Manual

To run the code

For Client

Java –jar PA2\_Client.jar workertype no of threads

For remote

First invoke client using Java –jar PA2\_Client.jar workertype no of threads

Then invoke worker using

To invoke worker parallel in multiple nodes

Start the number of nodes no of worker needed

Run PA\_worker.jar command:

Java –jar PA2\_Worker.jar

To run worker parallel on multiple nodes

pssh  -h hosts.txt  -x "-oStrictHostKeyChecking=no  -i pemfile" "java -jar /home/ec2-user/PA\_Worker.jar noofthreads"

Source Code

Client

import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Client {

public static void main(String args[])

{

LocalWorkerQueue RequestQueue;

SimpleQueueService s;

//take the file and workertype and nothreads

String filename=args[0];

String WorkerType=args[1];

int noThreads=Integer.parseInt(args[2]);

int noTasks=0;

BufferedReader br;

try {

br = new BufferedReader(new FileReader(filename));

String data,animototask ="";

data=br.readLine();

List<String> tasks=new ArrayList<String>(); // list of tasks

int size=0;

System.out.println("enter no of workers");

Scanner scan=new Scanner(System.in);

int noWorkers=scan.nextInt(); // no of workers to run

while(data!=null &&!(data.isEmpty()))

{

tasks.add(data);

data=br.readLine();

}

for(int i=0;i<tasks.size();i++)

{

if(!(WorkerType.equals("animoto")))

{

int time=Integer.parseInt(tasks.get(i).split(" ")[1]);

//based on sleep job set no of tasks

if(time==0)

{

noTasks=10000;

}

if(time==10)

{

noTasks=1000;

}if(time==1000)

{

noTasks=100;

}if(time==10000)

{

noTasks=10;

}

size=size+(noTasks\*noWorkers);

}

else

{

noTasks=160;

}

// if workertype is local create local queue insert tasks into local queue

//process the data

//create object of local worker and process each task in local queue

if(WorkerType.equals("local"))

{

long beginTime=System.currentTimeMillis();

RequestQueue=new LocalWorkerQueue();

for(int j=0;j<noTasks;j++)

{

RequestQueue.insertData(tasks.get(i));

}

size=RequestQueue.getsize();

LocalWorker l=new LocalWorker(RequestQueue,noThreads);

l.process();

while(l.ResponseQueue.getsize()!=size && l.flag!=false)

{

}

// System.out.println(size+"sie"+l.ResponseQueue.getsize());

if(l.ResponseQueue.getsize()==size)

{

System.out.println("process done");

long endTime=System.currentTimeMillis();

System.out.println("time taken"+(endTime-beginTime));

}

}

// if workertype is remote create sqs queue insert tasks into remote queue

//process the data

//Remote worker will process each task in sqs queue from different nodes

if(WorkerType.equals("remote"))

{

long beginTime=System.currentTimeMillis();

s=new SimpleQueueService("RequestQueue");

for(int j=0;j<noTasks\*noWorkers;j++)

{

s.insert\_sqs(tasks.get(i));

}

size=s.sizeofsqs();

s=new SimpleQueueService("ResponseQueue");

System.out.println(s+" "+size);

/\*

while(s.sizeofsqs()!=size)

{

}

if(s.sizeofsqs()==size)

{

System.out.println("process done");

long endTime=System.currentTimeMillis();

System.out.println("time taken"+(endTime-beginTime));

}

\*/

}

// if workertype is animoto create sqs queue insert url tasks into SQS queue

if(WorkerType.equals("animoto"))

{

animototask=animototask+"@"+tasks.get(i); // append all 60 images as one task

}

}

//add animoto url combined as one task to sqs

if(WorkerType.equals("animoto"))

{

animototask=animototask+"@";

s=new SimpleQueueService("RequestQueue");

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

{

s.insert\_sqs(animototask);

}

size=s.sizeofsqs();

s=new SimpleQueueService("ResponseQueue");

/\*while(s.sizeofsqs()!=size)

{

}

if(s.sizeofsqs()==size)

{

System.out.println("process done");

s.deleteQueue();

}\*/

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class LocalWorker {

private LocalWorkerQueue RequestQueue;

int totalThreads;

LocalWorkerQueue ResponseQueue;

Worker worker;

public LocalWorker(LocalWorkerQueue RequestQueue,int t)

{

this.RequestQueue=RequestQueue;

this.totalThreads=t;

}

public void process()

{

//create a pool of threads using executor service

ExecutorService executor = Executors.newFixedThreadPool(totalThreads);

int size=RequestQueue.getsize();

//in a loop process the tasks in local queue by each thread

ResponseQueue=new LocalWorkerQueue();

for(int j=1;j<=size;j++)

{

String s=RequestQueue.deleteQueue(j); // get the data from local queue into string

int time = Integer.parseInt(s.split(" ")[1]); // split the data to get the sleep time

worker=new Worker(time,j,ResponseQueue);

executor.execute(worker); //each thread process the tasks

ResponseQueue.insertData("task"+j+"completed");

}

executor.shutdown();

while (!executor.isTerminated()) {

}

//System.out.println("j is"+ResponseQueue.getsize());

}

}

class Worker implements Runnable

{

int key;

int time;

public Worker(int time,int key,LocalWorkerQueue ResponseQueue)

{

this.time=time;

this.key=key;

}

//perform the sleep task in thread

@Override

public void run() {

try {

Thread.sleep(time);

System.out.println("sleeping" + Thread.currentThread()+" ");

}

catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

LocalWorker

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class LocalWorker {

private LocalWorkerQueue RequestQueue;

int totalThreads;

LocalWorkerQueue ResponseQueue;

Worker worker;

boolean flag=true;

public LocalWorker(LocalWorkerQueue RequestQueue,int t)

{

this.RequestQueue=RequestQueue;

this.totalThreads=t;

}

public void process()

{

//create a pool of threads using executor service

ExecutorService executor = Executors.newFixedThreadPool(totalThreads);

int size=RequestQueue.getsize();

//in a loop process the tasks in local queue by each thread

ResponseQueue=new LocalWorkerQueue();

for(int j=1;j<=size;j++)

{

String s=RequestQueue.deleteQueue(j); // get the data from local queue into string

int time = Integer.parseInt(s.split(" ")[1]); // split the data to get the sleep time

worker=new Worker(time,j,ResponseQueue);

executor.execute(worker); //each thread process the tasks

ResponseQueue.insertData("task"+j+"completed");

}

executor.shutdown();

while (!executor.isTerminated()) {

}

flag=false;

//System.out.println("j is"+ResponseQueue.getsize());

}

}

class Worker implements Runnable

{

int key;

int time;

public Worker(int time,int key,LocalWorkerQueue ResponseQueue)

{

this.time=time;

this.key=key;

}

//perform the sleep task in thread

@Override

public void run() {

try {

Thread.sleep(time);

System.out.println("sleeping" + Thread.currentThread()+" ");

}

catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

LocalWorkerQueue

import java.util.HashMap;

public class LocalWorkerQueue {

HashMap<Integer,String> Tasks=new HashMap<Integer,String>();

int key=1;

public LocalWorkerQueue()

{

key=1;

}

// inserts the records into hashmap

public void insertData(String job)

{

Tasks.put(key, job);

key=key+1;

}

//get the size of hashmap

public int getsize()

{

return Tasks.size();

}

// deletes the records from hashmap

public String deleteQueue(int key)

{

String s=Tasks.get(key);

Tasks.remove(key);

return s;

}

}

RemoteWorker

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class RemoteWorker {

public static void main(String args[]) {

try {

boolean flag=true;

SimpleQueueService tasks=new SimpleQueueService("RequestQueue");

int totaTask=tasks.sizeofsqs();

//Scanner reader = new Scanner(System.in);

int totalThreads=Integer.parseInt(args[0]);

//create a pool of threads using executor service

ExecutorService executor = Executors.newFixedThreadPool(totalThreads);

//process the tasks by remote worker by each thread till there is a task in sqs

while(flag==true)

{

TaskInfo task=tasks.processRequest(); // get the messages for each task

Runnable worker = new RemoteWorkerThread(task);

totaTask--;

executor.execute(worker); // execute the worker thread

if(totaTask<=0)

flag=false;// stop the process when all tasks are processed by thread

}

executor.shutdown();

while (!executor.isTerminated()) {

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

RemoteWorkerThread.java

**public** **class** RemoteWorkerThread **implements** Runnable {

SimpleQueueService results;

AmazonDynamoDB dp=**new** AmazonDynamoDB();

**int** dupVal;

TaskInfo task;

RemoteWorkerThread(TaskInfo task)

{

results=**new** SimpleQueueService("ResponseQueue");

**this**.task=task;

}

@Override

**public** **void** run() {

**try** {

dp.init(); // initliaze the amazon db table

dupVal=dp.putddb(task.jobid); // check in dyanamo db if there are any duplicate

//System.out.println("dupval is"+dupVal);

**if**(dupVal==1) // if task is not duplicate process the task

{

**int** time = Integer.*parseInt*(task.job.split(" ")[1]);

Thread.*sleep*(time);

results.insert\_sqs(task.jobid+"is Finished"); //add the output to result queue after task is done

}

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Animoto .java

import java.io.File;

import java.io.FileInputStream;

import java.io.InputStream;

import java.net.URL;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import com.amazonaws.HttpMethod;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3Client;

import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

import com.amazonaws.services.s3.model.ObjectMetadata;

import com.amazonaws.services.s3.model.PutObjectRequest;

public class animoto {

public static void main(String args[])

{

int totalThreads=Integer.parseInt(args[0]);

SimpleQueueService tasks=new SimpleQueueService("RequestQueue");

int totalTask = tasks.sizeofsqs();

boolean flag=true;

//process the tasks by remote worker by each thread till there is a task in sqs

ExecutorService executor = Executors.newFixedThreadPool(totalThreads);

while(flag==true)

{

TaskInfo task=tasks.processRequest();

Runnable worker = new RemoteAnimotoThread(task); // execute the worker thread

totalTask--;

executor.execute(worker);

if(totalTask<=0)

flag=false; // stop the process when all tasks are processed by thread

}

executor.shutdown();

while (!executor.isTerminated()) {

}

}

}

class RemoteAnimotoThread implements Runnable {

static int totalThreads=0;

SimpleQueueService results=new SimpleQueueService("ResponseQueue");

AmazonDynamoDB dp=new AmazonDynamoDB();

int dupVal;

static BasicAWSCredentials credentials = new BasicAWSCredentials("AKIAIKJ7XKFQAKDOLAZA", "liQQanhZzwZu5NZZjATHdm4SYfotiR+118Ulgcdt");

private static String bucket;

static AmazonS3 s3client = new AmazonS3Client(credentials);

int img=0;

TaskInfo task;

RemoteAnimotoThread(TaskInfo task)

{this.task=task;

}

@Override

public void run() {

try {

//check if there are duplicates in dynamo db

dupVal=dp.putddb(task.jobid);

if(dupVal==1)

{

//split the task and assign the task to string url

String url[]=task.job.split("@");

System.out.println("url is"+url[1]);

String commands;

//create the url of each string in tasks

for(int i=1;i<url.length;i++)

{

URL urlno = new URL(url[i]);

img++;

//download the image into local folder

String commandString="wget "+url[i]+" -O "+img+".jpg";

Process process = Runtime.getRuntime().exec(commandString);

}

//after images are downloaded to local folder merge them into video using ffmpeg coomand

commands = "ffmpeg -i "

+ "" + "img%2d.jpg" + " " + "video.mpeg";

System.out.println(commands);

Process p =Runtime.getRuntime().exec(commands);

//create s3 bucket

bucket="ass3";

s3client.createBucket(bucket);

File f=new File("video.mpeg");

//for each video set content length and meradata of video

ObjectMetadata omd = new ObjectMetadata();

omd.setContentLength(f.length());

InputStream in = new FileInputStream(f);

//put the video into s3 bucket using putobject command

s3client.putObject(new PutObjectRequest(bucket, "1", in,omd));

//generate url for s3 object in the bucket

GeneratePresignedUrlRequest generatePresignedUrlRequest =

new GeneratePresignedUrlRequest(bucket, "1");

generatePresignedUrlRequest.setMethod(HttpMethod.GET);

//add the s3 bucket url to list of urls by generating url using command generatePresignedUrl

URL urls = s3client.generatePresignedUrl(generatePresignedUrlRequest);

System.out.println("Pre-Signed URL = " + urls.toString());

results.insert\_sqs(urls.toString());

}

} catch (Exception e) {

e.printStackTrace();

}

}}

SimpleQueueService.java

import java.util.List;

import java.util.Map;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Region;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.sqs.AmazonSQS;

import com.amazonaws.services.sqs.AmazonSQSClient;

import com.amazonaws.services.sqs.model.CreateQueueRequest;

import com.amazonaws.services.sqs.model.DeleteMessageRequest;

import com.amazonaws.services.sqs.model.DeleteQueueRequest;

import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;

import com.amazonaws.services.sqs.model.Message;

import com.amazonaws.services.sqs.model.ReceiveMessageRequest;

import com.amazonaws.services.sqs.model.SendMessageRequest;

public class SimpleQueueService {

String myQueueUrl;

AmazonSQS sqs;

BasicAWSCredentials credentials = new BasicAWSCredentials("AKIAIKJ7XKFQAKDOLAZA", "liQQanhZzwZu5NZZjATHdm4SYfotiR+118Ulgcdt");

CreateQueueRequest createQueueRequest;

//in constructor create queue for sqs using cerdentials

SimpleQueueService(String QueueName)

{

sqs = new AmazonSQSClient(credentials);

Region usWest2 = Region.getRegion(Regions.US\_WEST\_2);

sqs.setRegion(usWest2);

createQueueRequest = new CreateQueueRequest(QueueName);

myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();

}

//insert message in sqs queue passed as parameter tasks

public void insert\_sqs(String tasks)

{

sqs.sendMessage(new SendMessageRequest(myQueueUrl, tasks));

}

//process the queue and get messages from the queue return job and jobid stored in class variable TaskInfo

public TaskInfo processRequest()

{

ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest("RequestQueue");

List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();

String job = null;

String jobid = null;

TaskInfo t=new TaskInfo();

for(int i=0;i<messages.size();i++)

{

Message m=messages.get(i);

job=m.getBody();

jobid=m.getMessageId();

t.jobid=jobid;

t.job=job;

if(messages.size()>0)

{

CreateQueueRequest createQueueRequest = new CreateQueueRequest("RequestQueue");

String messageRecieptHandle = messages.get(0).getReceiptHandle();

sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageRecieptHandle));

}

}

return t;

}

//gives the size of approximate messages in sqs queue

public int sizeofsqs()

{

GetQueueAttributesRequest request = new GetQueueAttributesRequest();

request = request.withAttributeNames("ApproximateNumberOfMessages");

request = request.withQueueUrl(myQueueUrl);

Map<String, String> attrs = sqs.getQueueAttributes(request).getAttributes();

// get the approximate number of messages in the queue

int sizeOfMessages = Integer.parseInt(attrs.get("ApproximateNumberOfMessages"));

return sizeOfMessages;

}

//delete a queue

public void deleteQueue()

{

sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));

}

}

class TaskInfo

{

String jobid;

String job;

}

AmazonDynamoDb

import java.util.HashMap;

import java.util.Map;

import com.amazonaws.auth.BasicAWSCredentials;

import com.amazonaws.regions.Region;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;

import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;

import com.amazonaws.services.dynamodbv2.model.AttributeValue;

import com.amazonaws.services.dynamodbv2.model.ConditionalCheckFailedException;

import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;

import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;

import com.amazonaws.services.dynamodbv2.model.KeyType;

import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

import com.amazonaws.services.dynamodbv2.model.PutItemRequest;

import com.amazonaws.services.dynamodbv2.model.PutItemResult;

import com.amazonaws.services.dynamodbv2.model.ResourceInUseException;

import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;

import com.amazonaws.services.dynamodbv2.model.TableDescription;

import com.amazonaws.services.dynamodbv2.util.Tables;

public class AmazonDynamoDB {

static AmazonDynamoDBClient dynamoDB;

String tableName = "checkDuplicate";

public void init() throws Exception {

BasicAWSCredentials credentials = new BasicAWSCredentials("AKIAIKJ7XKFQAKDOLAZA", "liQQanhZzwZu5NZZjATHdm4SYfotiR+118Ulgcdt");

dynamoDB = new AmazonDynamoDBClient(credentials);

Region usWest2 = Region.getRegion(Regions.US\_WEST\_2);

dynamoDB.setRegion(usWest2);

try

{

if (Tables.doesTableExist(dynamoDB, tableName)) {

// System.out.println("Table: " + tableName + " is already ACTIVE");

} else {

// Create a table with a primary hash key named 'name', which holds a string

CreateTableRequest createTableRequest = new CreateTableRequest().withTableName(tableName)

.withKeySchema(new KeySchemaElement().withAttributeName("name").withKeyType(KeyType.HASH))

.withAttributeDefinitions(new AttributeDefinition().withAttributeName("name").withAttributeType(ScalarAttributeType.S))

.withProvisionedThroughput(new ProvisionedThroughput().withReadCapacityUnits(2L).withWriteCapacityUnits(2L));

TableDescription createdTableDescription = dynamoDB.createTable(createTableRequest).getTableDescription();

// Wait for it to become active

System.out.println("Waiting for " + tableName + " to become ACTIVE...");

Tables.waitForTableToBecomeActive(dynamoDB, tableName);

}

}

catch(ResourceInUseException e)

{

Tables.waitForTableToBecomeActive(dynamoDB, tableName);

}

}

public int putddb(String taskid) throws Exception {

try {

// Create table if it does not exist yet

Map<String, AttributeValue> item = newItem(taskid);

//tries to put the taskid in the table if the value is unique it will be inserted else it will throw constraint exception and return 0

PutItemRequest putItemRequest = new PutItemRequest().withTableName(tableName).withItem(item);

try{ PutItemResult putItemResult = dynamoDB.putItem(putItemRequest);

}catch( ConditionalCheckFailedException e)

{

return 0;

}

}

catch(Exception e)

{

}

return 1;

}

private static Map<String, AttributeValue> newItem(String name) {

Map<String, AttributeValue> item = new HashMap<String, AttributeValue>();

item.put("name", new AttributeValue(name));

return item;

}

}