

APACHE
LUCENE
EUROCON



Improved Search with Lucene 4.0

Robert Muir, Lucid Imagination
robert.muir@lucidimagination.com, 10/19/2011

Presented by

lucid
IMAGINATION



Introductions

- What: Examine improvements in 4.0
- Who am I?
 - **Lucene Committer & PMC Member**
 - **Lucid Imagination Employee**
- Many big changes coming!
- Examine three general areas:
 - **Indexing Improvements**
 - **Search Improvements**
 - **Performance Improvements**
- Future improvements

Indexing Improvements

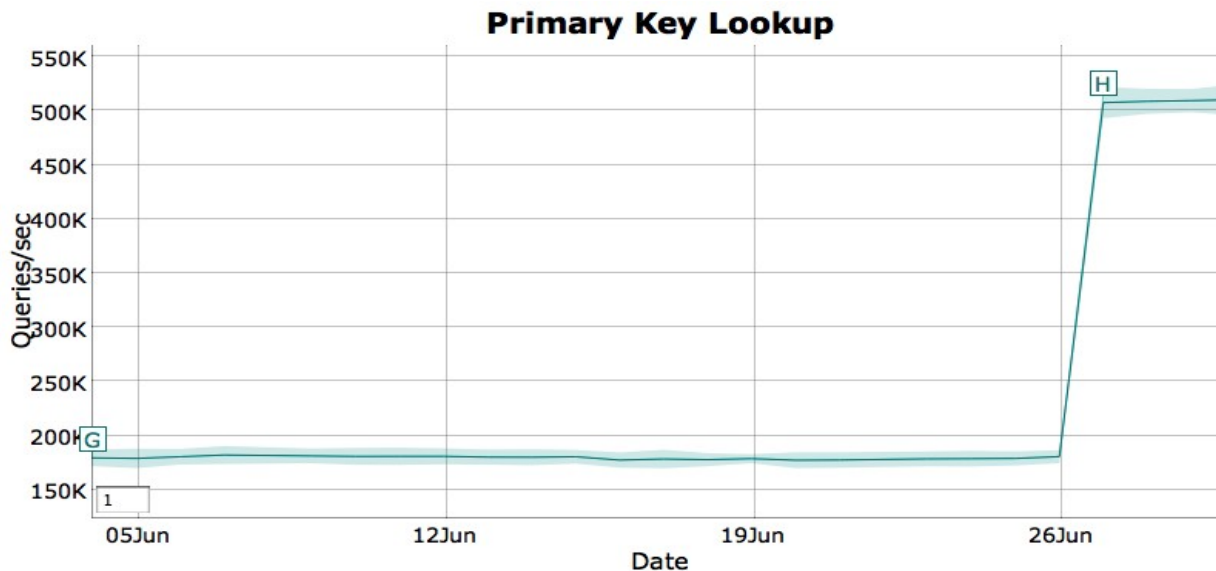
- **Codecs**
 - **Flexible index format**
- **Index DocValues**
 - **Efficient per-document lookup values**
- **Miscellaneous Improvements**
 - **Binary terms**
 - **Additional index statistics**

Codecs: Introduction

- Problem: “one size fits all” index format
- How to integrate future improvements?
 - **Example: more efficient index compression**
 - **But need to support old format, too.**
- How to optimize for your data?
 - **Without digging into the guts of indexer!**
- Idea: format should be pluggable

Codecs: Example use case

- Accelerate near-realtime reopen
- Speed up delete by term on ID field
 - “Primary key lookup”
- Idea: optimize ID field for this use case



Codecs: available formats

- Standard: Lucene 4.0 index format
- Pulsing: inline low frequency terms' postings
- Memory: loads field entirely into RAM
- Appending: supports append-only filesystem
- SimpleText: plaintext (not for production!!!!)
- PreFlex: supports Lucene 3.x index format

Codecs: Configuration

Lucene: use *CodecProvider* class

Solr: specify in `schema.xml`

```
<fieldType name="text_general" class="solr.TextField"  
  codec="SimpleText">
```

Codecs: Configuration

```
Terminal — vim — 61x14
field features
  term 1
    doc 20
      freq 1
      pos 12
  term 1,200
    doc 22
      freq 1
      pos 112
  term 1.35ghz
    doc 26
      freq 1
      pos 114
"solr/data/index/_0_1.pst" 4272L, 53640C
```

Index DocValues: Introduction

- Problem: lookup a per-document value
 - RAM-resident for things like scoring factors (e.g. pagerank)
 - Disk-resident for things like document versioning
- Existing workarounds in Lucene have limitations
 - Scoring limited to one byte norm
 - FieldCache requires uninversion to build
 - Stored fields are slow, geared towards summary results
- Idea: add performant, flexible per-document lookups.

Index DocValues: Limitations

- New feature, recently added to Lucene
- ✓ External scoring factors
- ✓ Memory-resident and disk-based lookup
- ✗ Docvalues sort-by-term
- ✗ Internal scoring factors (die norms, die)
- ✗ Integration with Solr

Index DocValues: Example

```
IndexDocValuesField dvField = new IndexDocValuesField("foo_boost");
doc.add(dvField);
dvField.setFloat(2f); // boost x2

...
final Source values = context.reader
    .docValues("foo_boost").getSource();

return new ExactDocScorer() {
    @Override
    public float score(int doc, int freq) {
        return (float) values.getFloat(doc) * sub.score(doc, freq);
    }
}
...
```

Binary Terms

- In Lucene 4.0, index terms are byte[]
 - **Do not need to be unicode text**
- Example: Localized Sort and Range
 - **Previous versions of Lucene: special encoder**
 - **Lucene 4.0: 50% space savings**

Localized Sort/Range Example

Lucene: use *CollationAnalyzer* class

Solr: specify in schema.xml

```
<fieldtype name="sort_de" class="solr.CollationField"  
  language="de" />
```

Additional Index Statistics

- New statistics in Lucene 4.0:
 - **totalTermFreq(term):** number of occurrences
 - **sumTotalTermFreq(field):** number of tokens
 - **sumDocFreq(field):** number of postings
 - **docCount(field):** number of docs with value
- Available from Lucene APIs
- Available from function queries
- Supports additional scoring algorithms...

Search Improvements

- Improved Scoring API
 - **Additional Scoring algorithms**
- Spellchecking improvements
- Miscellaneous
 - **Query parsing improvements**
 - **Deep paging support**

Scoring: Introduction

- Problem: “baked-in” vector space model
- How to integrate additional algorithms?
 - **Example: Language Models**
 - **Before 4.0: write custom Queries**
 - **Before 4.0: track certain statistics yourself**
- How to customize for your data?
 - **Without digging into the guts of postings lists!**
- Idea: separate “matching” from “scoring”

Scoring: additional algorithms

- BM25
- Language Models
- Divergence from Randomness
- Information-based Models

Scoring: Configuration

Lucene: use *Similarity* class

Solr: specify in schema.xml

```
<similarity class="solr.LMDirichletSimilarityFactory">  
  <float name="mu">1000</float>  
</similarity>
```

Spellchecking Improvements

- New DirectSpellChecker
 - No additional index needed
- Better Suggestions
 - Levenshtein vs. n-gram
- Exposes more configuration options
 - Tune to your collection

Spellchecking: Configuration

Lucene: use *DirectSpellChecker* class

Solr: specify in solrconfig.xml

```
<!-- a spellchecker built from a field of the main index -->  
<lst name="spellchecker">  
  <str name="name">default</str>  
  <str name="field">name</str>  
  <str name="classname">solr.DirectSolrSpellChecker</str>  
  ...
```

Query parsing Improvements

- Regular expression queries
 - `/mycompany.(com|org|net)/`
- Specify number of edits for fuzzy
 - `foobar~2`
- Wildcard escaping
 - `crazy\?*`
- Range query syntax improvements
 - **Mix inclusive/exclusive bounds**
 - **Support open-ended syntax in Lucene**

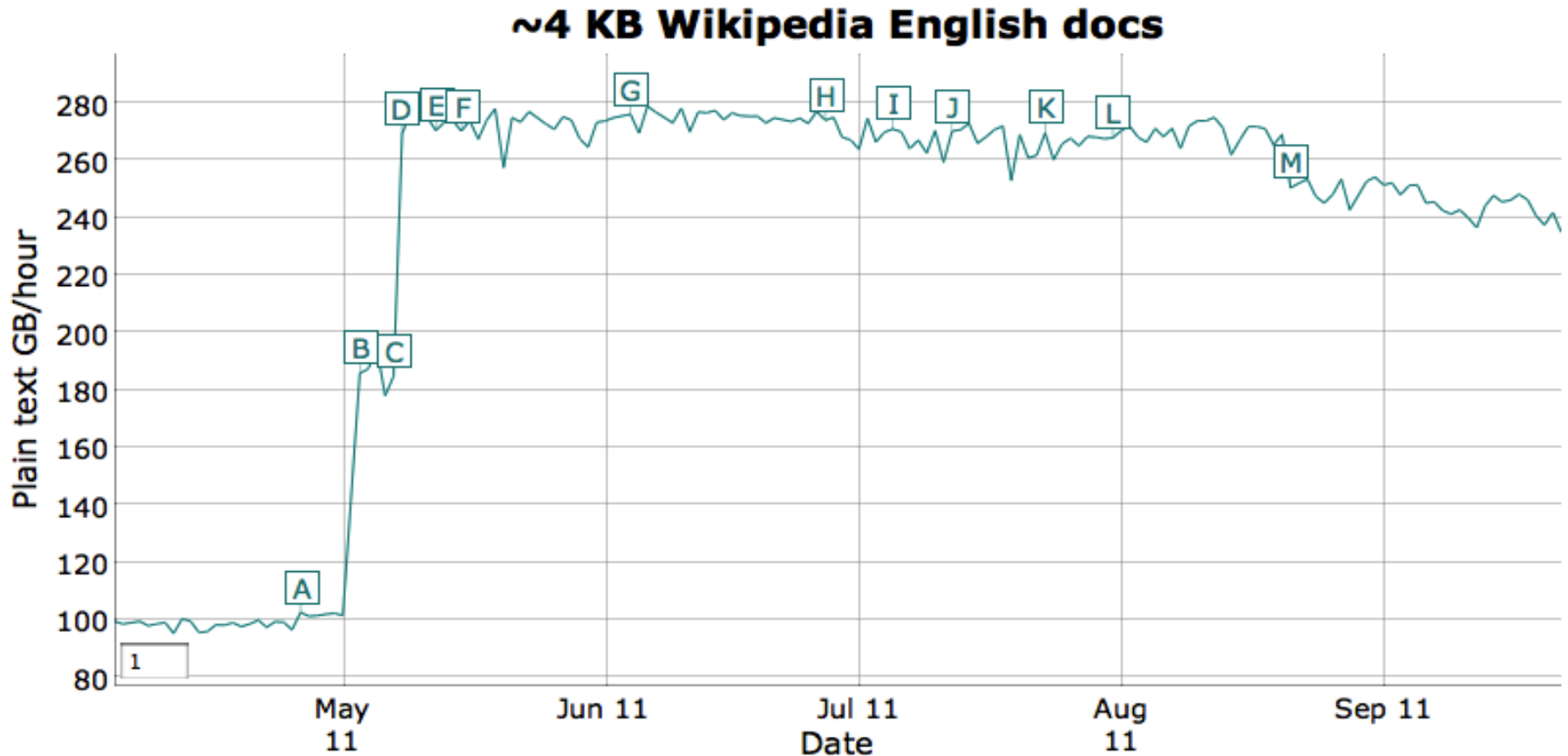
Deep-paging support

- Problem: users who page deep into results :)
- Normal paging in lucene:
 - Page 1: `search("foo", 20)` ← look at 1-20
 - Page 2: `search("foo", 40)` ← look at 21-40
- Deep paging in lucene:
 - Page 1: `searchAfter(null, "foo", 20)`
 - Page 2: `searchAfter(lastResult, "foo", 20)`

Performance Improvements

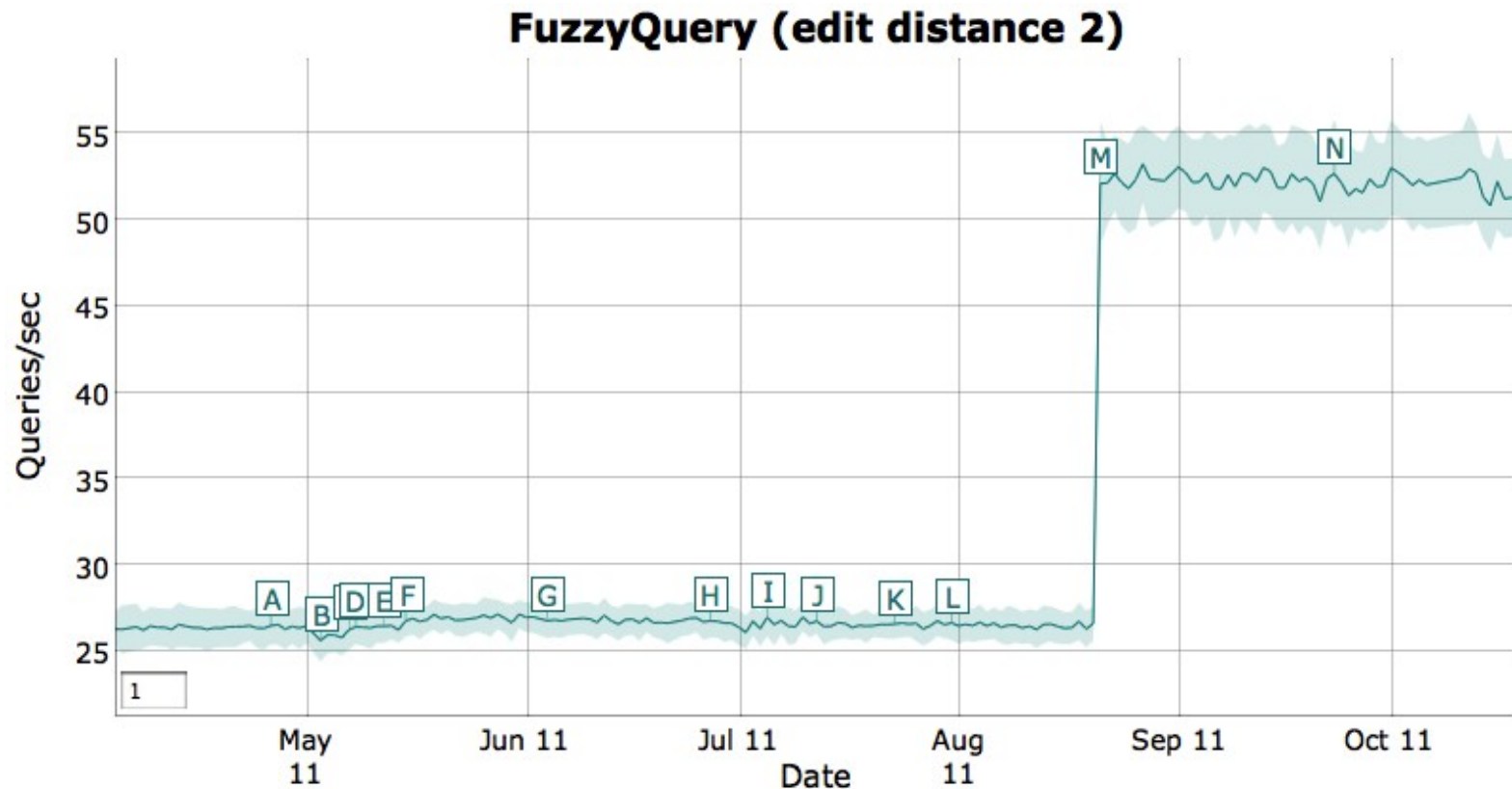
- Concurrent Flushing
- Fast Fuzzy Query
- Improved RAM Efficiency

Concurrent Flushing



<http://people.apache.org/~mikemccand/lucenebench/>

Fast Fuzzy Query



In Lucene 3 this thing is < 1 QPS!

Improved RAM Efficiency

- Lucene 4.0 uses much less memory
 - **Terms Index, Suggester, Synonyms : finite-state**
 - **Fieldcache: packed integers / utf-8**
- Example with Wikipedia collection:

“Memory footprint reduction from 389M to 90M after some off-the wall sorting and faceting”

Future Improvements

- Block Index Compression
 - **PFOR-delta, Simple8b, ...**
- Positional iterators from Scorers
 - **Offsets in postings lists (fast highlighting)**
 - **Proximity Scoring**
- Structured/Section Scoring (e.g. BM25F)
- Improved flexibility through Codec
 - **stored fields, term vectors, ...**
- Faster filtered search

Conclusion

- Lucene 4.0 will have many improvements, architectural, and API changes.
- A few of these were introduced here:
 - **Ability to customize the index format**
 - **Additional scoring algorithms**
 - **Performance Improvements**
- More are currently under development.
- For more information, look at CHANGES.txt in subversion, JIRA, ...

More Information

- Lucene Website
 - <http://lucene.apache.org>
- Users List
 - java-user@lucene.apache.org
- Lucene Revolution presentations
 - <http://www.lucidimagination.com/devzone/events/conferences/revolution/2010>
 - <http://www.lucidimagination.com/devzone/events/conferences/revolution/2011>
- Contact Info
 - robert.muir@lucidimagination.com
 - <http://www.lucidimagination.com/blog/author/robert-muir>