Carpe diem (Felix's blog)

I am a happy developer

- RSS

Search

Navigate…

- Blog
- Archives
- About Me
- Github
- Drawings

Java Fast IO Using java.nio API

Sep 28th, 2013

For modern computing, IO is always a big bottleneck to solve. I recently encounter a problem is to read a 355MB index file to memory, and do a run-time lookup base the index. This process will be repeated by thousands of Hadoop job instances, so a fast IO is a must. By using the `java.nio` API I sped the process from 194.054 seconds to 0.16 sec! Here’s how I did it.

The Data to Process

This performance tuning practice is very specific to the data I’m working on, so it’s better to explain the context. We have a long ip list (26 millions in total) that we want to put in the memory. The ip is in text form, and we’ll transform it into signed integer and put it into a java array. (We use signed integer because java doesn’t support unsigned primitive types…) The transformation is pretty straight forward:

```
public static int ip2integer (String ip_str){
  String [] numStrs = ip_str.split("\\.");
  long num;
  if (numStrs.length == 4){
    num =
        Long.parseLong(numStrs[0]) * 256 * 256 * 256
        + Long.parseLong(numStrs[1]) * 256 * 256
        + Long.parseLong(numStrs[2]) * 256
        + Long.parseLong(numStrs[3]);
    num += Integer.MIN_VALUE;
    return (int)num;
  } else {
    System.err.println("IP is wrong: "+ ip_str);
    return Integer.MIN_VALUE;
  }
}
```

However, reading ip in text form line by line is really slow.

Strategy 1: Line-by-line text processing

This approach is straight forward. Just a standard readline program in java.

```java
private int[] ipArray = new int[26123456];
public static void readIPAsText() throws IOException{
  BufferedReader br = new BufferedReader(new FileReader("ip.tsv"));
  DataOutputStream ds = new DataOutputStream(fos);
  String line;
  int i = 0;

  while ((line = br.readLine()) != null) {
    int ip_num = ip2integer(line);
    ipArray[i++] = ip_num;
  }
  br.close();
}
```

The result time was `194.054` seconds.

Strategy 2: Encode ip in binary format

The file size of the `ip.tsv` is 355MB, which is inefficient to store or to read. Since I'm only reading it to an array, why not store it as a big chunk of binary array, and read it back while I need it? This can be done by DataInputStream and DataOutputStream. After shrinking the file, the file size became 102MB.

Here's the code to read ip in binary format:

```java
public static void readIPAsDataStream() throws IOException{
  FileInputStream fis = new FileInputStream(new File("ip.bin"));
  DataInputStream dis = new DataInputStream(fis);
  int i = 0;
  try {
    while(true){
      ipArr[i++] = dis.readInt();
    }
  }catch (EOFException e){
    System.out.println("EOF");
  }
  finally {
    fis.close();
  }
}
```

The resulting time was `72` seconds. Much slower than I expected.

Strategy 3: Read the file using java.nio API

The `java.nio` is a new IO API that maps to low level system calls. With these system calls we can perform libc operations like `fseek`, `rewind`, `ftell`, `fread`, and bulk copy from disk to memory. For the C API you can view it from GNU C library reference.

The terminology in C and Java is a little bit different. In C, you control the file IO by file descriptors; while in `java.nio` you use a FileChannel for reading, writing, or manipulate the position in the file. Another difference is you can bulk copy directly using the `fread` call, but in Java you need an additional `ByteBuffer` layer to map the data. To understand how it work, it's better to read it from code:

```
public static void readIPFromNIO() throws IOException{
  FileInputStream fis = new FileInputStream(new File("ip.bin"));
  FileChannel channel = fis.getChannel();
  ByteBuffer bb = ByteBuffer.allocateDirect(64*1024);
  bb.clear();
  ipArr = new int [(int)channel.size()/4];
  System.out.println("File size: "+channel.size()/4);
  long len = 0;
  int offset = 0;
  while ((len = channel.read(bb))!= -1){
    bb.flip();
    //System.out.println("Offset: "+offset+"\tlen: "+len+"\tremaining:"+bb.hasRemaining());
    bb.asIntBuffer().get(ipArr,offset,(int)len/4);
    offset += (int)len/4;
    bb.clear();
  }
}
```

The code should be quite self-documented. The only thing to note is the byte-buffer's `flip()` method. This call convert the buffer from writing data to buffer from disk to reading mode, so that we can read the data to int array via method `get()`. Another thing worth to mention is java use big-endian to read and write data by default. You can use `ByteBuffer.order(ByteOrder.LITTLE_ENDIAN)` to set the endian if you need it. For more about `ByteBuffer` here's a good blog post that explains it in detail.

With this implementation, the result performance is `0.16` sec! Glory to the `java.nio`!

Posted by dryman (Felix Ren-Chyan Chern) Sep 28th, 2013 Java

Tweet 2 +1 0

Like Share 11 people like this. Sign Up to see what your friends like.

« Process Small Files on Hadoop using CombineFileInputFormat (2) Convert utf8 literals in Java »

Comments

9 Comments Carpe diem Login

Sort by Best Share Favorite

Join the discussion…

Duncan • a year ago

In strategy 2 you should decorate file input stream with a buffered input stream. And you'll see that strategy 2 is not so bad.

1 • Reply • Share ›

Felix Chern Mod Duncan • a year ago

I just tried it, but found I can't use BufferedInputStream with DataInputStream. One way to do is to use BufferedInputStream and read several bytes, then use ByteBuffer to convert it to int.

• Reply • Share ›

Duncan Felix Chern • a year ago

FileInputStream fis = new FileInputStream(new File("ip.bin"));

DataInputStream dis = new DataInputStream(new BufferedInputStream(fis, 64*1024));

• Reply • Share ›

Felix Chern Mod Duncan • a year ago

Thanks, I got it worked. The result time is 2.375. Much better than the non-buffered result!

• Reply • Share ›

jerry • a year ago

NIO = non blocking IO
NOT NEW IO

• Reply • Share ›

Felix Chern Mod jerry • a year ago

http://docs.oracle.com/javase/...
The first line states "New I/O APIs"

Non blocking is one of the features you can get from java nio, but not the only one (many of the API are still blocking actually)

• Reply • Share ›

Felix Chern Mod • a year ago

I also wrote the program in C; the performance is really close.

• Reply • Share ›

Brian • a year ago

BTW, I really admire your hardwork, keep going!

• Reply • Share ›

Brian • a year ago

"The java.nio is a new IO API that maps to low level system calls."
Does that mean this implementation yields similar performance when we simply implement these codes in C?

• Reply • Share ›

ALSO ON **CARPE DIEM** WHAT'S THIS?

Yet Another Monad Tutorial in 15 Minutes

1 comment • a year ago

sunny — Thank you for this very nice intro to monad. I find it very helpful!

Setting up Python on OSX Mountain Lion

7 comments • 2 years ago

Felix Chern — Glad you like it! I haven't update this post for a while. Thanks for reporting the issue.

Hadoop performance tuning best practices

2 comments • 10 months ago

Mihir — good one. How about using RawComparator instead of WritableComparator!

Convert uf8 literals in Java

1 comment • a year ago

Papick G. Taboada — You could use the ones from apache commons-lang...StringEscapeUtils.escapeJava("....");

Subscribe Add Disqus to your site Privacy

Recent Posts

- Writing 64 bit assembly on Mac OS X
- Integer Promotion Part 2
- Introducing Hadoop-FieldFormat
- Hadoop performance tuning best practices
- Setting up Jasper Server on Linux
- Capture path info in hadoop InputFormat class
- Capture directory context in Hadoop Mapper
- Yet Another Monad Tutorial in 15 Minutes
- Convert utf8 literals in Java
- Java fast IO using java.nio API

Latest Tweets

- Status updating...

Follow @idryman