SOURCE CODE

VIKRAM GAVISIDDESHWARA MOUNA GIRI PRANAV SAI DEENUMSETTI

[Year]

1. SHARED MEMORY
2. Sort\_Java.java

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.EOFException;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.RandomAccessFile;

import java.util.Comparator;

import java.util.List;

import java.util.PriorityQueue;

import java.util.Scanner;

public class Sort\_Java {

static int commandLine;

static String file\_name;

static double value;

public static void main(String args[]) throws InterruptedException, IOException {

String out = ("output\_sorted");

Scanner console = new Scanner (System.in);

System.out.print("Please enter input file name to be sorted");

file\_name = console.next();

RandomAccessFile randomFile = new RandomAccessFile(file\_name, "rw");

System.out.print("Please enter the number of threads [1, 2, 4 or 8] ");

commandLine = console.nextInt();

File\_ReadNSort[] run = new File\_ReadNSort[commandLine];

value = (randomFile.length()/commandLine);

double start = System.nanoTime();

for(int i=0;i<commandLine;i++)

{

long from=(long) (i\*value);

long to=(long) (from+value);

int c=0;

run[i] = new File\_ReadNSort(from, to, file\_name,c);

run[i].run();

}

for (int v = 0; v < commandLine; v++)

run[v].join();

mergeFiles(File\_ReadNSort.files\_finallist,out,File\_ReadNSort.string\_Comparator);

double Total = (System.nanoTime()-start)/1000000000.0;

System.out.println("Time Taken is "+(Total)+" sec");

double result\_throughput=(randomFile.length()/1073741824)/Total;

System.out.println("Throughput is "+ result\_throughput + " GB/sec");

}

public static void mergeFiles(List<File> file\_List, String outputfile, final Comparator<String> comp) throws IOException {

Buffer binaryBuffer ;

PriorityQueue<Buffer> buffer\_pq = new PriorityQueue<Buffer>(11,

new Comparator<Buffer>() {

public int compare(Buffer i, Buffer j) {

return comp.compare(i.peek(), j.peek());

}

}

);

for (File file\_name : file\_List) {

buffer\_pq.add(new Buffer(file\_name));

}

File output\_file=new File(outputfile);

BufferedWriter buffWriter = new BufferedWriter(new FileWriter(output\_file));

int flag = 0;

try {

while(buffer\_pq.size()>0) {

Buffer buff = buffer\_pq.poll();

String result = buff.pop();

buffWriter.write(result);

buffWriter.newLine();

++flag;

if(buff.isEmpty()) {

buff.buffReader.close();

buff.initialFile.delete();

} else {

buffer\_pq.add(buff);

}

}

} finally {

buffWriter.close();

for(Buffer bfb : buffer\_pq )

bfb.close();

}

}

}

class Buffer  {

public static int buff\_size = 2048;

public BufferedReader buffReader;

public File initialFile;

private String temp;

private boolean isEmpty;

public Buffer(File f) throws IOException {

this.initialFile = f;

this.buffReader = new BufferedReader(new FileReader(f), buff\_size);

reload();

}

public boolean isEmpty() {

return isEmpty;

}

private void reload() throws IOException {

try {

if((this.temp = buffReader.readLine()) == null){

isEmpty = true;

temp = null;

}

else{

isEmpty = false;

}

} catch(EOFException e) {

isEmpty = true;

temp = null;

}

}

public void close() throws IOException {

buffReader.close();

}

public String peek() {

if(isEmpty()) return null;

return temp.toString();

}

public String pop() throws IOException {

String value = peek();

reload();

return value;

}

}

1. File\_ReadNSort.java

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.RandomAccessFile;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

public class File\_ReadNSort extends Thread {

public static List<File> files\_finallist = new ArrayList<File>();

private long from;

private long to;

private String fileName;

int thread\_count = Sort\_Java.commandLine;

int block\_size = 1024/thread\_count;

int c ;

public static Comparator<String> string\_Comparator = new Comparator<String>() {

public int compare(String string1, String string2){

return string1.compareTo(string2);}};

public File\_ReadNSort(long from, long to, String fileName,int c) {

super();

this.from = from;

this.to = to;

this.fileName = fileName;

this.c=c;

}

@Override

public void run()

{

RandomAccessFile randomFile = null;

FileReader fname;

BufferedReader buffer\_reader = null;

String readLine = "";

long max\_divide\_size = (long) (Sort\_Java.value / block\_size) ;

long freeRuntimeMemory = Runtime.getRuntime().freeMemory();

if( max\_divide\_size < freeRuntimeMemory/2)

max\_divide\_size = freeRuntimeMemory/2;

try {

Sorted\_LinkedList<String> reader = new Sorted\_LinkedList<String>((int)max\_divide\_size);

randomFile = new RandomAccessFile(fileName, "rw");

fname = new FileReader(randomFile.getFD());

buffer\_reader = new BufferedReader(fname);

randomFile.seek(this.from);

while(readLine!= null )

{

long flag=0;

if( c >= Sort\_Java.value)

break;

while((flag <= max\_divide\_size) && (readLine = buffer\_reader.readLine()) != null )

{

if( flag >= Sort\_Java.value)

break;

flag+= readLine.length()+2;

c+= readLine.length()+2;

reader.add(readLine+" ");

}

files\_finallist.add(write\_To\_File(reader));

reader.clear();

}

}

catch (IOException e) {

e.printStackTrace();

}

finally

{

try {

buffer\_reader.close();

randomFile.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

public static File write\_To\_File(Sorted\_LinkedList<String> sorted\_list)

{

File temp\_File = null;

BufferedWriter file\_writer = null;

try {

temp\_File = File.createTempFile("Temp", ".tmp");

file\_writer = new BufferedWriter(new FileWriter(temp\_File));

for(String i : sorted\_list) {

file\_writer.write(i);

file\_writer.newLine();

}

temp\_File.deleteOnExit();

} catch (IOException e) {

e.printStackTrace();

} finally {

try{

file\_writer.close();

}catch(Exception ex){}

}

return temp\_File;

}

}

1. Sorted\_LinkedList.java

import java.util.Comparator;

import java.util.Iterator;

public class Sorted\_LinkedList<T> implements ListInterface<T>{

int num\_elements=0;

T elements[];

long size = 1000000000;

protected Comparator<T> comp\_obj;

protected boolean found;

protected int loc\_value;

int current\_pointer ;

public Comparator<String> string\_Comp = new Comparator<String>() {

public int compare(String string1, String string2){

return string1.compareTo(string2);}};

public Sorted\_LinkedList(int size)

{

elements = (T[]) new Object[size];

comp\_obj = new Comparator<T>()

{

public int compare(T element1, T element2)

{

return ((Comparable<T>)element1).compareTo(element2);

}

};

}

public void add(int index, T element)

{

throw new UnsupportedOperationException("Unsupported index-based add method");

}

public T set(int index, T newElement)

{

throw new UnsupportedOperationException("Unsupported index-based set method");

}

public boolean is\_full() {

if (num\_elements == size ){

return true;

}

return false;

}

public boolean is\_empty() {

if (num\_elements == 0){

return true;

}

return false;

}

public int size() {

return num\_elements;

}

protected void Find\_recursive(T target, int first, int last)

{

int result;

if (first > last)

{

found = false;

result = comp\_obj.compare(target,elements[loc\_value]);

if (result > 0)

loc\_value++;

}

else

{

loc\_value = (first + last) / 2;

result = comp\_obj.compare(target,elements[loc\_value]);

if (result == 0)

{

found = true;

}

else

if (result > 0)

Find\_recursive(target, loc\_value + 1, last);

else

Find\_recursive(target, first, loc\_value - 1);

}

}

public boolean add(T element)

{

loc\_value = 0;

found = false;

if (!is\_empty())

Find\_recursive(element, 0, num\_elements - 1);

for (int m = num\_elements; m > loc\_value; m--)

elements[m] =elements[m - 1];

elements[loc\_value] = element;

num\_elements++;

return true;

}

public boolean contains(T e) {

if (num\_elements > 0) {

for (int i = 0 ; i < num\_elements ; i++ ) {

if(elements[i].equals(e))

return true;

}

}

return false;

}

@Override

public Iterator<T> iterator() {

return new Iterator<T>()

{

private int previousPos = -1;

public boolean hasNext() {

return (previousPos < (size() - 1)) ;

}

public T next()

{

if (!hasNext())

throw new IndexOutOfBoundsException("Illegal invocation of next " +" in LBList iterator.\n");

previousPos++;

return elements[previousPos];

}

public void remove()

{

for (int i = previousPos; i <= num\_elements - 2; i++)

elements [i] = elements[i+1];

elements [num\_elements - 1] = null;

num\_elements--;

previousPos--;

} };

}

@Override

public T get(int index) {

T element\_value= elements[index];

return element\_value;

}

@Override

public int indexOf(T target) {

if (num\_elements > 0) {

for (int i = 0 ; i < num\_elements ; i++ ) {

if(elements[i] == target)

return i;

}

}

return -1;

}

@Override

public T remove(int index) {

T elem=null;

T current\_pointer = elements[index] ;

for (int i = index + 1 ; i < num\_elements ; i++) {

elements[index] = elements[i] ;

index++ ;

}

num\_elements-- ;

return elem;

}

public void clear() {

for (int i = 0 ; i < num\_elements ; i++)

elements[i]=null;

num\_elements=0;

}

}

1. ListInterface.java

public interface ListInterface<T> extends Iterable<T>

{

void add(int index, T element);

T set(int index, T newElement);

T get(int index);

int indexOf(T target);

T remove(int index);

}

2. HADOOP

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

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 HadoopSortSingleNode{

public static class CustomMapper extends Mapper<Object, Text, Text, Text>{

public void map (Object key, Text value, Context context)throws IOException, InterruptedException{

String Key = (value.toString()).substring(1,10);

String nextKey = (value.toString()).substring(11);

context.write(new Text(Key),new Text(nextKey));

}

}

public static class CustomReducer extends Reducer<Text, Text, Text, Text>{

public void reduce (Text key, Iterable<Text> values, Context context)throws IOException, InterruptedException{

for (Text text : values) {

context.write(new Text(key), text);

}

}

public static void main(String[] args) throws Exception{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "sort");

job.setJarByClass(HadoopSortSingleNode.class);

job.setMapperClass(CustomMapper.class);

job.setCombinerClass(CustomReducer.class);

job.setReducerClass(CustomReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

CONFIGS :

Namenode config for multicluster setup

core-site.xml

<property>

<name>fs.defaultFS</name>

<value>hdfs://172.31.34.244:8020</value>

</property>

hdfs-site.xml

<property>

<name>dfs.namenode.name.dir</name>

<value>file:/root/hadoop/file/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:/root/hadoop/file/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

mapred-site.xml

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

yarn-site.xml

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>172.31.34.244:8025</value>

/property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>172.31.34.244:8030</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>172.31.34.244:8050</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.nodemanager.disk-health-checker.min-healthy-disks</name>

<value>0</value>

</property>

++++++++++++++++++++++++++++++++++++++++++++++++++++++++

hadoop multinode nodein datanodes

core-site.xml

<property>

<name>fs.defaultFS</name>

<value>hdfs://172.31.34.244:8020</value>

</property>

mapred-site.xml

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

hdfs-site.xml

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///root/hadoop/file/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///root/hadoop/file/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

yarn-site.xml

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>172.31.34.244:8025</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>172.31.34.244:8030</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>172.31.34.244:8050</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.nodemanager.disk-health-checker.min-healthy-disks</name>

<value>0</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>

14000

</value>

</property>

<property>

<name>yarn.nodemanager.resource.cpu-vcores</name>

<value>1</value>

</property>

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

hadoop 128 single node config files

yarn-site.xml

<property>

<name>yarn.resourcemanager.hostname</name>

<value>localhost</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>

14000

</value>

</property>

<property>

<name>yarn.nodemanager.resource.cpu-vcores</name>

<value>1</value>

</property>

hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

mapred-site.xml

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

core-site.xml

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost/</value>

</property>

IP address of nodes used in hadoop cluster:

Namenode IP :172.31.34.244

datanode1: 172.31.45.197

datanode2:172.31.38.150

datanode3 :172.31.43.126

datanode4: 172.31.46.96

datanode5: 172.31.46.154

datanode6: 172.31.41.109

172.31.37.128 datanode7

172.31.41.109 datanode6

172.31.46.154 datanode5

172.31.46.96 datanode4

172.31.43.126 datanode3

172.31.38.150 datanode2

172.31.45.197 datanode1

* + - 1. menode

1. SPARK

"""sparkTeraSort.py"""

from pyspark import SparkContext

dataFile = "hdfs://localhost:9000/sparkproject/input"

sc = SparkContext("local", "128sort")

logData = sc.textFile(dataFile).cache()

pairs = logData.map(lambda x: (x.split(" ")[0], x))

sorted = pairs.sortByKey(ascending=True, numPartitions=None, keyfunc = lambda x: str(x))

result = sorted.values()

result.saveAsTextFile("hdfs://localhost:9000/sparkproject/output")