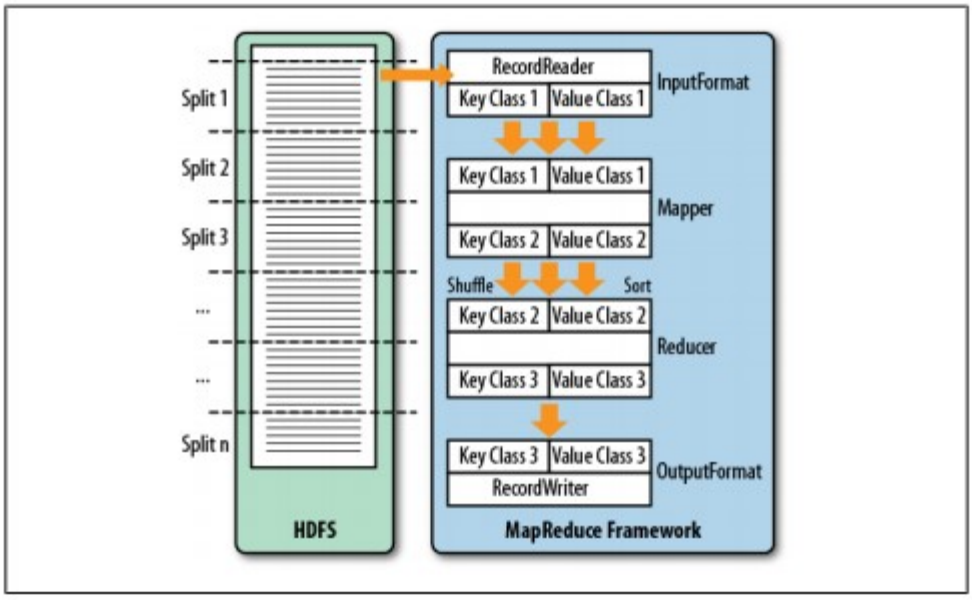
ASSIGNMENT NO: 11

AIM : Study of MapReduce in HBase.

INDEX TERMS: Map Reduce, MongoDB,

THEORY

MapReduce as a process was designed to solve the problem of processing in excess of terabytes of data in a scalable way. There should be a way to build such a system that increases in performance linearly with the number of physical machines added. That is what MapReduce strives to do. It follows a divide-and-conquer approach by splitting the data located on a distributed file system so that the servers (or rather CPUs, or more modern "cores") available can access these chunks of data and process them as fast as they can. The problem with this approach is that you will have to consolidate the data at the end. Again, MapReduce has this built right into it.



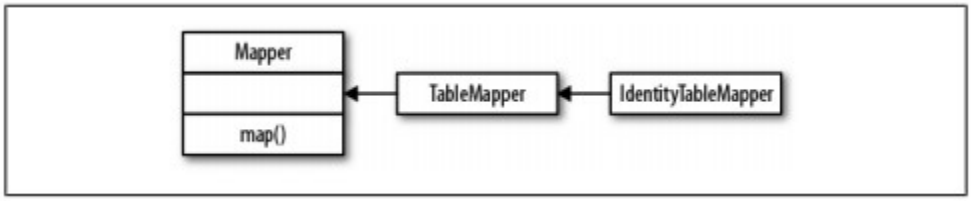
This (rather simplified) figure of the MapReduce process shows you how the data is processed. The first thing that happens is the split, which is responsible for dividing the input data into reasonably sized chunks that are then processed by one server at a time. This splitting has to be done in a somewhat smart way to make best use of available servers and the infrastructure in general. In this example, the data may be a very large logfile that is divided into pieces of equal size. This is good, for example, for Apache logfiles.

Classes

Above Figure also shows you the classes that are involved in the Hadoop implementation of MapReduce. Let us look at them and also at the specific implementations that HBase provides on top of them.

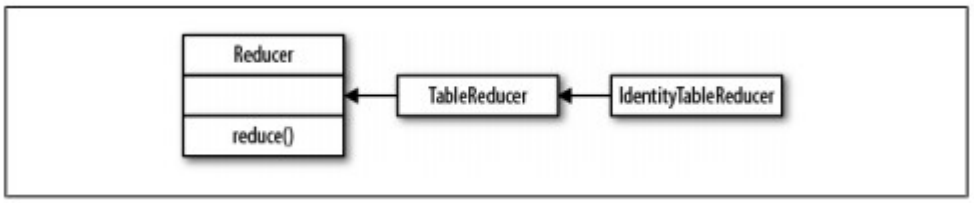
Mapper

The Mapper class(es) is for the next stage of the MapReduce process and one of its namesakes In this step, each record read using the RecordReader is processed using the map() method. Above Figure also shows that the Mapper reads a specific type of key/value pair, but emits possibly another type. This is handy for converting the raw data into something more useful for further processing. HBase provides the TableMapper class that enforces key class 1 to be an ImmutableBytes Writable, and value class 1 to be a Result type—since that is what the TableRecordReader is returning.



Reducer

The Reducer stage and class hierarchy is very similar to the Mapper stage. This time we get the output of a Mapper class and process it after the data has been shuffled and sorted. In the implicit shuffle between the Mapper and Reducer stages, the intermediate data is copied from different Map servers to the Reduce servers and the sort combines the shuffled (copied) data so that the Reducer sees the intermediate data as a nicely sorted set where each unique key is now associated with all of the possible values it was found with.



HBase MapReduce Read Example

The following is an example of using HBase as a MapReduce source in read-only manner. Specifically, there is a Mapper instance but no Reducer, and nothing is being emitted from the Mapper. There job would be defined as follows :

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config, "ExampleRead");
job.setJarByClass(MyReadJob.class);     // class that contains mapper
Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs
TableMapReduceUtil.initTableMapperJob(
  tableName,        // input HBase table name
  scan,             // Scan instance to control CF and attribute selection
  MyMapper.class,   // mapper
  null,             // mapper output key
  null,             // mapper output value
  job);
```

```
job.setOutputFormatClass(NullOutputFormat.class);   // because we aren't emitting anything from mapper

boolean b = job.waitForCompletion(true);

if (!b) {

  throw new IOException("error with job!");

}

public static class MyMapper extends TableMapper<Text, Text> {

  public void map(ImmutableBytesWritable row, Result value, Context context) throws InterruptedException, IOException {

    // process data for the row from the Result instance.

   }

}
```

HBase MapReduce Read/Write Example

The following is an example of using HBase both as a source and as a sink with MapReduce. This example will simply copy data from one table to another.

```
Configuration config = HBaseConfiguration.create();

Job job = new Job(config,"ExampleReadWrite");

job.setJarByClass(MyReadWriteJob.class);    // class that contains mapper

Scan scan = new Scan();

scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs

scan.setCacheBlocks(false);  // don't set to true for MR jobs

// set other scan attrs

TableMapReduceUtil.initTableMapperJob(

        sourceTable,      // input table

        scan,             // Scan instance to control CF and attribute selection

        MyMapper.class,   // mapper class

        null,             // mapper output key

        null,             // mapper output value
```

```
            job);
TableMapReduceUtil.initTableReducerJob(
            targetTable,      // output table
            null,             // reducer class
            job);
job.setNumReduceTasks(0);
boolean b = job.waitForCompletion(true);
if (!b) {
   throw new IOException("error with job!");
}
```

An explanation is required of what TableMapReduceUtil is doing, especially with the reducer. TableOutputFormat is being used as the outputFormat class, and several parameters are being set on the config (e.g., TableOutputFormat.OUTPUT_TABLE), as well as setting the reducer output key to ImmutableBytesWritable and reducer value to Writable. These could be set by the programmer on the job and conf, but TableMapReduceUtil tries to make things easier.

The following is the example mapper, which will create a Put and matching the input Result and emit it. Note: this is what the CopyTable utility does.

```
public static class MyMapper extends TableMapper<ImmutableBytesWritable, Put>  {

            public void map(ImmutableBytesWritable row, Result value, Context context) throws
IOException, InterruptedException {
                        // this example is just copying the data from the source table...
                        context.write(row, resultToPut(row,value));
            }

            private static Put resultToPut(ImmutableBytesWritable key, Result result) throws
IOException {
                        Put put = new Put(key.get());
                        for (KeyValue kv : result.raw()) {
                                    put.add(kv);
```

```
                }
                return put;
        }
}
```

There isn't actually a reducer step, so TableOutputFormat takes care of sending the Put to the target table. This is just an example, developers could choose not to use TableOutputFormat and connect to the target table themselves.

FAQS:
Q.1 Write TableInputFormatBase of Hbase to format input of Hbase-mapreduce operation?
Q2. Write methods of MapReduce implementation in Hbase?

Conclusion:
Outcome of the experiment is students are able to study MapReduce Operation with Hbase.