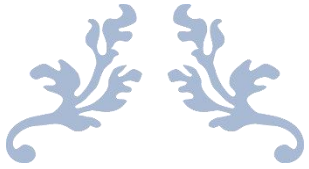


分布式系统第二次作业

1752124 朴雪

2019-12-8

同济大学
软件学院

目录

分布式系统第二次作业

1752124 朴雪

1.开发环境及注意事项

1.1 开发平台：

平台：win10
开发环境：IDEAC2019.3
语言：JAVA
云服务器：阿里云（部署 Hadoop 和 Cassandra 数据库）

1.2 配置：

项目配置：Maven 项目
本地配置 host：

```
# My hosts
39.97.175.111 master
39.97.175.111 iz2ze5corjxmmuzbwnxvhjz
```

开发环境配置：修改 JVM 内存配置为 

1.3 注意事项：

* 写入数据格式采用 string，其中不同数据格式造成不同文件大小
* 在服务器端向数据库中插入数据，由于数据量较大花费时间长可以改变重复次数进行测试，例如在 producer.java 文件中：

```
//      重复次数
    private static final int REPEAT_NUM = 256;
//    private static final int REPEAT_NUM = 1;
```

* 在执行第 2、3 问时，分别打开服务器上各自的数据库，关闭另一个

2.使用说明

2.1 数据测试

* 通过取消和增添注释来实现不同方法的写文件：

```
        // 方法一：outputStream方式
        for (int i = 0; i < THREAD_NUM; i++) {
            int[] numArr = Arrays.copyOfRange(Producer.NUM_ARR, from: i*newArrSize, to: (i+1)*newArrSize);
            streamExecutorService ws = new streamExecutorService(numArr,file, latch);
            pool.execute(ws);
        }
        // 方法二：objectStream方式
//        for (int i = 0; i < THREAD_NUM; i++) {
//            int[] numArr = Arrays.copyOfRange(Producer.NUM_ARR,i*newArrSize,(i+1)*newArrSize);
//            ObjectExecutorService ws = new ObjectExecutorService(numArr,file, latch);
//            pool.execute(ws);
//        }
        // 方法三：nio方式
//        for (int i = 0; i < THREAD_NUM; i++) {
//            int[] numArr = Arrays.copyOfRange(Producer.NUM_ARR,i*newArrSize,(i+1)*newArrSize);
//            NIOExecutorService ws = new NIOExecutorService(numArr,file, latch);
//            pool.execute(ws);
//        }
```

* 每次执行输出一个文件，通过手动改变线程常量值来实现多线程写文件并记录数值：

```
private final static int THREAD_NUM = 1;
```

其中线程数值即 THREAD_NUM 取 1，2，4，8，16，32

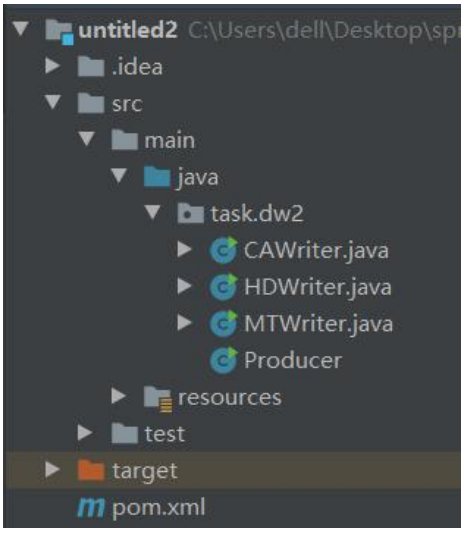
2.2 数据存储

前两问将数据以.txt 格式存入本地，后一问我直接在数据库中读取，其中存入本地时 main 函数中写入的路径及文件名可能需要改变，这里我将其存入 C 盘如下：

```
File file = new File( pathname: "C:\\doc\\dw\\wt_os_1.txt");
ExecutorService pool = Executors.newFixedThreadPool(THREAD_NUM);
```

3.代码结构

3.1 项目目录



主要代码在./src/main/java 的 task.dw2 包中，由 4 个 java 文件构成：

* Producer.java 为生成数据文件
* MTWriter.java 为普通写入：
  方法一：outputstream 方式写入
  方法二：objectstream 方式写入
  方法三：nio 方式写入
* HDWriter.java 为 HDFS 文件系统下的写入文件：
  方法一：本地写文件，然后上传 hdfs 系统
  方法二：hdfs 系统写文件
* CAWriter.java 为 Apacha Cassandra KVS 文件系统下的写入文件：
  方法一：单个插入
  方法二：批量插入

3.2 主要功能代码说明

3.2.1 生成数据

* 数据量及重复次数：

```
    // 2014*512*256
    private static final int ARR_SIZE = 263979008;
//    private static final int ARR_SIZE = 2014*512;

    // 2014*512
    public static final int MAX_NUM = 1031168;

//     重复次数
    private static final int REPEAT_NUM = 256;
//    private static final int REPEAT_NUM = 1;
```

* 生成函数：

```
public static void produceNum() {
    int idx = 0;
    for (int i = 0; i < MAX_NUM; i++) {
        for (int j = 0; j < REPEAT_NUM; j++) {
            NUM_ARR[idx++] = i+1;
        }
    }
}
```

3.2.2 写入数据

* outputstream 方式写入：

```java
@Override
public void run() {
    DataOutputStream dos = null;
    try {
        dos = new DataOutputStream(new BufferedOutputStream(new FileOutputStream(file,  append: true),  size: 1024*1024));
        for (int one:numArr)
        {
            dos.writeInt(one);
            if (one % 100000 == 0) {
                System.out.println(Thread.currentThread().getName() + "\t" + one);
            }
        }
        dos.flush();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            dos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        latch.countDown();
    }
}
```

* objectstream 方式写入：

```java
@Override
public void run() {
    ObjectOutputStream out = null;
    try {
        out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(file,  append: true), size: 1024*1024));
        out.writeObject(numArr);
    } catch (IOException e) {
        throw new RuntimeException(e);
    } finally {
        try {
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        latch.countDown();
    }
}
```

* nio 方式写入：

```java
@Override
public void run() {
    try {
        FileChannel channel = new RandomAccessFile(file, mode: "rw").getChannel();
        MappedByteBuffer buffer = channel.map(FileChannel.MapMode.READ_WRITE, file.length(),  size: 4 * numArr.length);
        for (int one: numArr) {
            buffer.putInt(one);
        }
        channel.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    } finally {
        latch.countDown();
    }
}
```

3.2.3 HDFS 文件系统

* 本地写文件：

```
@Override
public void run() {
    try {
        FileChannel channel = new RandomAccessFile(file, mode: "rw").getChannel();
        MappedByteBuffer buffer = channel.map(FileChannel.MapMode.READ_WRITE, file.length(), size: 4 * numArr.length);
        for (int one: numArr) {
            buffer.putInt(one);
        }
        channel.close();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        latch.countDown();
    }
}
```

3.2.4 Apacha Cassandra KVS 文件系统

* 建立连接：

```
/**
 * 建立连接
 */
private void connect()
{
    // addContactPoints:cassandra节点ip withPort:cassandra节点端口 默认9042
    cluster = Cluster.builder().addContactPoints("39.97.175.111").withPort(9042)
            .withCredentials( username: "cassandra", password: "cassandra").build();
    session = cluster.connect();
}
```

* 建立表和键空间：

```
/**
 * 创建键空间, 键名mydb
 */
private void createKeyspace()
{
    // 单数据中心 复制策略 :1
    String cql = "CREATE KEYSPACE if not exists mydb WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'}";
    session.execute(cql);
}

/**
 * 创建表, 表名 dw
 */
private void createTable()
{
    String cql = "CREATE TABLE if not exists mydb." + TABLE_NAME + " (id text,a int,PRIMARY KEY(id))";
    session.execute(cql);
}
```

* 查询：

```java
/**
 * 查询
 */
private void query()
{
    String cql = "SELECT * FROM mydb."+TABLE_NAME+";";

    ResultSet resultSet = session.execute(cql);
    System.out.print("这里是字段名：");
    for (ColumnDefinitions.Definition definition : resultSet.getColumnDefinitions())
    {
        System.out.print(definition.getName() + " ");
    }
    System.out.println();
    System.out.println(String.format("%s\t%s\t", "id","a"));
    System.out.println("=========================");
    for (Row row : resultSet)
    {
        System.out.println(String.format("%s\t%d\t", row.getString( s: "id"), row.getInt( s: "a")));
    }
}
```

* 插入执行器：

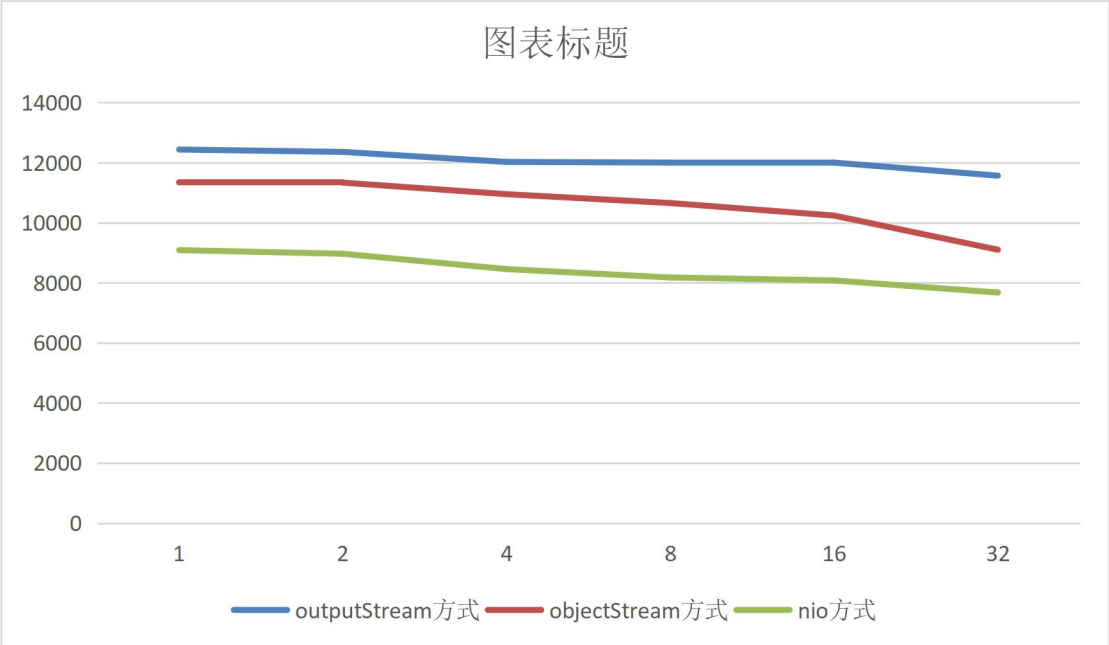
```java
/**
 * 插入执行器
 */
class CAExecutor implements Runnable {
    private int[] numArr;
    private CountDownLatch latch;
    private Session session;

    public CAExecutor(int[] numArr, Session session, CountDownLatch latch) {
        this.numArr = numArr;
        this.session = session;
        this.latch = latch;
    }

    @Override
    public void run() {
        for (int one: numArr) {
            String uuid = UUID.randomUUID().toString().replaceAll( regex: "-", replacement: "");
            String cql = String.format("INSERT INTO mydb.%s (id,a) VALUES (%s,%d);", uuid,one);
            session.execute(cql);
        }
        latch.countDown();
    }
}
```

* 批量插入执行器：

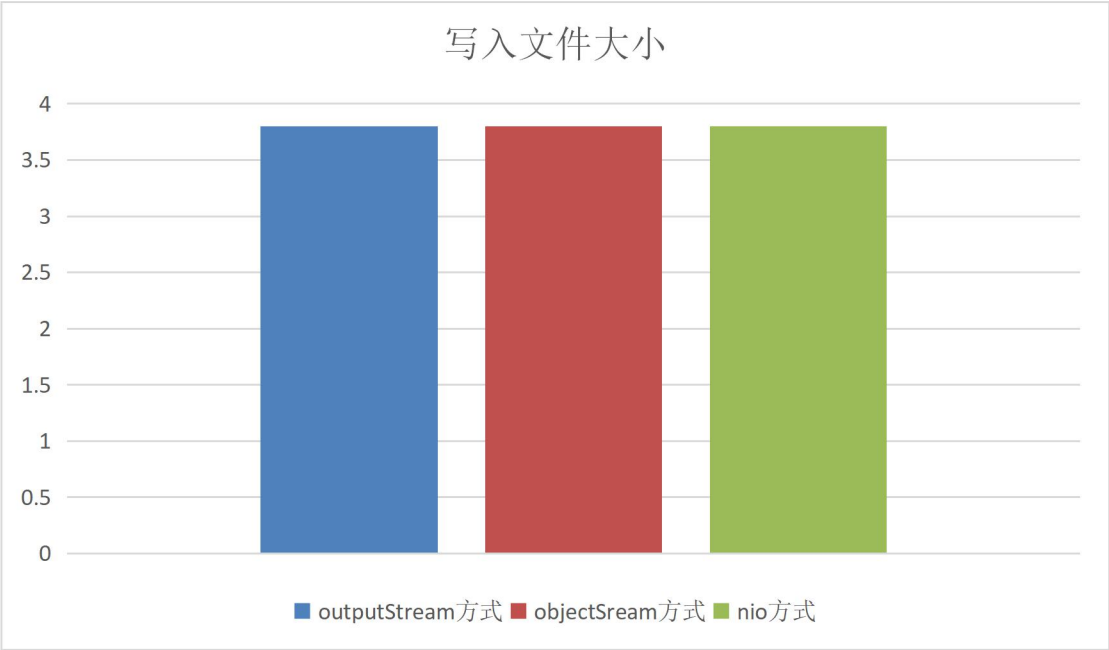
```java
@Override
public void run() {
    BatchStatement batch = new BatchStatement();
    PreparedStatement ps = session.prepare( s: "insert into mydb.dw(id,a) values(?,?)");
    for (int i = 0; i < numArr.length; i++) {
        String uuid = UUID.randomUUID().toString().replaceAll( regex: "-", replacement: "");
        BoundStatement bs = ps.bind(uuid,numArr[i]);
        batch.add(bs);
        if ((i+1) % 1024 == 0) {
            session.execute(batch);
            batch.clear();
            batch = new BatchStatement();
        }
    }
    session.execute(batch);
    batch.clear();
    latch.countDown();
}
```

4.数据测试

4.1 MTWriter



（纵坐标单位为 ms）

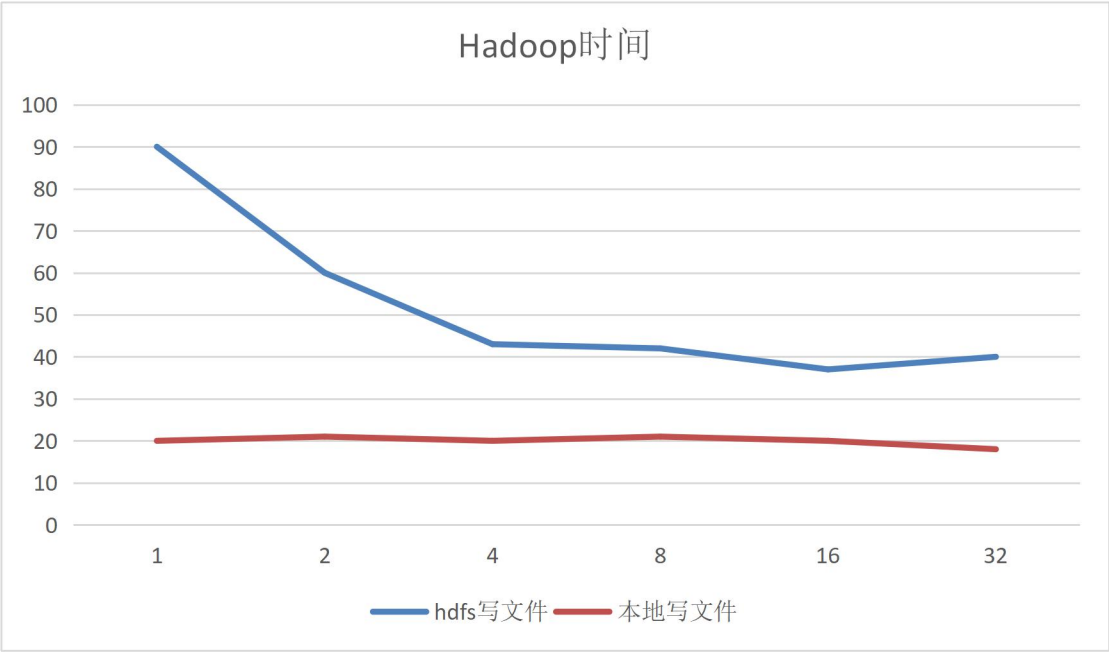


（纵坐标单位为 G）

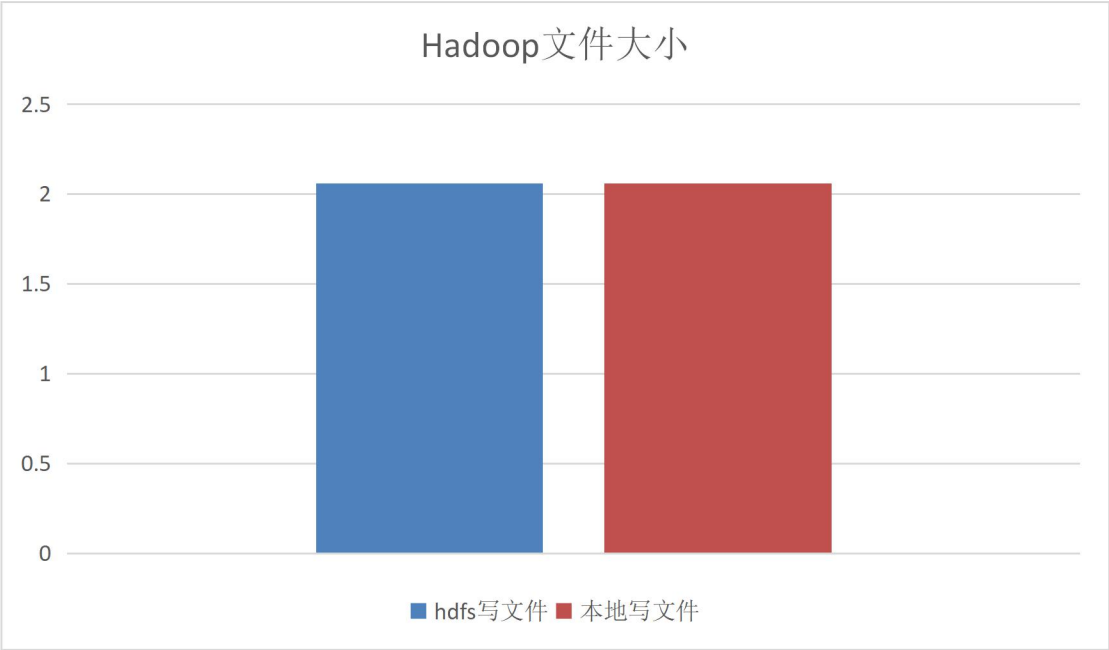
通过分析比对字节流方式写入、object 方式和 nio 方式的时间数据，可以发现写入同样字节的数据，字节流方式所用时间最多，nio 方式所用时间最少，因此效率由高到低为：nio 方式，object 方式，字节流方式。

通过分析比对字节流方式写入、object 方式和 nio 方式的空间数据，可以发现线程数越多，写入效率越高。

4.2 HDWriter



（纵坐标单位为 s）

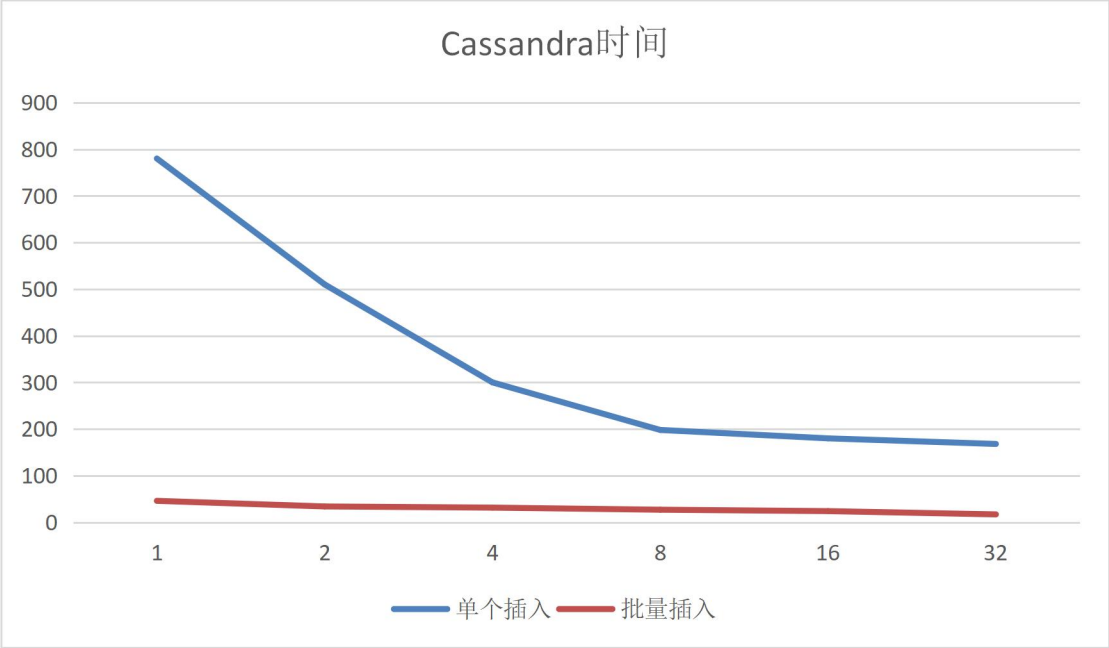


（纵坐标单位为 G）

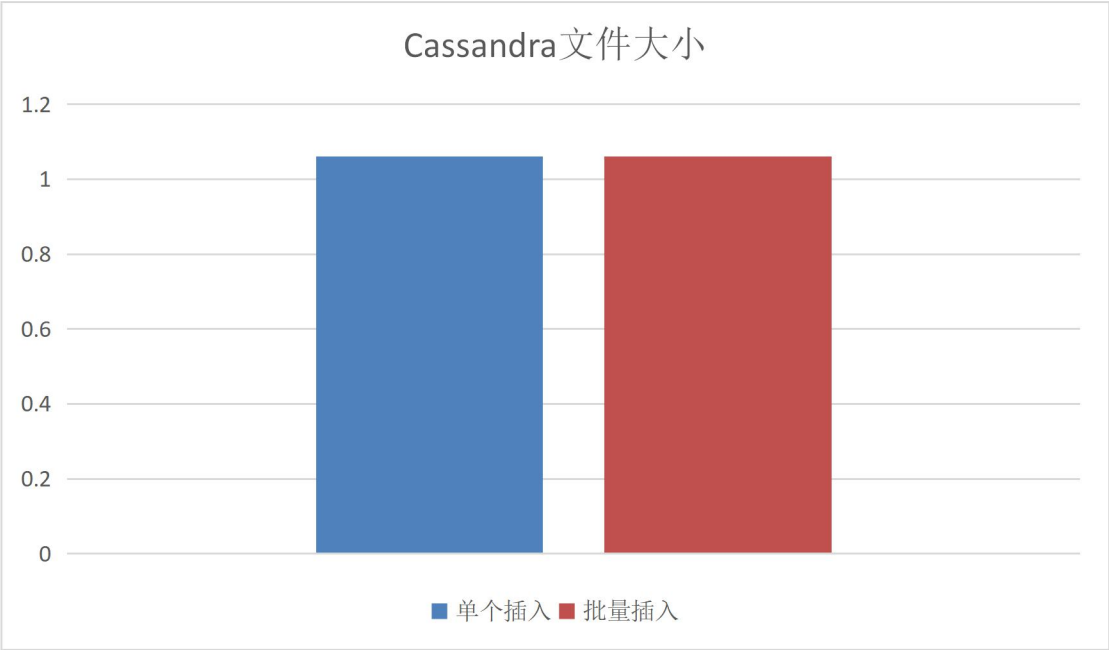
通过分析比对本地写文件和 hdfs 写文件的时间数据，可以发现 hdfs 写文件所用时间更多，效率更差，本地写文件然后上传 hdfs 系统更高效。

通过分析比对本地写文件和 hdfs 写文件的空间数据，可以发现线程数越多，写入效率越高。

4.3 CAWriter



（纵轴单位为 min）



（纵坐标单位为 G）

通过分析比对单个插入和批量插入的时间数据，可以发现单个插入所用时间更多，效率更差，批量插入更高效。

通过分析比对单个插入和批量插入的空间数据，可以发现线程数越多，写入效率越高。