**Explain the following terms in detail :**

* **What are the uses of counters**

• Hadoop MapReduce Counter provides a way to measure the progress or the number of operations that occur within MapReduce programs.

• Basically, MapReduce framework provides a number of built-in counters to measure basic I/O operations, such as FILE\_BYTES\_READ/WRITTEN and Map/Combine/Reduce input/output records.

• These counters are very useful especially when you evaluate some MapReduce programs.

• Besides, the MapReduce Counter allows users to employ your own counters.

• Since MapReduce Counters are automatically aggregated over Map and Reduce phases, it is one of the easiest way to investigate internal behaviors of MapReduce programs.

• To create user defined counter and increment it, use:

context.getCounter(<counter\_group>,<counter\_name> ).increment(<value>);

* **MR Unit testing is based on**

MrUnit Testing is based on Junit which is a testing framework for java. Junit has two methods of testing i.e. testing methods manual and automated testing. Junit uses the technique of first coding so it emphasizes on first creating the test data that can be tested once code is ready for implementation.

|  |  |  |
| --- | --- | --- |
|  |  | Features of Junit |
|  |  | • JUnit is an open source framework, which is used for writing and running tests. |
|  |  | • Provides annotations to identify test methods. |
|  |  | • Provides assertions for testing expected results. |
|  |  | • Provides test runners for running tests. |
|  |  | • JUnit tests allow you to write codes faster, which increases quality. |
|  |  | • JUnit is elegantly simple. It is less complex and takes less time. |
|  |  | • JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results. |
|  |  | • JUnit tests can be organized into test suites containing test cases and even other test suites. |
|  |  | • JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails. |
|  |  |  |

* **How testing is useful in industry**

Testing is the process of evaluating a system or its components.it is basically used to check the developed code meet the requirements or not.

|  |
| --- |
|  |
|  | According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item. |
|  | Testing can be conducted by the developer which is called as the unit testing or as in many it companies the testing is done by separate section which does the intense testing on the developed code so as it meets the clients expectations. |
|  | An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. But it depends on the product being developed and the expectations of the client based on the above criteria different SDLC are mentioned. |
|  | Different types of SDLC are |
|  | • V-Shaped Model |
|  | • Iterative Model. |
|  | • Spiral Model. |
|  | • Big Bang Model. |
|  | • Agile Model. |

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

**Mapreduce 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. For example, the MAP\_INPUT\_RECORDS counter counts the input records read by each map task and aggregates over all map tasks in a job, so that the final figure is the total number of input records for the whole job. Etc

## **File system counters**

File system countres 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 read (BYTES\_READ) :The number of bytes read by map tasks via the FileInputFormat.

BYTES\_WRITTEN counter is tracked by File Output Format

Bytes written (BYTES\_WRITTEN) :The number of bytes written by map tasks (for map-only jobs) or reduce tasks via the FileOutputFormat.

### Job Counters

Job counters are maintained by the jobtracker (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, TOTAL\_LAUNCHED\_MAPS counts the number of map tasks that were launched over the course of a job (including ones that failed).

* **Raw comparator VS Writable Comparator**

Implementing the**[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. As you may recall, a MR Job is composed of receiving and sending key-value pairs. The process looks like the following.

(K1,V1) –> Map –> (K2,V2)

(K2,List[V2]) –> Reduce –> (K3,V3)

The key-value pairs (K2,V2) are called the intermediary key-value pairs. They are passed from the mapper to the reducer. Before these intermediary key-value pairs reach the reducer, a shuffle and sort step is performed. The shuffle is the assignment of the intermediary keys (K2) to reducers and the sort is the sorting of these keys. In this blog, 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.

Implementing the **[WritableComparable](http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/WritableComparable.html" \t "_blank)**, you will compare (deserialized) objects, but in the latter approach, you will compare the keys using their corresponding raw bytes.

I conducted an empirical test to demonstrate the advantage of RawComparator over WritableComparable. Let’s say we are processing a file that has a list of pairs of indexes {i,j}. These pairs of indexes could refer to the i-th and j-th matrix element.

1, 2

3, 4

5, 6

...

...

0, 0

What we want to do is simply count the occurrences of the {i,j} pair of indexes. Our MR Job will look like the following.

(LongWritable,Text) –> Map –> ({i,j},IntWritable)

({i,j},List[IntWritable]) –> Reduce –> ({i,j},IntWritable)

* **Partitioner, Sort comparator, Group comparator**

**Group Comparator** – It 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.

**Partitioner** – It is used to decide the which key should go to which reducer, by default it uses the hash code of the object to decide the reducer but one can override the partitioner to send particular to particular reducer. This is mostly used in case of composite key, secondary sort.

**Sort Comparator-** Used to define how map output keys are sorted,SortComparator decides how map output keys are sorted. Ifthe property mapred.output.key.comparator.class is set, either explicitly or by calling setSortComparatorClass() on Job, then an instance of that class is used. (In the old API the equivalent method is setOutputKeyComparatorClass() on JobConf.)