Binary search in a sorted (memory-mapped ?) file in Java

I am struggling to port a Perl program to Java, and learning Java as I go. A central component of the original program is a Perl module that does string prefix lookups in a +500 GB sorted text file using binary search (essentially, "seek" to a byte offset in the middle of the file, backtrack to nearest newline, compare line prefix with the search string, "seek" to half/double that byte offset, repeat until found...)

I have experimented with several database solutions but found that nothing beats this in sheer lookup speed with data sets of this size. Do you know of any existing Java library that implements such functionality? Failing that, could you point me to some idiomatic example code that does random access reads in text files?

Alternatively, I am not familiar with the new (?) Java I/O libraries but would it be an option to memory-map the 500 GB text file (I'm on a 64-bit machine with memory to spare) and do binary search on the memory-mapped byte array? I would be very interested to hear any experiences you have to share about this and similar problems.

java nio large-files binary-search memory-mapping

edited Nov 17 '11 at 17:15
Charles
43k 11 80 118

asked Apr 10 '09 at 2:39
sds
178 1 3 7

8 Answers

I am a *big* fan of Java's `MappedByteBuffers` for situations like this. It is blazing fast. Below is a snippet I put together for you that maps a buffer to the file, seeks to the middle, and then searches backwards to a newline character. This should be enough to get you going?

I have similar code (seek, read, repeat until done) in my own application, benchmarked `java.io` streams against `MappedByteBuffer` in a production environment and posted the results on my blog (Geekomatic posts tagged 'java.nio') with raw data, graphs and all.

Two second summary? *My `MappedByteBuffer` -based implementation was about 275% faster.* YMMV.

To work for files larger than ~2GB, which is a problem because of the cast and `.position(int pos)` , I've crafted paging algorithm backed by an array of `MappedByteBuffer` s. You'll need to be working on a 64-bit system for this to work with files larger than 2-4GB because MBB's use the OS's virtual memory system to work their magic.

```
public class StusMagicLargeFileReader  {
    private static final long PAGE_SIZE = Integer.MAX_VALUE;
    private List<MappedByteBuffer> buffers = new ArrayList<MappedByteBuffer>();
    private final byte raw[] = new byte[1];

    public static void main(String[] args) throws IOException {
        File file = new File("/Users/stu/test.txt");
        FileChannel fc = (new FileInputStream(file)).getChannel();
        StusMagicLargeFileReader buffer = new StusMagicLargeFileReader(fc);
        long position = file.length() / 2;
        String candidate = buffer.getString(position--);
        while (position >=0 && !candidate.equals('\n'))
            candidate = buffer.getString(position--);
        //have newline position or start of file...do other stuff
    }
```

```
    StusMagicLargeFileReader(FileChannel channel) throws IOException {
        long start = 0, length = 0;
        for (long index = 0; start + length < channel.size(); index++) {
            if ((channel.size() / PAGE_SIZE) == index)
                length = (channel.size() - index *  PAGE_SIZE) ;
            else
                length = PAGE_SIZE;
            start = index * PAGE_SIZE;
            buffers.add(index, channel.map(READ_ONLY, start, length));
        }
    }
    public String getString(long bytePosition) {
        int page  = (int) (bytePosition / PAGE_SIZE);
        int index = (int) (bytePosition % PAGE_SIZE);
        raw[0] = buffers.get(page).get(index);
        return new String(raw);
    }
}
```

edited Aug 31 '11 at 19:45

community wiki
14 revs, 3 users 99%
Stu Thompson

2 I can't believe that the NIO buffers use an int as offset ruling out the possibility to use it with more than 2 GB. That is nearly stupid on todays machines. In this context, as fast as it is, this rules out the approach in the context given here. – dmeister Apr 10 '09 at 8:26

3 Note that the FileChannel.map() function takes a long, but ByteBuffer itself takes only ints. You can use files that are much larger than 2GB, just that any particular mapped view itself can only be 2GB. (for the record the Win32 OS has the same limitation) – Jason S Apr 10 '09 at 13:09

Good point, Jason S. – Stu Thompson Apr 10 '09 at 14:09

2 I've updated the code. I had tested on a small file, but with a real large file (I am benchmarking on a 360GB tar ball) it was a problem with some of the integers wrapping to negative numbers. – Stu Thompson Apr 16 '09 at 12:21

1 The number of buffers is fixed, based upon file size. The key is there in the construtor of `StusMagicLargeFileReader` , where the MBBs are instanced. The number of MBBs is based upon the file size. – Stu Thompson Dec 17 '09 at 19:04

I have the same problem. I am trying to find all lines that start with some prefix in a sorted file.

Here is a method I cooked up which is largely a port of Python code found here:
http://www.logarithmic.net/pfh/blog/01186620415

I have tested it but not thoroughly just yet. It does not use memory mapping, though.

```
public static List<String> binarySearch(String filename, String string) {
    List<String> result = new ArrayList<String>();
    try {
        File file = new File(filename);
        RandomAccessFile raf = new RandomAccessFile(file, "r");

        long low = 0;
        long high = file.length();

        long p = -1;
        while (low < high) {
            long mid = (low + high) / 2;
            p = mid;
            while (p >= 0) {
                raf.seek(p);

                char c = (char) raf.readByte();
                //System.out.println(p + "\t" + c);
                if (c == '\n')
                    break;
                p--;
            }
            if (p < 0)
                raf.seek(0);
            String line = raf.readLine();
            //System.out.println("-- " + mid + " " + line);
            if (line.compareTo(string) < 0)
                low = mid + 1;
            else
                high = mid;
        }

        p = low;
        while (p >= 0) {
```

```
            raf.seek(p);
            if (((char) raf.readByte()) == '\n')
                break;
            p--;
        }

        if (p < 0)
            raf.seek(0);

        while (true) {
            String line = raf.readLine();
            if (line == null || !line.startsWith(string))
                break;
            result.add(line);
        }

        raf.close();
    } catch (IOException e) {
        System.out.println("IOException:");
        e.printStackTrace();
    }
    return result;
}
```

answered Jun 15 '09 at 21:17

Andy

I am not aware of any library that has that functionality. However, a correct code for a external binary search in Java should be similar to this:

```
class ExternalBinarySearch {
final RandomAccessFile file;
final Comparator<String> test; // tests the element given as search parameter with
the line. Insert a PrefixComparator here
public ExternalBinarySearch(File f, Comparator<String> test) throws
FileNotFoundException {
    this.file = new RandomAccessFile(f, "r");
    this.test = test;
}
public String search(String element) throws IOException {
    long l = file.length();
    return search(element, -1, l-1);
}
/**
 * Searches the given element in the range [low,high]. The low value of -1 is a
special case to denote the beginning of a file.
 * In contrast to every other line, a line at the beginning of a file doesn't need
a \n directly before the line
 */
private String search(String element, long low, long high) throws IOException {
    if(high - low < 1024) {
        // search directly
        long p = low;
        while(p < high) {
            String line = nextLine(p);
            int r = test.compare(line,element);
            if(r > 0) {
                return null;
            } else if (r < 0) {
                p += line.length();
            } else {
                return line;
            }
        }
        return null;
    } else {
        long m  = low + ((high - low) / 2);
        String line = nextLine(m);
        int r = test.compare(line, element);
        if(r > 0) {
            return search(element, low, m);
        } else if (r < 0) {
            return search(element, m, high);
        } else {
            return line;
        }
    }
}
private String nextLine(long low) throws IOException {
    if(low == -1) { // Beginning of file
        file.seek(0);
    } else {
        file.seek(low);
    }
    int bufferLength = 65 * 1024;
    byte[] buffer = new byte[bufferLength];
    int r = file.read(buffer);
    int lineBeginIndex = -1;

    // search beginning of line
```

```
    if(low == -1) { //beginning of file
        lineBeginIndex = 0;
    } else {
        //normal mode
        for(int i = 0; i < 1024; i++) {
        if(buffer[i] == '\n') {
            lineBeginIndex = i + 1;
            break;
        }
        }
    }
    if(lineBeginIndex == -1) {
        // no line begins within next 1024 bytes
        return null;
    }
    int start = lineBeginIndex;
        for(int i = start; i < r; i++) {
            if(buffer[i] == '\n') {
                // Found end of line
                return new String(buffer, lineBeginIndex, i - lineBeginIndex + 1);
                return line.toString();
            }
        }
        throw new IllegalArgumentException("Line to long");
}
}
```

Please note: I made up this code ad-hoc: Corner cases are not tested nearly good enough, the code assumes that no single line is larger than 64K, etc.

I also think that building an index of the offsets where lines start might be a good idea. For a 500 GB file, that index should be stored in an index file. You should gain a not-so-small constant factor with that index because than there is no need to search for the next line in each step.

I know that was not the question, but building a prefix tree data structure like (Patrica) Tries (on disk/SSD) might be a good idea to do the prefix search.

answered Apr 10 '09 at 11:56

dmeister
20.2k 13 59 84

Thanks, I'll look into Patricia Tries (I don't yet see what a Trie would look like on-disk instead of in-memory) – sds Apr 10 '09 at 14:58

As for finding the beginning of a line, the original perl module just flushes partial lines with a readLine() after each seek. When you think about it, this doesn't interfere with the binary search itself. The text file has ~29x10^9 lines, so the index of byte offsets may itself get unwieldy fast. – sds Apr 10 '09 at 15:06

This is a simple example of what you want to achieve. I would probably first index the file, keeping track of the file position for each string. I'm assuming the strings are separated by newlines (or carriage returns):

```
RandomAccessFile file = new RandomAccessFile("filename.txt", "r");
List<Long> indexList = new ArrayList();
long pos = 0;
while (file.readLine() != null)
{
    Long linePos = new Long(pos);
    indexList.add(linePos);
    pos = file.getFilePointer();
}
int indexSize = indexList.size();
Long[] indexArray = new Long[indexSize];
indexList.toArray(indexArray);
```

The last step is to convert to an array for a slight speed improvement when doing lots of lookups. I would probably convert the `Long[]` to a `long[]` also, but I did not show that above. Finally the code to read the string from a given indexed position:

```
int i; // Initialize this appropriately for your algorithm.
file.seek(indexArray[i]);
String line = file.readLine();
        // At this point, line contains the string #i.
```

answered Apr 10 '09 at 3:06

Rick C. Petty
761 4 8

Are you going to have enough memory to keep the index List in memory? – Cem Catikkas Apr 10 '09 at 4:31

That depends upon the number of entries. One could always write the index out and use a LongBuffer, possibly mmap'd. – Rick C. Petty Apr 10 '09 at 4:43

It's a cool idea, but the text file is over 500GB, which pretty much rules this approach out. Anyways, even when you jump to the middle of some line with seek, subsequently calling a readLine() brings you to the nearest newline as well, adding little or no overhead. – sds Apr 10 '09 at 6:27

1 Just because the text file is huge doesn't imply that the index would be large, especially if each line is unique. Also, my method would not see into the middle of a line, you'd always seek to the start of the line you're interested in. – Rick C. Petty Apr 10 '09 at 18:13

If you are dealing with a 500GB file, then you might want to use a faster lookup method than binary search - namely a radix sort which is essentially a variant of hashing. The best method for doing this really depends on your data distributions and types of lookup, but if you are looking for string prefixes there should be a good way to do this.

I posted an example of a radix sort solution for integers, but you can use the same idea - basically to cut down the sort time by dividing the data into buckets, then using O(1) lookup to retrieve the bucket of data that is relevant.

```
Option Strict On
Option Explicit On

Module Module1

Private Const MAX_SIZE As Integer = 100000
Private m_input(MAX_SIZE) As Integer
Private m_table(MAX_SIZE) As List(Of Integer)
Private m_randomGen As New Random()
Private m_operations As Integer = 0

Private Sub generateData()
    ' fill with random numbers between 0 and MAX_SIZE - 1
    For i = 0 To MAX_SIZE - 1
        m_input(i) = m_randomGen.Next(0, MAX_SIZE - 1)
    Next

End Sub

Private Sub sortData()
    For i As Integer = 0 To MAX_SIZE - 1
        Dim x = m_input(i)
        If m_table(x) Is Nothing Then
            m_table(x) = New List(Of Integer)
        End If
        m_table(x).Add(x)
        ' clearly this is simply going to be MAX_SIZE -1
        m_operations = m_operations + 1
    Next
End Sub

 Private Sub printData(ByVal start As Integer, ByVal finish As Integer)
    If start < 0 Or start > MAX_SIZE - 1 Then
        Throw New Exception("printData - start out of range")
    End If
    If finish < 0 Or finish > MAX_SIZE - 1 Then
        Throw New Exception("printData - finish out of range")
    End If
    For i As Integer = start To finish
        If m_table(i) IsNot Nothing Then
            For Each x In m_table(i)
                Console.WriteLine(x)
            Next
        End If
    Next
End Sub

' run the entire sort, but just print out the first 100 for verification purposes
Private Sub test()
    m_operations = 0
    generateData()
    Console.WriteLine("Time started = " & Now.ToString())
    sortData()
    Console.WriteLine("Time finished = " & Now.ToString & " Number of operations =
" & m_operations.ToString())
    ' print out a random 100 segment from the sorted array
    Dim start As Integer = m_randomGen.Next(0, MAX_SIZE - 101)
    printData(start, start + 100)
End Sub

Sub Main()
    test()
    Console.ReadLine()
End Sub

End Module
```

edited Nov 30 '13 at 20:09

pts
33.2k 10 69 115

answered Jun 15 '09 at 21:42

Larry Watanabe
8,293 7 31 39

I had similar problem, so I created (Scala) library from solutions provided in this thread:

https://github.com/avast/BigMap

It contains utility for sorting huge file and binary search in this sorted file...

answered Dec 9 '14 at 18:20

Karry
121 1 2

I post a gist https://gist.github.com/mikee805/c6c2e6a35032a3ab74f643a1d0f8249c

that is rather complete example based on what I found on stack overflow and some blogs hopefully someone else can use it

```
import static java.nio.file.Files.isWritable;
import static java.nio.file.StandardOpenOption.READ;
import static org.apache.commons.io.FileUtils.forceMkdir;
import static org.apache.commons.io.IOUtils.closeQuietly;
import static org.apache.commons.lang3.StringUtils.isBlank;
import static org.apache.commons.lang3.StringUtils.trimToNull;

import java.io.File;
import java.io.IOException;
import java.nio.Buffer;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Path;

public class FileUtils {

    private FileUtils() {
    }

    private static boolean found(final String candidate, final String prefix) {
        return isBlank(candidate) || candidate.startsWith(prefix);
    }

    private static boolean before(final String candidate, final String prefix) {
        return prefix.compareTo(candidate.substring(0, prefix.length())) < 0;
    }

    public static MappedByteBuffer getMappedByteBuffer(final Path path) {
        FileChannel fileChannel = null;
        try {
            fileChannel = FileChannel.open(path, READ);
            return fileChannel.map(FileChannel.MapMode.READ_ONLY, 0,
fileChannel.size()).load();
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
        finally {
            closeQuietly(fileChannel);
        }
    }

    public static String binarySearch(final String prefix, final MappedByteBuffer
buffer) {
        if (buffer == null) {
            return null;
        }
        try {
            long low = 0;
            long high = buffer.limit();
            while (low < high) {
                int mid = (int) ((low + high) / 2);
                final String candidate = getLine(mid, buffer);
                if (found(candidate, prefix)) {
                    return trimToNull(candidate);
                }
                else if (before(candidate, prefix)) {
                    high = mid;
                }
                else {
                    low = mid + 1;
                }
            }
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
        return null;
    }

    private static String getLine(int position, final MappedByteBuffer buffer) {
        // search backwards to the find the proceeding new line
```

```
        // then search forwards again until the next new line
        // return the string in between
        final StringBuilder stringBuilder = new StringBuilder();
        // walk it back
        char candidate = (char)buffer.get(position);
        while (position > 0 && candidate != '\n') {
            candidate = (char)buffer.get(--position);
        }
        // we either are at the beginning of the file or a new line
        if (position == 0) {
            // we are at the beginning at the first char
            candidate = (char)buffer.get(position);
            stringBuilder.append(candidate);
        }
        // there is/are char(s) after new line / first char
        if (isInBuffer(buffer, position)) {
            //first char after new line
            candidate = (char)buffer.get(++position);
            stringBuilder.append(candidate);
            //walk it forward
            while (isInBuffer(buffer, position) && candidate != ('\n')) {
                candidate = (char)buffer.get(++position);
                stringBuilder.append(candidate);
            }
        }
        return stringBuilder.toString();
    }

    private static boolean isInBuffer(final Buffer buffer, int position) {
        return position + 1 < buffer.limit();
    }

    public static File getOrCreateDirectory(final String dirName) {
        final File directory = new File(dirName);
        try {
            forceMkdir(directory);
            isWritable(directory.toPath());
        }
        catch (IOException e) {
            throw new RuntimeException(e);
        }
        return directory;
    }
}
```

answered Sep 1 at 5:54

mikee805
35 6

If you truly want to try memory mapping the file, I found a tutorial on how to use memory mapping in Java nio.

answered Apr 10 '09 at 3:09

Eddie
41.9k 18 100 133
