**ASSIGNMENT 17.6 :**

1. **What are the uses of counters?**

* Counts the number of mappers and reducers launched(JOB COUNTER)
* Counts the number of bytes read and written(FILE SYSTEM COUNTER)
* Counts the number of tasks was launched and successfully ran(TASK COUNTER)
* Provides the amount of CPU and memory consumed and tells that it is appropriate or not for your job and cluster nodes.
* use of custom counters is in the debugging process – where it can also be used to determine the number of BAD records

1. **MR Unit testing is based on?**

MRUnit testing framework is based on JUnit and it can test Map Reduce programs written on 0.20 , 0.23.x , 1.0.x , 2.x version of Hadoop. JUnit is a [unit testing](https://en.wikipedia.org/wiki/Unit_testing) [framework](https://en.wikipedia.org/wiki/Software_framework) for the [Java programming language](https://en.wikipedia.org/wiki/Java_(programming_language)).

1. **How testing is useful in industry?**

**Uses of testing :**

* Validates to make sure that correct data is pulled into system from various source like RDBMS, weblogs, social media, etc.
* Used to Compare source data with the data pushed into the Hadoop system to make sure whether they match or not
* Used to Verify whether the right data is extracted and loaded into the correct HDFS location or not.
* Used to ensure whether Map Reduce process works correctly.
* Ensures whether Data aggregation or segregation rules are implemented on the data.
* Ensures whether the Key value pairs are generated.
* Used to Validate the data after Map Reduce process.
* Used to verify how the fast system can consume data from various data source.
* Used to verify the speed with which the queries or map reduce jobs are executed.

1. **Map reduce Task Counters ,File system counters ,Job Counter**

* **Map reduce task counters:**
* Task counters gather information about tasks over the course of their execution, and the results are aggregated over all the tasks in a job.
* Task counters are maintained by each task attempt, and periodically sent to the task tracker and then to the job tracker.
* Counter values are definitive only once a job has successfully completed. However, some counters provide useful diagnostic information as a task is progressing, and it can be useful to monitor them with the web UI.
* **File system counters:**
* File system counters track 2 main details, number of bytes read by the file system and number of bytes written.
* BYTES\_READ counter is tracked by File Input Format
* BYTES\_WRITTEN counter is tracked by File Output Format
* **Job counter:**
* Job counters are maintained by the job tracker (or application master in
* YARN), so they don’t need to be sent across the network, unlike all other counters, including user-defined ones.
* They measure job-level statistics, not values that change while a task is running.

1. **Raw comparator VS Writable Comparator**

**RAW COMPARATOR:**

Implementing the [org.apache.hadoop.io.RawComparator](http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/RawComparator.html" \t "_blank) interface will definitely help speed up your Map/Reduce (MR) Jobs. By implementing the RawComparator to compare the intermediary keys, this extra effort will greatly improve sorting. Sorting is improved because the RawComparator will compare the keys by byte.If we did not use RawComparator, the intermediary keys would have to be completely deserialized to perform a comparison.

**WRITABLE COMPARATOR:**

WritableComparator is a general-purpose implementation of RawComparator for WritableComparable classes. It provides two main functions. First, it provides a default implementation of the raw compare() method that deserializes the objects to be compared from the stream and invokes the object compare() method. Second, it acts as a factory for RawComparator instances (that Writable implementations have registered).

1. **Partitioner, Sort comparator, Group comparator**

**PARTITIONER:**

A partitioner works like a condition in processing an input dataset. The partition phase takes place after the Map phase and before the Reduce phase.

The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.

**SORT COMPARATOR:**

**SortComparator** decides how keys will be sorted in **input of reduce**. By default it uses natural ordering.

**GROUP COMPARATOR:**

**Group Comparator** decides which map output keys **will be united(grouped) into one key,** and of course all collections of values will be grouped too. Usually it takes a first key as the only one for summary collection.