Improving Bug Location Using Binary Class Relationships

Nasir Ali, Aminata Sabane, Yann-Gaël Guéhéneuc, and Giuliano Antoniol
What is bug location?

A modest proposal for updateable fields

Details
Type: New Feature
Priority: Major
Affects Version/s: 4.0-ALPHA
Component/s: cora/index
Labels: None
Lucene Fields: New

Description
I'd like to propose a simple design for implementing updateable fields in Lucene. This design has some limitations, so I'm not claiming it will be appropriate for every use case, and it's obvious it has some performance consequences, but at least it's a start...

This proposal uses a concept of "overlays" or "stacked updates", where the original data is not removed but instead it's overlaid with the new data. I propose to reuse as much of the existing APIs as possible, and represent updates as an IndexReader. Updates to documents in a specific segment would be collected in an "overlay" index specific to that segment, i.e. there would be as many overlay indexes as there are segments in the primary index.

A field update would be represented as a new document in the overlay index. The document would consist of just the updated fields, plus a field that records the id in the primary segment of the document affected by the update. These updates would be processed as usual via secondary IndexWriters, as many as there are primary segments, so the same analysis chains would be used, the same field types, etc.

SOURCE CODE

SCAM 201
How to find a buggy class(es)?

- FeedFactory.java
- RssProtocolProviderServiceLick.java
- RssSlickFixture.java
- ScTestRunner.java
- SipCommunicatorSlickRunner.java
- SipProtocolProviderServiceLick.java
- SipSlickFixture.java
- SlicklessTests.java
- TestAccountInstallation.java
- TestAccountInstallation.java
- TestAccountInstallation.java
How to find a buggy class(es)?

Files:
- FeedFactory.java
- RssProtocolProvider.java
- SipCommunicator.java
- SipProtocolProcessor.java
- SipSlickFixture.java
- SlicklessTests.java
- TestAccountInstallation.java

Proposal for updateable fields:

For implementing updateable fields in Lucene. This design claiming it will be appropriate for every use case, and it’s consequences, but at least it’s a start.

“overlays” or “stretched updates”, where the original data is not with the new data. I propose to reuse as much of the existing updates as an IndexReader. Updates to documents in a tied in an “overlay” index specific to that segment, i.e. there is as there are segments in the primary index.

Tied as a new document in the overlay index. The document fields, plus a field that records the id in the primary segment update. These updates would be processed as usual, as long as there are primary segments, so the same analysis field types, etc.
Problem with IR techniques

IR Techniques
Searching!

“Bugs” Bunny

Bug

LSI

Developer
Our Conjecture!

Send Email Feature

Binary Class Relationships (BCR)

"Good" Developer
Our Conjecture!

```
package org.nuxeo.upcoming;

import java.text.SimpleDateFormat;

public class DocumentCreationListener implements EventListener {
  // really just for the test code
  public static final String HANDLED_EVENT = "Did we handle the event?";

  public static final SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault());

  public DocumentCreationListener() {
    // 
    try {
     -handler(event) throws ClientException {
        // this is really just for the test to know we handled event
        if (event.isHandleEvent) {
          event.getContext().put("HANDLED_EVENT", "true");
          // check what type of event context
          if (!(ctx instanceof DocumentEventContext)) {
            return;
          }
        }
      }
    }
  }

  // get the event context for a document event
  DocumentEventContext context = event.event.getEvent();
```
Proposed Approach
Linguistic (textual) and BCRs of Object Oriented Systems (LIBCROOS)
IR Engine

- **Input:** Bug reports and source code
  - Filter (#43@$)
  - Stop words (the, is, an....)
  - Stemmer (attachment, attached -> attach)

- **IR Techniques**
  - Latent Semantic Indexing (LSI)
  - Vector Space Model (VSM)

- **Output**
  - Ranked list of linked bug reports and source code files based on textual similarity
Example of IR ranked list

- FeedFactory.java
  - 0.70
- RssProtocolProviderServiceLick.java
  - 0.65
- RssSlickFixture.java
  - 0.59
- ScTestRunner.java
  - 0.56
- SipCommunicatorSlickRunner.java
  - 0.35
- SipProtocolProviderServiceLick.java
  - 0.34
- SipSlickFixture.java
  - 0.27
- SlicklessTests.java
  - 0.19
- TestAccountInstallation.java
  - 0.12
- TestAccountInstallation.java
  - 0.09
- TestAccountInstallation.java
  - 0.06

A modest proposal for updateable fields

**Details**

- **Type:** New Feature
- **Priority:** Major
- **Resolution:** Unresolved
- **Affects Version/s:** 4.0-ALPHA
- **Fix Version/s:** None
- **Component/s:** core/index
- **Labels:** None
- **Lucene Field/s:** New

**Description**

I’d like to propose a simple design for implementing updateable fields in Lucene. This design has some limitations, so I’m not claiming it will be appropriate for every use case, and it’s obvious it has some performance consequences, but at least it’s a start.

This proposal uses a concept of “overlays” or “stashed updates”, where the original data is not removed but instead it’s overlaid with the new data. I propose to reuse as much of the existing APIs as possible, and represent updates as an IndexReader. Updates to documents in a specific segment would be collected in an “overlay” index specific to that segment, i.e. there would be as many overlay indexes as there are segments in the primary index.

A field update would be represented as a new document in the overlay index. The document would consist of just the updated fields, plus a field that records the id in the primary segment of the document affected by the update. These updates would be processed as usual via secondary IndexWriter, as many as there are primary segments, so the same analysis chains would be used, the same field types, etc.
Relational Model

• **Input**
  - IR generated ranked list
  - Source code OR binary classes

• **Binary Class Relationships**
  - PADL model creator
  - BCRs recovery
  - BCRs filter

• **Output**
  - Ranked list of linked bug reports and source code files based on BCRs
Example of BCRs

A modest proposal for updateable fields

Details
Type: New Feature
Priority: Major
Affects Version: 4.0-ALPHA
Components: core/index
Labels: None
Lucene Fields: New
Status: Open
Resolution: Unresolved
Fix Version: None

Description
I'd like to propose a simple design for implementing updateable fields in Lucene. This design does not claim it will be appropriate for every use case, and it's nice consequences, but at least it's a start.

The idea of "overlays" or "stamped updates", where the original data is not removed but instead is overlaid with the new data. I propose to reuse as much of the existing APIs as possible, and represent updates as an IndexReader. Updates to documents in a specific segment would be collected in an "overlay" index specific to that segment, i.e., there would be as many overlay indexes as there are segments in the primary index.

A field update would be represented as a new document in the overlay index. The document would consist of just the updated fields, plus a field that records the id in the primary segment of the document affected by the update. These updates would be processed as usual via secondary IndexWriter, as many as there are primary segments, so the same analytics chains would be used, the same field types, etc.
Ranker

• **Input**
  - IR generated ranked list
  - Relational model generated ranked list

• **Assign weights**
  - Assign weight to IR ranked list
  - Assign weight to relational model generated ranked list

• **Output**
  - New ranked list
Example of Ranker

A modest proposal for updateable fields

Log In

Details

Type: New Feature
Status: Open
Priority: Major
Affects Versions: 4.0-ALPHA
Fix Version/fix: None
Component/s: core/index
Labels: None
Lucene Fields: New

Description

I'd like to propose a simple design for implementing updateable fields in Lucene. This design

not claiming it will be appropriate for every use case, and it's

ice consequences, but at least it's a start...

of "overlays" or "stacked updates", where the original data is not
removed but instead it's overlaid with the new data. I propose to reuse as much of the existing
APIs as possible, and represent updates as an IndexReader. Updates to documents in a
specific segment would be collected in an "overlay" index specific to that segment, i.e. there
would be as many overlay indexes as there are segments in the primary index.

A field update would be represented as a new document in the overlay index. The document
would consist of just the updated fields, plus a field that records the id in the primary segment
of the document affected by the update. These updates would be processed as usual via
secondary IndexWriters, as many as there are primary segments, so the same analysis
chains would be used, the same field types, etc.
Case Study

• **Goal**: Investigate the effectiveness of LIBCROOS in improving the accuracy of VSM and LSI techniques

• **Quality focus**: Ability to decrease the rank of buggy classes in ranked list produced by IR techniques

• **Context**: Linking bug reports with source code of four open-source programs, ......
Research Questions (RQs)

R_{01}: Does LIBCROOS provide better accuracy, in terms of ranking, than IR techniques?

R_{02}: What are the important BCRs that help to improve IR techniques accuracy more than the others?
## Datasets

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Version</th>
<th>Bug Reports</th>
<th>Source Code Files</th>
<th>Classes</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rhino</td>
<td>1.5R4.1</td>
<td>41</td>
<td>111</td>
<td>111</td>
<td>94,078</td>
</tr>
<tr>
<td>Jabref</td>
<td>2.6</td>
<td>36</td>
<td>579</td>
<td>579</td>
<td>287,791</td>
</tr>
<tr>
<td>Lucene</td>
<td>3.1</td>
<td>89</td>
<td>434</td>
<td>434</td>
<td>111,117</td>
</tr>
<tr>
<td>muCommander</td>
<td>0.8.5</td>
<td>81</td>
<td>1,069</td>
<td>1,069</td>
<td>124,944</td>
</tr>
</tbody>
</table>
Case Studies Results (RQ1)

Y axis shows buggy classes’ rank and X axis shows approaches
Case Studies Results

<table>
<thead>
<tr>
<th>Relationship</th>
<th>LSI</th>
<th>LIBC.</th>
<th>Median</th>
<th>LSI</th>
<th>LIBC.</th>
<th>SD</th>
<th>VSM</th>
<th>LIBC.</th>
<th>Median</th>
<th>VSM</th>
<th>LIBC.</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>22.19</td>
<td>6.47</td>
<td>8</td>
<td>2.5</td>
<td>8</td>
<td>32.86</td>
<td>7.39</td>
<td>16.14</td>
<td>2.5</td>
<td>6.5</td>
<td>2</td>
<td>23.30</td>
</tr>
<tr>
<td>Aggregation</td>
<td>22.19</td>
<td>14.86</td>
<td>6</td>
<td>2.5</td>
<td>6</td>
<td>32.86</td>
<td>20.64</td>
<td>16.14</td>
<td>7.97</td>
<td>6.5</td>
<td>4</td>
<td>23.30</td>
</tr>
<tr>
<td>Association</td>
<td>22.19</td>
<td>19.92</td>
<td>8</td>
<td>2.5</td>
<td>8</td>
<td>32.86</td>
<td>28.47</td>
<td>16.14</td>
<td>13.5</td>
<td>6.5</td>
<td>6.5</td>
<td>23.30</td>
</tr>
<tr>
<td>Inheritance</td>
<td>22.19</td>
<td>10.72</td>
<td>4</td>
<td>2.5</td>
<td>4</td>
<td>32.86</td>
<td>15.33</td>
<td>16.14</td>
<td>5.75</td>
<td>6.5</td>
<td>3</td>
<td>23.30</td>
</tr>
<tr>
<td>Use</td>
<td>22.19</td>
<td>19.56</td>
<td>6</td>
<td>2.5</td>
<td>6</td>
<td>32.86</td>
<td>27.997</td>
<td>16.14</td>
<td>12.97</td>
<td>6.5</td>
<td>6.5</td>
<td>23.30</td>
</tr>
<tr>
<td>ALL</td>
<td>26.22</td>
<td>9.66</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>43.11</td>
<td>15.16</td>
<td>24.90</td>
<td>9.40</td>
<td>6</td>
<td>2</td>
<td>46.08</td>
</tr>
<tr>
<td>Aggregation</td>
<td>26.22</td>
<td>16.42</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>43.11</td>
<td>25.47</td>
<td>24.90</td>
<td>15.47</td>
<td>6</td>
<td>5</td>
<td>46.08</td>
</tr>
<tr>
<td>Association</td>
<td>26.22</td>
<td>24.90</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>43.11</td>
<td>40.50</td>
<td>24.90</td>
<td>23.88</td>
<td>6</td>
<td>6</td>
<td>46.08</td>
</tr>
<tr>
<td>Inheritance</td>
<td>26.22</td>
<td>15.85</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>43.11</td>
<td>25.45</td>
<td>24.90</td>
<td>14.87</td>
<td>6</td>
<td>3.5</td>
<td>46.08</td>
</tr>
<tr>
<td>Use</td>
<td>26.22</td>
<td>23.46</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>43.11</td>
<td>37.21</td>
<td>24.90</td>
<td>22.12</td>
<td>6</td>
<td>6</td>
<td>46.08</td>
</tr>
<tr>
<td>muCommander</td>
<td>36.72</td>
<td>11.59</td>
<td>7.5</td>
<td>3.5</td>
<td>7.5</td>
<td>89.33</td>
<td>24.96</td>
<td>39.88</td>
<td>13.23</td>
<td>9.5</td>
<td>3.5</td>
<td>105.20</td>
</tr>
<tr>
<td>Aggregation</td>
<td>36.72</td>
<td>19.54</td>
<td>7.5</td>
<td>4.5</td>
<td>7.5</td>
<td>89.33</td>
<td>43.76</td>
<td>39.88</td>
<td>20.37</td>
<td>9.5</td>
<td>5.5</td>
<td>105.20</td>
</tr>
<tr>
<td>Association</td>
<td>36.72</td>
<td>31.35</td>
<td>7.5</td>
<td>6.5</td>
<td>7.5</td>
<td>89.33</td>
<td>73.59</td>
<td>39.88</td>
<td>34.05</td>
<td>9.5</td>
<td>7.5</td>
<td>105.20</td>
</tr>
<tr>
<td>Inheritance</td>
<td>36.72</td>
<td>20.08</td>
<td>7.5</td>
<td>5.5</td>
<td>7.5</td>
<td>89.33</td>
<td>46.26</td>
<td>39.88</td>
<td>22.21</td>
<td>9.5</td>
<td>5</td>
<td>105.20</td>
</tr>
<tr>
<td>Use</td>
<td>36.72</td>
<td>29.94</td>
<td>7.5</td>
<td>6.5</td>
<td>7.5</td>
<td>89.33</td>
<td>69.59</td>
<td>39.88</td>
<td>32.69</td>
<td>9.5</td>
<td>8</td>
<td>105.20</td>
</tr>
<tr>
<td>Rhino</td>
<td>11.34</td>
<td>3.25</td>
<td>4.5</td>
<td>1</td>
<td>4.5</td>
<td>17.13</td>
<td>4.36</td>
<td>9.47</td>
<td>2.78</td>
<td>3</td>
<td>1</td>
<td>16.45</td>
</tr>
<tr>
<td>Aggregation</td>
<td>11.34</td>
<td>6.97</td>
<td>4.5</td>
<td>3</td>
<td>4.5</td>
<td>17.13</td>
<td>9.35</td>
<td>9.47</td>
<td>3.47</td>
<td>3</td>
<td>2</td>
<td>16.45</td>
</tr>
<tr>
<td>Association</td>
<td>11.34</td>
<td>9.41</td>
<td>4.5</td>
<td>3.5</td>
<td>4.5</td>
<td>17.13</td>
<td>14.66</td>
<td>9.47</td>
<td>7.72</td>
<td>3</td>
<td>3.5</td>
<td>16.45</td>
</tr>
<tr>
<td>Inheritance</td>
<td>11.34</td>
<td>6.31</td>
<td>4.5</td>
<td>2.5</td>
<td>4.5</td>
<td>17.13</td>
<td>9.61</td>
<td>9.47</td>
<td>4.5</td>
<td>3</td>
<td>2</td>
<td>16.45</td>
</tr>
<tr>
<td>Use</td>
<td>11.34</td>
<td>9.19</td>
<td>4.5</td>
<td>3.5</td>
<td>4.5</td>
<td>17.13</td>
<td>14.53</td>
<td>9.47</td>
<td>7.72</td>
<td>3</td>
<td>2.5</td>
<td>16.45</td>
</tr>
</tbody>
</table>

Table I

Descriptive statistics LIBCROOS, LSI, and VSM. LIBC. and SD represent LIBCROOS and standard deviation respectively.
RQs Answers

- **R_{01}**: LIBCROOS helps to decrease the rank of culprit classes and put culprit classes higher in the ranked lists when compared to “traditional” IR techniques alone.

- **R_{02}**: All BCRs helps to decrease the rank of culprit classes. However, inheritance is the most important BCR to decrease the ranking, then aggregation, use, and association.