大数据管理技术 第一次小组实习

林汇平 1800013104 | 奚佳琪 1800012923

- 实习要求：参考助教提供的HDFS文件创建读写操作的Java代码(在hadoop安装相关的文档中)，编程实现在 HDFS 中创建大批量小文件(文件内容随意)，分析小文件的 Block 是多大，使用hdfs存储小文件的缺点是什么，你什么改进措施吗?
- 报告内容：请在报告中详细写明你的实验步骤、技术方法、实习体会等，附上相应的代码段和截图。

---

1. 在HDFS 中创建大批量小文件的HDFSwrite类java代码：

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FSDataInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

//本程序实现在 HDFS 中创建大批量小文件(文件内容、文件名称与文件数量皆可指定)，在全分布式Hadoop运行成功。
public class HDFSwrite {
    public static void main(String[] args) throws Exception {
        int i = 1;
        while(i<10000) {
        Configuration conf=new Configuration();
        conf.set("fs.defaultFS", "hdfs://Master:9000");
        conf.set("fs.hdfs.impl", "org.apache.hadoop.hdfs.DistributedFileSystem");
        FileSystem fs = FileSystem.get(conf);
        String fileName = i+"";
        Path file = new Path(fileName);
        createFile(fs,file);
        readFile(fs,file);
        fs.close();
        ++i;
        }
    }

public static void createFile(FileSystem fs, Path file) throws IOException{
    byte[] buff = "Hi! This is a file!".getBytes();
    FSDataOutputStream os =fs.create(file);
    os.write(buff,0,buff.length);
    System.out.println("Create:"+file.getName());
    os.close();
    }
public static void readFile(FileSystem fs, Path file) throws IOException{
    FSDataInputStream in = fs.open(file);
    BufferedReader d = new BufferedReader(new InputStreamReader(in));
    String content = d.readLine();
    System.out.println(content);
```

```
    d.close();
    in.close();
    }
}
```

其中，文件内容、文件名称与文件数量可以任意指定。此版本代码建立的文件都在root目录下。默认建立文件数量为10000，文件名称从1开始编号，文件内容为"Hi! This is a file!"。

运行结果如下：（截图仅显示前9个小文件），可以看到这些文件的属主与权限。

```
phoenix@Master:~/myapp$ hdfs dfs -ls
Found 11 items
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 1
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 2
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 3
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 4
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 5
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 6
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 7
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 8
-rw-r--r--   3 phoenix supergroup         19 2020-07-14 10:16 9
```

2. 查看小文件的块大小：

```
phoenix@Master:~/myapp$ hdfs dfs -du 1
19  1
phoenix@Master:~/myapp$ hdfs dfs -du 2
19  2
phoenix@Master:~/myapp$ hdfs dfs -du 3
19  3
```

```
phoenix@Master:~/myapp$ hdfs dfs -stat %o 1
134217728
phoenix@Master:~/myapp$ hdfs dfs -stat %o 2
134217728
phoenix@Master:~/myapp$ hdfs dfs -stat %o 3
134217728
phoenix@Master:~/myapp$ hdfs dfs -stat %o 4
134217728
phoenix@Master:~/myapp$ hdfs dfs -stat %o 5
134217728
```

我们可以通过-du命令看到这些小文件的大小都是19Byte（19字节），并通过-stat命令的%o选项看到这些小文件的块大小都是134217728Byte，即128MB。HDFS的文件在物理上是分块储存(Block)，块的大小可以通过配置参数(dfs.blocksize)来规定，默认大小在Hadoop2.x版本中是128MB。对于小文件，一个文件占用一个块，每个文件的块大小就是128MB。即这些小文件的块大小是128MB。

设置成这个大小的原因为：(1) 如果HDFS的块设置太小，会增加寻址时间，程序一直在找块的开始位置。(2) 如果HDFS的块设置太大，从磁盘传输数据的时间会明显大于定位这个块开始位置所需的时间。导致程序在处理这块数据时，会非常慢。

3. 使用HDFS存储大量小文件：

首先，考虑到如下的HDFS文件读写流程：

读文件时，client端发送读文件请求给namenode，如果文件不存在，返回错误信息，否则，将该文件对应的block及其所在datanode位置发送给client。 client收到文件位置信息后，与不同datanode建立socket连接并行获取数据。

写文件时，client端发送写文件请求，namenode检查文件是否存在，如果已存在，直接返回错误信息，否则，发送给client一些可用datanode节点。client将文件分块，并行存储到不同节点的datanode上。发送完成后，datanode同时发送信息给namenode和client，client收到反馈后向namenode发送确认信息。namenode同时收到client和datanode的确认信息后，提交写操作。

经过对上述读写文件的流程分析，我们可以看到HDFS在批量读取/处理/存储小文件时的缺点：

(1) 小文件大block导致的内存浪费。

在HDFS中，任何block、文件或者目录在内存中均以对象的形式存储，文件的元数据（包括文件被分成了哪些blocks，每个block存储在哪些服务器的哪个block块上），都是存储在namenode上的。HDFS的每个文件、目录、数据块（即每个对象）约占150字节，如果有1000 0000个小文件，每个文件占用一个block，则namenode大约需要2G空间。如果存储1亿个文件，则namenode需要20G空间。小文件过多，会过多占用namenode的内存，并浪费block，而集群存储的数据却很少。因此，namenode内存容量严重制约了集群的扩展。

(2) 小文件相对大文件的效率低下。

对于大量的小文件而言，相比数据读写等处理时间，寻道花费时间更多。在读小文件时，访问大量小文件速度远远小于访问几个大文件。因为HDFS最初是为流式访问大文件开发的，如果访问大量小文件，需要不断的从一个datanode跳到另一个datanode，严重影响性能。

具体举例来说，一个128 MB的文件在namenode上通过两个namespace对象（1个inode + 1个块）表示，消耗大约300个字节的内存。相比之下，每个1 MB的128个文件由256个命名空间对象（128个inode + 128个块）表示，并消耗大约38,400个字节。

类似的，处理大量小文件速度远远小于处理同等大小的大文件的速度。因为每一个小文件要占用一个slot，而task启动将耗费大量时间甚至大部分时间都耗费在启动task和释放task上。

4. 可能的改进措施：

(1) 将大批量的小文件“打包”成少量的大文件。

打包成大文件后，在文件前端记录下每个小文件的具体信息，如文件大小、文件名称、文件头寻址方式等。将大文件作为单独的文件用HDFS的存储方式去存储。

这样的存储方式可以提高namenode和block的利用效率，访问大量小文件时只需先访问一个大文件，然后在大文件内部进行读取。但这样的方式使得小文件“紧密排列”，更适合文件的存储与读，而不适合文件的删除与写，因为这样有可能导致整个大文件需要重新组织。会比单独写小文件更耗时。如果小文件遵循HDFS设计理念，一次写多次读，则可以忽略这一缺陷。

(2) 方法(1)的改进：将小文件按大小分类，并分别打包。

分类方式如：小于1MB的认定为1MB，大于等于1MB小于2MB的认定为2MB，大于等于2MB小于4MB的认定为4MB……一直到大于等于64MB小于128MB的认定为128MB。然后将类别相同的小文件合并成128MB(默认的block size)，并建立索引，索引存储方式可以类似于(1)的方式写在大文件头部。索引方式可以是一般的定长hash。那么这样即使对小文件有写或者删除操作，只要改动不是很大，使得文件被修改后仍在同类中，就可以较快地进行修改。

(3) 基于访问相关性特点的小文件合并与索引。[1]

将同类型的、大概率被先后访问的小文件组织成大文件，如改进方法(1)中的打包方式。然后以建立两层访问机制：首先读取索引文件，即用户访问的文件所在的block对应的索引，从而用户在之后的文件访问中，可以直接通过索引而不必与namenode交互，然后再读取数据文件。这一方法有助于提高连续访问相关小文件的效率。

---

1. 改进思路参考Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li, Ying Li. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files. In Proceedings of IEEE SCC’2010. pp.65~72 ↩