**i am Trying to Analyze the scenario**

**I have a hadoop cluster with three racks (R1,R2,R3). each rack consist of 10 servers and each server had 10 TB of storage.**

**Out of three RACKS,  RACK3 (R3) is complete lost.**

In this case,

Will NameNode replicate data blocks resides on RACK3 data nodes?

If yes,

how much time will take to replicate data blocks (10\*10 TB=100TB) from dead rack nodes  to running rack's data nodes?

In this case will see any performance issues (CPU/IO/ Network latencies) on remaining RACKS?

What about the jobs (Task trackers) running on the RACK3 data nodes?  Because  task tracker will also go down when I lost RACK3.

Will job tracker will rerun/initiate  the jobs on running RACKS data nodes?

What if i don't have enough storage to hold 100 TB of data on remaining RACK nodes?

Is there any way to control / instruct hadoop cluster not to replicate data blocks in case of RACK failure?

https://mail.google.com/mail/u/0/images/cleardot.gif

**PAUL  QUESTION  IS:**

I’m confused as to the three admin files.

hdfs-site.xml

core-site.xml

Mapred-site.xml

In pseudo mode are these three files read by the system for purpose of installation & configuration?

In pseudo mode there is only one node, one master node, one data node, one Job Tracker node and one task tracker node  and so I am wondering why the configuration files are necessary?

If the admin files are required for pseudo mode and distributed mode are they separate xml files or are there switches in the xml files to support eh different configurations such as replication, number of nodes etc.?

Are the files typically in different locations?

**ANSWER:**

All the three files are mandatory for both pseudo-mode and distributed mode. Here follows the purpose of each file in brief.

**hdfs-site.xml - contains the details of where to store the meta data of namenode and data of datanode.**

**mapred-site.xml - To configure the tasktracker properties like number of mappers and number of reducers for the cluster.**

**core-site.xml is the core where we will be mentioning the IP address of namenode.**

**MULTIPLE FILES READING:**

The logic looks fine, but ....

I was little curious if you had a chance to run this code in your current environment.

I know your current environment (whatever version you are teaching us) and the Prod environment currently supports MRV1 code only.

Coz, this is not MRV1 code.... This is MRV2 code.

1) So, we will be **very happy and glad**,  **if you teach us** the **new API/Features of MRV2**.

2) And also send us the new VM setup for the  MRV2 environment.

Otherwise,

1) we will end up with this dead version (MVR1)

and

we can't run this program **as is** in our current environment unless we make so many changes to all the 3 files (Map, Reduce and Configurator).

Mentioned below are **some of the required changes**  (more changes are required in reality....) that we have to do to run these

programs in our current environment.......

For example in the   
    1**) We have to change all the import statements**.  
         example: change the following statement   
               import org.apache.hadoop.***mapreduce***.Mapper;  
          to   
               import org.apache.hadoop.***mapred***.Mapper;      And also we have to import   
               import org.apache.hadoop.mapred.MapReduceBase;  
  
      There are many such changes we have to do to run this program in our current environment for the imports section.  
  
  
**2) Map Program**

**a) We have to change the class definition from   
MapDemo1 extends Mapper<LongWritable, Text, Text,Text>  
         to   
MapDemo1 extends MapperBase  implements  Mapper<LongWritable, Text, Text, IntWritable>**

***And also we need to change the method Syntax ....etc,***

**3) For reducer logic also**

**a)  We have to change the class definition from  
        class ReduceDemo extends Reducer<Text, Text, Text, FloatWritable>     
        to   
        class ReduceDemo extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>**

***And also we need to change the method Syntax ....etc,***

***4)   Finally we will end up changing most of the code in the Configurator section  i.e,***

***The code in the main method of your class.***

Conclusion:

Better if you can spend two classes on the MRV2 to, so that we will gain confidence and will be ready to work

in either environment (both MRV1 and MRV2).

**Attached the code for Total Word count & Multiple Files Input FYI**

.

**1) Multiple Files Input:**

**If all the input files are in one single directory the use the FileInputFormat.addInputPath.**

**If there more directories use the other flavors of the addInputPaths or setInputPaths...  
  
Note:**  Some times, some of then input files can have different formats, then we have to assign different Map class  
           for each path/file (group of files in a path). We can work on these scenarios once Sathya completes  
           the JOINS topic. The best or common example is master/detail or Header/Detail. Where the Header and   
           Detail file formats are different expect both will have a common key to join the files together.

**2) For the Total Word Count:**

    Since we are interested in the total number of words,

**a) Map - program:** Make sure to use Null Writable for the output parameters in the map interface implements clause  
       and in the output parameters of map method.  
       What happens here is that, when you make the keys Null Writable (output parm of map), the shuffler process runns,  
        it finds all the keys set to null and groups the data into this null category and sends it to the Reduce Process.

**b) The reduce process** takes this as input and thes the only available key (Null key) and sums up all the list of words  
       and sends the output back.  So make sure to use NullWritable for the reducer input, output KEYS and in the reduce method  
       input/output parameters.  
  
  c) Finally, since we are using **NullWritable** for the KEYS in Map/Reduce **output parameters**, be sure to set **NullWritable**  
      in  setMapOutputKeyClass and setOutputKeyClass of the Configuration/Driver class.

MapReduce has four phases:

* map,
* combine,
* shuttle and sort,
* reduce.

**Map phase** is done by mappers. Mappers run on unsorted input key/values pairs. The same physical nodes that keeps input data run also mappers. Each mapper emits zero, one or multiple output key/value pairs for each input key/value pair. Output key/value pairs are called intermediate key/value pairs. Their type is usually different from input key/value pair type. Mapper must be supplied by programmers.

**Combine phase** is done by combiners. Combiner should combine key/value pairs with the same key together. Each combiner may run zero, once or multiple times. Framework decides whether and how many times to run the combiner, programmer has no control over it. Combiners output key/value pair type must be the same as its input key/value pair types.

**Shuffle and sort phase** is done by framework. Data from all mappers are grouped by the key, split among reducers and sorted by the key. Each reducer obtains all values associated with the same key. Programmer may supply custom compare function for sorting and partitioner for data split. All key/value pairs going to the same reducer are sorted by the key, but there is no global sorting.

**Reducer** obtains sorted key/[values list] pairs sorted by the key. Values list contains all values with the same key produced by mappers. Each reducer emits zero, one or multiple output key/value pairs for each input key/value pair. Output key/value pair type is usually different from input key/value pair type. Reducer must be supplied by programmers.

If the algorithm requires multiple MapReduce iterations, each combiner may increment global counter. Driver program would read the counter after the reduce phase. It then decides whether next iteration is needed or not.

Straggler is either map or reduce task that takes unusually long time to complete

**Local aggregation (combining of key/value pairs) done inside the mapper.**

***What is speculative execution (also called backup tasks)? What problem does it solve?***

Identical copy of the same task is executed on multiple nodes. Output of the fastest task used.  
**Speculative execution helps if the task is slow because of hardware problem. It does not help if the distribution of values over keys is skewed**

***Explain mapper lifecycle.***

**The answer is:**

**Initialization method is called before any other method is called. It has no parameters and no output.**

**Map method is called separately for each key/value pair. It process input key/value pairs and emits intermediate key/value pairs.**

**Close method runs after all input key/value have been processed. The method should close all open resources. It may also emit key/value pairs.**

***Explain reducer lifecycle.***

The answer is:

Initialization method is called before any other method is called. It has no parameters and no output.

Reduce method is called separately for each key/[values list] pair. It process intermediate key/value pairs and emits final key/value pairs. **Its input is a key and iterator over all intermediate values associated with the same key.**

Close method runs after all input key/value have been processed. The method should close all open resources. It may also emit key/value pairs. 

**3.5 Relational Joins**

***Describe reduce side join between tables with one-on-one relationship.***

The answer is:

Mapper produces key/value pairs with join ids as keys and row values as value. Corresponding rows from both tables are grouped together by the framework during shuffle and sort phase.

Reduce method in reducer obtains join id and two values, each represents row from one table. Reducer joins the data.

***Describe reduce side join between tables with one-to-many relationship.***

The answer is:

We assume that the join key is primary key in table called S. Second table is called T. In other words, the table S in on the ‘one’ side of the relationship and the table T is on the ‘many’ side of the relationship.

**We have to implement mapper, custom sorter, partitioner and reducer**.

Mapper produces key composed from join id and table flag. Partitioner splits the data in such a way, that all key/value pairs with the same join id goes to the same reducer. Custom sort puts key/value pair generated from the table S right before key/value pair with the same join id from the table T.

Reducers input looks like this:   
((JoinId1, s)-> row)  
((JoinId1, t)-> [rows])  
((JoinId2, s)-> row)  
((JoinId2, t)-> [rows])  
...  
((JoinIdn, s), row)  
((JoinIdn, t), [rows])  
  
The reducer joins all rows from s pair with all rows from following t pair.

***Describe reduce side join between tables with many-to-many relationship.***

The answer is:

We assume that data are stored in tables called S and T. The table S is smaller. We have to implement mapper, custom sorter, partitioner and reducer.

Mapper produces key composed from join id and table flag. Partitioner splits the data in such a way, that all key/value pairs with the same join id goes to the same reducer. Custom sort puts the key/value pairs generated from the table S is right before all key/value pair with the data from the table T.

Reducers input looks like this:   
((JoinId1, s)-> [rows])  
((JoinId1, t)-> [rows])  
((JoinId2, s)-> [rows])  
((JoinId2, t)-> [rows])  
...  
((JoinIdn, s), [rows])  
((JoinIdn, t), [rows])  
  
The reducer buffers all rows with the same JoinId from the table S into the memory and joins them with following T table rows.

All data from the smaller table must fit into the memory – the algorithm has scalability bottleneck problem.

***Describe map side join between two database tables.***

The answer is:

Map side join works only if following assumptions hold:

* both datasets are sorted by the join key,
* both datasets are partitioned the same way.

Mapper maps over larger dataset and reads corresponding part of smaller dataset inside the mapper. As the smaller set is partitioned the same way as bigger one, only one map task access the same data. As the data are sorted by the join key, we can perform merge join O(n).

***Describe memory backed join.***

The answer is:

Smaller set of data is loaded into the memory in every mapper. Mappers loop over larger dataset and joins it with data in the memory. If the smaller set is too big to fit into the memory, dataset is loaded into memcached or some other caching solution.

***Which one is faster? Map side join or reduce side join?***:

Map side join is faster.

What is MapReduce?

It is a framework or a programming model that is used for processing large data sets over clusters of computers using distributed programming.

What are ‘maps’ and ‘reduces’?

‘*Maps*‘ and ‘*Reduces*‘ are two phases of solving a query in HDFS. ‘Map’ is responsible to read data from input location, and based on the input type, it will generate a *key value pair,*that is, an intermediate output in local machine*.* ’Reducer’ is responsible to process the intermediate output received from the mapper and generate the final output.

What are the four basic parameters of a mapper?

The four basic parameters of a mapper are*LongWritable, text, text and IntWritable*. The first two represent input parameters and the second two represent intermediate output parameters.

What are the four basic parameters of a reducer?

The four basic parameters of a reducer are *text, IntWritable, text, IntWritable*. The first two represent intermediate output parameters and the second two represent final output parameters.

What do the master class and the output class do?

Master is defined to update the Master or the job tracker and the output class is defined to write data onto the output location.

What is the input type/format in MapReduce by default?

By default the type input type in MapReduce is ‘text’.

Is it mandatory to set input and output type/format in MapReduce?

No, it is not mandatory to set the input and output type/format in MapReduce. By default, the cluster takes the input and the output type as ‘text’.

What does the text input format do?

In text input format, each line will create a line object, that is an hexa-decimal number. Key is considered as a line object and value is considered as a whole line text. This is how the data gets processed by a mapper. The mapper will receive the ‘key’ as a ‘*LongWritable*‘ parameter and value as a ‘*text*‘ parameter.

What does job conf class do?

MapReduce needs to logically separate different jobs running on the same cluster. ‘*Job conf class*‘  helps to do job level settings such as declaring a job in real environment.  It is recommended that Job name should be descriptive and represent the type of job that is being executed.

What does conf.setMapper Class do?

*Conf.setMapper* class sets the mapper class and all the stuff related to map job such as reading a data and generating a *key-value pair* out of the mapper.

What do sorting and shuffling do?

Sorting and shuffling are responsible for creating a unique key and a list of values. Making similar keys at one location is known as *Sorting*. And the process by which the intermediate output of the mapper is sorted and sent across to the reducers is known as *Shuffling*.

What does a split do?

Before transferring the data from hard disk location to map method, there is a phase or method called  the ‘*Split Method*‘. Split method pulls a block of data from HDFS to the framework. The *Split class* does not write anything, but reads data from the block and pass it to the mapper. Be default, Split is taken care by the framework. Split method is equal to the block size and is used to divide block into bunch of splits.

How can we change the split size if our commodity hardware has less storage space?

If our commodity hardware has less storage space, we can change the split size by writing the ‘*custom splitter*‘. There is a feature of customization in Hadoop which can be called from the main method.

What does a MapReduce partitioner do?

*A MapReduce partitioner* makes sure that all the value of a single key goes to the same reducer, thus allows evenly distribution of the map output over the reducers. It redirects the mapper output to the reducer by determining which reducer is responsible for a particular key.

How is Hadoop different from other data processing tools?

In Hadoop, based upon your requirements, you can increase or decrease the number of mappers without bothering about the volume of data to be processed. this is the beauty of parallel processing in contrast to the other data processing tools available.

Can we rename the output file?

Yes we can rename the output file by implementing *multiple format output class*.

Why we cannot do aggregation (addition) in a mapper? Why we require reducer for that?

We cannot do aggregation (addition) in a mapper because, sorting is not done in a mapper. Sorting happens only on the reducer side. Mapper method initialization depends upon each input split. While doing aggregation, we will lose the value of the previous instance. For each row, a new mapper will get initialized. For each row, input split again gets divided into mapper,  thus we do not have a track of the previous row value.

What is Streaming?

Streaming is a feature with Hadoop framework that allows us to do programming using MapReduce in any programming language which can accept standard input and can produce standard output. It could be Perl, Python, Ruby and not necessarily be Java. However, customization in MapReduce can only be done using Java and not any other programming language.

What is a Combiner?

A ‘Combiner’ is a mini reducer that performs the local reduce task. It receives the input from the mapper on a particular node and sends the output to the reducer. Combiners help in enhancing the efficiency of MapReduce by reducing the quantum of data that is required to be sent to the reducers.

What is the difference between an HDFS Block and Input Split?

*HDFS Block* is the physical division of the data and ***Input Split* is the logical division of the data.**

What happens in a textinputformat?

In *textinputformat*, each line in the text file is a record. *Key* is the byte offset of the line and *value* is the content of the line. For instance, **Key: longWritable, value: text**.

What do you know about keyvaluetextinputformat?

In *keyvaluetextinputformat*, each line in the text file is a ‘*record*‘. The first separator character divides each line. Everything before the separator is the *key* and everything after the separator is the *value*. For instance, **Key: text, value: text.**

What do you know about Sequencefileinputformat?

*Sequencefileinputformat*is an input format for reading in sequence files. *Key* and *value* are user defined. It is a specific compressed binary file format which is optimized for passing the data between the output of one MapReduce job to the input of some other MapReduce job.

What do you know about Nlineoutputformat?

***Nlineoutputformat* splits ‘n’ lines of input as one split.**