分布式实验报告

Java文件IO操作及HDFS编程

作 者 姓 名： 张亦驰

学 号： 1751593

指 导 教 师： 饶卫雄

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 2](#_Toc26742728)

[1.1 题目分析 2](#_Toc26742729)

[1.2 解题思路 3](#_Toc26742730)

[2. 代码实现 4](#_Toc26742731)

[2.4 CAWriter.java 14](#_Toc26742732)

[3. 结果分析 18](#_Toc26742733)

[3.1 MTWriter 18](#_Toc26742734)

[3.2 HDWriter 19](#_Toc26742735)

[3.3 CAWriter 19](#_Toc26742736)

# 1 分析

## 1.1 题目分析

此题着重考察Java多线程文件IO操作及HDFS、Cassandra基本CRUD操作，难点在于多个写线程情况下如何保证数据不被覆盖以及Hadoop、Cassandra在Linux系统上的配置。此外，应从结果分析得到不同写策略对性能带来的影响。

## 1.2 解题思路

此题重点在于第一小问，第二、三问主要在配置环境上较为耗时，环境配置成功以后，只需要使用相应的API即可实现，故以下着重介绍我第一题的解题思路。

#### 1.2.1 采用阻塞队列的生产者消费者模式

生产者Producer作为数据源，写线程作为各个消费者使用数据，阻塞队列作为缓冲区，实现生产者与消费者之间的同步。主线程启动一个生产者线程与N个消费者线程，N个消费者异步工作。

当缓冲区满，生产者写入操作阻塞；当缓冲区为空，消费者取出操作阻塞。采用concurrency包的阻塞队列，避免使用synchronized同步块重量级锁而导致的阻塞所有试图访问同步块线程的状况，提高效率。

此种方法每次只在缓冲区已满的情况下占用内存最大，最大为缓冲区大小，占用内存资源少。

#### 1.2.2 直接划分各写线程的任务

生产者Producer只作为数据源，当所有数据都被缓存到内存中之后，主线程划分任务块，将所有的任务分配给各个写线程。

此种方法免去了生产者消费者线程同步产生的时间开销，写入效率极高，但消耗内存资源太大，约为1G。

#### 1.2.3 写入策略分析

**一、字符串转byte数组**

起初考虑到直观感受，我将每个int值转为string，为了使用RandomAccessFile多线程写入，又将string转为byte数组输出。

由于用string表示的数值，每个数所需要的字节数与其自身位数有关，比如12345共5位，需要5个字节存储，而字节长度的不固定又导致了raf.seek()字节数的不确定，造成了很大的麻烦。我采用的解决方法是将每个数256份平均分给N个线程，只有当所有消费者线程都已写入256/N个数后，生产者线程才被允许进行下一个数的生产，以便计算偏移量。

但这种方法的弊端在于，每次写入的字节数实在是太少，最多也就7个字节（1031168），所需要的写入时间几乎与生产者生产一个数的时间相差无几，且消费者线程数目远大于生产者线程数，这就导致了长时间的等待，最终结果甚至需要18分钟完成全部工作。

**注：由于效率实在令人难以接受，且此producer与其余策略的producer属性、方法差距较大，故此代码未被纳入4个java文件中，但作为附件一同打在压缩包中。**

**二、int转化为byte数组**

这是我最终采用的策略，并通过设定每次写入1024B、2048B、4096B三种子策略，来达到观察分析写入性能的目的。

不同于法一的节约空间，此法将效率提升到极致，直接把1G的大数据划分成N块喂给每个消费者线程。采用的主要数据结构为List<byte[]>，生产者将每个数的256个本体（对应1024B）或每两个数的256个本体（2048B）或每四个数的256个本体（4096B），转化为一个byte[]并将其传入List中，消费者线程每次写数据只需要取出一个byte[]，在确认写入位置的情况下，可直接写入。

# 2. 代码实现

#### 2.1 Producer.java

import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \** ***@author*** *19028  
 \** ***@date*** *2019/11/22 21:33  
 \*/*public class Producer {  
 private int size;  
 private int MAX\_NUM;  
 private int method;  
 private List<byte[]> data = new ArrayList<>();  
  
 Producer(int size, int MAX\_NUM, int method) {  
 this.size = size;  
 this.MAX\_NUM = MAX\_NUM;  
 this.method = method;  
 }  
  
 private byte[] intToByteArray(int value) {  
 byte[] src = new byte[4];  
 src[0] = (byte) ((value >> 24) & 0xFF);  
 src[1] = (byte) ((value >> 16) & 0xFF);  
 src[2] = (byte) ((value >> 8) & 0xFF);  
 src[3] = (byte) (value & 0xFF);  
 return src;  
 }  
  
 public List<byte[]> produce() {  
 switch (method) {  
 case 1: {  
 //每份数据为1024字节  
 for (int toProduce = 1; toProduce <= MAX\_NUM; toProduce++) {  
 int curPos = 0;  
 byte[] bytes = new byte[size \* 4];  
 byte[] eachArray = intToByteArray(toProduce);  
 for (int i = 0; i < size; i++) {  
 System.*arraycopy*(eachArray, 0, bytes, curPos, eachArray.length);  
 curPos += 4;  
 }  
 data.add(bytes);  
 }  
 break;  
 }  
 case 2: {  
 //每份数据为2048字节  
 int toProduce = 1;  
 while (toProduce <= MAX\_NUM) {  
 int curPos = 0;  
 byte[] bytes = new byte[size \* 4 \* 2];  
 for (int i = 0; i < 2; i++) {  
 byte[] eachArray = intToByteArray(toProduce);  
 for (int j = 0; j < size; j++) {  
 System.*arraycopy*(eachArray, 0, bytes, curPos, eachArray.length);  
 curPos += 4;  
 }  
 toProduce++;  
 }  
 data.add(bytes);  
 }  
 break;  
 }  
 case 3: {  
 //每份数据为4096字节  
 int toProduce = 1;  
 while(toProduce <= MAX\_NUM) {  
 int curPos = 0;  
 byte[] bytes = new byte[size \* 4 \* 4];  
 for (int i = 0; i < 4; i++) {  
 byte[] eachArray = intToByteArray(toProduce);  
 for (int j = 0; j < size; j++) {  
 System.*arraycopy*(eachArray, 0, bytes, curPos, eachArray.length);  
 curPos += 4;  
 }  
 toProduce++;  
 }  
 data.add(bytes);  
 }  
 break;  
 }  
 }  
 return data;  
 }  
}

#### 2.2 MTWriter.java

import java.io.\*;  
import java.util.List;  
import java.util.Scanner;  
import java.util.concurrent.\*;  
  
  
*/\*\*  
 \** ***@author*** *19028  
 \** ***@date*** *2019/11/23 10:21  
 \*/*public class MTWriter {  
 private final static int SIZE = 256;  
 private final static int MAX\_NUM = 2014 \* 512;  
 private int threadSize;  
 private CyclicBarrier cyclicBarrier;  
 private File file;  
 private int method;  
  
 public static void main(String[] args) throws Exception {  
 Scanner scanner = new Scanner(System.in);  
 System.out.println("-----------------请输入线程数----------------");  
 int threadSize = scanner.nextInt();  
 System.out.println("---------------------------请输入写策略-------------------------\n" +  
 "| 数字1:RandomAccessFile.write(byte[])（单次写入1024字节） |\n" +  
 "| 数字2:RandomAccessFile.write(byte[])（单次写入2048字节） |\n" +  
 "| 数字3:RandomAccessFile.write(byte[])（单次写入4096字节） |\n" +  
 "--------------------------------------------------------------");  
 int method = scanner.nextInt();  
 MTWriter mtWriter = new MTWriter(new File("data.txt"), threadSize, method);  
 mtWriter.start();  
 }  
  
 private MTWriter(File out, int threadSize, int method) {  
 this.file = out;  
 this.threadSize = threadSize;  
 this.cyclicBarrier = new CyclicBarrier(threadSize + 1);//加上主线程  
 this.method = method;  
 }  
  
 private void start() throws Exception {  
 int sliceSize = setSliceSize();  
 Producer producer = new Producer(SIZE, MAX\_NUM, method);  
 final long startTime = System.currentTimeMillis();  
  
 //得到装有每个数生成256次的总列表  
 List<byte[]> list = producer.produce();  
 int startPos = 0;  
 Thread thread;  
 for (int i = 0; i < threadSize; i++) {  
 List<byte[]> slice = list.subList(startPos, sliceSize \* (i + 1));  
 thread = new Thread(new SliceWriterThread(slice, file, cyclicBarrier, MAX\_NUM / threadSize));  
 thread.start();  
 startPos += sliceSize;  
 }  
 cyclicBarrier.await();  
 System.out.println("总时间开销: " + (System.currentTimeMillis() - startTime));  
 }  
  
 private int setSliceSize() {  
 int sliceSize = 0;  
 switch (method) {  
 case 1: {  
 sliceSize = MAX\_NUM / threadSize;  
 break;  
 }  
 case 2: {  
 sliceSize = MAX\_NUM / threadSize / 2;  
 break;  
 }  
 case 3: {  
 sliceSize = MAX\_NUM / threadSize / 4;  
 break;  
 }  
 }  
 return sliceSize;  
 }  
  
 static class SliceWriterThread extends Thread {  
 private List<byte[]> slice;  
 private CyclicBarrier cyclicBarrier;  
 private long startPos;  
 private File file;  
  
 SliceWriterThread(List<byte[]> slice, File file, CyclicBarrier cyclicBarrier, int startPos) {  
 this.slice = slice;  
 this.cyclicBarrier = cyclicBarrier;  
 this.startPos = startPos \* 4;  
 this.file = file;  
 }  
  
 @Override  
 public void run() {  
 RandomAccessFile randomAccessFile;  
 try {  
 randomAccessFile = new RandomAccessFile(file, "rw");  
 randomAccessFile.seek(startPos);  
 for (byte[] bytes : slice) {  
 randomAccessFile.write(bytes);  
 }  
 cyclicBarrier.await();  
 randomAccessFile.close();  
 } catch (IOException | InterruptedException | BrokenBarrierException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

#### 2.3 HDWriter.java

import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.FSDataOutputStream;  
import org.apache.hadoop.fs.FileSystem;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IOUtils;  
  
import java.io.\*;  
import java.util.List;  
import java.util.Scanner;  
import java.util.concurrent.BrokenBarrierException;  
import java.util.concurrent.CyclicBarrier;  
  
  
*/\*\*  
 \** ***@author*** *19028  
 \** ***@date*** *2019/11/23 10:21  
 \*/*public class HDWriter {  
 private final static int *SIZE* = 256;  
 private final static int *MAX\_NUM* = 2014 \* 512;  
 private int threadSize;  
 private CyclicBarrier cyclicBarrier;  
 private int method;  
 private File file;  
  
 public static void main(String[] args) throws Exception {  
 Scanner scanner = new Scanner(System.*in*);  
// System.out.println("-----------------请输入线程数----------------");  
// int threadSize = scanner.nextInt();  
 System.*out*.println("---------------------------请输入写策略-------------------------\n" +  
 "| 数字1:RandomAccessFile.write(byte[])（单次写入1024字节） |\n" +  
 "| 数字2:RandomAccessFile.write(byte[])（单次写入2048字节） |\n" +  
 "| 数字3:RandomAccessFile.write(byte[])（单次写入4096字节） |\n" +  
 "--------------------------------------------------------------");  
 int method = scanner.nextInt();  
 int[] threadSizes = {32, 16, 8, 4, 2, 1};  
 for (int threadSize : threadSizes) {  
 HDWriter hdWriter = new HDWriter(new File("data.txt"), threadSize, method);  
 hdWriter.start();  
 }  
 }  
  
 private HDWriter(File out, int threadSize, int method) {  
 this.file = out;  
 this.threadSize = threadSize;  
 this.cyclicBarrier = new CyclicBarrier(threadSize + 1);//加上主线程  
 this.method = method;  
 }  
  
 public void upload() {  
 try {  
 Configuration configuration = new Configuration();  
 configuration.set("dfs.client.use.datanode.hostname", "true");//坑1  
 configuration.set("fs.defaultFS", "hdfs://aliyun:9000");//坑2  
 System.*setProperty*("HADOOP\_USER\_NAME", "root");  
 FileSystem fileSystem = FileSystem.*get*(configuration);  
 InputStream inputStream = new FileInputStream("data.txt");  
 OutputStream outputStream = fileSystem.create(new Path("/data.txt"));  
 IOUtils.*copyBytes*(inputStream, outputStream, configuration);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void start() throws BrokenBarrierException, InterruptedException {  
 int sliceSize = setSliceSize();  
 Producer producer = new Producer(*SIZE*, *MAX\_NUM*, method);  
 final long startTime = System.*currentTimeMillis*();  
  
 //得到装有每个数生成256次的总列表  
 List<byte[]> list = producer.produce();  
 int startPos = 0;  
 Thread thread;  
 for (int i = 0; i < threadSize; i++) {  
 List<byte[]> slice = list.subList(startPos, sliceSize \* (i + 1));  
 thread = new Thread(new SliceWriterThread(slice, file, cyclicBarrier, MAX\_NUM / threadSize));  
 thread.start();  
 startPos += sliceSize;  
 }  
 cyclicBarrier.await();  
 upload();  
 System.out.println("总时间开销: " + (System.currentTimeMillis() - startTime));  
 }  
  
 private int setSliceSize() {  
 int sliceSize = 0;  
 switch (method) {  
 case 1: {  
 sliceSize = MAX\_NUM / threadSize;  
 break;  
 }  
 case 2: {  
 sliceSize = MAX\_NUM / threadSize / 2;  
 break;  
 }  
 case 3: {  
 sliceSize = MAX\_NUM / threadSize / 4;  
 break;  
 }  
 }  
 return sliceSize;  
 }  
  
 static class SliceWriterThread extends Thread {  
 private List<byte[]> slice;  
 private CyclicBarrier cyclicBarrier;  
 private long startPos;  
 private File file;  
  
 private SliceWriterThread(List<byte[]> slice, File file, CyclicBarrier cyclicBarrier, int startPos) {  
 this.slice = slice;  
 this.cyclicBarrier = cyclicBarrier;  
 this.startPos = startPos \* 4;  
 this.file = file;  
 }  
  
 @Override  
 public void run() {  
 RandomAccessFile randomAccessFile;  
 try {  
 randomAccessFile = new RandomAccessFile(file, "rw");  
 randomAccessFile.seek(startPos);  
 for (byte[] bytes : slice) {  
 randomAccessFile.write(bytes);  
 }  
 cyclicBarrier.await();  
 randomAccessFile.close();  
 } catch (IOException | InterruptedException | BrokenBarrierException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

## 2.4 CAWriter.java

import com.datastax.driver.core.Cluster;  
import com.datastax.driver.core.HostDistance;  
import com.datastax.driver.core.PoolingOptions;  
import com.datastax.driver.core.Session;  
import com.datastax.driver.core.querybuilder.Insert;  
import com.datastax.driver.core.querybuilder.QueryBuilder;  
import java.nio.ByteBuffer;  
import java.util.List;  
import java.util.Scanner;  
import java.util.concurrent.BrokenBarrierException;  
import java.util.concurrent.CyclicBarrier;  
  
*/\*\*  
 \** ***@author*** *19028  
 \** ***@date*** *2019/12/8 17:57  
 \*/*public class CAWriter {  
 private Cluster cluster;  
 private final static int *SIZE* = 256;  
 private final static int *MAX\_NUM* = 2014 \* 512;  
 private int threadSize;  
 private int method;  
 private CyclicBarrier cyclicBarrier;  
  
  
 public static void main(String args[]) throws BrokenBarrierException, InterruptedException {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("---------------------------请输入写策略-------------------------\n" +  
 "| 数字1:RandomAccessFile.write(byte[])（单次写入1024字节） |\n" +  
 "| 数字2:RandomAccessFile.write(byte[])（单次写入2048字节） |\n" +  
 "| 数字3:RandomAccessFile.write(byte[])（单次写入4096字节） |\n" +  
 "--------------------------------------------------------------");  
 int method = scanner.nextInt();  
 int[] threadSizes = {32, 16, 8, 4, 2, 1};  
 for (int threadSize : threadSizes) {  
 CAWriter caWriter = new CAWriter(threadSize, method);  
 caWriter.start();  
 }  
 }  
  
 private CAWriter(int threadSize, int method) {  
 this.threadSize = threadSize;  
 this.method = method;  
 this.cluster = Cluster.*builder*().addContactPoint("47.97.205.133").withPort(9042)  
 .withCredentials("dananguo", "Zyc190285975").build();  
 this.cyclicBarrier = new CyclicBarrier(threadSize + 1);//加上主线程  
 }  
  
 private void createTable() {  
 Session session = cluster.connect("mydb");  
 String cql = "DROP TABLE" + " data ";  
 session.execute(cql);  
 cql = "create table data\n" +  
 "(\n" +  
 "\tid int primary key,\n" +  
 "\tdatablock blob\n" +  
 ");";  
 session.execute(cql);  
 }  
  
 private int setSliceSize() {  
 int sliceSize = 0;  
 switch (method) {  
 case 1: {  
 sliceSize = *MAX\_NUM* / threadSize;  
 break;  
 }  
 case 2: {  
 sliceSize = *MAX\_NUM* / threadSize / 2;  
 break;  
 }  
 case 3: {  
 sliceSize = *MAX\_NUM* / threadSize / 4;  
 break;  
 }  
 }  
 return sliceSize;  
 }  
  
 private void start() throws BrokenBarrierException, InterruptedException {  
 createTable();  
 int sliceSize = setSliceSize();  
 Producer producer = new Producer(*SIZE*, *MAX\_NUM*, method);  
 final long startTime = System.*currentTimeMillis*();  
  
 //得到装有每个数生成256次的总列表  
 List<byte[]> list = producer.produce();  
 int startPos = 0;  
 Thread thread;  
 for (int i = 0; i < threadSize; i++) {  
 List<byte[]> slice = list.subList(startPos, sliceSize \* (i + 1));  
 thread = new Thread(new SliceWriterThread(slice, cyclicBarrier, cluster, i));  
 thread.start();  
 startPos += sliceSize;  
 }  
 cyclicBarrier.await();  
 System.*out*.println("总时间开销: " + (System.*currentTimeMillis*() - startTime));  
 }  
  
 static class SliceWriterThread extends Thread {  
 private List<byte[]> slice;  
 private CyclicBarrier cyclicBarrier;  
 private Cluster cluster;  
 private int id;  
  
 private SliceWriterThread(List<byte[]> slice, CyclicBarrier cyclicBarrier, Cluster cluster, int id) {  
 this.slice = slice;  
 this.cyclicBarrier = cyclicBarrier;  
 this.cluster = cluster;  
 this.id = id \* slice.size();  
 }  
  
 @Override  
 public void run() {  
 Session session = cluster.connect("mydb");  
 ByteBuffer buffer;  
 try {  
 int id = this.id;  
 for (byte[] bytes : slice) {  
 buffer = ByteBuffer.*wrap*(bytes);  
 Insert insert = QueryBuilder.*insertInto*("mydb", "data").  
 value("id", id).  
 value("datablock", buffer);  
 session.execute(insert);  
 id++;  
 }  
 cyclicBarrier.await();  
 } catch (Exception e) {  
 e.printStackTrace();  
 System.*out*.println("插入失败");  
 }  
 System.*out*.println("finish");  
 }  
 }  
}

#### 2.5 代码综述

HDWriter、CAWriter都是基于MTWriter修改而来，其中替换了相应的HDFS、CassandraAPI，以达到在各自文件系统上测试的目的。

# 3. 结果分析

## 3.1 MTWriter

此图Y轴单位为ms，X轴为写线程数。

由图可得，线程数与时间总体成反比，且在4线程之前，线程数翻倍，所耗时间近乎减半，到4线程之后，由于CPU核数限制，即便线程数继续增长，也不会带来性能的显著提升，甚至由于CPU占用率过高，会使得消耗时间出现波动。

同时，能够明显的看到，单次写入数据量的增加会提高写入效率，据此推测，频繁取数据也会消耗大量时间。

## 3.2 HDWriter

XY轴含义同上。由于Hadoop不支持多线程写入，所以我采用的实现手段是先用多线程写入本地文件，再将本地文件上传到阿里云HDFS中，理想的结果应该是相同的上传时间+写入时间，曲线趋势应该也是线程数与消耗时间成反比，但出现此图的原因，很有可能是网络波动带来的影响。

## 3.3 CAWriter

XY轴含义同上。Cassandra支持多线程写入，在直接写入数据库的情况下，网络波动可分担到每次写入上，所以最终呈现的结果具有较高价值。

由此可得，线程数越大，写入效率越高；单次写入数据越大，写入效率越高。由于Cassandra默认支持最大32线程并发，因此能得到较好的下降曲线，且随着线程数的继续增长，性能逐渐达到瓶颈，三条曲线趋于拟合。