Lucene Java 2.9: Numeric Search, Per-Segment Search, Near-Real-Time Search, and the new TokenStream API

Uwe Schindler
Lucene Java Committer
uschindler@apache.org

PANGAEA® - Publishing Network for Geoscientific & Environmental Data (www.pangaea.de)
MARUM, Center for Marine Environmental Sciences, Bremen, Germany (www.marum.de)
New features in Lucene Java 2.9

- Lucene now includes high-performance handling of numeric fields. Such fields are indexed with a trie structure, enabling simple to use and much faster numeric range searching without having to externally pre-process numeric values into textual values.
- Smarter, more scalable multi-term queries (wildcard, range, etc)
- Per segment searching and caching (can lead to much faster reopen among other things)
- A freshly optimized Collector/Scorer API
- Scoring is now optional when sorting by field, or using a custom Collector, gaining sizable performance when scores are not required
- Near real-time search capabilities added to IndexWriter
- A new Attribute based TokenStream API
- A new QueryParser framework in contrib with a core QueryParser replacement impl included
- New Query types
- Improved Unicode support and the addition of Collation contrib
- New analyzers (PersianAnalyzer, ArabicAnalyzer, SmartChineseAnalyzer)
- New fast-vector-highlighter

Source: Release notes of Lucene Java 2.9
Numeric Fields
Problems with 2.4’s RangeQueries/-Filters

- Classical **RangeQuery** hits TooManyClausesException on large ranges and is very slow.
- **ConstantScoreRangeQuery** is faster, cacheable, but still has to visit a large number of terms.
- Both need to enumerate a large number of terms from **TermEnum** and then retrieve **TermDocs** for each term.
- The number of terms to visit grows with number of documents and unique values in index (especially for float/double values)
TrieRange: How it works

```
    4
   / \   /
  42  44 445 446 448 521 522 632 633 634 641 642 644
```

range
Supported Data Types

• **Native data type:** `long, int` (standard Java signed). All “tricks” like padding are **not needed**! These types are internally made unsigned, each trie precision is generated by stripping off least significant bits (using `precisionStep` parameter). Each value is then converted to a sequence of 7bit ASCII chars, result is prefixed with the number of bits stripped, and indexed as term. Only 7 bits/char are used because of most efficient bit layout in index (8 or more bits would split into two or more bytes when UTF-8 encoded).

• **double, float:** Converter to/from IEEE-754 bit layout that sorts like a signed `long/int`

• **Date/Calendar:** Convert to UNIX time stamp with e.g. `Date.getTime()`
Speed

- Upper limit on number of terms, independent of index size. This value depends only on precision.

- **Term numbers**: 8 bit approx. 400 terms, 4 bit approx. 100 terms, 2 bit approx. 40 terms

- **Query time**: in most cases <100 ms with 1,000,000 docs index, 13 numeric fields, precision 8 bit
How to use (indexing)

- New convenience class **NumericField** that optionally also stores the numeric value as string. Provides various setters for different data types.

- The work is done by **NumericTokenStream**, which “tokenizes” the number into the binary encoded trie terms.

```java
Directory directory = new RAMDirectory();
Analyzer analyzer = new WhitespaceAnalyzer();
IndexWriter writer = new IndexWriter(directory, analyzer,
    IndexWriter.MaxFieldLength.UNLIMITED);
for (int i = 0; i < 20000; i++) {
    Document doc = new Document();
    doc.add(new Field("id", String.valueOf(i),
        Field.Store.YES, Field.Index.NOT_ANALYZED_NO_NORMS));
    doc.add(new NumericField("newNumeric", 4,
        Field.Store.YES, true).setIntValue(i));
    writer.addDocument(doc);
}
writer.close();
```
How to use (searching)

- New classes: **NumericRangeQuery**, **NumericRangeFilter** with “static” ctors per data type.
- Old RangeQuery & co. is deprecated and replaced by **TermRangeQuery**
- 4 different modes for **MultiTermQueries**: Conventional with scoring (not recommended), constant score with Filter or BooleanQuery, automatic constant score dependent on term count. WildcardQuery, PrefixQuery and FuzzyQuery are MTQs since 2.9, too.

```java
IndexSearcher searcher = new IndexSearcher(directory, true);
Query query = NumericRangeQuery.newIntRange("newNumeric", 4, 10, 10000, true, false);
TopDocs docs = searcher.search(query, null, 10);
assertNotNull("Docs is null", docs);
assertEquals(9990, docs.totalHits);
for (int i = 0; i < docs.scoreDocs.length; i++) {
    ScoreDocs sd = docs.scoreDocs[i];
    assertTrue(sd.doc >= 10 && sd.doc < 10000);
}
```
NumericField and FieldCache

• **NumericFields** can be loaded into **FieldCache** and will be used for sorting.
• **FieldCache.AUTO / SortField.AUTO** deprecated.
• New range filter implementation based completely on using the FieldCache: **FieldCacheRangeFilter**. Similar API like NumericRangeFilter, but also supports string(index) fields.
Per-Segment Search
Segments in Lucene

- Each index consists of various segments placed in the index directory. All documents are added to new in-RAM segment files, merged to on-disk files after flushing (each document is initially one segment!).
- Lucene writes segments incrementally and then can merge them.
- Optimized index consists of one segment.
- `IndexReader.reopen()` adds new/changed segments after commit to segments of an already existing `IndexReader` (lower I/O cost in contrast to re-opening the whole `IndexReader`).
Problems

- **FieldCache** used for sorting is keyed against the IndexReader instance.
- After reopen the whole FieldCache is invalid and needs to be reloaded.
- Long “warming” time for sorted queries (and also function queries in Solr) after reopen.
What has changed?

- **IndexSearcher** now works directly on segments (cf. MultiSearcher), results are merged by **Collectors** (TopDocsCollector,…). Non-expert API stays unchanged.

- **FieldCache** therefore also works on segments ⇒ sorting warmup after reopening IndexReaders is much faster, as only FieldCaches for new/changed segments have to be rebuilt.

- Scoring decoupled from **Collector**.

<table>
<thead>
<tr>
<th>Remote Name</th>
<th>Size</th>
<th>Type</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>d2b.cfs</td>
<td>5804,717</td>
<td>CFS-Datei</td>
<td>20.09.2009 01:51:25</td>
</tr>
<tr>
<td>_d2b_2p.del</td>
<td>77,843</td>
<td>DEL-Datei</td>
<td>25.09.2009 23:08:06</td>
</tr>
<tr>
<td>d4h.cfs</td>
<td>24079,695</td>
<td>CFS-Datei</td>
<td>23.09.2009 15:48:38</td>
</tr>
<tr>
<td>_d4h_c.del</td>
<td>268</td>
<td>DEL-Datei</td>
<td>25.09.2009 16:08:26</td>
</tr>
<tr>
<td>d4I.cfs</td>
<td>37710,376</td>
<td>CFS-Datei</td>
<td>23.09.2009 15:49:42</td>
</tr>
<tr>
<td>_d4w.cfs</td>
<td>66912,228</td>
<td>CFS-Datei</td>
<td>23.09.2009 16:10:36</td>
</tr>
<tr>
<td>_d4w_e.del</td>
<td>1,259</td>
<td>DEL-Datei</td>
<td>24.09.2009 09:48:22</td>
</tr>
<tr>
<td>d57.cfs</td>
<td>41917,094</td>
<td>CFS-Datei</td>
<td>23.09.2009 19:49:04</td>
</tr>
<tr>
<td>d5l.cfs</td>
<td>63936,632</td>
<td>CFS-Datei</td>
<td>23.09.2009 20:09:12</td>
</tr>
<tr>
<td>_d5l_2.del</td>
<td>17</td>
<td>DEL-Datei</td>
<td>24.09.2009 11:28:30</td>
</tr>
<tr>
<td>_d5u.cfs</td>
<td>40342,241</td>
<td>CFS-Datei</td>
<td>24.09.2009 09:48:24</td>
</tr>
<tr>
<td>_d5u_5.del</td>
<td>708</td>
<td>DEL-Datei</td>
<td>25.09.2009 03:26:22</td>
</tr>
<tr>
<td>_d64.cfs</td>
<td>21268,388</td>
<td>CFS-Datei</td>
<td>24.09.2009 12:48:19</td>
</tr>
<tr>
<td>_d64_2.del</td>
<td>246</td>
<td>DEL-Datei</td>
<td>25.09.2009 03:28:22</td>
</tr>
<tr>
<td>d62.cfs</td>
<td>118650</td>
<td>CFS-Datei</td>
<td>25.09.2009 14:28:05</td>
</tr>
<tr>
<td>d62_1.del</td>
<td>10</td>
<td>DEL-Datei</td>
<td>25.09.2009 14:48:07</td>
</tr>
<tr>
<td>d70.cfs</td>
<td>61466</td>
<td>CFS-Datei</td>
<td>25.09.2009 14:48:05</td>
</tr>
<tr>
<td>_d70_1.del</td>
<td>9</td>
<td>DEL-Datei</td>
<td>25.09.2009 15:08:08</td>
</tr>
<tr>
<td>_d71.cfs</td>
<td>83393</td>
<td>CFS-Datei</td>
<td>25.09.2009 15:08:06</td>
</tr>
<tr>
<td>_d72.cfs</td>
<td>62401</td>
<td>CFS-Datei</td>
<td>25.09.2009 15:28:05</td>
</tr>
<tr>
<td>_d73.cfs</td>
<td>413404</td>
<td>CFS-Datei</td>
<td>25.09.2009 15:48:09</td>
</tr>
<tr>
<td>_d73_1.del</td>
<td>13</td>
<td>DEL-Datei</td>
<td>25.09.2009 16:08:26</td>
</tr>
<tr>
<td>d74.cfs</td>
<td>4737351</td>
<td>CFS-Datei</td>
<td>25.09.2009 16:08:24</td>
</tr>
<tr>
<td>d75.cfs</td>
<td>11427390</td>
<td>CFS-Datei</td>
<td>25.09.2009 16:08:28</td>
</tr>
<tr>
<td>_d76.cfs</td>
<td>275541</td>
<td>CFS-Datei</td>
<td>25.09.2009 23:08:07</td>
</tr>
<tr>
<td>segments.gen</td>
<td>20</td>
<td>GEN-Datei</td>
<td>25.09.2009 23:08:06</td>
</tr>
<tr>
<td>segments_4w7</td>
<td>3632</td>
<td>Datei</td>
<td>25.09.2009 23:08:08</td>
</tr>
</tbody>
</table>
New Collector class

• Replacement for HitCollector.
• Gets notification about IndexReader change (together with new document ID base). This method can be used to change FieldCache arrays used during collecting to new IR.
  • collect() method gets document ID from current reader. It may map it by adding the current base to get a global ID.
  • collect() method no longer gets a score. It gets a notification about change of the underlying Scorer instance and can call Scorer.score() if needed. This can be used to skip scoring for queries that don’t care about score.
• Old HitCollectors can be wrapped by a special HitCollectorWrapper (they get called with rebased doc IDs and wrapper calls Scorer.score() for each hit).
Near-Real-Time Search
NRT additions

- Directly get an `IndexReader` from `IndexWriter` containing also uncommitted (in-memory) changes: `IndexWriter.getReader()`
- Supports `reopen()` as usual.
- Callback for warming merged segments: `IndexWriter.setMergedSegmentWarmer()`
New Attribute-based
TokenStream API
Tokenizers, TokenFilters, TokenStreams

• **TokenStream** is base class for **Tokenizer** and **TokenFilter**
• Decorator pattern (**TokenFilter** adds functionality to a **Tokenizer**)
• Implementation part of each **Tokenizer** / **TokenFilter** should be final
• **Lucene 2.4**: **Token** class holds all **attributes** of a token: term, position increment, start/end offset, type and flags (e.g. **part of speech information passed between TokenFilters**), payload
Lucene 2.9: Attributes instead of Tokens

- Introduces stronger typing and arbitrary attributes into the analysis process
- Easier to code custom TokenStreams by focusing only on needed attributes
- Helps set Lucene up for more flexible indexing options in the near future (LUCENE-1458)
- **Downside:** Some extra work transitioning your existing TokenStreams for 3.0:
  \[
  \text{next}(\text{Token}) \Rightarrow \text{incrementToken()}
  \]
public final class LengthFilter extends TokenFilter {

    private final int min;
    private final int max;

    public LengthFilter(TokenStream in, int min, int max) {
        super(in);
        this.min = min;
        this.max = max;
    }

    public Token next(final Token reusableToken) throws IOException {
        for (Token nextToken = input.next(reusableToken);
            nextToken != null; nextToken = input.next(reusableToken)) {
            int len = nextToken.termLength();
            if (len >= min && len <= max) {
                return nextToken;
            }
        }
        return null;
    }
}
public final class LengthFilter extends TokenFilter {

    private final int min;
    private final int max;

    public LengthFilter(TokenStream in, int min, int max) {
        super(in);
        this.min = min;
        this.max = max;
    }

    public Token next(final Token nextToken) throws IOException {
        for (Token nextToken = input.next(nextToken); nextToken != null;
             nextToken = input.next(nextToken)) {
            int len = nextToken.termLength();
            if (len >= min && len <= max) {
                return nextToken;
            }
        }
        return null;
    }
}

public final class LengthFilter extends TokenFilter {

    private final int min;
    private final int max;

    private final TermAttribute termAtt;

    public LengthFilter(TokenStream in, int min, int max) {
        super(in);
        this.min = min;
        this.max = max;
        termAtt = (TermAttribute) addAttribute(TermAttribute.class);
    }

    public boolean incrementToken() throws IOException {
        while (input.incrementToken()) {
            int len = termAtt.termLength();
            if (len >= min && len <= max) {
                return true;
            }
        }
        return false;
    }
}
public final class LengthFilter extends TokenFilter {

    private final int min;
    private final int max;
    private final TermAttribute termAtt;

    public LengthFilter(TokenStream in, int min, int max) {
        super(in);
        this.min = min;
        this.max = max;
        termAtt = (TermAttribute) addAttribute(TermAttribute.class);
    }

    public boolean incrementToken() throws IOException {
        while (input.incrementToken()) {
            int len = termAtt.termLength();
            if (len >= min && len <= max) {
                return true;
            }
        }
        return false;
    }

    public Token next(final Token reusableToken) throws IOException {
        for (Token nextToken = input.next(reusableToken);
            nextToken != null;
            nextToken = input.next(reusableToken)) {
            int len = nextToken.termLength();
            if (len >= min && len <= max) {
                return nextToken;
            }
        }
        return null;
    }
}
Websites

• NumericRangeQuery example: www.pangaea.de
  (PANGAEA® - Publishing Network for Geoscientific & Environmental Data)

• PANGAEA Framework for Metadata Portals: www.panFMP.org

• Lucene Java 2.9.0: lucene.apache.org/java/docs/
Happy coding with **Lucene 2.9.0**!

Thank You!