MAPREDUCE TYPES AND FORMATS

Hadoop uses the MapReduce programming model for the data processing of input and output for the map and to reduce functions represented as key-value pairs. They are subject to parallel execution of datasets situated in a wide array of machines in a distributed architecture. The programming paradigm is essentially functional in nature in combining while using the technique of map and reduce. This article introduces the MapReduce model, and in particular, how data in various formats, from simple text to structured binary objects are used.

MAPREDUCE TYPES.

Mapping is the core technique of processing a list of data elements that come in pairs of keys and values. The map functions applies to individual elements defined as key-value-pairs of a list and produces a newlist. The general idea of map and reduce function of Hadoop can be illustrated as follows:-

map: (K1, V1) -> list (K2, V2)

reduce: (K2, list (V2)) -> list (K3, V3)

The Default MapReduce Job.

The default mapper is Identitymapper, which writes the input key and value unchanged to the output.

- Mapreduce take Two arguments (input &output)
- Public class Identify Mapper <K, V> extends MapReduceBase Implements Mapper <K, V, K, V)

```
  {
  public void map (K key, V val, output collector
                    <K, V> output, Reporter, reporter)
  throw IO Exception
   {
   output.collect(key, value ;
   }
  }
```

- IdentityMapper is a generic type, which allows it to work with any key or value types, with the restriction that the map input and output keys are same type, and map input and output values are same type.

- map output key is Longwritable.
- map output value is Text.

* Input Formats

Hadoop can process many different types of data formats, from flat text files to data bases.

- Different formats are available.

* Input Splits and Records

Input split is a chunk of the input that is processed by a single map. The inputsplit represents the data to be processed by a Map. It returns the length in bytes and has a reference to the input data. It presents a byte-oriented view on the Input and is the responsibility of the Record ~~Readed~~ Reader of the job to process this and present a record-oriented view.

- In most cases, we do not deal with Inputsplit directory because they are created by an ~~Inputsplit~~ inputformat.

eg:
```
Public Interface Inputformat <K, V> {
      Inputsplit [] getsplits ( Job conf job.
              IO Exception;
      RecordReader <K, V> get RecordReader (Input-
                  Jobconf job, throws IOException;

  }
```

* The fileInputformat is the baseclass for the file-data source. It has the responsibility to Identify

the files that are to be included as the job input and the definition for generating the split.

- * Hadoop also includes processing of unstructured data that often comes internal formats. the TextInput Format is the default inputFormat for such data.
- * The SequenceInputFormat takes up binary inputs and stores sequences of binary key-value pairs.
- * DatabaseInputFormat provides the capability to read data from relational database using JDBC.

Text Input

Hadoop excel at processing unstructured text.

TextInputFormat

It is the default inputFormat. Each record is a line of input. The key, a longwritable is the byte offset within the file of the beginning of the line. The value is the contents of the line, excluding any line terminators and is packaged an a text object.

KeyValue TextInput Format.

It's keys being simply the offset within the file are not normally very useful. It is -

common for each line in a file to be a key-value pair, seperated by a delimiter such as a tab character.

NlineInputFormat

With TextInput Format and KeyvalueTextinputformat, each mapper recieves a variable no of lines of input. The number depends on the size of the split and the length of the lines

XML

- Most XML parcer, operate on whole XML document so it a large XML document is madeup of multiple input split - then it is a challenge to parse these individually
- Hadoop comes with a class for this purpose called stream XML RecordReader.

Binary Input

Hadoop MapReduce is not just restricted to processing textout data - it has support for binary formats too.

SequenceFile Input Format.

Hadoop's sequence File format store sequences of binary key-value pairs. They are well suited as a format for mapreduce data since they are splitable, they support chairperson as a part of the format and they can store arbitery type using variety of serialization Framework.

Multiple Inputs

Although the input to a MapReduce job may consist of multiple input files all of the input is interpreted by single InputFormat and a single Mapper.

Database Input (& output)

DBInputformat is an input format for redding data from a relational database using JDBC. Because it doesn't have any shording capabilities you need to be careful not to overwhelms the database you are reading from by running too many mappers.

SequenceFile as Input.Format

It is a varient of sequenceFileInputFormat that converts the sequence File's key and values to Text Objects. The conversion is performed by calling to String() on the keys and values. This format makes sequence file suitable input for streaming

OUTPUT FORMATS

The output format classes are similar to their corresponding input format classes- and work in the reverse direction.

* Text Output Formats

It is the default output format that writes records as plaintext files, whereas key-values can be any types, and transforms- them into a string by working they to string() method. The key-value character is seperated by- the tab charracter, although this can be customised by manipulating the seperator property of the- text output Formats.

* Binary Output.

There is sequence fileInput format to- write a sequence of binary output to a file. Binary outputs are particularly useful if the- output becomes input to a further mapReduce Job.

* The outputformats for relational databases- and to HBase are handled by DB outputFormat.

* Multiple Outputs

It allows writing data to files whose names are derived from output keys and values, or in fact from an arbitery string

Lazy output formats

Sometimes file output formats will create output files even if they are empty lazy-output formats is wrapper output formats which ensuses that the output file will be created only when the record is emitted for given partition.

✱ MAPREDUCE FEATURES.

(a)

1) Counters

Hadoop counters provides a way to measure the progress or the no: of operations that occur within map/reduce job. Counters in Hadoop-MapReduce are a useful channel for gathering statistics about the mapreduce job, for quality control or for application-level. They are also useful for problem diagonisis.

- Counters represents Hadoop Global counters, defined either by the mapreduce frame work or applications.
- Counters are branched into groups. each comprising of counters from a particular Enum class.

✱ Hadoop candidates validate that :-

- The correct no. of bytes was read and written
- The correct no. of tasks was launched & successfully

The amount of CPU and memory consumed is appropriate for our jobs and cluster nodes.

2) Built-In Counters

Hadoop maintains some built-in hadoop counters for every job and these report various metrics, like, there are counters for the number of bytes and records, which allows as to confirm that the expected amount of input is consumed and the expected amount of output is produced.

- Hadoop counters are divided into groups and there are several groups for the built-in counters. Each group either contains task counters or job counter.
- These are several groups for the hadoop built-in counters.
- Hadoop task counters are maintained by each task attempt and periodically sent to the application master so they can be globally aggregated.

3) User-Defined Java Counters

Hadoop MapReduce permits user code to define set of counters. Then it increment them as desired in the mapper or reducer. Like in Java to define counters is user, 'enums'.

- A job may define an arbitery number of 'enums' each with an arbitrary number of fields. The name of the enum is the group name. The enum's fields are the counternames.

i) Dynamic counters in Hadoop

Java enum's fields are defined at compile time. So we cannot create new counters at runtimeusing enums. So, we use dynamic counters to create new counters at runtime. But dynamic counter is not defined at compiletime

ii) Readable counter names

By default, a counter's name is the enum's fully qualified Java classname. These names are not vesy readablewhen they appear on the web UI ~~for~~ in the console, So hadoop providesaway to change the display names using resousse bundles

iii) Retrieving counters

In addition, to being available via the web UI and the commandline, you can retrieving counter values using the Java API.

iv) User-defined Streaming Counters

A streaming MapReduce prgm can increment counters by sending a specially formatted line to the standard error stream. which is co-opted as a control channel in this case.

* Sorting

The ability to sort data is at the heart of MapReduce. Even if your application isn't concerned with sorting per se, it may be able to use the sorting stage that MapReduce provides to organise its data.

i) Preperation

We are going to sort the weather dataset by temperature. storing temperatures as Text objects doesn't work sorting purposes, since signed integers don't sort l exit graphically.

ii) Partial Sort

The reducer output will be lot of files each of which is sorted within itself based on the key

. N number of mappers will simply generate N number of reducers will sort these files individually.

iii) Total Sort

- The reducer output will be a single file having all the output sorted based on the key
- All key value pairs from a particular key will reach a particular reducer. This will happen through partioners at mapper level. combiners at mapper level will act as semireducers and

send values of a particular key to reducer.

iv) Secondary sort

We will be able to control the ordering of the values along will the keys. That is sorting can be done on two or more field values.

* Joins

MapReduce can perform join between large dataset, but writing the code to do join from scratch is fairly involved. Rather than writing MapReduce prgm, you might consider using a higher-level framework such as pig, Hive or cascading in which join operation are a core part of the implementation.

- We have two datasets.
  for eg: the weather station database and the weather records - we want to reconcile the two.
- How we implement the join depends on how large the datasets and how they are partitional.
- If one dataset is large but the other one is small enough to be distributed to each node in the cluster.

* Side data distribution

Side data ~~distribut~~ can be defined as extra read only data needed by a job to process the main dataset. The challenge is to make side-data available to all the map or reduce. tasks in a convenient and efficient function.

* Distributed Cache

Rather than serializing side data in the job configuration, it is preferable to distribute dataset using Hadoop's distributed cache mechanism. This provides a service for copying files and archives to the task ~~[illegible]~~ nodes in-time for the task to use them when they run. To save network bandwidth, files are normally copied to any particular node once per job.

AH
04/11/201