**Session 17**

**Assignment 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 launched and ran successfully (TASK COUNTER)
* Provides a way to measure the progress or the number of operations that occur within MapReduce programs.
* 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
* Basically, MapReduce framework provides a number of built-in counters to measure basic I/O operations, such as FILE\_BYTES\_READ, FILE\_BYTES\_WRITTEN and Map/Combine/Reduce input/output records.

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

* MR Unit testing framework is based on JUnit and it can test MapReduce programs written on 0.20, 0.23.x, 1.0.x, 2.x version of Hadoop.

**NOTE:** 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?**

* Validates the data to make sure that the pulled data into the system from various source like RDBMS, weblogs, social media, etc. is correct to the requirements.
* Used to compare source data with the data pushed into the Hadoop system to make sure whether they match or not.
* To verify whether the right data is extracted and loaded into the correct HDFS location or not.
* Used to ensure whether MapReduce process works correctly.
* Ensures whether Data aggregation or segregation rules are implemented on the data correctly.
* Ensures whether the Key value pairs are generated.
* Used to validate the data obtained after MapReduce process.
* Used to verify how the fast system can consume data from various data source.
* Used to check the speed with which the queries or MapReduce 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 the data of the bytes read and output produced by the MapReduce.
* Counter values are definitive only once after the successful completion of the job. 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, they are
* Number of bytes read by the file system.
* Number of bytes written by the system.
* BYTES\_READ counter is tracked by File Input Format.
* BYTES\_READ: The number of bytes read by map tasks via FileInputFormat.
* BYTES\_WRITTEN counter is tracked by File Output Format.
* BYTES\_WRITTEN: The number of bytes written by map tasks (for map-only jobs) or reduce tasks via FileOutputFormat.
* **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.
* For example, the count of number of map tasks that were launched over the course of a job (including the ones failed) can be counted using the job counter ‘TOTAL\_LAUNCHED\_MAPS’.

1. **Raw comparator VS Writable Comparator**

* **Raw Comparator:**
* Implementing the [Raw Comparator](http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/RawComparator.html) interface will definitely help speed up your Map/Reduce (MR) Jobs.
  + Implementing code: [org.apache.hadoop.io.RawComparator](http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/RawComparator.html)
* By implementing the Raw Comparator to compare the intermediate keys, this extra effort will greatly improve sorting.
* The Raw Comparator will compare the keys by bytes, so the sorting of the data is improved.
* If we do not use Raw Comparator, the intermediate keys have to be completely deserialized to perform a comparison.
* **Writable Comparator:**
* IntWritable implements the Writable Comparable interface.
* Writable Comparator is a general-purpose implementation of Raw Comparator for Writable Comparable 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 Raw Comparator instances (that Writable implementations have registered).
* The Writable Comparator user the Raw Comparator for implementing and thus both are independent.

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.
* The data passed from a single partitioner is processed by a single Reducer. That is, for each partitioner there is a reducer.
* Partitioner controls the partitioning of the keys of the intermediate map-outputs. The total number of partitions is the same as the number of reduce tasks for the job.
* **Sort Comparator:**
* **Sort Comparator** 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.