1. Explain in brief

● Sequence File Format

● NLine Input Format

● DB Input Format

● DB Output Format

1. Sequence File Format

We know that Hadoop’s performance is drawn out when we work with a small number of files with big size rather than a large number of files with small size. If the size of a file is smaller than the typical block size in Hadoop, we consider it as a small file. Due to this, a number of metadata increases which will become an overhead to the NameNode. To solve this problem sequence files are introduced in Hadoop. Sequence files act as a container to store the small files.

Sequence files are flat files consisting of binary key-value pairs. When Hive converts queries to MapReduce jobs, it decides on the appropriate key-value pairs to be used for a given record. Sequence files are in the binary format which are able to split and the main use of these files is to club two or more smaller files and make them as a one sequence file.

There are three SequenceFile Writers based on the SequenceFile.CompressionType used to compress key/value pairs:

Writer : Uncompressed records.

RecordCompressWriter : Record-compressed files, only compress values.

BlockCompressWriter : Block-compressed files, both keys & values are collected in 'blocks' separately and compressed. The size of the 'block' is configurable.

The actual compression algorithm used to compress key and/or values can be specified by using the appropriate CompressionCodec.

The recommended way is to use the static createWriter methods provided by the SequenceFile to chose the preferred format.

The SequenceFile.Reader acts as the bridge and can read any of the above SequenceFile formats.

1. NLine Input Format

NLineInputFormat With TextInputFormat and KeyValueTextInputFormat, each mapper receives a variablenumber of lines of input. The number depends on the size of the split and the length of the lines. If you want your mappers to receive a fixed number of lines of input, then NLineInputFormat is the InputFormat to use. Like with TextInputFormat, the keys are the byte offsets within the file and the values are the lines themselves. N refers to the number of lines of input that each mapper receives. With N set to 1 (the default), each mapper receives exactly one line of input. The mapreduce.input.lineinputformat.linespermap property controls the value of N. By way of example, consider these four lines again:

On the top of the Crumpetty Tree

The Quangle Wangle sat,

But his face you could not see,

On account of his Beaver Hat.

If, for example, N is 2, then each split contains two lines. One mapper will receive the first two key-value pairs:

(0, On the top of the Crumpetty Tree)

(33, The Quangle Wangle sat,)

And another mapper will receive the second two key-value pairs:

(57, But his face you could not see,)

(89, On account of his Beaver Hat.)

The keys and values are the same as those that TextInputFormat produces. The difference is in the way the splits are constructed.

Usually, having a map task for a small number of lines of input is inefficient (due to the overhead in task setup), but there are applications that take a small amount of input data and run an extensive (i.e., CPU-intensive) computation for it, then emit their output. Simulations are a good example. By creating an input file that specifies input parameters, one per line, you can perform a parameter sweep: run a set of simulations in parallel to find how a model varies as the parameter changes.

1. DB Input Format

DBInputFormat is an input format for reading data from a relational database, using JDBC. Because it doesn’t have any sharding capabilities, you need to be careful not to overwhelm the database from which you are reading by running too many mappers. For this reason, it is best used for loading relatively small datasets, perhaps for joining with larger datasets from HDFS using MultipleInputs. The corresponding output format is DBOutputFormat, which is useful for dumping job outputs (of modest size) into a database.

1. DB Output Format

he OutputFormat interface determines where and how the results of your job are persisted. Hadoop comes with a collection of classes and interfaces for different types of formats, and customization is done by extending one of these. You are probably already familiar with the default OutputFormat: a line separated, tab delimited text file of key-value pairs. This is the TextOutputFormat. But for many types of data, most obviously numbers, text serialization wastes space, which translates into longer run times and more resources consumed. To avoid the drawbacks of text files, Hadoop provides the SequenceFileOutputformat which can write the binary representation of objects instead of converting it to text, and compress the result. Below is the class hierarchy Hadoop provides:

FileOutputFormat (implements interface OutputFormat) – base class for all OutputFormats

MapFileOutputFormat – a format for partially indexed keys

SequenceFileOutputFormat – compressed binary key-value data

SequenceFileAsBinaryOutputFormat – compressed raw binary data

TextOutputFormat – line separated, tab delimited text file of key-value pairs

MultipleOutputFormat – abstract class for writing a file paramaterized by the key-value

MultipleTextOutputFormat – write to multiple files in the standard line separated, tab delimited format

MultipleSequenceFileOutputFormat – write to multiple files in the compressed file format

public class DBOutputFormat<K extends DBWritable,V>

extends OutputFormat<K,V>

A OutputFormat that sends the reduce output to a SQL table.

DBOutputFormat accepts <key,value> pairs, where key has a type extending DBWritable. Returned RecordWriter writes only the key to the database with a batch SQL query.