Unit # 3

- Job scheduling , YARN.
- Failures in classic Map-reduce and YARN,
- Anatomy of Map Reduce job run,
- Classic Map-reduce.
- Shuffle & Sort

Job Scheduling in Hadoop

We know now that Hadoop is a distributed computing framework that need to handle many jobs simultaneously on a distributed cluster of many machines (nodes). Thus needless to say, it needs a mechanism to decide which job runs where and when on the available resources. If we try to see the responsibility of the job scheduler it must ensure minimum following:

- Cluster resources (CPU, memory, disk) are used efficiently.
- Jobs don't conflict or overload certain nodes in the cluster.
- Jobs are processed in a fair, timely manner, especially when multiple users submit jobs simultaneously.

This responsibility in Hadoop till version <2.0 was assumed by Master Node also known as **Job Tracker**. It was solely responsible to manages the entire lifecycle of Map Reduce jobs, from submission to completion. The previous versions had a very simple way. Generally, they ran in order of submission using a Hadoop FIFO scheduler. The order is as described below :

1. Job Submission:

A user submits a job (e.g., a Map Reduce job) with its configuration (input data, output path, number of map/reduce tasks, etc.) to the **Job Tracker**.

2. Job Initialisation:

The **Job Tracker** then splits the input data. Each split is assigned to a map task, which will be scheduled to run on one of the **Task Trackers**. The Job Tracker also schedules reduce tasks, which will start after the map phase produces intermediate data.

Job Scheduling in Hadoop (contd)

3. Task Assignment :

Task Trackers periodically keep sending a heartbeats to the Job Tracker, reporting the number of free slots. The JobTracker assigns map and reduce tasks to TaskTrackers as soon as it finds any Task tracker having a free slot.

The JobTracker tries to take full care to schedule map tasks on the same node where the data resides (or as close as possible), reducing network overhead and improving performance.

4. Task Execution :

Once the JobTracker assigns a task to a TaskTracker, the TaskTracker launches the task. It monitors the task's progress and reports back to the JobTracker.

5. Task Completion :

After the map and reduce tasks complete, the TaskTrackers send the results back to the JobTracker, which marks the job as finished and informs the client of its completion.

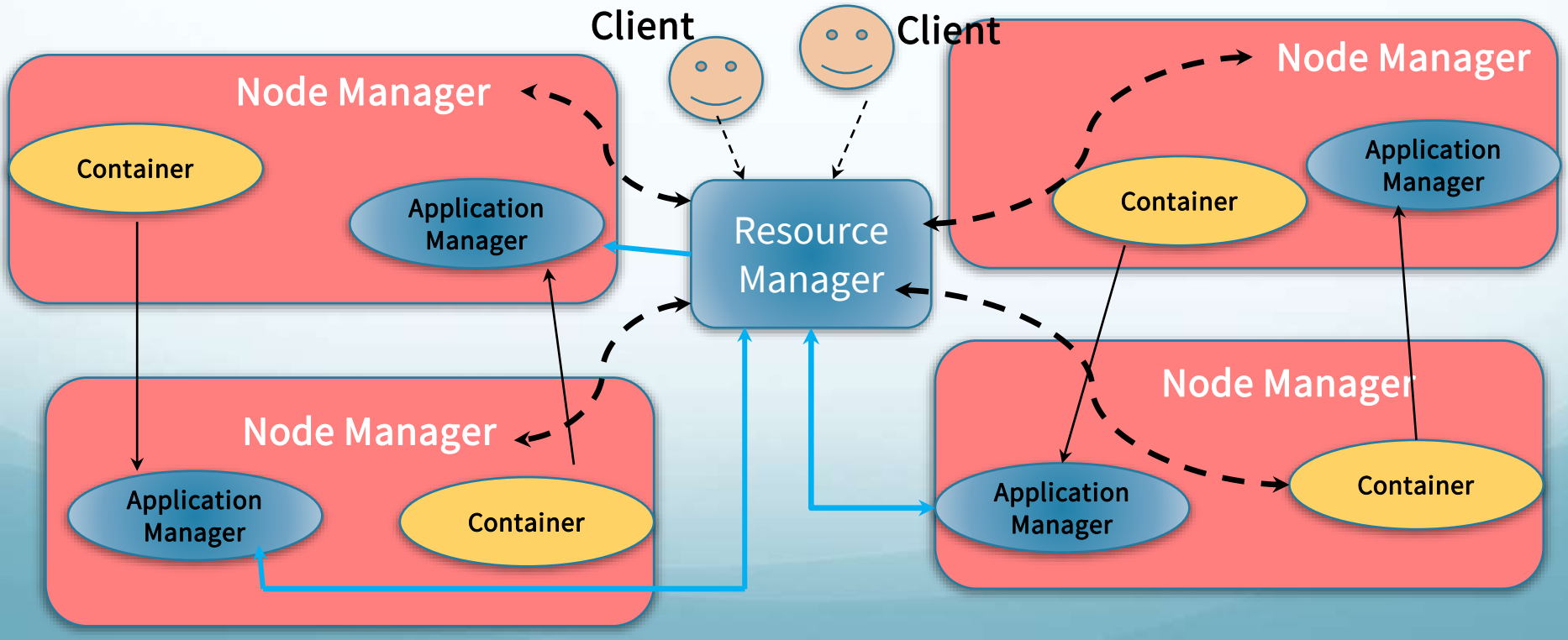
This simple method was also known as Map-Reduce Framework way for scheduling and monitoring the tasks and was used prior to Hadoop 2.

Note: Practical -3 objective is to find out the states of task which can be any of the above state in Hadoop.

YARN (Yet Another Resource Scheduler)

Since Hadoop 2, we have YARN (Yet Another Resource Negotiator). It forms a core component of the Hadoop ecosystem in Hadoop 2.x. The basic idea behind the YARN is introduction of separate daemons , *ResorceManager*, *ApplicationMaster*, and *NodeManager* as shown and explained on subsequent slides.

This architecture of Hadoop 2.x YARN provides a general purpose data processing platform which is not just limited to the **MapReduce** discussed in previous slides. Rather, it allows different data processing engines like graph processing, interactive processing, stream processing ,batch processing etc to run and process data stored in HDFS.



YARN (contd)

1. ResourceManager

The ResourceManager is the master daemon responsible for managing and allocating cluster resources across different distributed applications. It is the main authority for resource allocation within the YARN framework as seen in the block diagram. Resource Manager has Scheduler as main constituent running. The scheduler is responsible for allocating the resources to the running application. The scheduler is pure scheduler it means that it performs no monitoring no tracking for the application and even doesn't guarantees about restarting failed tasks either due to application failure or hardware failures. There are three types of Schedulers . Its discussed on subsequent slide.

2. NodeManager

The NodeManager is the worker daemon running on each node in the cluster. It is responsible for executing tasks, monitoring resource usage (CPU, memory, disk) of the running containers and reporting the status back to the ResourceManager. They manages all containers and tracks the health of the node itself by continously sending the heartbeat. NodeManager also manages auxiliary services running on cluster for example like sorting and shuffling lwhich are generally loaded by the NM during startup.

YARN (contd)

3. ApplicationMaster

For each application running in YARN , there is an ApplicationMaster that manages the application's lifecycle, including resource requests, task execution, and failure handling. One application master runs per application. The AM acquires containers from the RM's Scheduler and coordinate with corresponding NMs to start the application's individual tasks.

It negotiates resources with the ResourceManager and manages the allocation of containers from the NodeManagers to execute tasks in the containers and handles any task failures.

4. Containers

A Container is the basic unit of execution in YARN, representing a bundle of physical resources (CPU, memory, and disk) allocated on a node for running an application task. The ApplicationMaster requests containers from the ResourceManager, and once allocated, the NodeManager launches containers where tasks (like mappers, reducers, or other jobs) are executed.

YARN (contd)

Types of Hadoop Scheduler

There are mainly 3 types of Schedulers in Hadoop:

1. FIFO (First In First Out) Scheduler.
2. Capacity Scheduler.
3. Fair Scheduler.

1. FIFO (First In First Out) Scheduler.

As the name suggests FIFO i.e. First In First Out, so the tasks or application that comes first will be served first. This is the default Scheduler used in Hadoop. The tasks are placed in a queue and the tasks are performed in their submission order. In this method, once the job is scheduled, no intervention is allowed. So sometimes the high-priority process has to wait for a long time since the priority of the task does not matter in this method.

Advantage:

- No need for configuration
- First Come First Serve
- simple to execute

Disadvantage:

- Priority of task doesn't matter, so high priority jobs need to wait
- Not suitable for shared cluster

YARN (contd)

2. Capacity Scheduler.

In Capacity Scheduler we have multiple job queues for scheduling the tasks. The Capacity Scheduler divides resources into multiple queues, each with a configured capacity. Different users can be assigned to different queues. The scheduler guarantees that each queue will not exceed its resource allocation but can use spare capacity from other queues when available.

Advantage:

- Best for working with Multiple clients or priority jobs in a Hadoop cluster
- Maximizes throughput in the Hadoop cluster

Disadvantage:

- More complex
- Not easy to configure for everyone

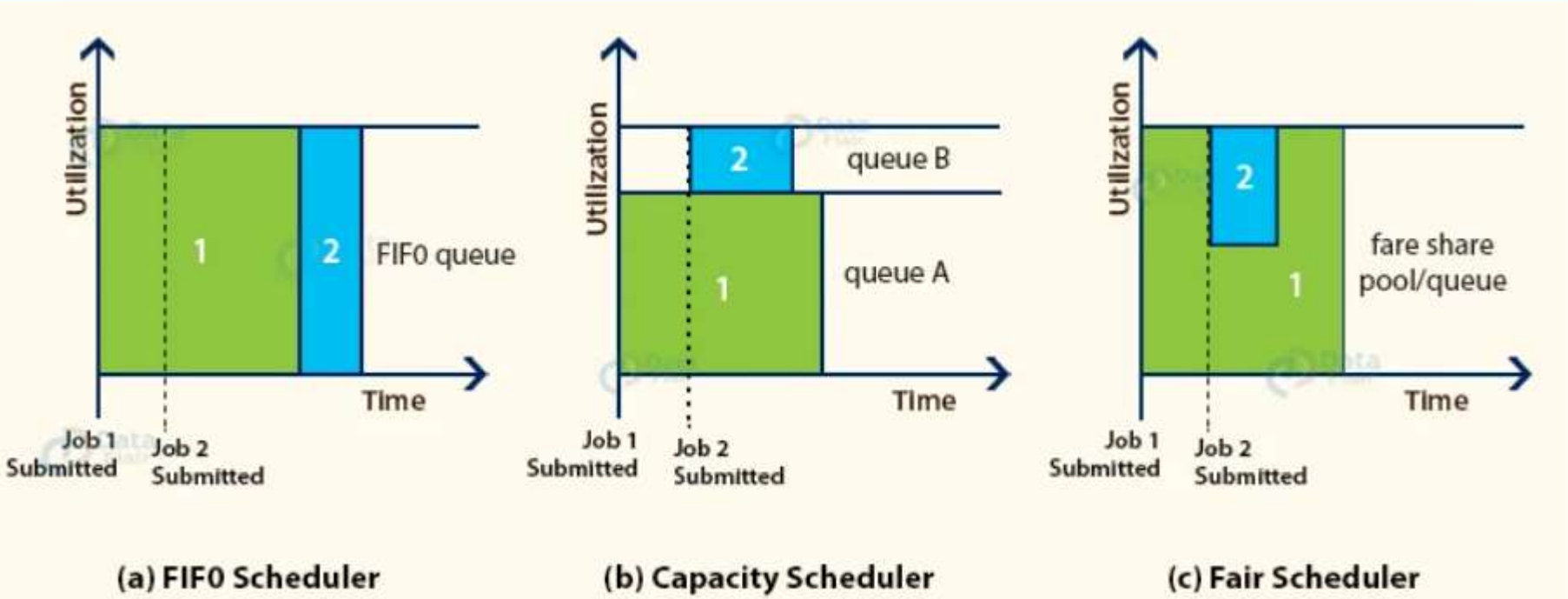
3. Fair Scheduler.

Fair Scheduler is very much similar to that of the capacity scheduler except that it ensure that all applications get an equal share of cluster resources. When the single application is running, then that app uses the entire cluster resources. When other applications are submitted, the free up resources are assigned to the new apps so that every app eventually gets roughly the same amount of resources. FairScheduler enables short apps to finish in a reasonable time without starving the long-lived apps.

YARN (contd)

Advantage:

- Best for working with Multiple clients or priority jobs in a Hadoop cluster , BUT
- it can limit the concurrent running task in a particular pool or queue.



Failures in classic Map-reduce and YARN

In both classic MapReduce (Hadoop 1.x) and YARN-based MapReduce (Hadoop 2.x and beyond), failures are inevitable when running large-scale distributed computations due to various reasons like hardware failures, software bugs, or network issues. Hadoop is designed to handle these failures gracefully, ensuring that tasks can be re-executed or rerouted to different nodes without the entire job failing.

The different types of Failures in classic Map-reduce and YARN and their mitigation methods are listed and described below :

1. Failures in classic Map-reduce

As we have seen in previous slides that , In Hadoop 1.x (referred to as classic MapReduce), the job execution and resource management were tightly coupled into the JobTracker and TaskTracker components. Below are common types of failures and how Hadoop 1.x handled them:

(a) Task tracker Failure

Failure Scenario:

A Task Tracker may fail due to node crashes, network issues, or hardware faults.

Failure Handling :

The Job Tracker receives periodic heartbeats from each TaskTracker. If the JobTracker doesn't receive a heartbeat from a TaskTracker within a certain timeout period, it marks the TaskTracker as failed. In this case , All tasks running on the failed TaskTracker are reassigned to other active TaskTrackers.

Failures in classic Map-reduce and YARN (contd)

(b) Job tracker Failure

Failure Scenario:

If the JobTracker fails, the entire job execution system comes to a halt, as it is responsible for both job scheduling and task execution tracking.

Failure Handling :

In Hadoop 1.x, there is no automatic failover mechanism for the JobTracker. If it crashes, all running jobs are lost, and the cluster administrator must manually restart the JobTracker and resubmit the jobs.

Thus it is a critical limitation for large clusters or critical jobs as it cannot be recovered without manual intervention.

(c) Task Failure

Failure Scenario:

Map and Reduce tasks may fail due to various reasons, such as hardware faults, software bugs, memory issues, or disk failures.

Failure Handling :

If a task fails, the JobTracker automatically retries the task up to a predefined number of attempts (default is 4 attempts). If the task consistently fails, the job will eventually fail.

To handle critical tasks, the JobTracker can launch a duplicate (speculative) task on a different node. The result from the task that finishes first is taken as the valid output, and the other task is killed.

Failures in classic Map-reduce and YARN (contd)

(d) DataNode Failure

Failure Scenario:

In Hadoop's HDFS (Hadoop Distributed File System), DataNodes store blocks of data. If a DataNode fails or becomes unavailable, the map or reduce tasks that rely on the data stored on that node can be affected.

Failure Handling :

HDFS stores multiple copies (replicas) of each data block (default is 3 replicas). If a DataNode fails, the JobTracker can schedule the tasks on another node that holds a replica of the data. This ensures data availability even if some nodes fail.

When a DataNode failure is detected, HDFS automatically triggers a process to create new replicas of the blocks stored on that DataNode.

(e) Network Failure

Failure Scenario:

Network failures can interrupt communication between the JobTracker, TaskTrackers, and DataNodes.

Failure Handling :

Hadoop 1.x retries failed network operations, and if the failure persists, tasks are moved to other nodes.

If a TaskTracker becomes unreachable due to network issues, the JobTracker may reassign its tasks to other TaskTrackers.

Failures in classic Map-reduce and YARN (contd)

2. Failures in YARN

As we have seen in previous slides that , with the introduction of YARN in Hadoop 2.x, resource management and job execution are separated. YARN improves failure handling by distributing the responsibilities that were previously managed by the single JobTracker across multiple components. Here's how YARN handles failures:

(a) NodeManager Failure

Failure Scenario:

A NodeManager may crash, leading to the failure of any tasks running on that node.

Failure Handling :

The ResourceManager receives heartbeats from NodeManagers. If a NodeManager fails to send heartbeats, the ResourceManager marks it as dead and reschedules the tasks on other available NodeManagers.

YARN re-executes any tasks that were running on the failed NodeManager on other available nodes.

(b) ResourceManager Failure

Failure Scenario:

If the ResourceManager crashes, resource allocation across the cluster stops.

Failure Handling :

Unlike Hadoop 1.x, YARN supports ResourceManager High Availability (HA), where multiple ResourceManagers can run in an active-standby

Failures in classic Map-reduce and YARN (contd)

configuration. If the active ResourceManager fails, a standby ResourceManager takes over without interrupting running jobs.
This significantly improves fault tolerance compared to the single JobTracker in Hadoop 1.x.

(c) ApplicationMaster Failure

Failure Scenario:

If the ApplicationMaster fails, the job it manages may be disrupted which It is currently handling .

Failure Handling :

YARN can automatically restart the ApplicationMaster on another node in case of failure, with job progress preserved. By default, YARN retries the AM up to a configurable number of times.
The ApplicationMaster can be restarted with its state preserved, allowing the job to continue from where it left off, rather than restarting from the beginning.

(d) Task (Container) Failure

Failure Scenario:

Tasks are run in containers, which encapsulate the resources (CPU, memory) required for each task. A map or reduce task may fail due to hardware errors, software issues, or resource contention.

Failures in classic Map-reduce and YARN (contd)

Failure Handling :

Similar to Hadoop 1.x, YARN retries failed tasks. If a task fails multiple times, it is marked as failed, and depending on the retry limit, the entire job may fail. If a task (container) fails on a particular NodeManager, the ApplicationMaster reschedules it on another available NodeManager.

(d) DataNode Failure

Failure Scenario:

If a DataNode in HDFS fails, any tasks relying on the data stored on that node may be affected.

Failure Handling :

YARN continues to use HDFS for storing data, which replicates blocks of data across multiple DataNodes. If a DataNode fails, YARN can run tasks on other nodes with copies of the data.

Just like in Hadoop 1.x, HDFS automatically re-replicates blocks from the failed DataNode to maintain the required number of replicas.

(e) Network Failure

Failure Scenario:

Network issues between NodeManagers, ResourceManagers, or clients can cause tasks to fail.

Failure Handling :

YARN retries tasks affected by network issues and can reschedule them on other nodes if network failures persist.

Failures in classic Map-reduce and YARN (contd)

Summary :

Failure Type	Classic MapReduce (Hadoop	YARN (Hadoop 2.x)
Node Failure	TaskTracker failure triggers task execution	NodeManager failure task re-execution.
Master Failure	JobTracker failure causes the entire system to halt	ResourceManager supports HA; standby takes over if the active fails
Task Failure	Tasks retried; speculative execution supported	Tasks retried; speculative execution supported with containers
Task Scheduling	Fixed number of map/reduce slots; inefficient resource use.	Dynamic resource allocation via containers; improved efficiency.
Application Failure	Single point of failure with the JobTracker.	ApplicationMaster failure triggers retries; tasks continue running.
Data Node Failure	HDFS replicates data, failed tasks are rescheduled on other nodes.	HDFS continues to replicate data, failed tasks rescheduled on other nodes.

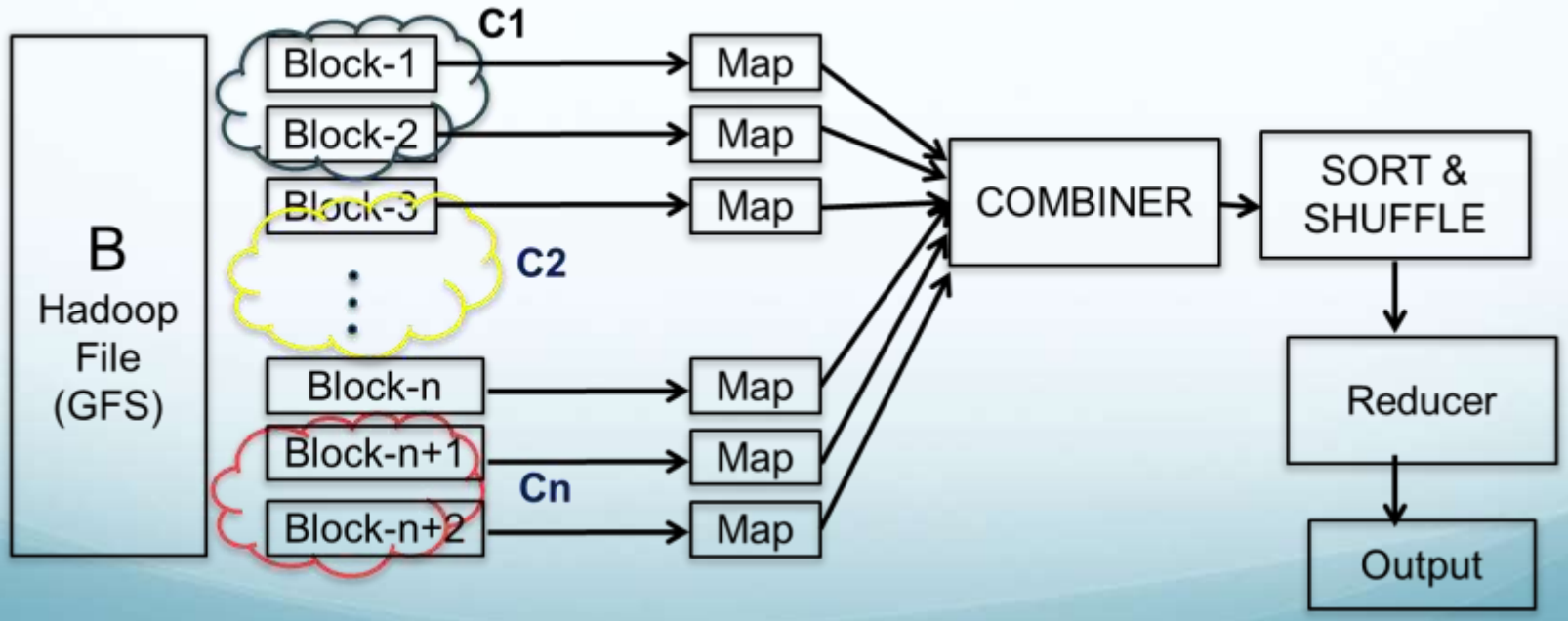
Shuffle and Sort

Shuffle and Sort are critical phases in Big data Hadoop's MapReduce processing frameworks. Both these phases are key to enabling efficient data processing and are fundamental in transforming and aggregating data at scale.

Shuffle and Sort Phase

The shuffle phase refers to the process of recovering data across different nodes in a distributed system, ensuring that data with the same key is brought together on the same node.

PPT#5, slide#6

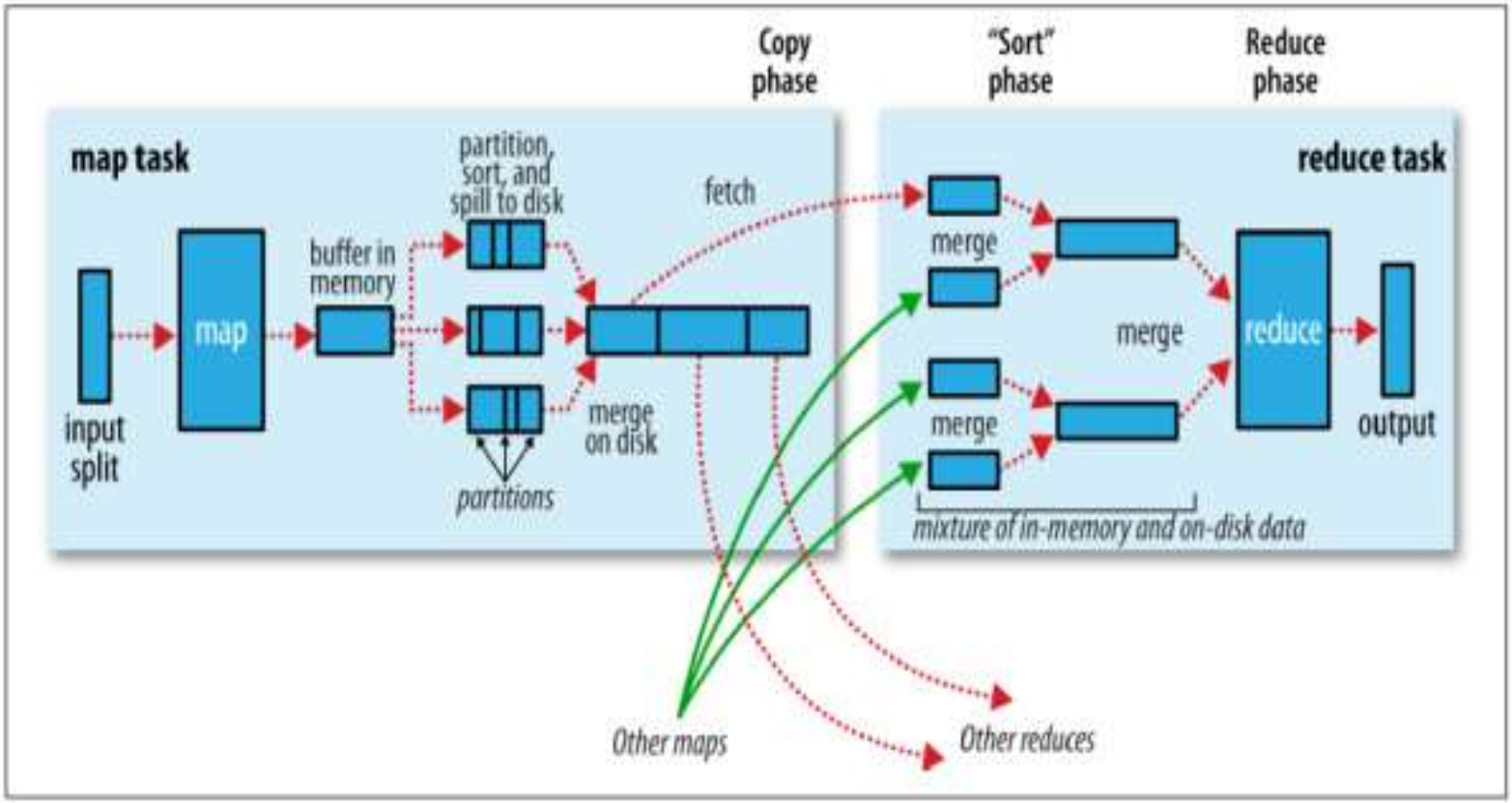


Shuffle and Sort (contd)

After the Map step, the MapReduce framework, intermediate key-value pairs are generated .These pairs are shuffled and sorted by key (usually the hashtags). Map reduce operation gives the guarantee that the input to reducer is sorted by a key. The process by which the system performs the sort and transfers the output to reducer as inputs is known as Shuffle. The goal here is to group all occurrences of the same hashtag together. Refer example on slide#5 , PPT5. In many ways the Shuffle is the heart of the MapReducer and it is where the magic happens.

- When the map function starts producing outp+
- ut, it is not simply written to disk but temporary stored in a circular buffer memory (100MB default size) as shown on fig below. When the buffer is filled say 80% its then only the content is moved to the disk. This is known as spilling.

The process of writing to circular memory buffer and spilling to disk can run concurrently. If the circular buffer gets filled however then the map will block until the spill is complete

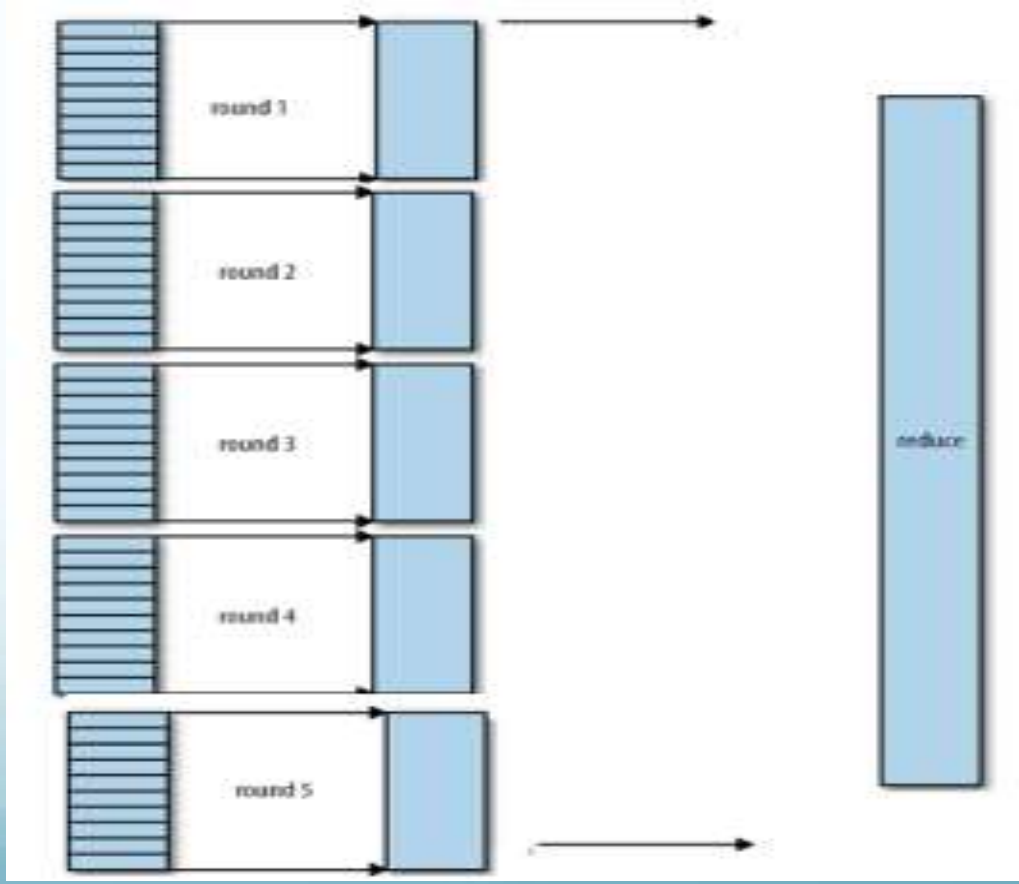


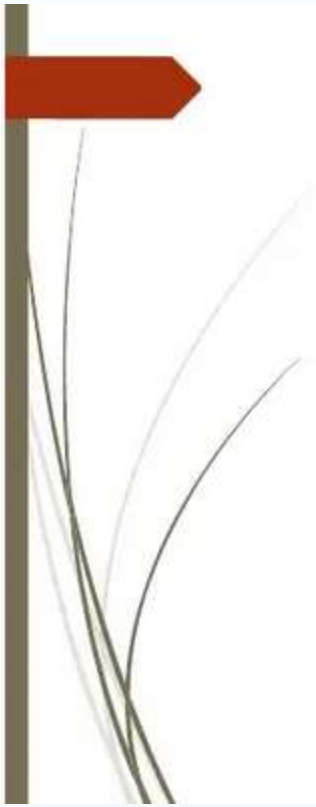
Shuffle and Sort (contd)

Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to based on hash key.

- As map tasks complete successfully, they notify their application master using the heartbeat mechanism. Therefore, for a given job, the application master knows the mapping between map outputs and reducer.
- When all the map outputs have been copied, the reduce task moves into the sort phase (which should properly be called the merge phase, as the sorting was carried out on the map side), which merges the map outputs, maintaining their sort ordering.

- This is done in rounds. For example, if there are say 50 map outputs and the merge factor say is 10 there would be five rounds. Each round would merge 10 files into 1, so at the end there would be 5 intermediate files.
- A thread in the reducer periodically asks the application for map output node until it has retrieved them all.
- Nodes do not delete map outputs from disk as soon as the first reducer has retrieved them, as the reducer may subsequently fail. Instead, they wait until they are told to delete them by the application master, which is after the job has completed.





Thanks