|  |
| --- |
|  |
|  | |
|  | | **What are the uses of counters?**  Counters are the parameters which used to get the information about what is going on during the process of the map reduce. There are built in counters which are defined in the Hadoop which helps us know different parameters about are peripheral to the analysis you are performing. |
|  | | Examples |
|  | | Hadoop inbuilt counters |
|  | | 1) Task Counters |
|  | | -Map input records (MAP\_INPUT\_RECORDS):- The number of input records consumed by all the maps in the job. Incremented every time a record is read from a Record Reader and passed to the maps map ( ) method by the framework. |
|  | | -Split raw bytes (SPLIT\_RAW\_BYTES):- The number of bytes of input-split objects read by maps. These objects represent the split metadata (that is, the offset and length within a file) rather than the split data itself, so the total size should be small. |
|  | |  |
|  | | 2) Job Counters |
|  | | - Launched map tasks (TOTAL\_LAUNCHED\_MAPS):-The number of map tasks that were launched. Includes tasks that were started speculatively |
|  | | -Launched reduce tasks (TOTAL\_LAUNCHED\_REDUCES) The number of reduce tasks that were launched. Includes tasks that were started speculatively. |
|  | |  |
|  | | User can also define his own counter in order to count the number of occurrences of a particular job, error etc. Hence one can get precise information about job being executed and can do modifications to achieve the optimization in the code. |
|  | |  |
|  | |  |
|  | |  |
|  | | Q. MR Unit testing is based on |
|  | | MR Unit Testing is based on Junit which is a testing framework for java. Junit has two methods of testing methods manual testing and automated testing. J unit uses the technique of first testing and the coding so it emphasizes on first creating the test data that can be tested once the 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. |
|  | |  |
|  | | Q. How testing is useful in industry |
|  | | Testing is the process of evaluating a system or its component(s). It is basically used to check whether the developed code meets 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. |
|  | |  |
|  | |  |
|  | | Q.-- Mapreduce Task Counters,File system counters,Job Counter |
|  | | Task Counter – They gather information about the specific task over the execution of the task and the results then summed up at the end of the job. Task counters are maintained by each task. The value of the task counter are sent periodically to the tasktracker. |
|  | | e.g.PHYSICAL\_MEMORY\_BYTES, VIRTUAL\_MEMORY\_BYTES, and COMMITTED\_HEAP\_BYTE. |
|  | | Job Counters-Job counters are maintained by jobtracker, which measures the job level statistics. Job counter maintains the statistics related to the particular job, and it is maintained by application master. The value of the job counters do not change during the execution of the job. |
|  | | e.g. Launched map tasks(TOTAL\_LAUNCHED\_MAPS)The number of map tasks that were launched. Includes tasks that were started Speculatively. |
|  | | Launched reduce tasks (TOTAL\_LAUNCHED\_REDUCES) The number of reduce tasks that were launched. Includes tasks that were started speculatively. |
|  | |  |
|  | | File System counters - File system maintains different aspects related to data read and written into the hdfs there are three counters in this type. |
|  | | 1. FILE\_BYTES\_WRITTEN. |
|  | | 2. HDFS\_BYTES\_READ. |
|  | | 3. HDFS\_BYTES\_WRITTEN. |
|  | | Q- Raw comparator VS Writable Comparator |
|  | | The key-value pairs (K2,V2) are called the intermediary key-value pairs. They are passed from the mapper to the reducer but in between shuffle sort will take place. The sorting can be done by two methods |
|  | | 1) Writable Comparable method |
|  | | 2) Raw Comparator |
|  | |  |
|  | | Writable comparator method –The writable comparable method uses the raw comparator hence it has to deserialized first and then sent to the raw comparator. |
|  | | E.g. |
|  | | publicclassIndexPair implementsWritableComparable<IndexPair> { |
|  | | privateIntWritablei; |
|  | | privateIntWritable j; |
|  | | //.... |
|  | | /\*\* |
|  | | \* Constructor. |
|  | | \* @paramii. |
|  | | \* @param j j. |
|  | | \*/ |
|  | | publicIndexPair(inti, intj) { |
|  | | this.i = newIntWritable(i); |
|  | | this.j = newIntWritable(j); |
|  | | } |
|  | | //.... |
|  | | @Override |
|  | | publicintcompareTo(IndexPair o) { |
|  | | intcmp = i.compareTo(o.i); |
|  | | if(0!= cmp) |
|  | | returncmp; |
|  | | returnj.compareTo(o.j); |
|  | | } |
|  | | //.... |
|  | | } |
|  | | //.... |
|  | | } |
|  | | If we use the raw comparator the comparasion will be done byte by byte hence will be faster |
|  | | publicclassIndexPairComparator extendsWritableComparator { |
|  | | protectedIndexPairComparator() { |
|  | | super(IndexPair.class); |
|  | | } |
|  | |  |
|  | | @Override |
|  | | publicintcompare(byte[] b1, ints1, intl1, byte[] b2, ints2, intl2) { |
|  | | inti1 = readInt(b1, s1); |
|  | | inti2 = readInt(b2, s2); |
|  | |  |
|  | | intcomp = (i1 < i2) ? -1: (i1 == i2) ? 0: 1; |
|  | | if(0!= comp) |
|  | | returncomp; |
|  | |  |
|  | | intj1 = readInt(b1, s1+4); |
|  | | intj2 = readInt(b2, s2+4); |
|  | | comp = (j1 < j2) ? -1: (j1 == j2) ? 0: 1; |
|  | |  |
|  | | returncomp; |
|  | | } |
|  | | } |
|  | | Here we can see that direct comparison is done using the compare method. |
|  | | Q.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.) |
|  | |  |