***Q1.What are the uses of counters?***

* Counters are used to see the progress and operations of what all happened inside each map reduce program.
* For file\_bytes\_read/written and Map/Combine/Reduce input/output records it is used to measure all those I/O operations .
* It is the major part of map reduce programs.
* Own counter set is possible when map reduce program is used.
* Map and reduce are two major phases so it will automatically aggregate.
* And each part of map and reduce internal behavior will be found.

***Two types of counters:***

1. ***Hadoop Built-In counters:***

There are some built in counters available:

*MapReduce Task Counters –*

At the time of execution it will collect the task specific information like number of i/p records.

*FileSystem Counters –*

The information during byte read/write will be collected.

*FileInputFormat Counters –*

Will collect the information about the number of bytes which will read through the FileInputFormat.

*FileOutputFormat Counters –*

Number of bytes which all read through FileOutputFormat will be collected.

***2.Job Counters :***

* Job trackers will make use of this counter.
* Statics like number of tasks launched for each job will be collected.

1. **User Defined Counters**

* This can enable the user to make their own counters using the similar functionalities which were used in programming languages.
* Eg: [Java](http://www.guru99.com/java-tutorial.html)'enum'-defining user defined functions.

-----------------------------------------------------------------------------------------------

**Q2. MR Unit testing is based on**

MRunit able to track the craft test input.

By pushing the MRUnit through mapper and reducer /mapper or reducer one can get verified output in the JUnit.

One can decode in Junit by make using of JUnit test as driver.

Using PipelineMapReduceDriver one can test the overflow of map/reduce jobs.

PipelineMapReduceDriver can check the flow of map/reduce jobs.

Under MRUnit the artitioners will not have any test unit.

TDD is allowed in partitioners under MRUnit and allows light weight unit test which helps to accommodate the architecture and construction of Hadoop.

--------------------------------------------------------------------------------------------------

**Q3. How testing is useful in industry**

Testing in previous year will be done in intermediate stage which will be rushed just before the software is handed over to client.

But there are some software development agencies which do not show interest to devote the resources in order to complete the entire process.

Testing was a easy one which will be just like a tick inside a box.

The policy results in a incomplete process which has functions as a massive variance to the clients need.

Some software which all delivered will not be used.

There where about 80-90% of the project which costs so high and those price need to be reduced.

there will be over run of cost or time due to high profile or publicity which will cause a huge damage for the IT firm or client as well the development industry.

Some IT projects need to be abandoned.

There will be project failure due to testing regime.

This will cause a huge loss in industry due to project failure.

Trust is important for all the consultancies and development houses which leads to win and retain in the business.

Test plan is not act as a great solution to a project.

Testing provides the IT industry with great solution which reduces the error.

It is to check whether there is robustness in the IT system as well reduce the cost of identifying the errors in prior.

***Q4. Mapreduce Task Counters, File system counters, Job Counter***

***Hadoop Built-In counters:***

Built-in counter will be available for each job.

Built-in counter groups are-

* *MapReduce Task Counters* – will Collect all the task specific information like - input records count during the time of execution.
* *FileSystem Counters -* Collects all the information regarding number of bytes read or written each task.
* *FileInputFormat Counters –* will Collects all the information regarding the number of bytes read through FileInputFormat.
* *FileOutputFormat Counters –* will Collects the information regarding number of bytes written via FileOutputFormat.
* *Job Counters –* are used by JobTracker.

Statistics collected by job counter which includes the number of task which were launched for each job.

--------------------------------------------------------------------------------------------------

***Q5. Raw comparator VS Writable Comparator***

* [org.apache.hadoop.io.RawComparator](http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/RawComparator.html) is implemented which makes the map/reduce job to get speed up.
* The job of MR is to send the key-value pair.
* And the process will be like: (K1,V1) –> Map –> (K2,V2)
* (K2,List[V2]) –> Reduce –> (K3,V3)
* K2,V@ are key-pair values which will be called as intermediary key-value pairs.
* And they will be passed on from the mapper to the reducer.
* In-between the passing rom the mapper to reducer they will shuffle and sort.
* Shuffle will be due to keys(K2) which will passed to the reducer and during sort the keys will be sorted.
* By making use of the RawComparator one is able to compare the intermediate keys which helps to improve the sorting by comparing the keys and the bytes.
* If in case the RawComparator is not used then the intermediary keys should be deserialized for comparison.
* RawComparator snippet:
* RawComparator will be directly implemented by extending the WritableComparator
* utility methods of WritableComparator can be used.

|  |  |
| --- | --- |
| 1  1 | public class IndexPairComparator extends WritableComparator {      protected IndexPairComparator() {          super(IndexPair.class);      }        @Override      public int compare(byte[] b11, int s11, int l11, byte[] b12, int s12, int l12) {          int i11 = readInt(b11, s11);          int i12 = readInt(b12, s12);            int comp = (i11 < i12) ? -1 : (i11 == i12) ? 0 : 1;          if(0 != comp)              return comp;            int j11 = readInt(b11, s11+4);          int j12 = readInt(b12, s12+4);          compa = (j11 < j12) ? -1 : (j11 == j12) ? 0 : 1;            return compa;      }  }   * By implementing the raw byte comparison of the intermediary keys one can * improve the speed of the comparison of deserialized objects |

----------------------------------------------------------------------------------------------------

***Q6. Partitioner, Sort comparator, Group comparator***

*Partitioner:*

It is used to control the key partition of all the intermediate map-outputs.

The or the subset which is used is the deriving part of the partition by means of hash function.

total number of partitions = number of reduce tasks (for each job).

So the will be control in the m reduce task of intermediate key which is sent for each reduction(as per record).

*Sort comparator:*

For example key = (Attribute1, Attribute2).

Use Sort Comparator for keys (Attribute1, Attribute2).

When sort it used first Attribute1 will sorted followed by Attribute2.

For example:

Key= (2010, 32) // year, temperature.

The year and temperature should be sorted then sort comparator should be used.

public static class KeyComparator extends WritableComparator {

protected KeyComparator() {

super(CompositeKey.class, true);

}

@Override

public int compare (WritableComparable w1, WritableComparable w12) {

CompositeKey ip11 = (CompositeKey) w1;

CompositeKey ip12 = (CompositeKey) w12;

int result = CompositeKey.compare(ip11.getYear(), ip12.getYear());

if (result != 0) {

return result;

}

return CompositeKey.compare(ip11.getTemperature(), ip12.getTemperature());

}

}

**GroupComparator**:

once the composites key\value reaches the reducer .

(**a-1**,{1-10})

(**a-2**,{2-20})

This will happen due to the unique key used

(a,{**1-10,2-20**})

**[[single reduce method call.]]**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |