Why Is Big Data Important?

Note that Big Data solutions aren't a replacement for your existing warehouse solutions.

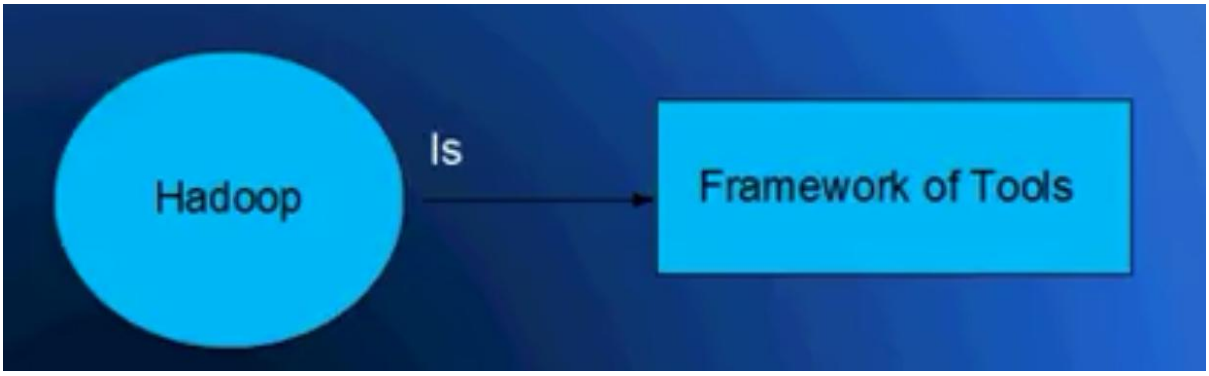
when you use Big Data technologies:

- **Big Data solutions are ideal for analyzing not only raw structured data, but semistructured and unstructured data from a wide variety of sources.**

- **Big Data solutions are ideal when all, or most, of the data needs to be analyzed versus a sample of the data; or a sampling of data isn't nearly as effective as a larger set of data from which to derive analysis.**

- **Big Data solutions are ideal for iterative and exploratory analysis when business measures on data are not predetermined.**

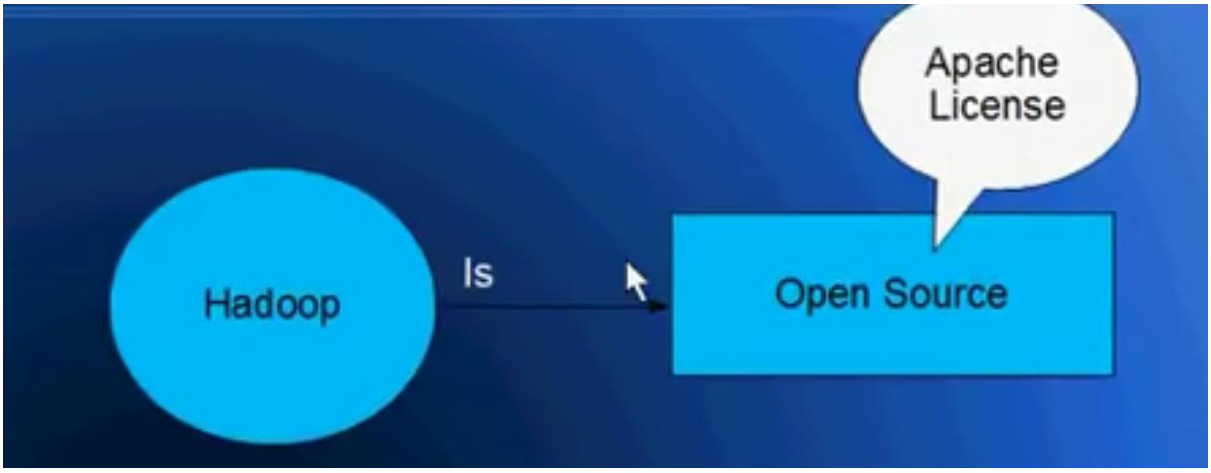
What is Hadoop and History



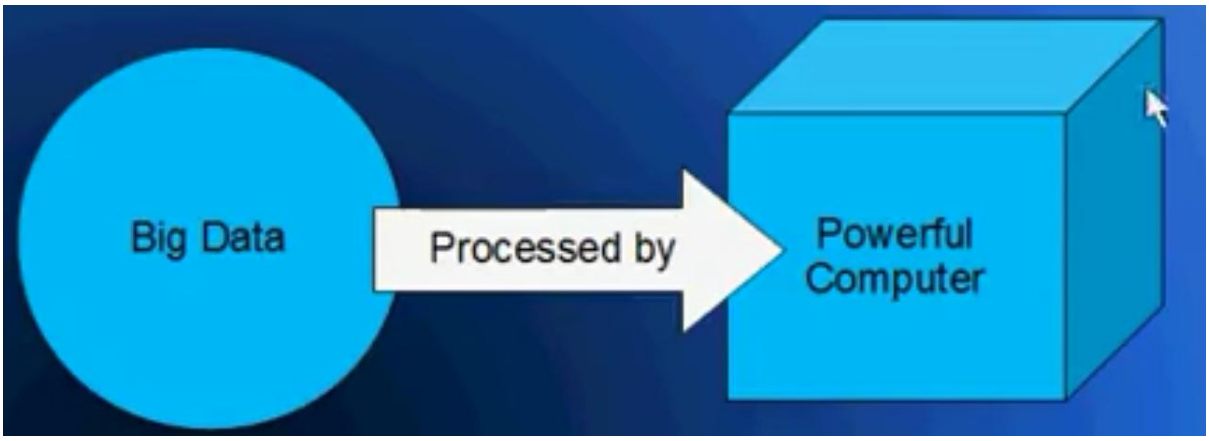
- **Open source software framework of tools designed for storage and processing of large scale data on clusters of commodity hardware.**
- **Created by Doug Cutting and Mike Carafella in 2005.**
- **Cutting named the program after his son's toy elephant.**

The object of the tools is to support running of application on the big data

Hadoop is open source set of tools distributed under apache license.



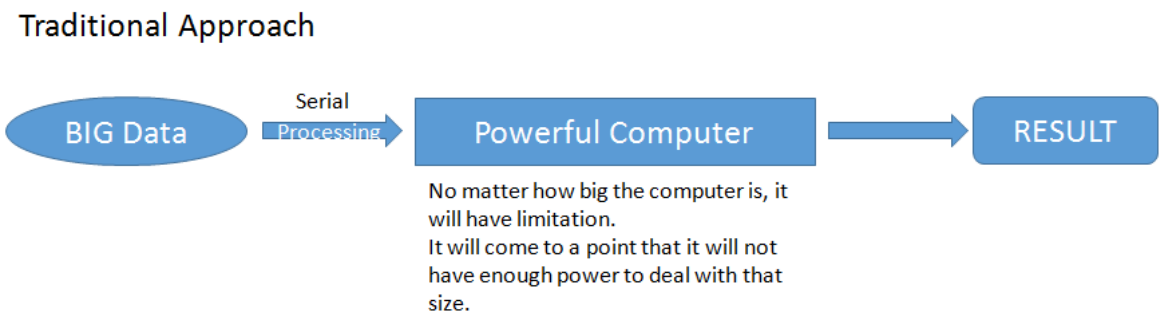
Traditional Approach



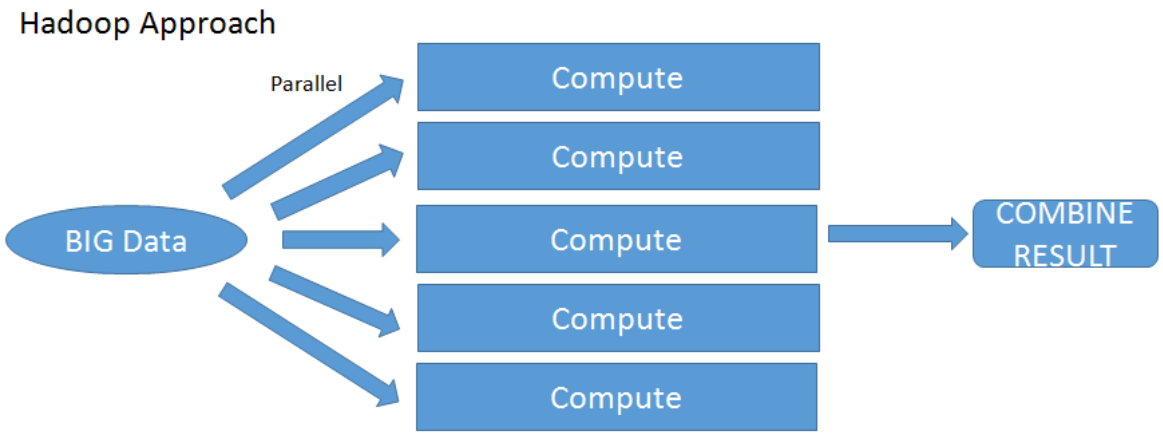
Limitation to the traditional approach:

Hadoop Approach:

Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. In short, Hadoop framework is capable enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for a huge amounts of data.

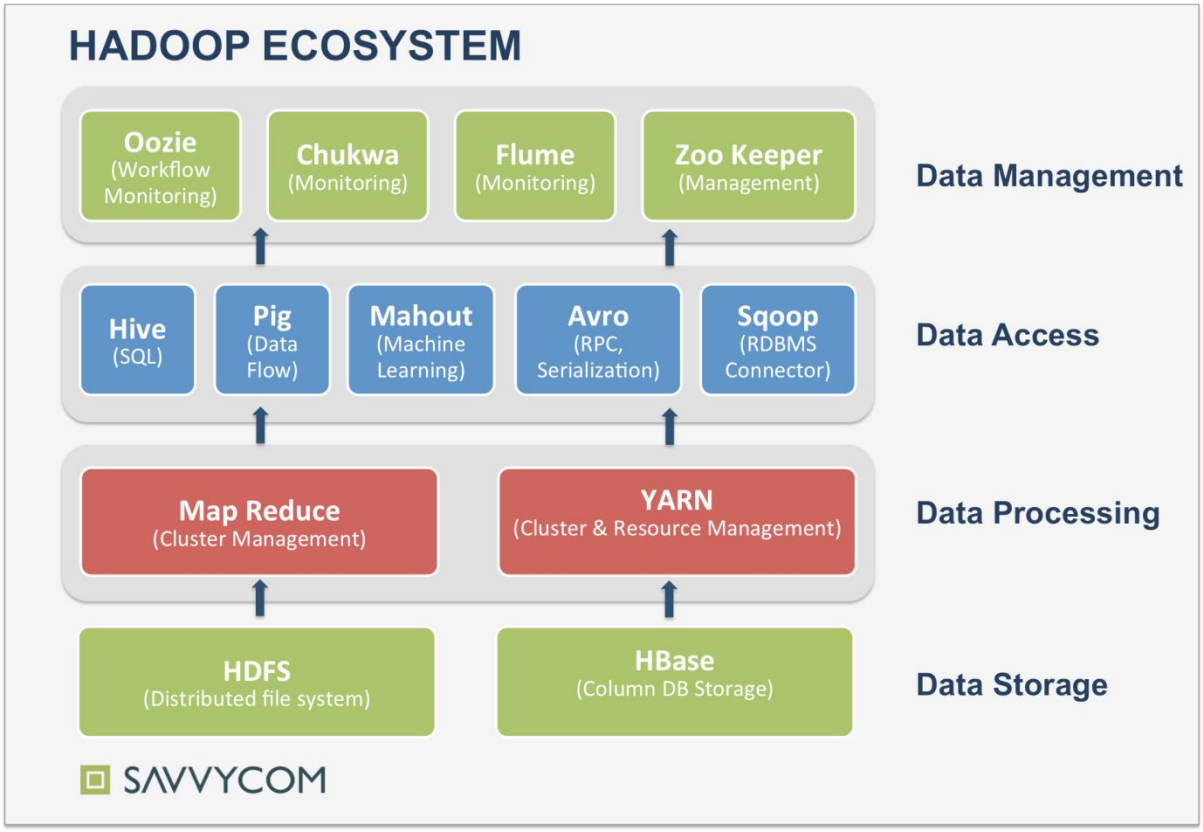


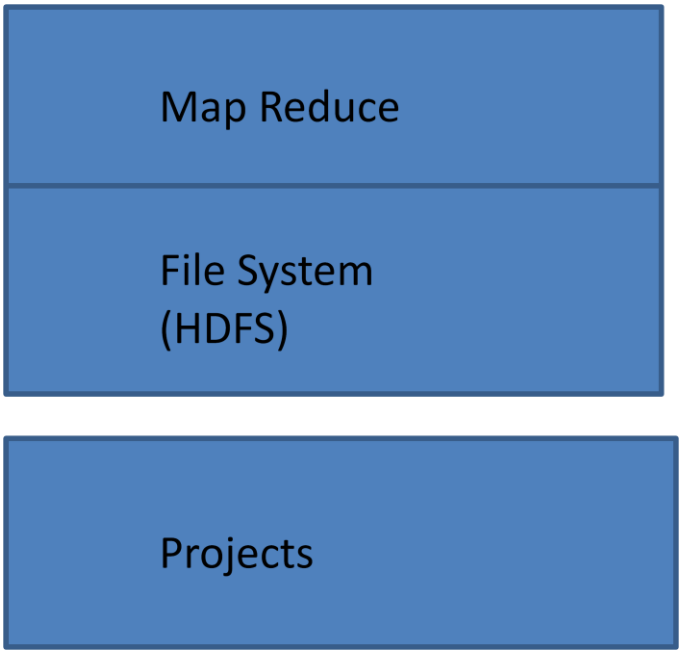
Problem: *Only so much data could be processed on a certain point because it is NOT SCALABLE.*



Solution: *Distributed* - process data in parallel rather than in serial.

Simple Hadoop Architecture





The Hadoop projects that are covered in this book are described briefly here:

Common

A set of components and interfaces for distributed filesystems and general I/O (serialization, Java RPC, persistent data structures).

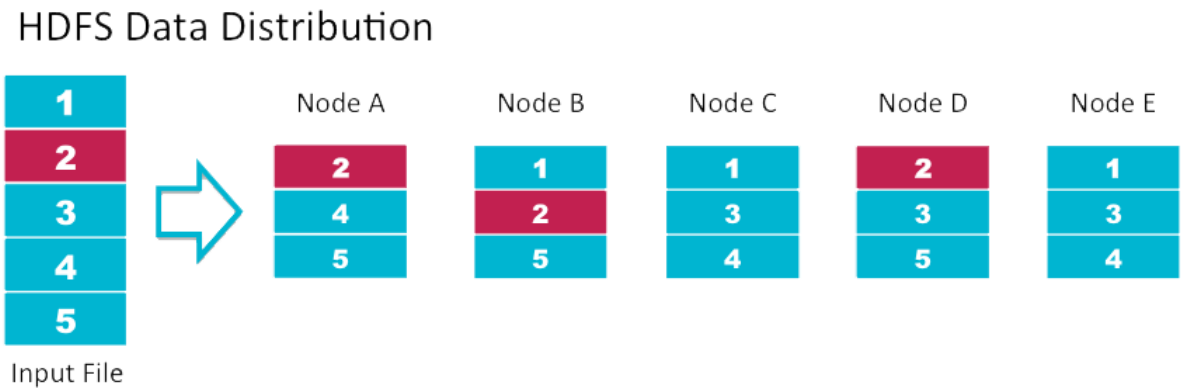
Avro

A serialization system for efficient, cross-language RPC, and persistent data storage.

MapReduce
A distributed data processing model and execution environment that runs on large clusters of commodity machines.

HDFS
A distributed filesystem that runs on large clusters of commodity machines.

- Files are split into blocks
- Blocks are split across many machines at load time
  - Different blocks from the same file will be stored on different machines
- Blocks are replicated across multiple machines
- The NameNode keeps track of which blocks make up a file and where they are stored



Pig
A data flow language and execution environment for exploring very large datasets.
Pig runs on HDFS and MapReduce clusters.
or A high level platform for creating MapReduce programs using language called Pig Latin. Similar to that of SQL for RDBMS system

Hive
A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to MapReduce jobs) for querying the data.
or A data warehouse infrastructure for providing data summarization ,query and analysis .

HBase
A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).
or An open source non relational distributed database written in java running on top of HDFS

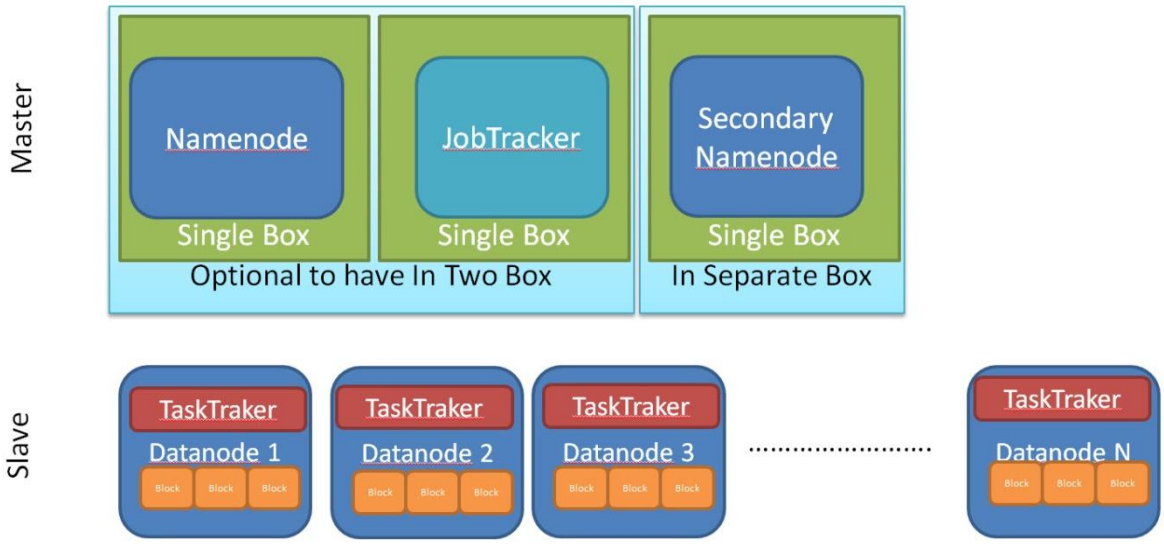
ZooKeeper
A distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.

Sqoop
A tool for efficiently moving data between relational databases and HDFS.

Apache Hadoop is designed to have Master Slave architecture:

Master: Namenode, JobTracker

Slave: {DataNode, TaskTraker}, ….. {DataNode, TaskTraker}



HDFS is one primary components of Hadoop cluster and HDFS is designed to have Master-slave architecture.

Master: NameNode

Slave: {Datanode}…..{Datanode}

- The Master (NameNode) manages the file system namespace operations like opening, closing, and renaming files and directories and determines the mapping of blocks to DataNodes along with regulating access to files by clients

- Slaves (DataNodes) are responsible for serving read and write requests from the file system's clients along with perform block creation, deletion, and replication upon instruction from the Master (NameNode).

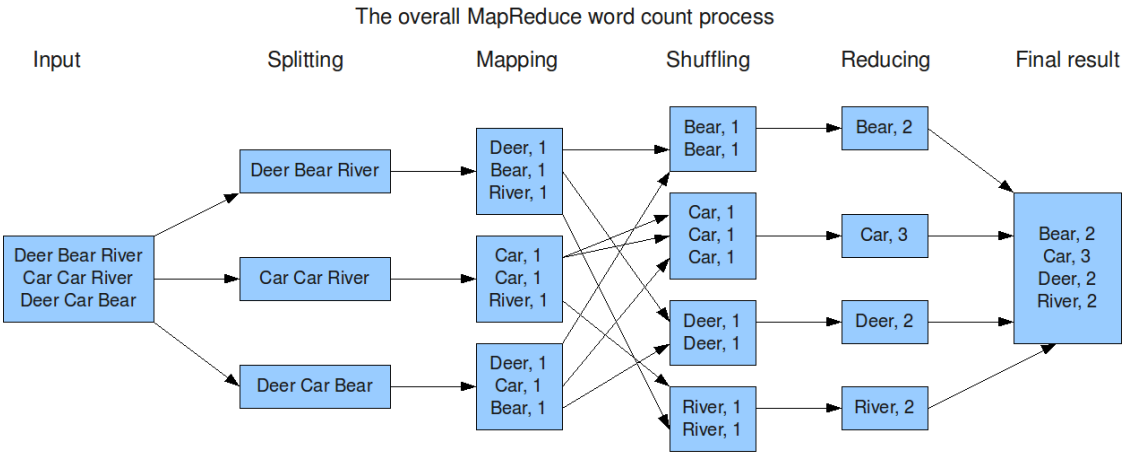
Map/Reduce is also primary component of Hadoop and it also have Master-slave architecture

Master: JobTracker

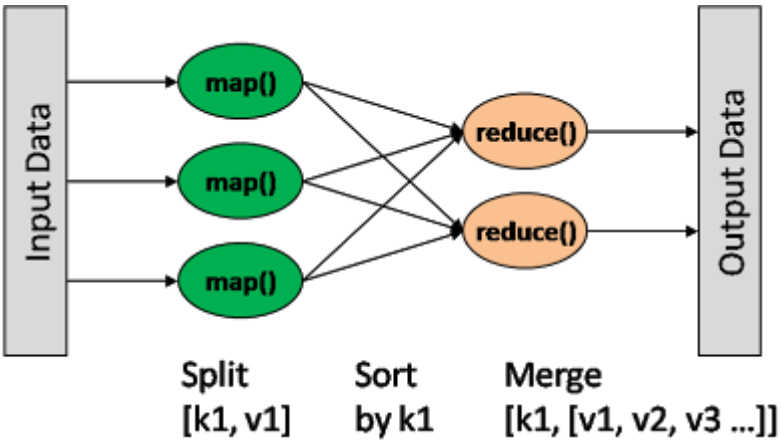
Slaves: {tasktraker}……{Tasktraker}

- Master {Jobtracker} is the point of interaction between users and the map/reduce framework. When a map/reduce job is submitted, Jobtracker puts it in a queue of pending jobs and executes them on a first-come/first-served basis and then manages the assignment of map and reduce tasks to the tasktrackers.

- Slaves {tasktracker} execute tasks upon instruction from the Master {Jobtracker} and also handle data motion between the map and reduce phases.



- First, we divide the input in three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mapper and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.



Creating our Hadoop program. Hadoop natively runs compiled Java programs, so we need to complete WordCounter.java in order to solve our problem.

This involves three steps:

(a) creating the mapper class that takes the input and produces <key, value> pairs;

(b) creating the reducer class that takes the <key, value> pairs produced by the mapper and combined them as appropriate to solve the problem; and

(c) creating the WordCouner's main() function to use our mapper and reducer. Taking these one at a time:

Creating the Mapper Class. Inside WordCounter.java, find the comment:

```
// replace this comment with the mapper class
```

and replace it with the following class:

```
/* Our 'mapper' class emits (word, "1") pairs,
 *  for later counting by our 'reducer' class.
 */
public static class TokenizerMapper     extends Mapper{
 public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
  StringTokenizer itr = new StringTokenizer( value.toString() );
  while ( itr.hasMoreTokens() ) {
    word.set( itr.nextToken() );
    context.write(word, one);
  }
 }
 private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();
}
```

As you can see, this class extends a Hadoop generic class named Mapper, and overrides its map() method. This map() method has three parameters: a key (which our method does not use, but could), a value (consisting of a line of text from a file), and a context (to which we will write our results), all of which are passed to it by Hadoop.

The body of the method tokenizes the value parameter into words, and then uses a loop to iterate through those words. For each word, the loop writes the pair (word, "1") to the context. When the loop completes, our mapper has done its work!

Creating the Reducer Class. Still inside WordCounter.java, find the comment:

```
// replace this comment with the reducer class
```

and replace it with the following class:

```
/* Our 'reducer' class receives a sequence of (word, "1") pairs
*  and for each word in the sequence, adds up its "1"s...
*/
public static class IntSumReducer   extends Reducer {
  public void reduce(Text key, Iterable values, Context context)
                    throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
        sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
  }
  private IntWritable result = new IntWritable();
}
```

This class extends the Hadoop Reducer class, overwriting its reduce() method. Like the map() method, the reduce() method takes three parameters: a key, which is one of the word values emitted by the mapper phase; a values, which is an iterable sequence of the values that were combined with that key; and a context to which the reducer will write its results.

The body of the method declares a sum variable to keep track of the number of occurences of the key, and then uses a loop to iterate through the values. Since the mapper produces "1" values, each val in values is "1", so the loop converts that "1" from a string to the int 1 and adds it to sum. When control leaves the loop, the method sets instance variable result to sum and writes the pair (key, result) to the context. That completes our reducer!

reference: https://cs.calvin.edu/courses/cs/374/exercises/12/lab/

pseudo code:

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

Example Scenario

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

If the above data is given as input, we have to write applications to process it and produce results such as finding consumption exceeded 30 .

This is a walkover for the programmers with finite number of records. They will simply write the logic to produce the required output, and pass the data to the application written.

But, think of the data representing the electrical consumption of all the largescale industries of a particular state, since its formation.

When we write applications to process such bulk data,

- They will take a lot of time to execute.
- There will be a heavy network traffic when we move data from source to network server and so on.

Input Data

The above data is saved as **sample.txt**and given as input. The input file looks as shown below.

```
1979  23  23  2  43  24  25  26  26  26  26  25  26 25
1980  26  27  28 28  28  30  31  31  31  30  30  30 29
1981  31  32  32 32  33  34  35  36  36  34  34  34 34
1984  39  38  39 39  39  41  42  43  40  39  38  38 40
1985  38  39  39 39  39  41  41  41  00  40  39  39 45
```

```
//Mapper class
  public static class E_EMapper extends MapReduceBase implements
  Mapper<LongWritable ,/*Input key Type */
  Text,                /*Input value Type*/
  Text,                /*Output key Type*/
  IntWritable>        /*Output value Type*/
  {

    //Map function
    public void map(LongWritable key, Text value,
    OutputCollector<Text, IntWritable> output,
    Reporter reporter) throws IOException
    {
      String line = value.toString();
      String lasttoken = null;
      StringTokenizer s = new StringTokenizer(line,"\t");
      String year = s.nextToken();

      while(s.hasMoreTokens())
        {
          lasttoken=s.nextToken();
        }

      int avgprice = Integer.parseInt(lasttoken);
```

```
      output.collect(new Text(year), new IntWritable(avgprice));
    }
  }



//Reducer class
public static class E_EReduce extends MapReduceBase implements
Reducer< Text, IntWritable, Text, IntWritable >
{

  //Reduce function
  public void reduce( Text key, Iterator <IntWritable> values,
    OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException
    {
      int maxavg=30;
      int val=Integer.MIN_VALUE;

      while (values.hasNext())
      {
        if((val=values.next().get())>maxavg)
        {
          output.collect(key, new IntWritable(val));
        }
      }

    }
}
```

What is Cloudera?

Cloudera is revolutionizing enterprise data management by offering the first unified Platform for Big Data: The Enterprise Data Hub. Cloudera offers enterprises one place to store, process, and analyze all their data, empowering them to extend the value of existing investments while enabling fundamental new ways to derive value from their data.

Founded in 2008, Cloudera was the first, and is currently, the leading provider and supporter of Apache Hadoop for the enterprise. Cloudera also offers software for business critical data challenges including storage, access, management, analysis, security, and search.

Cloudera was the first commercial provider of Hadoop-related software and services and has the most customers with enterprise requirements, and the most experience supporting them, in the industry. Cloudera's combined offering of differentiated software (open and closed source), support, training, professional services

What are Cloudera's products?

Cloudera's platform, which is designed to specifically address customer opportunities and challenges in Big Data, is available in the form of free/unsupported products (CDH or Cloudera Express, for those interested solely in a free Hadoop distribution), or as supported, enterprise-class software (Cloudera Enterprise - in Basic, Flex, and Data Hub editions) in the form of an annual subscription. All the integration work is done for you, and the entire solution is thoroughly tested for enterprise requirements and fully documented.