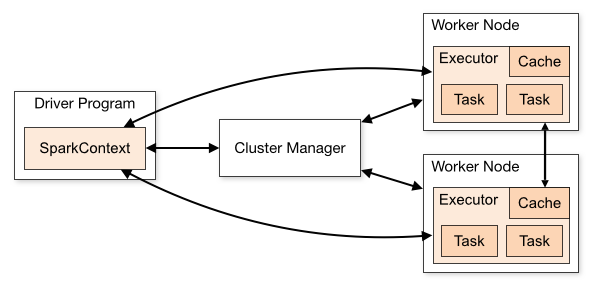
1. How are worker, executor and task related to each other?



In spark, Unlike MapReduce, an application will have processes, called Executors, running on the cluster on its behalf even when it’s not running any jobs. This approach enables data storage in memory for quick access, as well as lightning-fast task startup time. Spark uses master/slave architecture.  It has one central coordinator (Driver) that communicates with many distributed workers (executors). The driver and each of the executors run in their own Java processes

When we submit an application to the cluster with spark-submit this is what happens internally:

1. A standalone application starts and instantiates a SparkContext instance (and it is only then when you can call the application a driver).
2. The driver program asks for resources to the cluster manager to launch executors.
3. The cluster manager launches executors.
4. The driver process runs through the user application. Depending on the actions and transformations over RDDs task are sent to executors.
5. Executors run the tasks and save the results.
6. If any worker crashes, its tasks will be sent to different executors to be processed again.

2. What are the key features of Spark?

1. **Lighting 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. It stores this intermediate processing data in-memory. It uses the concept of a Resilient Distributed Dataset (RDD), which allows it to transparently store data on memory and persist it to disc only it’s needed. This helps to reduce most of the disc read and write – the main time consuming factors – of data processing.

1. **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.

1. **Support for Sophisticated Analytics:**

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.

1. **Real Time Stream Processing:**

Spark can handle real time streaming. Map-reduce mainly handle and process the data stored already. However Spark can also manipulate data in real time using Spark Streaming. This is built in functionality. Spark Streaming recovers lost work.

1. **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. 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.

1. **Active and Expanding Community:**

Apache Spark is built by a wide set of developers from over 50 companies. The project started in 2009 and as of now more than 250 developers have contributed to Spark already! It has active mailing lists and JIRA for issue tracking.

3. What is Spark Driver?

A Spark driver is a JVM process that hosts SparkContext for a Spark application. It is the master node in a Spark application.

The driver is the process where the main method runs. First it converts the user program into tasks and after that it schedules the tasks on the executors.

The spark driver is the program that declares the transformations and actions on RDDs of data and submits such requests to the master.

4. What are the benefits of Spark over MapReduce?

* **Performance:**

Apache Spark processes data in-memory while Hadoop MapReduce persists back to the disk after a map or reduce action.

Spark needs a lot of memory. Much like standard DBs, it loads a process into memory and keeps it there until further notice, for the sake of caching. MapReduce, however, kills its processes as soon as a job is done, so it can easily run alongside other services with minor performance differences.

Spark has the upper hand as long as we’re talking about iterative computations that need to pass over the same data many times. But when it comes to one-pass ETL-like jobs, for example, data transformation or data integration, then MapReduce is the deal

* **Ease of Use:**

Spark has comfortable APIs for Java, Scala and Python, and also includes Spark SQL. It’s easy to write user-defined functions. It even includes an interactive mode for running commands with immediate feedback.

Hadoop MapReduce is written in Java and is infamous for being very difficult to program. MapReduce doesn’t have an interactive mode, although Hive includes a command line interface.

* **Data Processing:**

Spark can process graphs and use the existing machine-learning libraries. Thanks to its high performance, Spark can do real-time processing as well as batch processing.

Hadoop MapReduce is great for batch processing.

* **Failure Tolerance (**not a benefit of over MR):

Spark has retries per task and speculative execution—just like MapReduce. Nonetheless, because MapReduce relies on hard drives, if a process crashes in the middle of execution, it could continue where it left off, whereas Spark will have to start processing from the beginning.

5. What is Spark Executor?

Executor is a distributed agent that is responsible for executing tasks.

Executor typically runs for the entire lifetime of a Spark application which is called static allocation of executors. they reports heartbeat and partial metrics for active tasks to HeartbeatReceiver RPC Endpoint on the driver. It provide in-memory storage for RDDs that are cached in Spark applications. Executors can run multiple tasks over its lifetime, both in parallel and sequentially.