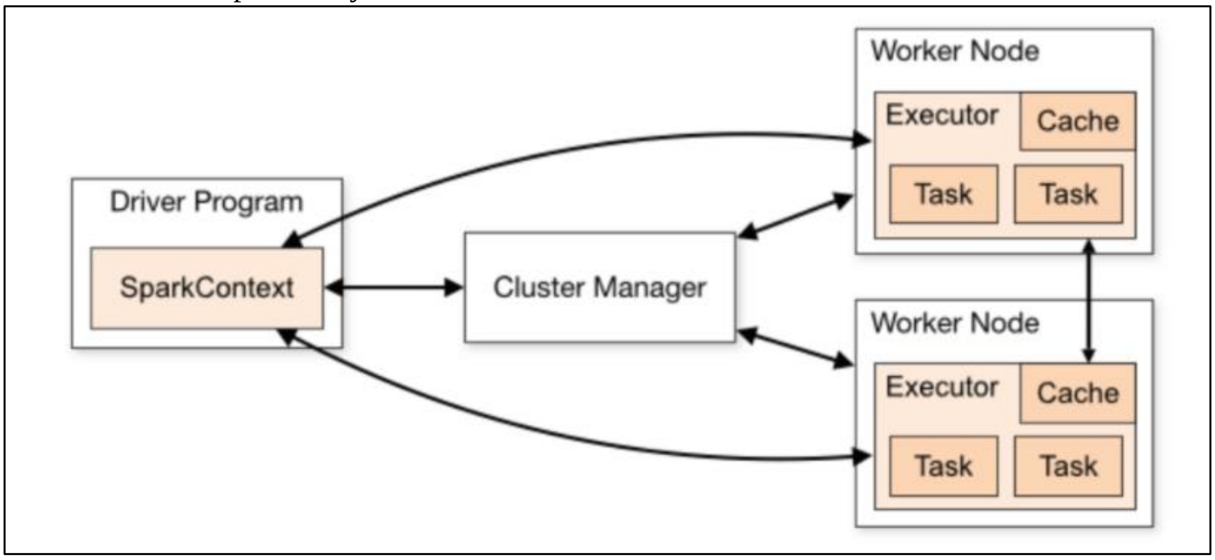
Big Data And Hadoop
Assignment 12.2

Problem Statement:
Answer the given questions.

1. **How are worker, executor and task related to each other?**

Answer: The relation between worker, executor and task can be understood by realizing the flow of execution of a job in Apache Spark as follows:

a. When a client submits a spark user-application code, the driver implicitly converts the code containing transformations and actions into a logical directed acyclic graph (DAG).
b. It then converts DAG into physical execution plan with set of stages. After creating the physical execution plan, it creates small physical execution units referred to as tasks under each stage. Then tasks are bundled to be sent to the Spark Cluster.
c. The cluster manager then launches executors on the worker nodes on behalf of the driver, for the execution of these tasks.
d. This flow can pictorially be seen as follows:



e. Worker nodes are machines that run executors. One JVM (= 1 UNIX process) is allocated per Worker.
f. Each Worker can spawn one or more Executors.
g. Executors run tasks. They can execute one or more tasks using thread pool.

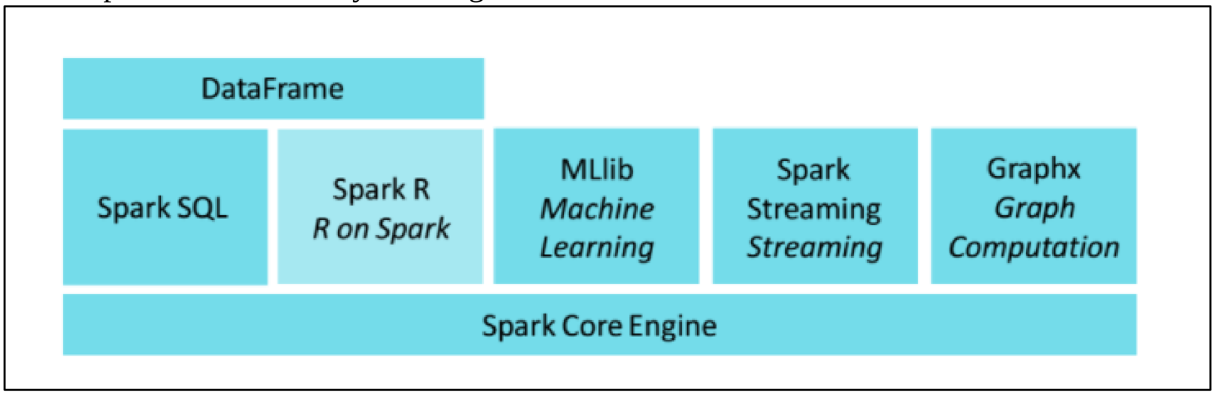
2. **What are the key features of Spark?**

Answer: Key features of Spark are as follows:

a. Lightning fast processing: Spark enables applications in Hadoop clusters to run up to 100x faster in memory, and 10x faster even when running on disk. Spark makes it possible by reducing number of read/write to disc.
b. Data caching (In-Memory Processing): It stores the intermediate processing data in-memory.
c. Resilient Distributed Datasets (RDD): It uses the concept of Resilient Distributed Dataset (RDD), which allows it to transparently store data in memory and persist it to

disc on when it’s needed. This helps to reduce most of the disc read and write – the main time consuming factors – of data processing.The main abstraction Spark provides is a resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to persist an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.

d. DAG based execution: The user application code is converted into DAG, a set of Vertices and Edges, where vertices represent the RDDs and the edges represent the Operation to be applied on RDD. In Spark DAG, every edge is directed from earlier to later in the sequence. On calling of Action, the created DAG is submitted to DAG Scheduler which further splits the graph into the stages of the task.
e. Ability to Integrate with Hadoop and Existing HadoopData: Spark can run independently. Apart from that it can run on Hadoop 2’s YARN cluster manager, and can read any existing Hadoop data. That’s a BIG advantage! It can read from any Hadoop data sources for example HBase, HDFS etc. This feature of Spark makes it suitable for migration of existing pure Hadoop applications, if that application use-case is really suiting Spark.
f. Unified platform: In addition to simple “map” and “reduce” operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. Not only that, users can combine all these capabilities seamlessly in a single workflow.



g. Ease of Use as it supports multiple languages: Spark lets you quickly write applications in Java, Scala, or Python. This helps developers to create and run their applications on their familiar programming languages. It comes with a built-in set of over 80 high-level operators. We can use it interactively to query data within the shell too. It has a rich set of APIs.
h. Real Time Stream Processing: Spark can handle real time streaming. Map-reduce mainly handles and process the data stored already. However Spark can also manipulate data in real time using Spark Streaming. Not ignoring that there are other frameworks with their integration we can handle streaming in Hadoop.
i. Integration: Spark has the capability to reuse the same code for batch and stream processing, even joining streaming data to historical data.
j. Automatic Parallelization of Complex Flows: When constructing a complex pipeline of MapReduce jobs, the task of correctly parallelizing the sequence of jobs is left to you. Thus, a scheduler tool such as Apache Oozie is often required to carefully

construct this sequence. With Spark, a whole series of individual tasks is expressed as a single program flow that is lazily evaluated so that the system has a complete picture of the execution graph. This approach allows the core scheduler to correctly map the dependencies across different stages in the application, and automatically parallelize the flow of operators without user intervention.

k. Interactive Shell: Spark also lets you access your datasets through a simple yet specialized Spark shell for Scala and Python. With the Spark shell, developers and users can get started accessing their data and manipulating datasets without the full effort of writing an end-to-end application. Exploring terabytes of data without compiling a single line of code means you can understand your application flow by literally test-driving your program before you write it up.

3. What is Spark Driver?

Answer: Apache Spark follows a master/slave architecture with two main daemons and a cluster manager –

a. Master Daemon – (Master/Driver Process)
b. Worker Daemon –(Slave Process)

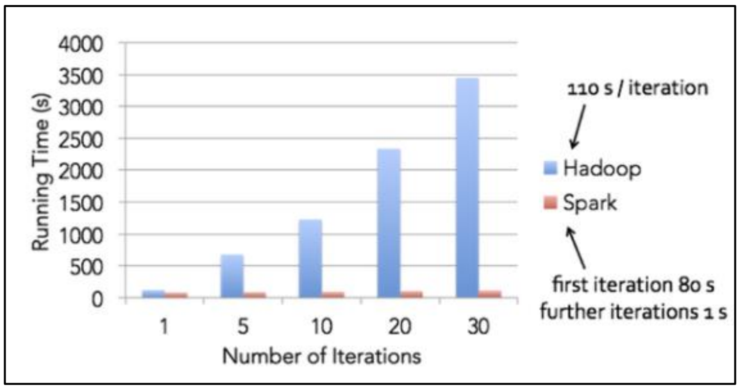
Role played by driver in spark application is as follows:

a. Spark Driver is the master node of the spark application. It is the central point and the entry point of the Spark Shell (Scala, Python, and R).
b. The driver program runs the main () function of the application and is the place where the Spark Context is created.
c. It acquires executors on cluster nodes to run the tasks, cache data. It sends application code to executors and tasks for executors to run.
d. Spark Driver contains various components – DAGScheduler, TaskScheduler, BackendScheduler and BlockManager responsible for the translation of spark user code into actual spark jobs executed on the cluster.
e. The driver program that runs on the master node of the spark cluster, schedules the job execution and negotiates with the cluster manager. Connects to a cluster manager to allocate resources across applications.
f. It translates the RDD's into the execution graph and splits the graph into multiple stages.
g. Driver stores the metadata about all the Resilient Distributed Databases and their partitions.
h. Cockpits of Jobs and Tasks Execution -Driver program converts a user application into smaller execution units known as tasks. Tasks are then executed by the executors i.e. the worker processes which run individual tasks.
i. Driver exposes the information about the running spark application through a Web UI at port 4040.

4. What are the benefits of Spark over MapReduce?

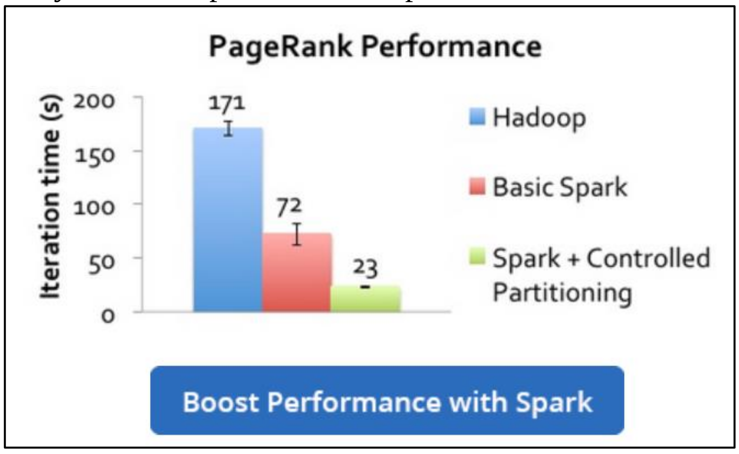
Answer: Benefits of Spark over map-reduce are as follows:

a. Map-reduce execution is disk-intensive. It writes the intermediate outputs to the disk. This activity of writing to disk is timing consuming and hence, may result in a performance hit. Spark, on the other hand uses in-memory processing. Hence, its performance is better than map-reduce. The figure below shows time consumption in map-reduce and spark.

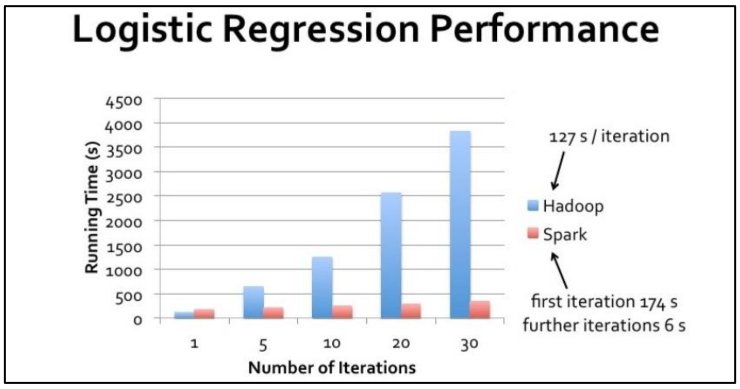


Spark is 100x times faster than map-reduce, while 10x times faster with disk.

b. Map-reduce mainly handles and process the data stored already. However Spark can also manipulate data in real time using Spark Streaming.
c. Hadoop map-reduce is written in java, whereas, Spark is written in scala.
d. Spark is easy to program and does not require any abstraction; Hadoop MapReduce is more difficult to program and requires abstractions.
e. Spark has an inbuilt interactive mode. Map-reduce does not have an inbuilt interactive mode, except tools like pig and hive.
f. Programmers can modify the data in real-time through Spark streaming. Map-reduce allows you to just process a batch of stored data.
g. Spark provides support for many languages, Hence, the user need to step out of his comfort zone to design spark jobs.
h. Writing Hadoop map-reduce pipelines is complex and lengthy process. Writing spark code is always compact than map-reduce code.
i. Graph processing: Most graph processing algorithms like page rank perform multiple iterations over the same data and this requires a message passing mechanism. We need to program MapReduce explicitly to handle such multiple iterations over the same data. Also, each MapReduce iteration has very high latency, and the next iteration can begin only after the previous job has completely finished. Also, message passing requires scores of neighboring nodes in order to evaluate the score of a particular node. These computations need messages from its neighbors (or data across multiple stages of the job), a mechanism that MapReduce lacks. Introduction of Apache Spark solved these problems to a great extent. Spark contains a graph computation library called GraphX which simplifies our life.



j. Iterative machine learning algorithms: Almost all machine learning algorithms work iteratively. As we have seen earlier, iterative algorithms involve I/O bottlenecks in the MapReduce implementations. MapReduce uses coarse-grained tasks (task-level parallelism) that are too heavy for iterative algorithms. Spark with the help of Mesos – a distributed system kernel, caches the intermediate dataset after each iteration and runs multiple iterations on this cached dataset which reduces the I/O and helps to run the algorithm faster in a fault tolerant manner. Spark has a built-in scalable machine learning library called MLlib which contains high-quality algorithms that leverages iterations and yields better results than one pass approximations sometimes used on MapReduce.



k. Considering Spark’s benchmarks, it should be more cost-effective since less hardware can perform the same tasks much faster, especially on the cloud where compute power is paid per use.

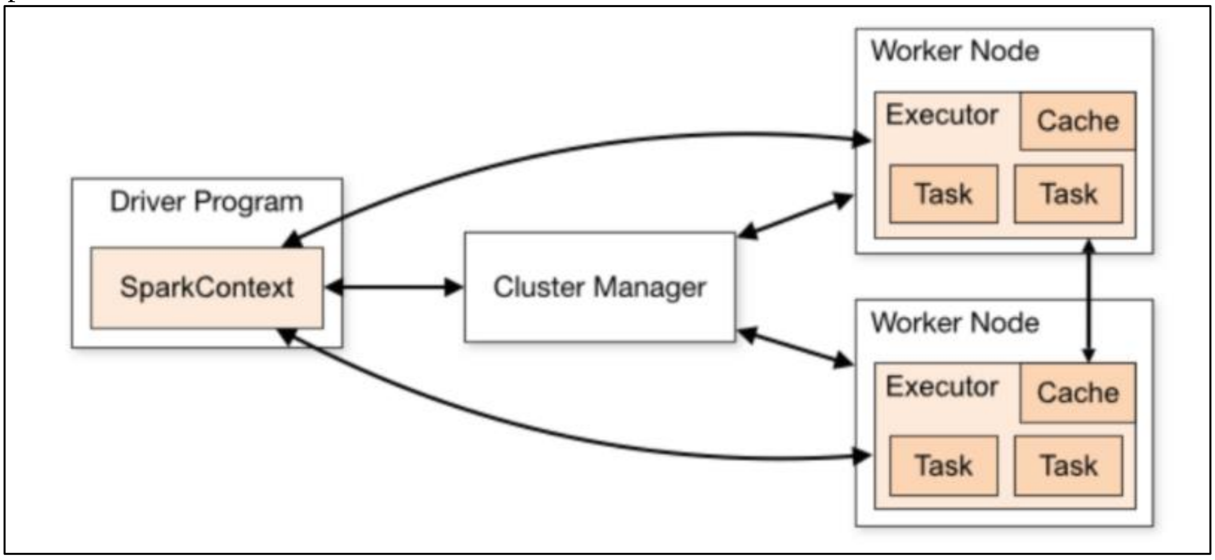
5. What is Spark Executor?

Answer:

a. Executors are worker node processes in charge of running individual tasks in a given Spark job.

b. They are launched at the beginning of a Spark application and typically run for the entire lifetime of an application.

c. Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run.



d. When an application is submitted for execution, the driver program ask for resources to the cluster manager to launch executors. Once they have run the task they send the results to the driver. The cluster manager launches executors. Depending on the actions and transformations over RDDs task are sent to executors. Executors run the tasks and save the results.