Why are Lucene's stored fields so slow to access

Problem

I have a Lucene index that has some large fields (about 50 KB each) and some small fields (about 50 bytes each). I need to access (iterate) one of the small fields for say 1/10 of the documents. For some reason, such operation is very slow, unreasonably so for such a small field.

Cause

Lucene provides a number of “policies” of how to access fields of a document. (See class org.apache.lucene.document.FieldSelector.) They specify when and how fields are loaded from the index. It turns out that the default is to load all fields in the document as soon as a Document is requested by, say IndexReader. (See class org.apache.lucene.index.FieldsReader, in particular, how it implements the doc(n, FieldSelector) function.) Therefore, when you load a small field, the large fields are also loaded, causing performance problem if you repeat the operation many times.

Solution

The class org.apache.lucene.document.FieldSelectorResult provides several “policies” that you can use. The most interesting one w.r.t. our problem is FieldSelectorResult.LAZY_LOAD. It basically specifies that a field is lazily loaded (i.e. loaded only when needed).

To use this policy, create a FieldSelector object.

```
FieldSelector lazyFieldSelector = new FieldSelector() {□public FieldSelectorResult accept(String
```

When you request the document from an IndexReader, pass this object too.

```
IndexReader reader;...// Open the index reader...Document doc = reader.document(docId, lazyFieldS
```

Note that to get the field, use the Document's getFieldable(String) method instead of getField(String). This is according to the API reference.

```
Fieldable fieldable = document.getFieldable(fieldName);String value = fieldable.stringValue();//
```

Solution

Within a document, stored fields are read sequentially. (See Index File Formats.) In theory, accessing the first fields should be faster than reading the last ones.

Fields are ordered and their orders are stored implicity in the .fnm file. The order that the fields are read from should be the same as the order that you create the fields. To gain performance, create frequently used (and small) stored fields first.

Cause

For some reason, this is still slower than indexing the field and then iterate through all the terms in the field. I looked closer and found another bottleneck.

A Lucene index stores the lengths of the fields in terms of character count, not byte count; Also, a character can be more than a byte long. As we have seen, Lucene stores and processes the fields sequentially. Even if it does not load a field, it must read the whole content of a field to get to the next field. If a large field is not loaded but is before a small field that is loaded, the processing time depends on the length of both fields, not just the small ones.

Solution

The problem will not happen if the field you need to iterate is placed before the large fields, and if you ask the FieldSelector to stop at the field you want.

Say you want to iterate only field "field1". Then create a FieldSelector that only loads field1 and stops at this field. When creating the index, remember to put the large fields after field1.

```
String fieldToIterate = "field1";...FieldSelector lazyFieldSelector = new FieldSelector() {□publ
```

The rest of the code should be the same.

Revision #2
Created Thu, Sep 11, 2008 3:18 PM by Chan, Wing Kai

Updated Thu, Sep 18, 2008 6:20 PM by Chan, Wing Kai