Open Source Search

Doug Cutting
cutting@apache.org

Information Retrieval Seminar
December 05, 2005
IBM Research Lab
Haifa, Israel

Lucene pre-history: Xerox PARC

- Text Database (TDB) 1988-1993
  - in Common Lisp
  - B-Tree based (with optimizations)
  - phrase & ranked searching
  - vectors & clustering
  - used only in some prototypes
- Lessons
  - seek per <term,doc> pair too slow
  - wanted real users

Lucene pre-History: Apple ATG

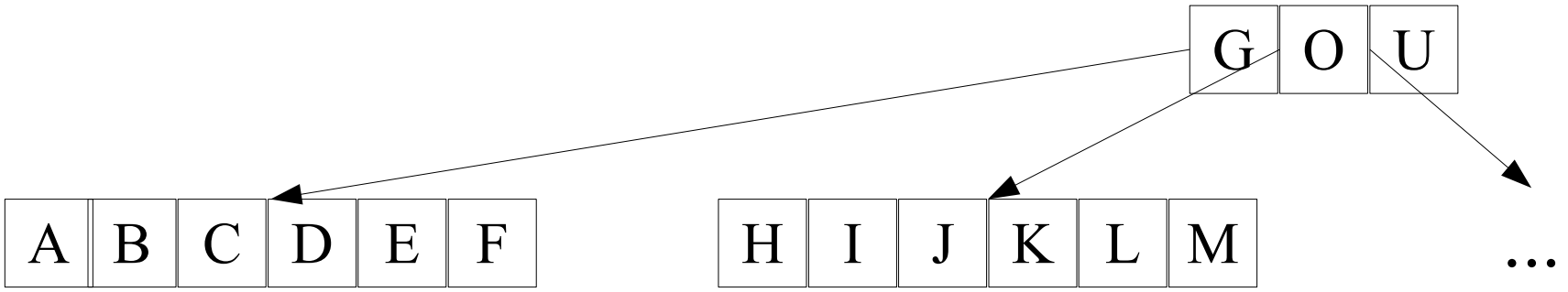
- V-Twin, 1993-1996
  - in C++
  - B-tree based, with optimizations
    - slow to update large collections except by merging indexes
  - no proximity: ranked search only
  - used in Sherlock (1998), Spotlight (2004) etc.
- Lessons:
  - batching seeks still too slow
    - B-trees fragment; index merging 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
  - closed-sourced code can be lost

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



Seek versus Transfer

- update by merging
  - merge sort takes log(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

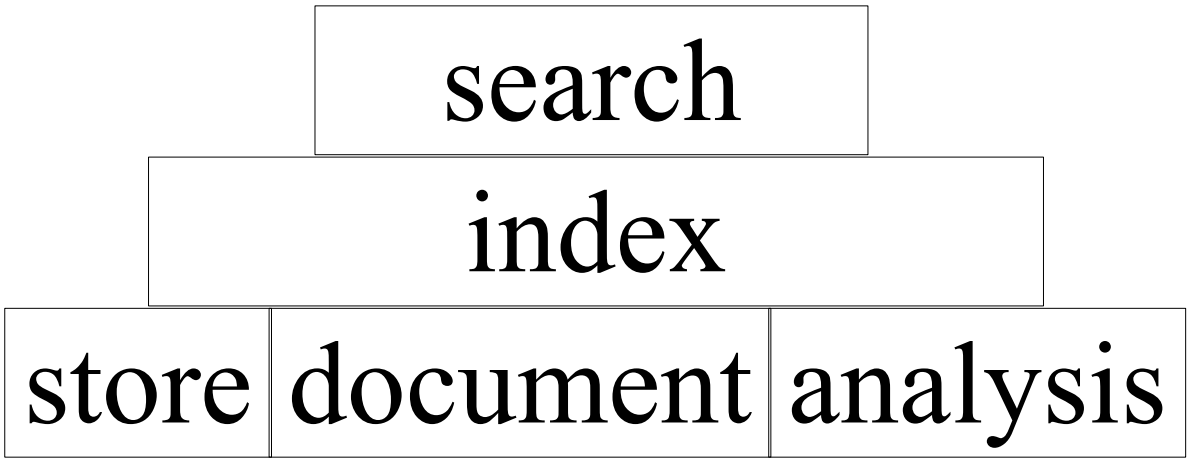
Lucene History

- 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 engines
- no config files, dynamic field typing
- simple, well-documented API
  - no user subclassing required
- support commonly used features
  - fields, booleans, proximity, tf*idf ranking
- scalable & incremental
  - aimed for ~10M document indexes on single CPU

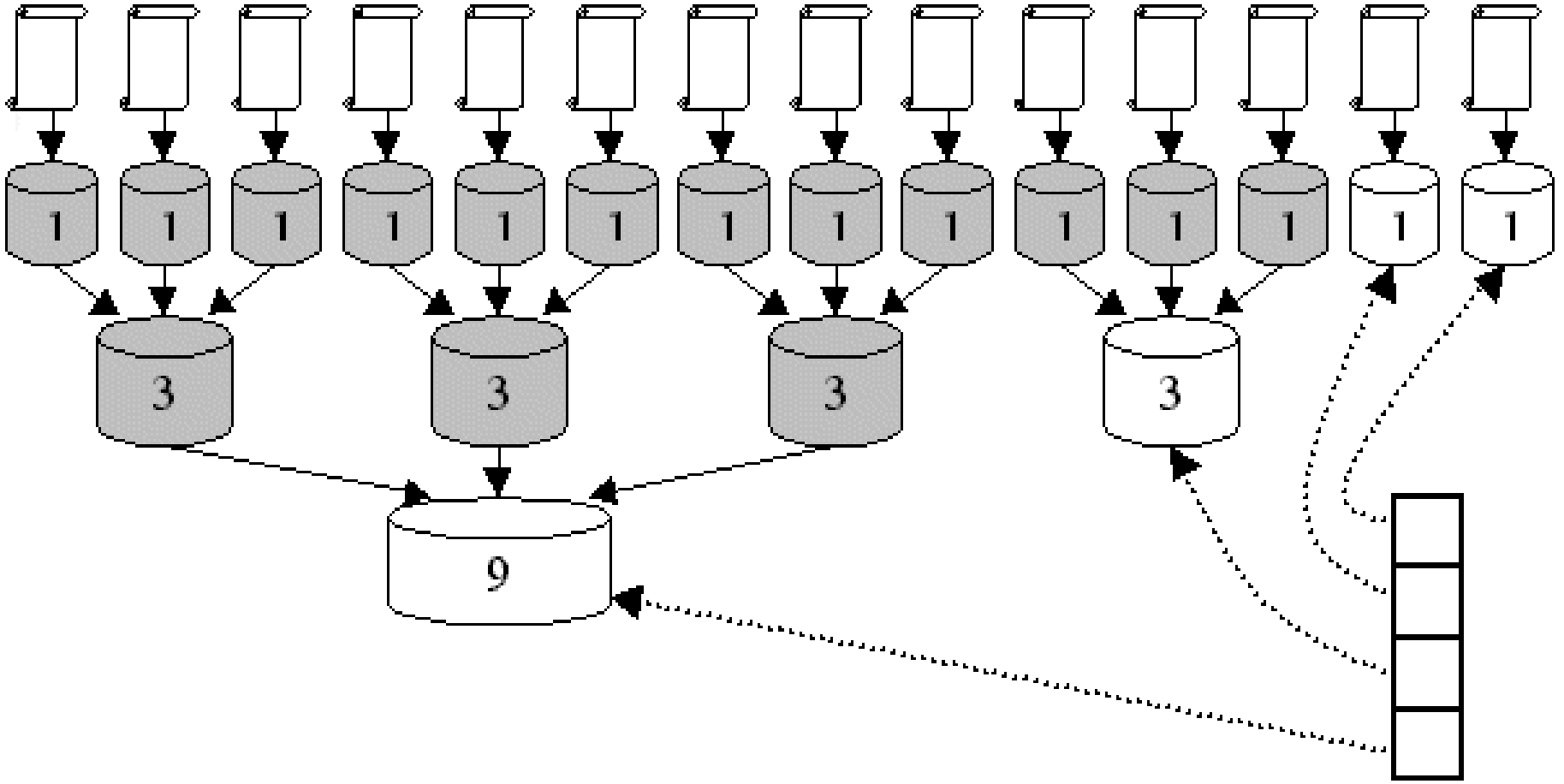
Lucene Architecture



Lucene Indexing Algorithm

- maintain a stack of segment indexes
- create index for each incoming document
- push new indexes onto the stack
- let *b*=10 be the merge factor; M=∞
- for (size = 1; size < M; size *= b) {
  if (∃ b indexes of size docs on top of the stack) {
  pop them off the stack;
  merge them into a single index;
  push the merged index onto the stack;
  } else {
  break;
  }
  }

Lucene Indexing Algorithm



Lucene Indexing Algorithm: notes

- multiway merge: process at transfer rate
- average $b*\log_b(N)/2$ 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
- 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.
  - translated to C, C++, C#, Python, Perl & Ruby
- 2.0 release nearing completion
  - 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

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 indexing & search

Nutch Documents

field	*stored*	*indexed*	*analyzed*
`url`	Yes	Yes	Yes
`anchor`	No	Yes	Yes
`content`	No	Yes	Yes
`site`	Yes	Yes	No
`lang`	Yes	Yes	No
…			

Nutch Analysis

- Defined w/ JavaCC
- Words are (<letter> | [0-9_&])+
  - or acronyms: <letter> [.] (<letter> [.])+
  - or CJK
- First word in each anchor gets big position increment, to inhibit cross-anchor matches.
- No stop list or stemmer.
- URLs, email, etc. tokenized same as other text.

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 removed
  - unless in phrase or required.
  - can use N-grams if in phrase
- Plugins can extend for new fields

Nutch N-Grams

- Very common terms are indexed with neighbors.
  - E.g., w/ “the”, “http”, “www”, “http-www” & “org”:
  - “Buffy the Vampire” is indexed as
    buffy, buffy-the+0, the, the-vampire+0, vampire,
  - “http://www.nutch.org/” is indexed as:
    http, http-www+0, http-www-nutch+0,
    www, www-nutch+0, nutch, nutch-org+0, org.
  - terms are field specific
  - improves performance of phrase searches

Nutch N-Gram Query

- For explicit phrase query: “Buffy the Vampire”:
  - for content field, translated to:
    content:“buffy-the the-vampire”
  - much faster b/c we don't have to search for “the”
- For query: http://www.nutch.org/:
  - implicit phrase: “http www nutch org”
  - for URL field, translated to:
    url:“http-www-nutch www-nutch nutch-org”

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 is 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

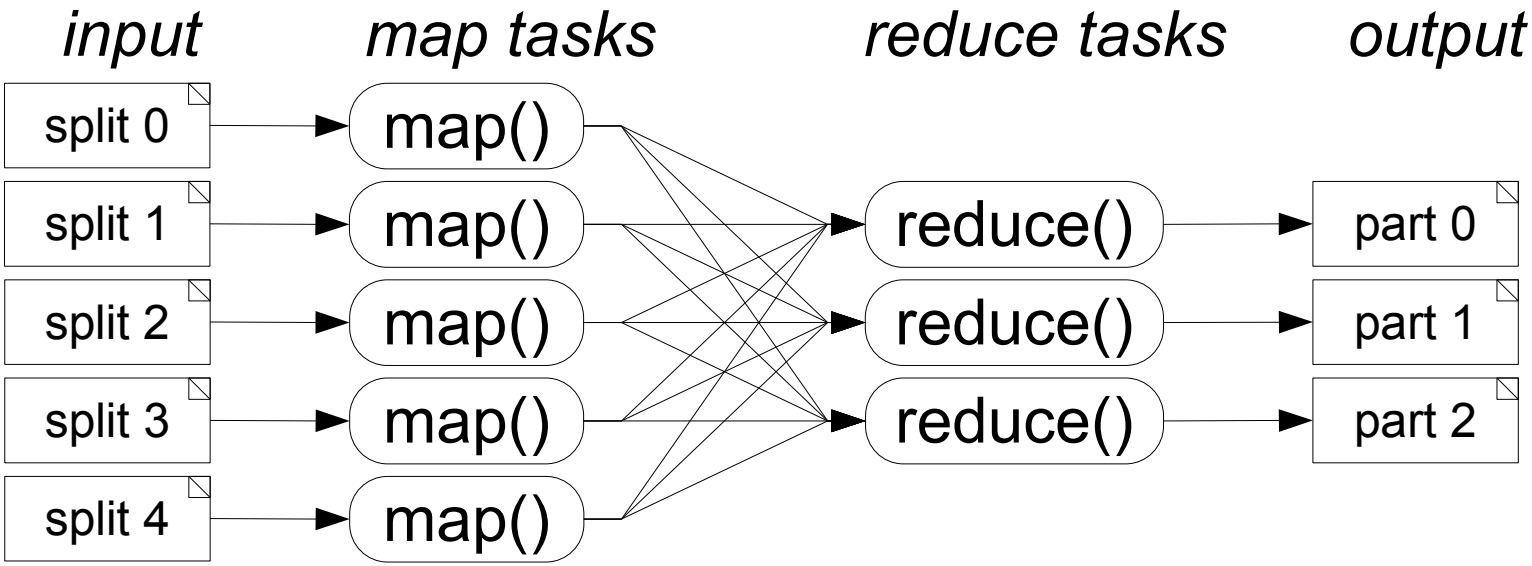
NDFS

- modelled after Google's GFS
- single *namenode*
  - maps name → <blockId>*
  - maps blockId → <host:port>$^{replication\_level}$
- 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) → <k', v'>*
  - reduce(k', <v'>*) → <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



Nutch on MapReduce & NDFS

- Nutch's major algorithms converted in 2 weeks.
- Before:
  - several were undistributed scalabilty 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 verticals: Internet Archive
- scaling well
  - 200M pages indexed on 35 boxes
  - 50M pages crawled & indexed in 24 hours on 20 boxes
  - linear scaling on 200 boxes

Nutch Future

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

Thanks!

http://lucene.apache.org/