Lucene: Generic Data Indexing

Mike Cannon-Brookes
CEO, Atlassian Software Systems
Java Champion
WARNING

The following presentation contains egregious product placement and lots of text.
Indexing:

"The process of converting a collection of data into a format suitable for easy search and retrieval."
### Issue Navigator

Displaying issues 1 to 10 of 10 matching issues.

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Reporter</th>
<th>Assignee</th>
<th>Status</th>
<th>Res</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONF-6076</td>
<td>Error page if new password doesn't match Crowd password validation</td>
<td>Matt Ryall</td>
<td>Unassigned</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6075</td>
<td>Document process for moving from evaluation to commercial cluster license</td>
<td>Matt Ryall</td>
<td>Unassigned</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6072</td>
<td>zip _src from tiny moe served without caching headers on extrenet</td>
<td>Scott Farquhar</td>
<td>Matthew Jarmen</td>
<td>Reopened</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6071</td>
<td>$[base url] is not being substituted correctly in daily update email.</td>
<td>Bob Di Marzo</td>
<td>Unassigned</td>
<td>Needs Verification</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6070</td>
<td>Migrate c.a.c to hibernate user repository</td>
<td>Christopher Owen</td>
<td>Unassigned</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6069</td>
<td>Link to Plugin Repository is 'plugin/repository link'</td>
<td>David Sull</td>
<td>Unassigned</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6068</td>
<td>Global Activity link on Space Activity page is not correct</td>
<td>Agnes Rw</td>
<td>Christopher Owen</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6066</td>
<td>Intermittent problem loading /stylesheets and javascript on extranet</td>
<td>Scott Farquhar</td>
<td>Christopher Owen</td>
<td>In Progress</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6065</td>
<td>RSS parsing and content-type problems</td>
<td>Matt Ryall</td>
<td>Unassigned</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
<tr>
<td>CONF-6064</td>
<td>Group picker for page restrictions silently hides non-member groups, may need clarifying sentence added</td>
<td>David Sull</td>
<td>Unassigned</td>
<td>Open</td>
<td>UNRESOLVED</td>
<td>10-Mar-07</td>
</tr>
</tbody>
</table>
JIRA: Lucene History

- 1.4 - Use DB for all queries, Lucene only if full text search - results 'merged'
- 2.0 - Use Lucene for all search, Java for permissioning - results iterated over and non-view stripped
- 2.2 - Use Lucene for all queries including perms - sorting still done in Java
- 2.4 - Use Lucene for all queries, retrieving issues and displaying - no DB access at all!
- 3.0 - Switch "stats" over to using Lucene via HitCollectors
JIRA: Statistics

All from Lucene!
Lucene: Full Text Search

- Text Analysis & Stemming
  - “Michael jogs in the park” > “michael, jog, park”

- Proximity Queries
  - “cat NEAR dog”

- Wildcard Queries
  - “jog*”, “j?g”

- Results returned scored by relevance
Lucene: Generic Data Indexing (GDI)

- Fast retrieval of complex data objects
  - Built from one database, multiple databases, files, anywhere
  - Not a single table - just use a database index
Lucene: Generic Data Indexing (GDI)

- Powerful pre-built query tools
  - RangeQuery, BooleanQuery etc

“select issues created between 2001 and 2004, with no components, no versions, still unresolved that have > 4 votes”
Lucene: Generic Data Indexing (GDI)

• Results returned sorted in *custom order*

• Sort, SortField
Lucene: Generic Data Indexing (GDI)

- AOP-like result filtering and hit collection
  - QueryFilter and HitCollector
Lucene: Generic Data Indexing (GDI)

- Integrated full text search - only *if* you need it!
- “Free!”

select issues created between 2001 and 2004, with no components, no versions, still unresolved that have > 4 votes

_and match the query “dash*”_
Database V1

Issues

<table>
<thead>
<tr>
<th>Issue</th>
<th>Summary</th>
<th>Assignee</th>
<th>Reporter</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRA-1</td>
<td>Buy milk</td>
<td>Fred</td>
<td></td>
</tr>
<tr>
<td>JRA-2</td>
<td>Collect laundry</td>
<td>Bill</td>
<td>Fred</td>
</tr>
</tbody>
</table>

Query: select * from issues where assignee = 'fred'
Database V2

<table>
<thead>
<tr>
<th>Issues</th>
<th></th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Summary</td>
<td>Issue</td>
</tr>
<tr>
<td>JRA-1</td>
<td>Buy milk</td>
<td>JRA-1</td>
</tr>
<tr>
<td>JRA-2</td>
<td>Collect laundry</td>
<td>JRA-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JRA-2</td>
</tr>
</tbody>
</table>

Query: select * from issues, fields where fields.field = 'Assignee' and fields.value = 'fred' And fields.issue = issues.issue
Database V3

FieldValues
  ├── IssueFieldValue
  │    └── Components
  │         └── Issues
  │                 └── Versions
  │                        └── Attachments
  │                                                └── Comments
  │                                                                                                                                                 └── IssueProperties

Query: ? ? ?
How is Lucene fast?

Objects

<table>
<thead>
<tr>
<th>reporter</th>
<th>Bill</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignee</td>
<td>Fred</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reporter</th>
<th>Jane</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignee</td>
<td>Fred</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reporter</th>
<th>Fred</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignee</td>
<td>Bill</td>
</tr>
</tbody>
</table>
How is Lucene fast?

Objects

<table>
<thead>
<tr>
<th>reporter</th>
<th>assignee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Fred</td>
</tr>
<tr>
<td>Jane</td>
<td>Fred</td>
</tr>
<tr>
<td>Fred</td>
<td>Bill</td>
</tr>
</tbody>
</table>

Docs

- docId: 1
- reporter: Bill
- assignee: Fred
- component: 1
- component: 4
- component: 5
- created: 20070320
- field: value ...

- docId: 2
- reporter: Jane
- assignee: Fred

- docId: 3
- reporter: Fred
- assignee: Bill
**How is Lucene fast?**

<table>
<thead>
<tr>
<th>Objects</th>
<th>Docs</th>
<th>Inverted Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>reporter</td>
<td>Bill</td>
<td>reporter</td>
</tr>
<tr>
<td>assignee</td>
<td>Fred</td>
<td>field</td>
</tr>
<tr>
<td>reporter</td>
<td>Jane</td>
<td></td>
</tr>
<tr>
<td>assignee</td>
<td>Fred</td>
<td></td>
</tr>
<tr>
<td>reporter</td>
<td>Fred</td>
<td></td>
</tr>
<tr>
<td>assignee</td>
<td>Bill</td>
<td></td>
</tr>
<tr>
<td></td>
<td>docId: 1</td>
<td>Fred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jane</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bill</td>
</tr>
<tr>
<td></td>
<td>docId: 2</td>
<td>Fred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bill</td>
</tr>
<tr>
<td></td>
<td>docId: 3</td>
<td>assignee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>doc ids</td>
</tr>
</tbody>
</table>
How is Lucene fast?

<table>
<thead>
<tr>
<th>Objects</th>
<th>Docs</th>
<th>Inverted Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>reporter</td>
<td>Bill</td>
<td>reporter</td>
</tr>
<tr>
<td>assignee</td>
<td>Fred</td>
<td></td>
</tr>
<tr>
<td>reporter</td>
<td>Jane</td>
<td>Jane</td>
</tr>
<tr>
<td>assignee</td>
<td>Fred</td>
<td>Bill</td>
</tr>
<tr>
<td>reporter</td>
<td>Fred</td>
<td></td>
</tr>
<tr>
<td>assignee</td>
<td>Bill</td>
<td></td>
</tr>
</tbody>
</table>
Advantages For Generic Data Indexing

- Store Denormalised Data
  - Issue object, fields - one single Lucene document
  - No record de-duplication needed as per SQL query

- Native Java API
  - Useful for things like sorting where a DB can't do it
    - Java specific sort algorithm for issue keys
    - Version sequencing - v. complex to do in DB
Advantages For Generic Data Indexing

- **Constant time & capabilities**
  - Our apps are cross platform, OS, JDK, database
  - Lucene works pretty much the same across all of them unlike SQL
  - Local file system access (most commonly) which is faster than DB as no network time

- **Constant index format**
  - Readable from Java, C, Perl, Ruby etc

- **QueryFilters & HitCollectors!**
HitCollector

- Call back object for hit collection
- Great for statistical operations where content / score is irrelevant
  - JIRA - StatusHitCollector for ‘bucketing’
- Fast because:
  - Retrieve only fields you need
  - Minimum number of loops

![Project Summary](image)
public abstract void collect(int doc, float score)

Called once for every non-zero scoring document, with the document number and its score.

If, for example, an application wished to collect all of the hits for a query in a BitSet, then it might:

```java
Searcher searcher = new IndexSearcher(indexReader);
final BitSet bits = new BitSet(indexReader.maxDoc());
searcher.search(query, new HitCollector() {
    public void collect(int doc, float score) {
        bits.set(doc);
    }
});
```
QueryFilter

• AOP for query results

• BitSet representing possible matches
  • Complex in SQL to do, ends up being done in Java
  • Permissioning on top of any search
    • Construct once per request
  • Results are cached in the filter
QueryFilter

public BitSet bits(IndexReader reader) throws IOException

Description copied from class Filter
Returns a BitSet with true for documents which should be permitted in search results, and false for those that should not.

Specified by:
bits in class Filter

Throws:
IOException
Atlassian: Examples Of Lucene Usage

- **JIRA** - User driven queries an arbitrary data model
- Plugins index/search their own 'fields' - future proof!
- QueryFilters for permissions - cached per request
- HitCollectors for all statistics / dashboard
Atlassian: Examples Of Lucene Usage

- Confluence - Full text search of wiki pages
  - SearchExtractor allows plugins to add meta data to documents
  - Also used to search attachment contents & metadata
    - eg image file sizes
  - Arbitrary ‘page set’ retrieval
  - QueryFilter used extensively for security
Bamboo - Build telemetry statistics via Lucene

- Fast over millions of rows - data on every test/suite/build run, ever.
- Use Lucene to aggregate data into useful statistics
- HitCollectors used extensively for telemetry data

Recent Failures

- Average time to fix a failure: 2 days, 19 hours, 16 minutes
- Average number of builds between fixes: 1 builds
- The longest time taken to fix a failure is 2 days, 19 hours, 16 minutes
- The greatest number of builds taken to fix a failure is 1, from failure sta

96% Successful
Successful Builds: 24 / 25
Average Duration: 5 minutes
Problems For GDI

- One Big Singleton
  - Updates require serialization - indexes are write once, read many
  - Jira vs Confluence different access/write strategies

- Delete / Update Operations
  - Lucene wasn’t built for fast changing data
  - Delete operation is just a flag op & Update requires delete / re-add

- Writing Is Expensive
  - Opening/closing reader/writers proportional to index size
Problems For GDI

- **Timing Of index.optimise()**
  - Indexes get fragmented - optimise() defrags
  - Tricky to time this as v. slow on large indexes
  - Eden space strategy can solve this
    - Small index for ‘updated’ data, large for ‘old’ data
    - Optimize large index rarely, small frequently - like GC.
    - MultiIndexSearcher allows search on multiple indexes like one
Problems For GDI

- Non Transactional
  - DB can have data that index misses, or vice versa
  - Compass is a solution here - haven’t tested
  - Otherwise, architect correct design knowing Lucene

- Optionitis
  - Write settings can require a lot of knowledge and tuning
    - eg MAX_MERGE_DOCS

- Local storage - can be a problem in a cluster
  - See my other presentation for clustering strategies!
GDI Lucene Usage Models

- **JIRA**
  - Synchronous indexing > tricky locking problems at scale
  - More updates than creates > heavier index load
    - Fixed with Lucene 2.1!
  - Slower updates, statistics always correct

- **Confluence**
  - Asynchronous indexing
  - Updates are queued, flushed every minute
  - Clusterable and faster ‘net’ time for user
  - ‘Recent Updates’, ‘Search’ up to 1 min inaccurate
Tips

- Use derived data only, so can be recreated at will
- Store -1 for null because nature of fields
  - Can't query Lucene for 'lack of a field' - ie “No Component”
- Keep open a single searcher and 'flip it' after writing
- ThreadLocals are valuable in web apps
  - Use for Searchers, BooleanQueries and QueryFilters that are expensive to create per search but don’t change per request (10s of queries per request)
- Understand Lucene to adjust your usage to your app
- Index dates to highest granularity possible, prevent term explosion
  - Remember Lucene storage? YYMMDD vs YYMMDDHHmmSSSS
Links

- Luke - useful tool to examine indexes
  - http://www.getopt.org/luke

- Lucene In Action - awesome book
  - http://www.lucenebook.com

- Compass - Lucene abstraction framework
  - http://www.opensymphony.com/compass
Q & A

P.S. Java guru? Atlassian needs engineers!
- Sydney or San Francisco.

http://www.atlassian.com/about/jobs

Email me: mike@atlassian.com