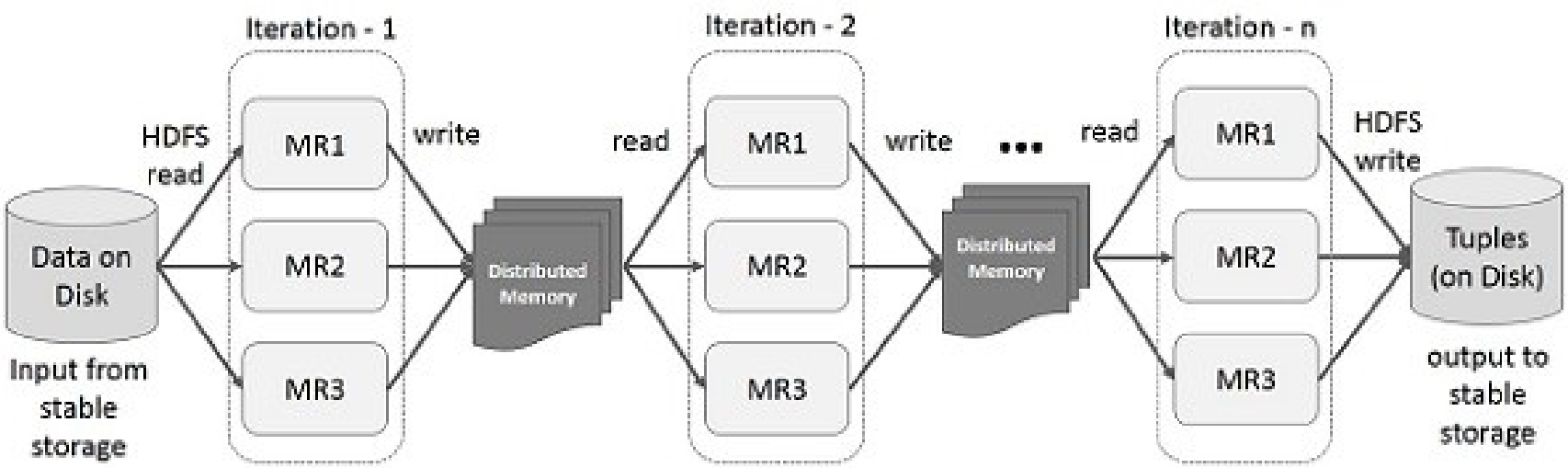
Data Sharing using Spark RDD

- The key idea of spark is Resilient Distributed Datasets (RDD); it supports in-memory processing computation.
- This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs.
- Data sharing in memory is 10 to 100 times faster than network and Disk.

Iterative Operations on Spark RDD

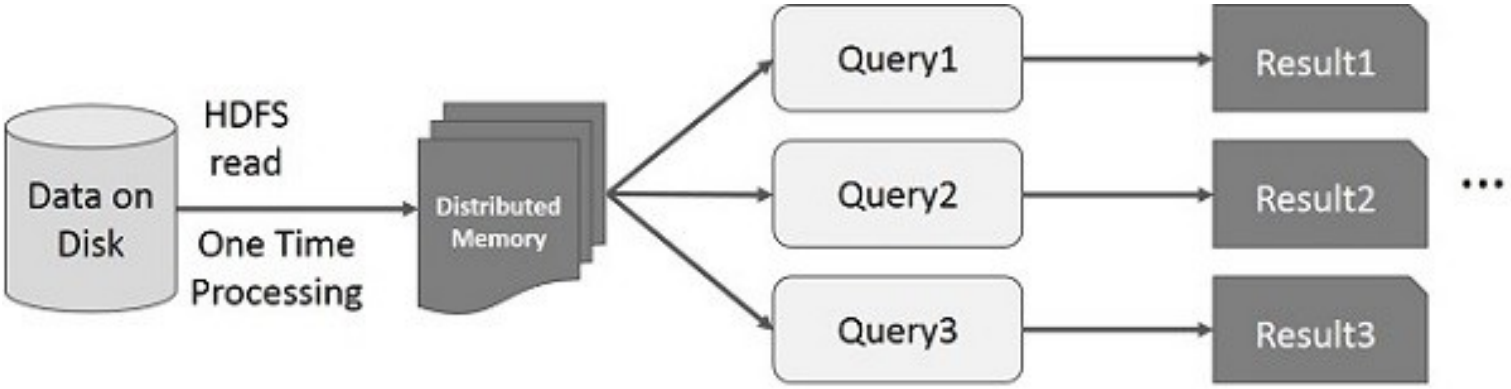
- The illustration given below shows the iterative operations on Spark RDD.
- It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.
- Note – If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.

Iterative Operations on Spark RDD



Interactive Operations on Spark RDD

- This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



Interactive Operations on Spark RDD

- By default, each transformed RDD may be recomputed each time you run an action on it.
- However, you may also persist an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it.
- There is also support for persisting RDDs on disk, or replicated across multiple nodes.

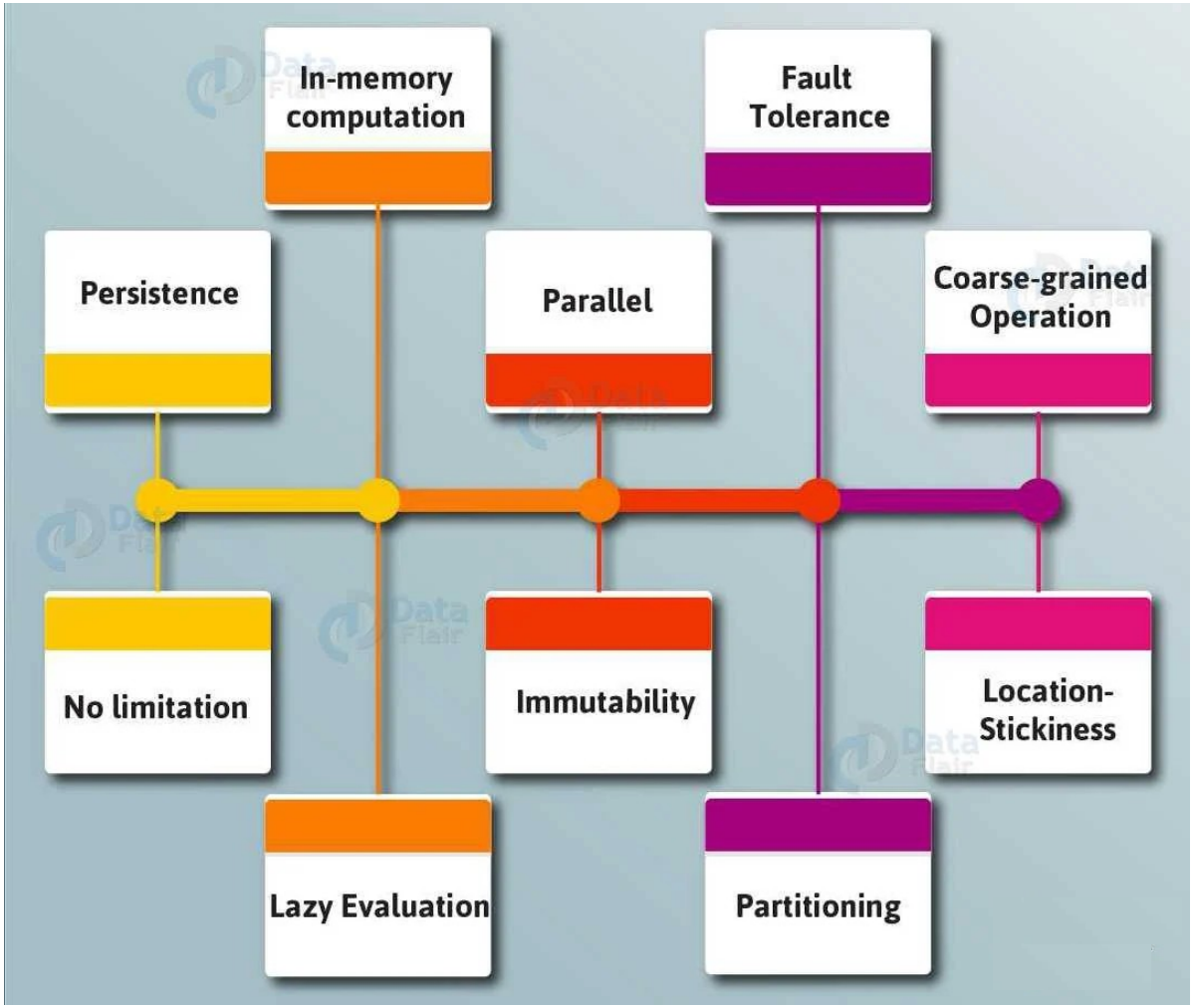
Ways to create Spark RDD

- Parallelized collections
  - By invoking parallelize method in the driver program, we can create parallelized collections.
- External datasets
  - One can create Spark RDDs, by calling a textFile method. Hence, this method takes URL of the file and reads it as a collection of lines.
- Existing RDDs
  - Moreover, we can create new RDD in spark, by applying transformation operation on existing RDDs.

Spark RDDs operations

- There are two types of operations, which Spark RDDs supports:
- Transformation Operations
  - It creates a new Spark RDD from the existing one. Moreover, it passes the dataset to the function and returns new dataset.
- Action Operations
  - In Apache Spark, Action returns final result to driver program or write it to the external data store.

RDD Features



RDD Features

- In-memory computation
  - Basically, while storing data in RDD, data is stored in memory for as long as you want to store.
  - It improves the performance by an order of magnitudes by keeping the data in memory.

RDD Features

- Lazy Evaluation
  - Spark Lazy Evaluation means the data inside RDDs are not evaluated on the go.
  - Basically, only after an action triggers all the changes or the computation is performed. Therefore, it limits how much work it has to do

RDD Features

- Fault Tolerance
  - If any worker node fails, by using lineage of operations, we can re-compute the lost partition of RDD from the original one. Hence, it is possible to recover lost data easily. Learn Fault Tolerance in detail.
- Immutability
  - Immutability means once we create an RDD, we can not manipulate it. Moreover, we can create a new RDD by performing any transformation. Also, we achieve consistency through immutability.

RDD Features

- Persistence
  - In in-memory, we can store the frequently used RDD. Also, we can retrieve them directly from memory without going to disk. It results in the speed of the execution.
  - Moreover, we can perform multiple operations on the same data.
  - It is only possible by storing the data explicitly in memory by calling persist() or cache() function.

RDD Features

- Partitioning
  - Basically, RDD partition the records logically. Also, distributes the data across various nodes in the cluster.
  - Moreover, the logical divisions are only for processing and internally it has no division. Hence, it provides parallelism.
- Parallel
  - While we talk about parallel processing, RDD processes the data parallelly over the cluster.

RDD Features

- Location-Stickiness
  - To compute partitions, RDDs are capable of defining placement preference. Moreover, placement preference refers to information about the location of RDD.
  - Although, the DAGScheduler places the partitions in such a way that task is close to data as much as possible. Moreover, it speeds up computation.

RDD Features

- Coarse-grained Operation
  - Generally, we apply coarse-grained transformations to Spark RDD. It means the operation applies to the whole dataset not on the single element in the data set of RDD in Spark.
- Typed
  - There are several types of Spark RDD. Such as: RDD [int], RDD [long], RDD [string].
- No limitation
  - There are no limitations to use the number of Spark RDD. We can use any no. of RDDs. Basically, the limit depends on the size of disk and memory.

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com