Open Source Platforms for Search

Doug Cutting
Yahoo!
Apache Software Foundation
cutting@apache.org

International Workshop on Intelligent Information Access
July 6, 2006
Helsinki, Finland
What am I?

- not an academic (anymore)
- not a commercial software developer (anymore)
- open-source guy
  - developer
  - architect
  - manager
  - evangelist (but not a zealot)
# What Distinguishes Open Source?

<table>
<thead>
<tr>
<th>model</th>
<th>research</th>
<th>commerce</th>
<th>open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>license</td>
<td>various</td>
<td>various</td>
<td>OSI-approved</td>
</tr>
<tr>
<td>artifacts</td>
<td>publications</td>
<td>products</td>
<td>code</td>
</tr>
<tr>
<td>payment</td>
<td>university</td>
<td>private</td>
<td>various</td>
</tr>
<tr>
<td>motivator</td>
<td>peer</td>
<td>money</td>
<td>various</td>
</tr>
<tr>
<td>community</td>
<td>academic</td>
<td>closed</td>
<td>open</td>
</tr>
</tbody>
</table>
Lucene pre-history: Xerox PARC

- Text Database (TDB) 1988-1993
  - in Common Lisp
  - B-Tree based (with optimizations)
  - phrase & ranked searching
  - vectors & clustering
  - research prototype, led to several publications

- Lessons
  - seek per <term,doc> pair too slow
  - wanted real users
Lucene pre-History: Apple ATG

• V-Twin, 1993-1996
  - in C++
  - B-tree based, with optimizations
    • slow to update large collections except by merging indexes
  - no proximity: ranked search only

• Lessons:
  - batching seeks still too slow
    • B-trees fragment; index merging required
  - developers don't like subclassing
Lucene pre-History: Excite

• Architext (1996-1998)
  − C++, originally written by Graham Spencer
  − merge-based indexing (4-stage process)
  − 2-level files (subset of keys in RAM w/ pointers to data)
  − no fields, tf*idf ranking, w/ boolean proximity
  − 250M page indexes, ~1000 searches/second peak

• Lessons
  − merging indexes scales optimally
  − server-side rocks, but C++ fragile
  − code can be lost
Digression: Seek versus Transfer

- **B-Tree**
  - requires seek per access
  - unless to recent, cached page
  - so can buffer & pre-sort accesses
  - but, w/ fragmentation, must still seek per page
Digression:
Seek versus Transfer

- update by merging
  - merge sort takes $\log(\text{updates})$, at transfer rate
  - merging updates is linear in db size, at transfer rate

- if 10MB/s xfer, 10ms seek, 1% update of TB db
  - 100B entries, 10kB pages, 10B entries, 1B pages
  - seek per update requires 1000 days!
  - seek per page requires 100 days!
  - transfer entire db takes 1 day
Lucene History

- 1997-98: written in 3 months, part-time
- 1998: Licensed to one client
- 2000: open source on Sourceforge.net
  - GPL at first, then LGPL
- 2001: moved to Apache
- 2005: Apache top-level project
Original Lucene Goals

• in Java
  – new environment, no existing search technology
• simple, well-documented API
  – no user subclassing required
  – no config files – dynamic field typing
• support commonly used features
  – fields, booleans, proximity, tf*idf ranking
• scalable & incremental
  – aimed for ~10M document indexes on single CPU
Lucene Architecture

search
index
store|document|analysis
Lucene Indexing Algorithm
**Lucene Indexing Algorithm: notes**

- multiway merge: process at transfer rate
- average $b \cdot \log_b(N)/2$ indexes
  - $N=1M$, $b=2$ gives just 20 indexes
  - fast to update and not too slow to search
- optimization
  - single-doc indexes kept in RAM, saves system calls
- batch indexing w/ $M=\infty$, merge all at end
  - equivalent to external merge sort, optimal
- segment indexing w/ $M<\infty$
Lucene Search Algorithms

- merge streams of postings, ordered by <doc, pos>
  - can skip ahead in stream
  - collect only top sorting hits
  - space proportional to query size
- lots of operators
  - boolean, phrase, span, range, etc.
- scoring is (modified) tf*idf by default
Lucene Status

- 1.4.3 release **widely used**
  - wikipedia, eclipse, etc. (anyone here?)
  - translated to C, C++, C#, Python, Perl & Ruby
- 2.0 release recently completed
  - api cleanups
  - lots of new features, bug fixes & optimizations
- users don't subclass
  - but developers do, to extend
- if I'd tried to sell it
  - it would have created less commerce
Rapid Adoption Facilitators

- abstract external interfaces
  - anywhere i/o is performed, language processed, etc.
  - provide default implementations that most folks use
- concrete internal implementations
  - keeps things finite & tractable
  - can easily make private internals public & abstract later
- minimal
- always complete, useful, & working
  - users become developers
Lucene Future

- no central planning
  - Andrew Morton: “Whatever people send me.”
- wish list
  - extensible index format
  - easy federation
  - web service
- what would you add?
Nutch

- web search application
  - crawler
  - link graph
    - link analysis
    - anchor text
  - document format detection & parsing
  - language, charset detection & processing
  - extensible crawling, parsing, indexing & search
# Nutch Documents

<table>
<thead>
<tr>
<th>field</th>
<th>stored</th>
<th>indexed</th>
<th>analyzed</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>anchor</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>content</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>lang</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nutch Queries

- By default:
  - require all query terms
  - search url, anchors and content
  - reward for proximity
- E.g., search for “search engine” is expanded to:
  - + (url:search^x anchor:search^y content:search^z)
  - + (url:engine^x anchor:engine^y content:engine^z)
  - url:“search engine”~p^a
  - anchor:“search engine”~q^b
  - content:“search engine”~r^c
Query Parsing

• Certain characters cause implicit phrases:
  - dash, plus, colon, slash, dot, apostrophe and atsign
  - URL & email are thus phrase searches
  - e.g., http://www.nutch.org/ = “http www nutch org”,
    doug@nutch.org = “doug nutch org”, etc.

• Stop words indexed, but normally ignored
  - unless in phrase or required.
  - can use N-grams if in phrase

• Plugins can extend for new fields
Nutch Search Performance Tricks

- index N-grams for very common words
  - “the” in content, “www” in url, etc.
  - convert phrase queries (implicit or explicit) to N-grams
  - stop word search still permitted, but very rare
- convert common binary terms to bit-vectors
  - lang:en, content-type:pdf, etc.
- sort indexes by page score
  - stop searching when \( N \approx 1000 \) matches found
- distributed search
  - partition by document
Nutch Scalability Goals

• Scale to entire web
  - pages on millions of different servers
  - billions of pages
  - complete crawl takes weeks
  - very noisy

• Support high traffic
  - thousands of searches per second

• State-of-the-art search quality
Scalability

• To meet scalability goals:
  - multiple simultaneous fetches
    (~100 pages/second / CPU, ~10M / day)
  - parallel, distributed db update
    (100M pages @ 100 pages/second / CPU)
  - distributed search
    (2-20M pages, 1-40 searches/second / CPU)
Initial Scalability

• Initial implementation was scalable...
  – parallel processes on multiple machines
  – some serial bottlenecks, but w/ plans to resolve
  – 100M web pages demonstrated
... but not to billions of pages

• scales better than other open source options
• but large installations are operationally onerous
  – manually monitoring multiple machines is painful
  – data-interchange and space-allocation difficult
• with single operator
  – hard to use more than a handful of machines
  – effectively limited to ~100M pages
Hadoop

• new project, spun out of Nutch
• platform for distributed computing
• foundation for all Nutch operations
Hadoop's DFS

- modelled after Google's GFS
- single namenode
  - maps name → <blockId>*
  - maps blockId → <host:port>\textsuperscript{replication\_level}
- many datanodes, one per disk generally
  - map blockId → <byte>*
  - poll namenode for replication, deletion, etc. requests
- client code talks to both
MapReduce

- Platform for reliable, scalable computing.
- All data is sequences of <key,value> pairs.
- Programmer specifies two primary methods:
  - map(k, v) $\rightarrow$ <k', v'>*
  - reduce(k', <v'>*) $\rightarrow$ <k', v'>*
  - also partition(), compare(), & others
- All v' with same k' are reduced together, in order.
  - bonus: built-in support for sort/merge!
MapReduce job processing
Hadoop Status

- Very active development
- Currently tested on up to 600 nodes
- Latest sort benchmark
  - 300 node cluster
  - 3TB of random data in DFS
  - sorted in under 3 hours
  - output to DFS
- Still lots of room for improvement
Nutch on Hadoop

• Nutch's major algorithms converted in 2 weeks.

• Before:
  - several were undistributed scalability bottlenecks
  - distributable algorithms were complex to manage
  - collections larger than 100M pages impractical

• After:
  - all are scalable, distributed, easy to operate
  - code is substantially smaller & simpler
  - should permit multi-billion page collections
Nutch Status

• used in production
  – intranet: Oregon State University
  – vertical: Creative Commons
  – larger-scale vertical: Internet Archive
  – lots of others (you?)

• scaling well
  – 200M pages indexed on 35 boxes
  – 50M pages crawled & indexed in 24 hours on 20 boxes
Nutch Future

• not centrally planned!
• wish list
  – web-based config
  – better incremental updates
  – shingle-based dedup
  – spam detection
• what would you add?
Apache is Community

- technology is only the artifact
- open source license doesn't make a community
- without community
  - technology stagnates
  - is not used
- with community
  - technology evolves
  - meets needs of community
  - survives
Thanks!

http://lucene.apache.org/