CS623, Research Project

Hadoop MapReduce

Olivia Arroyo, 4/2024

OVERVIEW

Structured

Traditional datasets that have rows, columns and clearly defined data attributes.

- DBMS tables
- Excel, CSV, SQL databases

Semi-structured

Does not conform to fixed fields, but does contain tags and markers.

- XML, HTML, JSON

Unstructured

Data that doesn't reside in a traditional row-column database.

- Images, Videos, PDF files, email attachments.

Big Data

Heterogeneous mix of structured , semi-structured and unstructured data.

(Dhavapriya, 2016), (Maitrey, 2015), (Sagiroglu,2013)

Hadoop MapReduce

Hadoop is the most popular open source implementation of the MapReduce framework.

The framework is designed to handle large datasets (multi-terabytes) and processes them in parallel across multiple computing nodes.

Supported by many programming languages: Java, C++, Python

A cloud computing technology.

(Dittrich, 2012), (Ghazi, 2015), (Maitrey, 2015)

SIMPLICITY

SCALABILITY

FAULT-TOLERANCE

HISTORY

Origins

MapReduce and the Google File System was created by Google in 2003 to address the need of processing big data.

They also built a DBMS known as Big Table.

Open-Sourcing

Apache Hadoop was created as an open source implementation of the MapReduce system.

(Maitrey, 2015), (Pothuganti, 2015), (Salo, 2012)

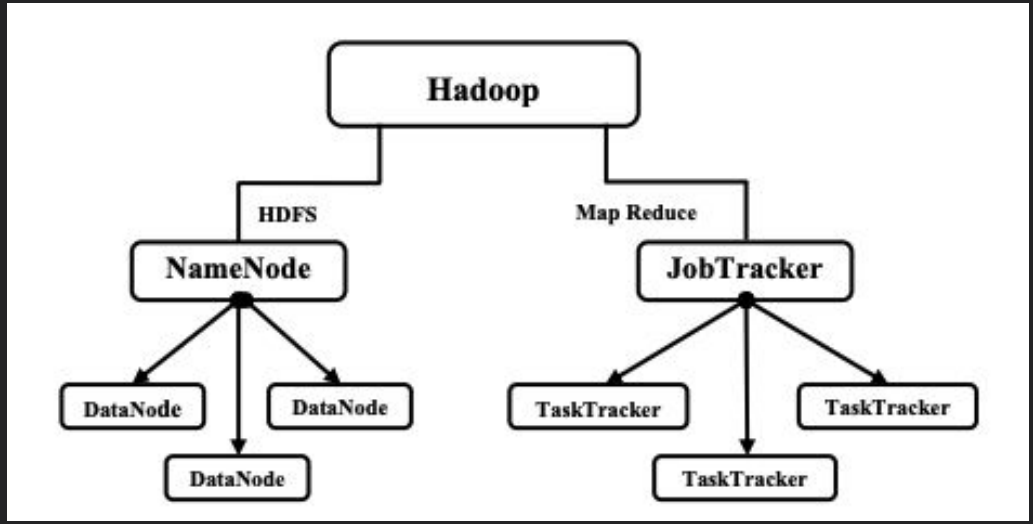
FEATURES

Hadoop Distributed File System (HDFS)

Data Storage

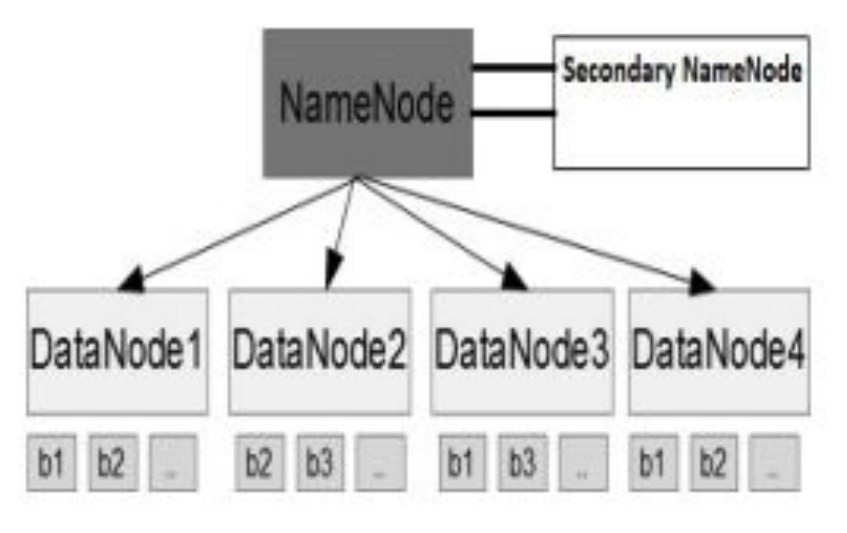
MapReduce Framework

Data Processing



(Ghazi, 2015), (Pothuganti, 2015)

HADOOP



NameNode

Runs on Manager node and dictates what the DataNode computers do with data.

It determines how files are split, where they are stored, and reviews the health of the file system.

DataNode

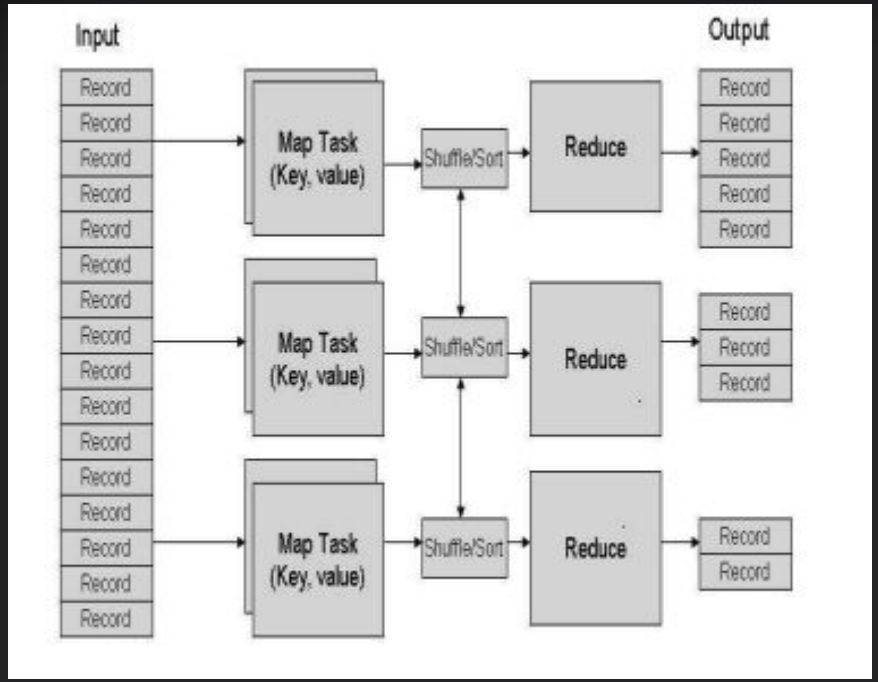
Controlled by the NameNode. They store data blocks and service read/write requests on files stored on the HDFS.

Secondary NameNode

Regularly reads the file system, logs the changes, and applies them onto a fsimage file.

(Dhavapriya, 2016), (Ghazi, 2015), Pothuganti,2015)

MapReduce



Map Phase

Splits input data into groups of key/value pairs using parallel processing.

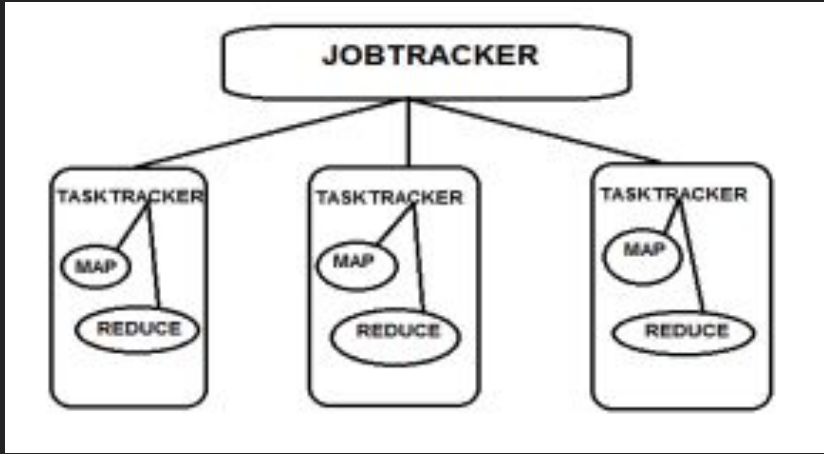
Groups the key/value pairs by their keys and feeds them to the reduce phase.

Reduce Phase

Aggregates and sorts the output from the Map phase to produce the final key/value pair outputs.

(Dhavapriya, 2016),(Dittrich, 2012), (Ghazi, 2015), Pothuganti,2015)

MapReduce



JobTracker

Runs on the manager node and manages job assignments, communicates with the NameNode, and monitors the health of the TaskTrackers that it oversees.

TaskTracker

Operates on the worker nodes and accepts and executes from distributed by the JobTracker. The JobTracker manages the task slots on teach TaskTracker to ensure there is availability to run each MapReduce operation.

(Dhavapriya, 2016), (Ghazi, 2015), Pothuganti,2015)

Sample Code

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
            extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        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);
            }
        }
    }

    public static class IntSumReducer
            extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context
        ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

Mapper Class

Reducer Class

(MapReduce Tutorial, 2024)

Rising Data Complexity

The demand to process vast amounts of data is increasing. These datasets hold valuable insights, driving the need for effective computational tools that can process large scale data.

Applications

The Hadoop MapReduce Framework allows users a solution for cloud-base large scale data processing and analytics.

Features

Hadoop MapReduce enables distributed processing of large datasets across clusters, offering simple, scalable and fault tolerant solutions for efficient data analysis and computation.

The Future

As big data continues to grow, the Hadoop MapReduce framework will need to continue to progress. New technologies, such as Apache Spark, have gained popularity due to their performance advantages.

CONCLUSION

REFERENCES

Dhavapriya, M., & Yasodha, N. (2016). Big data analytics: challenges and solutions using Hadoop, map reduce and big table. *International Journal of Computer Science Trends and Technology (IJCST)*, *4*(1), 5-14.

Dittrich, J., & Quiané-Ruiz, J. A. (2012). Efficient big data processing in Hadoop MapReduce. *Proceedings of the VLDB Endowment*, *5*(12), 2014-2015.

Ghazi, M. R., & Gangodkar, D. (2015). Hadoop, MapReduce and HDFS: a developers perspective. *Procedia Computer Science*, *48*, 45-50.

Maitrey, S., & Jha, C. K. (2015). MapReduce: simplified data analysis of big data. *Procedia Computer Science*, *57*, 563-571.

Mapreduce Tutorial. Apache Hadoop 3.4.0 – MapReduce Tutorial. (2024, March 4). https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Pothuganti, A. (2015). Big data analytics: Hadoop-Map reduce & NoSQL databases. *International Journal of Computer Science and Information Technologies*, *6*(1), 522-527.

Sagiroglu, S., & Sinanc, D. (2013, May). Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)* (pp. 42-47). IEEE.

Salo, J. (2012). Pattern definition of MapReduce. In *Proceedings of VikingPLoP 2012 Conference* (p. 97).