SPARK SHARED VARIABLE: ACCUMULATORS

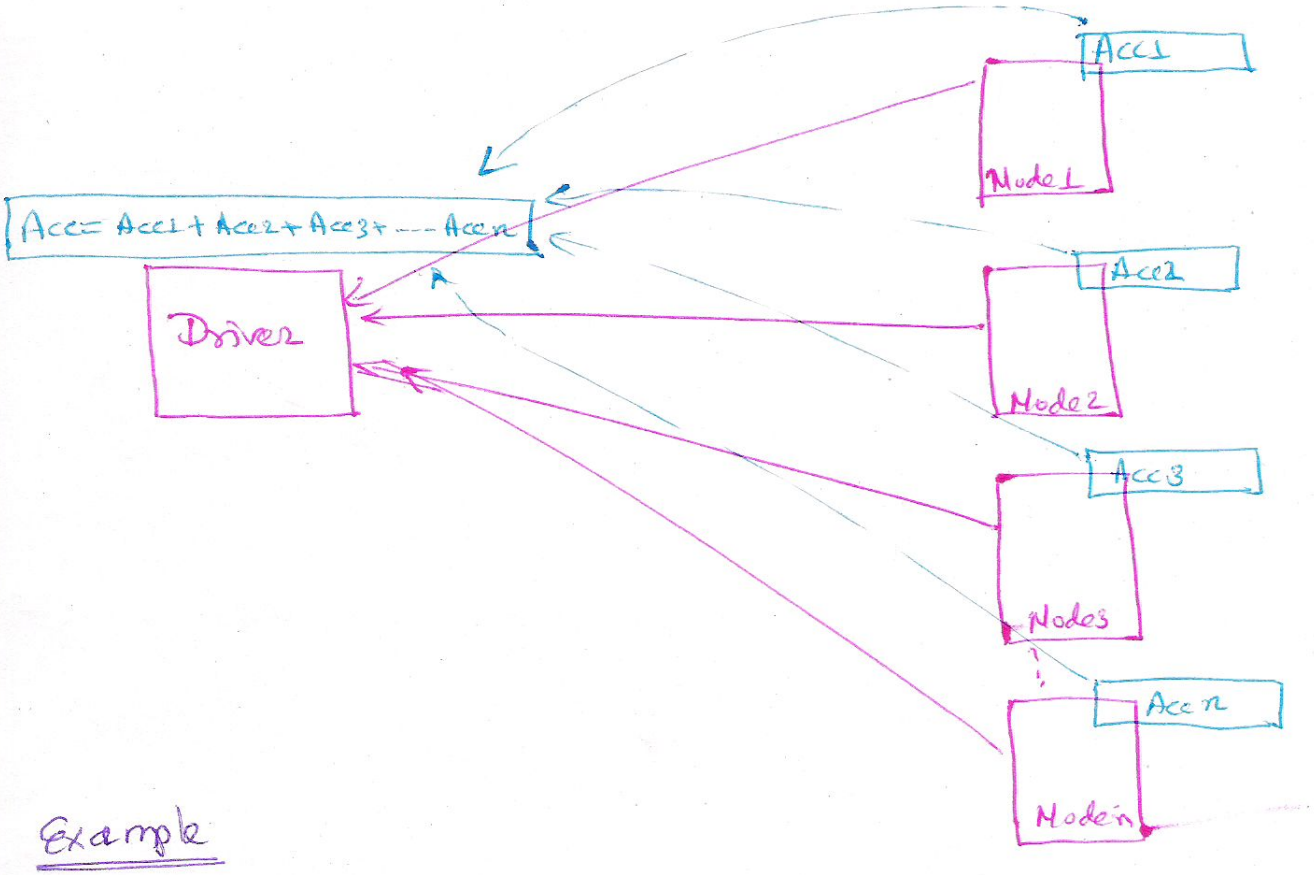
By www.HadoopExam.com

Note: These instructions should be used with the HadoopExam Apache Spark: Professional Trainings. Where it is executed and you can do hands on with trainer.

Apache Spark Shared Variable

Accumulator

⇒ Accumulator is a type of communication pattern of aggregation of result.



Example

⇒ In this example, we counts the number of elements in an RDD of integers using an Accumulator.

⇒ While at the same time, ~~summing~~ summing the values in the RDD using a reducer ~~of~~ action.

```
Val count : Accumulator[Int] = sc.accumulator(0)
Val result = sc.parallelize ( Array (1,2,3))
        .map( i => {
                    count += 1;
                  })
        .reduce ((x,y) => x+y)
assert ( count.value ===3 )
assert ( result ===6)
```

⇒ In above example, we used an Int for the Accumulator, but any numeric value type can be used.

⇒ Spark Accumulators are write-only variables from the worker perspective.

⇒ Any attempt to read its value during the task does not make sense because there is no shared state between workers.

⇒ Local accumulator value reflects only the state for current partition.

Are Accumulator Always Reliable ?

① Accumulator updates are sent back to the driver when a task is successfully completed.

④ So Accumulators results are guaranteed to be correct when you are certain that each task will have been executed exactly once.

True Reliability :-

⇒ Accumulators update must be performed inside Action only.

⇒ Spark guarantees that each tasks update to the accumulator will only be applied once i.e. restarted tasks will not update the value. (only in Actions)

⇒ If you are using transformations, you must be aware that each tasks update may be applied more than once, if tasks or job stages are restarted.

Example

```
Val count = sc.accumulator(0)
list.foreach(x => {
    if (x != "") {
        count += 1;
    }})
```

Empty string

Custom Code in Action

⇒ If task failed due to some exception in code.

⇒ Spark will retry 4 times.

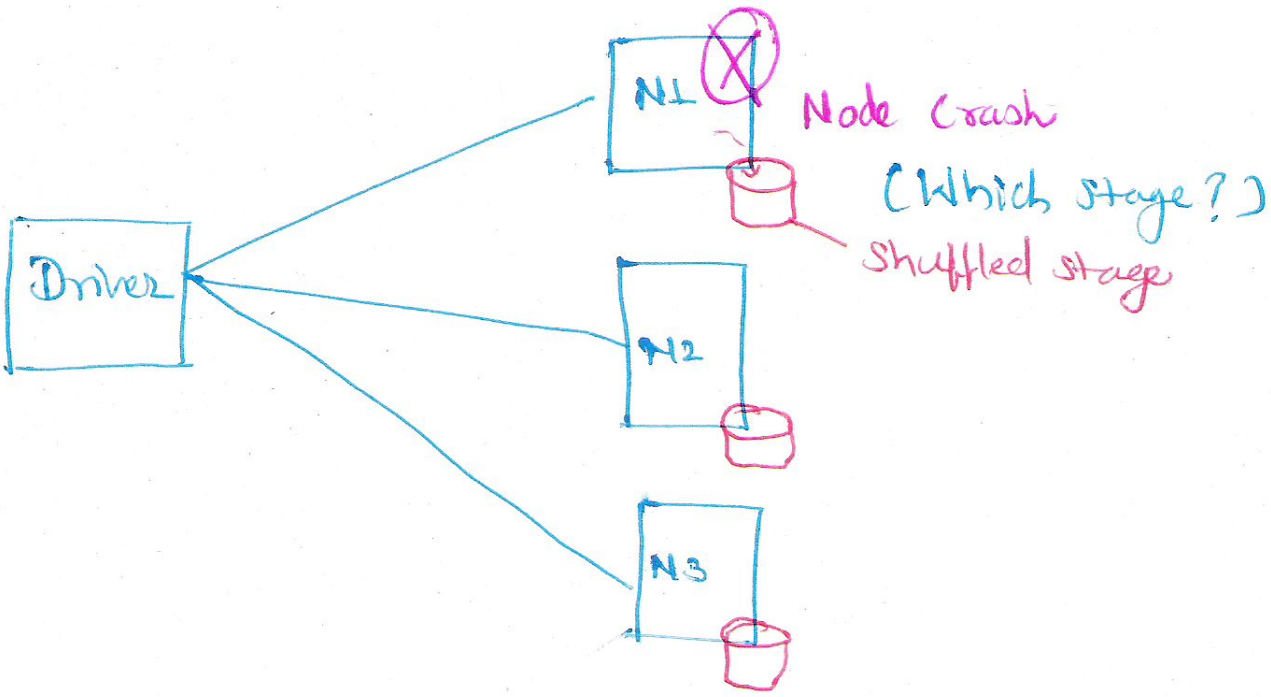
⇒ It is possible all 4 times task will fail and raise exception.

⇒ If by chance, task succeeds then Spark will continue and just update the Accumulator value for successfull state and failed state accumulators values are ignored.

Handling Properly

⇒ Stage failure

① If an executor node crashes, hardware failure.



⇒ Node goes down in shuffle stage, as shuffle output stored locally. and node goes down then shuffled output goes.

⇒ Spark goes back to the stage that generated the shuffle output, looks at which tasks need to be re-run and will run on other node.

⇒ map task executed on failed task, previously as well as on new task, which will cause Accumulator to get corrupted.

⇒ So here Accumulator will give wrong result.

Slow Task : Speculative Execution

⇒ If a task is Running slow, then Spark can launch speculative copy of that task on another active node.

[Not Handled:, hence Accumulator will give wrong value]

Cached RDD

⇒ If an RDD is huge, and cannot reside in memory, then spill over disk.

⇒ So whenever the RDD is used it will re-run the Map operation to get the RDD and again Accumulator will be updated by it.

[Not Handled: Accumulator will give wrong value]

⇒ So it may happen some function run multiple time on same data.

⇒ So Spark does not provide any gaurantee for Accumulator getting updated because of map (Transformation) operation.

⇒ So Always use Accumulator in action in Spark.

⇒ Or use it for only debugging operation.