***Assignment 20.3***

***Problem Statement:***

***Explain in brief Writable and Writable Comparable in Hadoop with an example.***

**Writable in Hadoop:**

* Writable - interface in Hadoop.
* Where it will  acts as wrapper class for all the primitive data type available in Java.
* This makes the int of java as IntWritable in Hadoop
* And  String of Java as Text in Hadoop.
* Writables I nhadoop were used for creating the serialized data types.
* Hadoop frame work were in need of Writable type of interface for performing the following tasks:
* Implement serialization
* For Transferring the data between clusters and networks.
* And they Store the deserialized data to the local disk present in the system.
* writable Implementation will be similar to implementation of interface which is in Java.
* The keyword **‘implements’** is used for doing this process .
* And overriding - default writable method.
* Writable - serializing data as well will reduce the data size.
* Within the networks the data can be exchanged.
* Read and write fields will be separated in order to read data from network and write data into respective disk.
* Hadoop data which is present inside should accept writable and comparable interface properties.
* When key is made as IntWritable in the Mapper class
* And those key will be sent to the reducer class.
* There will be an intermediate phase which will be between Mapper and Reducer class
* i.e., shuffle and sort.
* where each key has to be compared with other keys.
* When the keys are not comparable, then shuffle and sort cannot be executed or may be done with a high amount of overhead.

**For example,**

* The steps to make a custom type in Java is as follows:

public class add {

int a;

int b;

public add() {

this.a = a;

this.b = b;

}

}

Custom type can be done in Hadoop using Writables.

* For implementing Writables, we need few more methods in Hadoop:

public interface Writable {

void readFields(DataInput in);

void write(DataOutput out);

}

* ReadFields will read the data from network
* write will write the data into the local disk.
* Both were needed for data transfer through clusters.
* DataInput and DataOutput classes are part of java.io which will contain methods in order to serialize the most of the basic data types.

If in case one needs to make a composite key in Hadoop by combining two Writables then following steps need to be followed:

public class add implements Writable{

public int a;

public int b;

public add(){

this.a=a;

this.b=b;

}

public void write(DataOutput out) throws IOException {

out.writeInt(a);

out.writeInt(b);

}

public void readFields(DataInput in) throws IOException {

a = in.readInt();

b = in.readInt();

}

public String toString() {

return Integer.toString(a) + ", " + Integer.toString(b);

}

}

* One can create custom Writables which is similar to custom data type in Java.
* but it needs two additional methods

i.)write

ii.)readFields.

* Custom writable will travel through networks which will reside in other systems.
* This custom type cannot be compared with each other by default custom type,
* If incase we need to make them comparable with each other.

**Writable Comparable in Hadoop:**

* When a key is taken as a default IntWritable then it will possesses with comparable feature.
* Due to RawComparator which acts on the variable .
* It will compare the key with other keys inside the network
* When Writable is not present then it won’t be executed.
* IntWritable, LongWritable and Text have a RawComparator by default .
* It will execute this comparable phase for them.
* WritableComparable is needed in order to use the RawComparator which helps the custom Writable.
* WritableComparable - sub interface of Writable, an it possess the feature of Comparable in it.

*If one create our custom type writable is there any need for WritableComparable?*

* Then one needs to make the custom type into comparable they need to compare this type with the other types.
* custom type is taken as a key.
* key type is taken as WritableComparable instead of Writable.
* Which means custom type will be compared with other types.
* As well be sorted accordingly.
* Or else, keys will not be compared with each other keys
* As well will be passed through the network.

**For example,**

WritableComparable will implemented similar to Writable

But in additional ‘CompareTo’ will be used along.

public interface WritableComparable extends Writable, Comparable

{

void readFields(DataInput in);

void write(DataOutput out);

int compareTo(WritableComparable o)

}

*We can make custom type a WritableComparable by following the method below:*

public class add implements WritableComparable{

public int a;

public int b;

public add(){

this.a=a;

this.b=b;

}

public void write(DataOutput out) throws IOException {

out.writeint(a);

out.writeint(b);

}

public void readFields(DataInput in) throws IOException {

a = in.readint();

b = in.readint();

}

public int CompareTo(add c){

int presentValue=this.value;

int CompareValue=c.value;

return (presentValue < CompareValue ? -1 : (presentValue==CompareValue ? 0 : 1));

}

public int hashCode() {

return Integer.IntToIntBits(a)^ Integer.IntToIntBits(b);

}

}

This will help the read fields as well the write field to compare the data faster within the network.

Using Writable and WritableComparables in Hadoop one can serialized custom type with less difficulty.

This make the developers to choose their required custom types which is based on their requirement.