**Assignment 20.3**

**Problem Statement:**

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

**Answer:**

**Writable in Hadoop:**

Writable is an interface in Hadoop. Writable in Hadoop acts as a wrapper class to almost all the primitive data type of Java. That is how int of java has become IntWritable in Hadoop and String of Java has become Text in Hadoop. Writables are used for creating serialized data types in Hadoop.

Hadoop frame work definitely needs Writable type of interface in order to perform the following tasks:

Implement serialization

Transfer data between clusters and networks

Store the deserialized data in the local disk of the system

Implementation of writable is similar to implementation of interface in Java. It can be done by simply writing the keyword ‘implements’ and overriding the default writable method.

Writable is a strong interface in Hadoop which while serializing the data, reduces the data size enormously, so that data can be exchanged easily within the networks. It has separate read and write fields to read data from network and write data into local disk respectively. Every data inside Hadoop should accept writable and comparable interface properties.

When we write a key as IntWritable in the Mapper class and send it to the reducer class, there is an intermediate phase between the Mapper and Reducer class i.e., shuffle and sort, where each key has to be compared with many other keys. If the keys are not comparable, then shuffle and sort phase won’t be executed or may be executed with 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;

}

}

Similarly we can make a custom type 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);

}

Here, readFields, reads the data from network and write will write the data into local disk. Both are necessary for transferring data through clusters. DataInput and DataOutput classes (part of java.io) contain methods to serialize the most basic types of data.

Suppose we want to make a composite key in Hadoop by combining two Writables then follow the steps below:

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);

}

}

Thus we can create our custom Writables in a way similar to custom types in Java but with two additional methods, write and readFields. The custom writable can travel through networks and can reside in other systems.

This custom type cannot be compared with each other by default, so again we need to make them comparable with each other.

**Writable Comparable in Hadoop:**

If a key is taken as IntWritable, by default it has comparable feature because of RawComparator acting on that variable and it will compare the key taken with the other keys in network and If Writable is not there it won’t be executed.

By default, IntWritable, LongWritable and Text have a RawComparator which can execute this comparable phase for them. We need to have WritableComparable to use RawComparator help the custom Writable.

WritableComparable can be defined as a sub interface of Writable, which has the feature of Comparable too. If we have created our custom type writable, then why do we need WritableComparable?

We need to make our custom type, comparable if we want to compare this type with the other.

We want to make our custom type as a key, then we should definitely make our key type as WritableComparable rather than simply Writable. This enables the custom type to be compared with other types and it is also sorted accordingly. Otherwise, the keys won’t be compared with each other and they are just passed through the network.

For example,

The implementation of WritableComparable is similar to Writable but with an additional ‘CompareTo’ method inside it.

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);

}

}

These read fields and write make the comparison of data faster in the network.With the use of these Writable and WritableComparables in Hadoop, we can make our serialized custom type with less difficulty. This gives the ease for developers to make their custom types based on their requirement.