**Recipe Objective: Explain Accumulator shared variables in Spark**

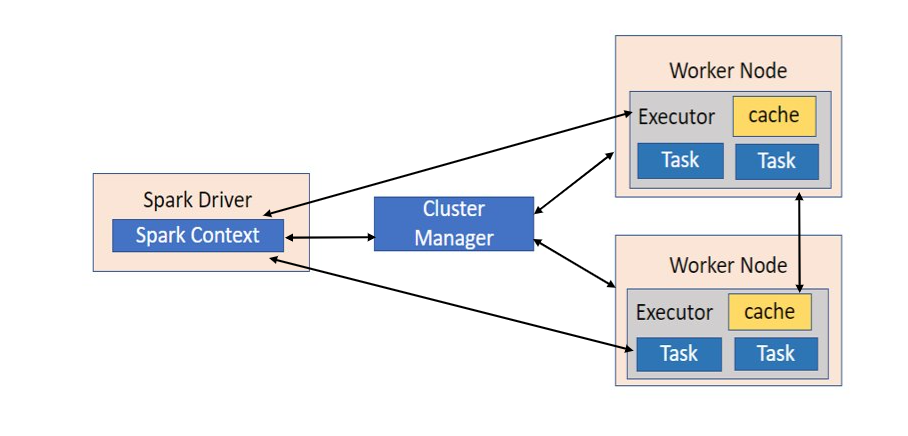
Accumulators are read-only shared variables provided by [Spark](https://www.projectpro.io/apache-spark-tutorial/tutorial-introduction-to-apache-spark). Accumulators are only "added" to through an associative and commutative operation and can be efficiently supported in parallel. They can be used to implement counters (as in MapReduce) or sums. Spark natively supports accumulators of numeric types, and programmers can add support for new types.

An accumulator is a sort of incremental variable that tasks running on nodes can add to while the driver program can read the value. That means tasks running on different machines can increment their value and this aggregated information is available back in the driver program as read-only.

**1. Spark Job Execution**

Let's review the concepts of these two critical shared variables, but before that, here is a diagram showing how the Spark job is executed in the cluster.

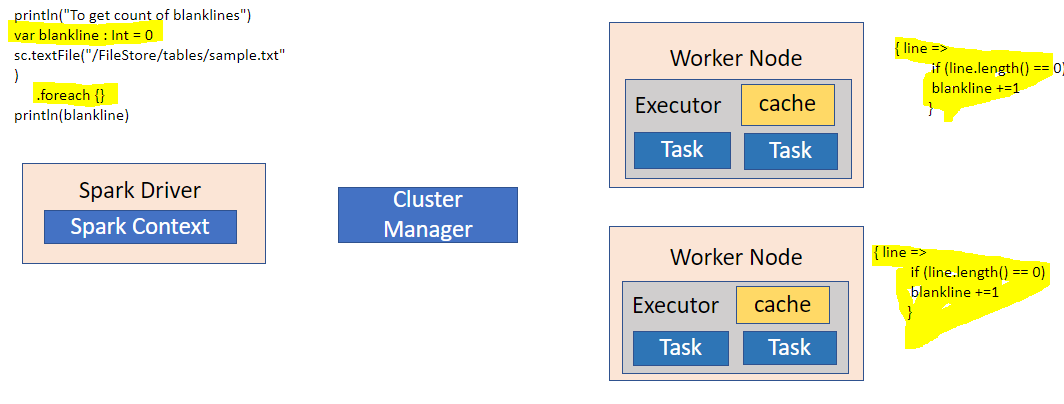
Driver node submits Spark job to the cluster manager. First, it creates the spark context that coordinates with the cluster manager for cluster resources (worker nodes), and in response cluster manager allocates worker nodes from the cluster. Spark context or driver program then launches Executors on the worker nodes. Multiple executors can be launched on a single worker node. Each Executor then launches numerous tasks that run concurrently. General practice shows that each Executor should be configured to launch a maximum of 5 tasks concurrently; otherwise, contention between tasks will degrade the overall Spark job.



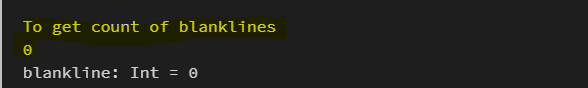
**Count of BlankLines without Accumulator**

We need tasks on different worker nodes processing files line by line, and if a line has some BlankLine, we need to have a count of such BlankLines. Driver program will initialize a variable **"blankline"** and then open a file **sample.txt** for processing. The code within foreach is executed in parallel on multiple nodes and does some processing on each line of data shared.

The execution of the above code is as follows, and it shows how this code is executing in the cluster, that is, which part is running on the driver node and which part is executing on worker nodes.



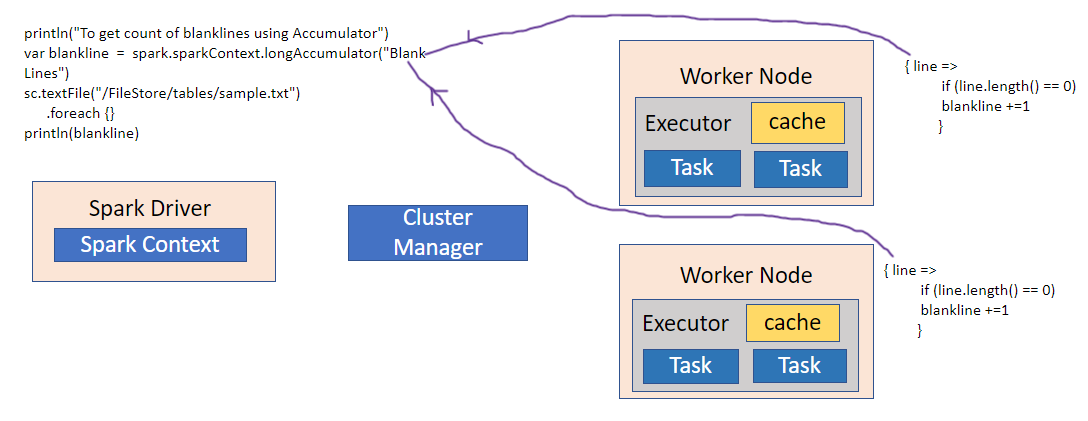
If you observe below figure **"blankline"** variable on the driver side is disconnected with the **"blankline"** variable sent to the worker nodes and tasks. Tasks executing on different nodes can increment the variable, but that information is lost, and once foreach finishes and driver code displays BlankLines variable, it will show 0. All information aggregated on different tasks is lost and does not communicate back to driver code. you view the output of the above code as below



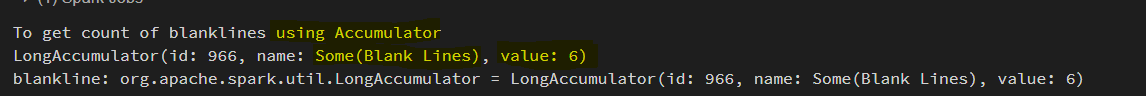
**Count of BlankLines with Accumulator**

We need tasks on different worker nodes are processing file line by line, and if a line has some BlankLine, then we need to have a count of such BlankLines. Driver program will initialize a variable **"blankline"** and then open a file **sample.txt** for processing. The code within foreach is executed in parallel on multiple nodes and does some processing on each line of data shared.

The execution of the above code is as follows, and it shows how this code is executing in the cluster, that is, which part is running on the driver node and which part is executing on worker nodes.



If you observe below figure **"blankline"** variable on the driver side is disconnected with the **"blankline"** variable sent to the worker nodes and tasks. Tasks executing on different nodes can increment the accumulator variable at driver only once and after its job is finished, i.e., foreach. Ad driver code displays BlankLines variable. It will show 6. Here, if you observe, we took the accumulator as "Blank Lines," and it displayed its value as "6", which is our expected value.



For accumulator updates performed inside actions only, Spark guarantees that each task's update to the accumulator will only be applied once, i.e., restarted tasks will not update the value. In transformations, users should know that each task's update may be used more than once if tasks or job stages are re-executed. Accumulators do not change the lazy evaluation model of Spark. If they are being updated within an operation on an RDD, their value is only updated once that RDD is computed as part of an action.

As a user, you can create named or unnamed accumulators. Here we used the named accumulator "Blank Lines." As seen in the image below, a named accumulator ("Blank Lines") will display in the web UI for the stage that modifies that accumulator. Spark displays the value foreach accumulator modified by a task in the "Tasks" table.

