mow

M M X

Miracle OpenWorld 2010

Making freetext search with Lucene.Net work for you

Anders Lybecker

- Consultant
  - Solution Architect
  - Miracle Projects A/S
- Expertise
  - .Net
  - SQL Server
  - Freetext Search

aly@miracleas.dk / +45 25 277 147

Agenda

- Lucene Intro
- Indexing
- Searching
- Analysis
  - Options
  - Patterns
  - Multilingual
  - What not to do!
- „Did you mean...“ functionality
- Performance factors for indexing and searching

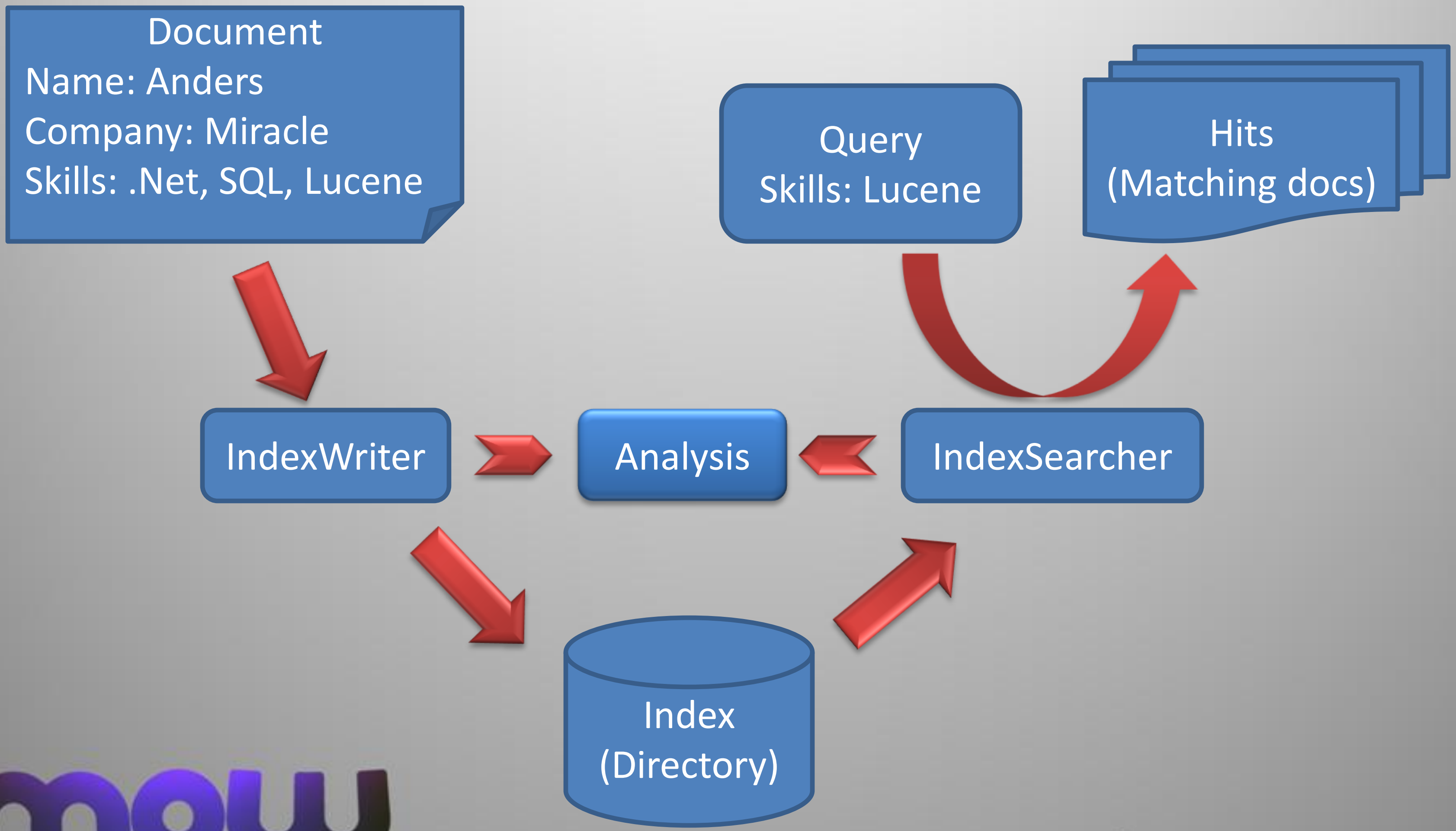
What is Lucene

- Information retrieval software library
  - Also know as a search engine
- Free / open source
- Apache Software Foundation
- Document Database
  - Schema free
- Inverted Index
- Large and active community

Who uses Lucene?

- MySpace, LinkedIn, Technorati, Wikipedia, Monster.com, SourceForge, CNET Reviews, E. On, Expert-Exchange, The Guardian - Jobs, Akamai, Eclipse, JIRA, Statsbiblioteket - the State and University Library in Århus – Denmark, AOL, Disney, Furl, IBM OmniFind Yahoo! Edition, Hi5, TheServerSide, Nutch, Solr

Basic Application



Document
Name: Anders
Company: Miracle
Skills: .Net, SQL, Lucene

Query
Skills: Lucene

Hits
(Matching docs)

IndexWriter

Analysis

IndexSearcher

Index
(Directory)

Querying

1. Construct Query
   – E.g via QueryParser
2. Filter
   – Limiting the result, E.g security filters
   – Does not calculate score (Relevance)
   – Caching via CachingWrapperFilter
3. Sort
   – Set sort order, default Relevance

Demo

Types of Queries

Name	Description
TermQuery	Query by a single Term – Word
PrefixQuery	Wildcard query – like Dog*
RangeQuery	Ranges like AA-ZZ, 22-44 or 01DEC2010-24DEC2010
BooleanQuery	Container with Boolean like semantics – Should, Must or Must Not
PhraseQuery	Terms within a distance of one another (slop)
WildcardQuery	E.g. A?de* matches Anders
FuzzyQuery	Phontic search via Levenshtein distance algorithm

Query Parser

- Default Query Parser Syntax
  - conference
  - conference AND lucene <=> +conference +lucene
  - Oracle OR MySQl
  - C# NOT php <=> C# -php
  - conference AND (Lucene OR .Net)
  - “Miracle OpenWorld“
  - title:MOW2010
  - M?racle
  - Mir*
  - schmidt~ schmidt, schmit, schmitt
  - price:[12 TO 14]
- Custom Query parsers
  - Use Irony, ANTLR …

Analysis

- Converting your text into Terms
  - Lucene does NOT search your text
  - Lucene searches the set of terms created by analysis
- Actions
  - Break on whitespace, punctuation, caseChanges, numb3rs
  - Stemming (shoes -> shoe)
  - Removing/replacing of Stop Words
    - The quick brown fox jumps -> The quick brown fox jumps
  - Combining words
  - Adding new words (synonyms)

Demo

Field Options

- Analyzed, Not Analyzed, Analyzed No Norms, Not Analyzed No Norms
- Stored – Yes, No, Compress

Index	Store	TermVector	Example usage
Not Analyzed No Norms*	Yes	No	Identifiers (Primary keys, file names), SSN, Phone No, URLs, names, Dates and textual fields for sorting
Analyzed	Yes	Positions + Offsets	Title, Abstract
Analyzed	No	Positions + Offsets	Main content body
No	Yes	No	Document type, Primary keys (if not used for searching)
Not Analyzed	No	No	Hidden keywords

* Norms are used for Relevance ranking

Field Options

- Norms
  - Boosts and field length normalization
  - Use for ranking
    - Default: shorter fields has higher rank
- Term Vectors
  - Miniature inverted index
  - Term frequency pairs
  - Positional information of each Term occurrence (Position and Offset)
  - Use with
    - PhraseQuery
    - Highlighter
    - "More Like This“

Copy Fields

- It’s common to want to index data more than one way
- You might store an unanalyzed version of a field for searching
  - – And store an analyzed version for faceting
- You might store a stemmed and non-stemmed version of a field
  - – To boost precise matches

Multilingual

- Generally, keep different languages in their own fields or indexes
- This lets you have an analyzer for each language
  - Stemming, stop words, etc.

Wildcard Querying

- Scenario
  - Search for *soft
  - Leading wildcards require traversing the entire index
- Reversing Token Filter
  - Reverse the order, and leading wildcards become trailing
  - *soft -> tfos*

What can go wrong?

- Lots of things
  - You can’t find things
  - You find too much
  - Poor query or indexing performance
- Problems happen when the terms are not what you think they are

Case: Slow Searches

- They index 500,000 books
- Multiple languages in one field
  - So they can’t do stemming or stop words
- Their worst case query was:
  - “The lives and literature of the beat generation”
- It took 2 minutes to run
- The query requires checking every doc containing “the” & “and”
  - And the position info for each occurrence

Bi-grams

- Bi-grams combine adjacent terms
- “The lives and literature “ becomes “The lives” “lives and” “and literature”
- Only have to check documents that contain the pair adjacent to each other.
- Only have to look at position information for the pair
- But can triple the size of the index
  - Word indexed by itself
  - Indexed both with preceding term, and following term

Common Bi-grams

- Form bi-grams only for common terms
- “The” occurs 2 billion times. “The lives” occurs 360k.
- Used the only 32 most common terms
- Average response went from 460 ms to 68ms.

Spell Checking

- „Did you mean...“
- Spell checker starts by analyzing the source terms into n-grams

Index Structure	Example
word	kings
gram3	kin, ing, ngs
gram4	king, ings
start3	kin
start4	king
end3	ngs
end4	ings

Demo

Trie Fields – Numeric ranges

- Added in v2.9
- 175 is indexed as hundreds:1 tens:17 ones:175
    - TrieRangeQuery:[154 TO 183] is executed as tens:[16 TO 17] OR ones:[154 TO 159] OR ones:[180 TO 183]
- Configurable precisionStep per field
- 40x speedup for range queries

Synonyms

- Synonym filter allows you to include alternate words that the user can use when searching
- For example, theater, theatre
  - Useful for movie titles, where words are deliberately mis-spelled
- Don't over-use synonyms
  - It helps recall, but lowers precision
- Produces tokens at the same token position
  - "local | theater | company"

    theatre

Other features

- Find similar documents
  - Selects documents similar to a given document, based on the document's significant terms
- Result Highlighter
- Tika
  - Rich document text extraction
- Spatial Search
- …

Demo

General Performance Factors

- Use local file system
- Index Size
  - Stop Word removal
  - Use of stemming
- Type of Analyzer
  - More complicated analysis, slower indexing
  - Turn off features you are not using (Norms, Term Vectors etc.)
- Index type (RAMDirectory, other)
- Occurrences of Query Terms
- Optimized Index
- Just add more RAM :-)

Indexing Performance Factors

- Re-use the IndexWriter
- IndexWriter.SetRAMBufferSizeMB
  - Minimum # of MBs before merge occurs and a new segment is created
  - Usually, Larger == faster, but more RAM
- IndexWriter.SetMergeFactor
  - How often segments are merged
  - maller == less RAM, better for incremental updates
  - Larger == faster, better for batch indexing
- IndexWriter.SetMaxFieldLength
  - Limit the number of terms in a Document
- Reuse Document and Field instances

Search Performance Factors

- Use ReadOnly IndexReader
- Share a single instance of IndexSearcher
  - Reopen only when nessecary and pre warm-up
- Query Size
  - Stop Words removal, Bi-grams …
- Query Type(s)
  - WildcardQuery rewrites to BooleanQuery with all Terms
- Use FieldSelector
  - Select only the stored fields needed
- Use Filters with cache
- Search an “all” field instead of many fields with the sate Query Terms

Demo

Questions?

Resources

- Anders Lybecker’s Blog
  - http://www.lybecker.com/blog/
- Lucene
  - http://lucene.apache.org/java/docs/
- Lucene.Net
  - http://lucene.apache.org/lucene.net/
- Lucene Wiki
  - http://wiki.apache.org/lucene-java/
- Book: Lucene In Action
- Luke - Lucene Index Exploration Tool
  - http://www.getopt.org/luke/

Relevans Scoring

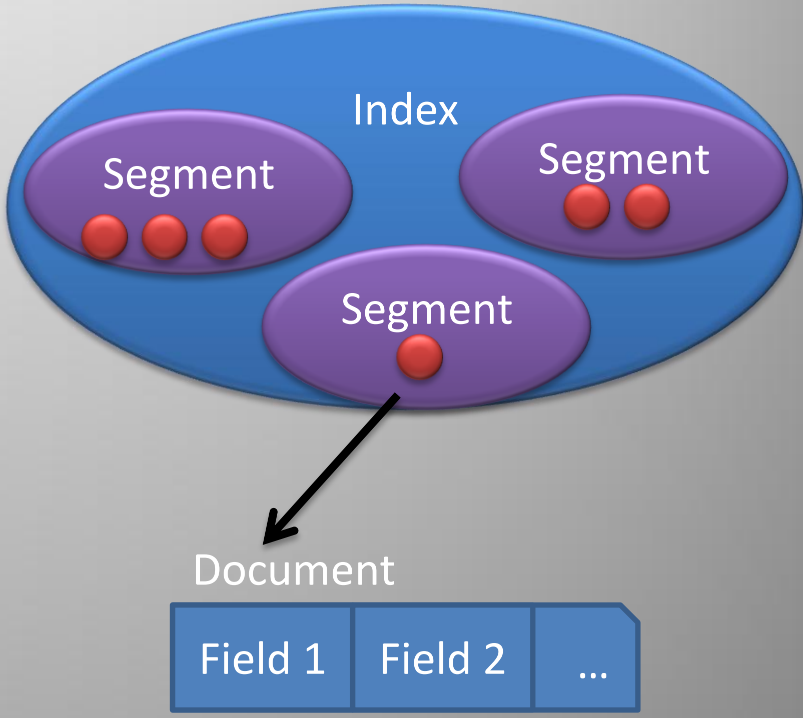
$$\sum_{t\,in\,q}(tf(t\,in\,d)\times idf(t)^2\times boost(t.field\,in\,d)\times lengthNorm(t.field\,in\,d))$$
$$\times\, coord(q,d)\times queryNorm(q)$$

Factor	Description
tf(t in d)	Term frequency factor for the term (t) in the document (d), ie how many times the term t occurs in the document.
idf(t)	Inverse document frequency of the term: a measure of how “unique” the term is. Very common terms have a low idf; very rare terms have a high idf.
boost(t.field in d)	Field & Document boost, as set during indexing. You may use this to statically boost certain fields and certain documents over others.
lengthNorm(t.field in d)	Normalization value of a field, given the number of terms within the field. This value is computed during indexing and stored in the index norms. Shorter fields (fewer tokens) get a bigger boost from this factor.
coord(q, d)	Coordination factor, based on the number of query terms the document contains. The coordination factor gives an AND-like boost to documents that contain more of the search terms than other documents.
queryNorm(q)	Normalization value for a query, given the sum of the squared weights of each of the query terms.

Index Structure

- Document
  - Grouping of content
- Field
  - Properties of the Document
- Term
  - Unit of indexing – often a word

- Index
- Segment
  - File – an index by it self
  - Lucene write segments incrementally



Phonetic Analysis

- Creates a phonetic representation of the text, for "sounds like" matching
- PhoneticFilterFactory. Uses one of
  - Metaphone
  - Double Metaphone
  - Soundex
  - Refined Soundex
  - Nysis

- Components of a Analyzer
  - CharFilters
  - Tokenizers
  - TokenFilters

CharFilters

- Used to clean up/regularize characters before passing to
- TokenFilter
- Remove accents, etc. MappingCharFilter
- They can also do complex things, we’ll look at
- HTMLStripCharFilter later.

Tokenizers

- Convert text to tokens (terms)
- Only one per analyzer
- Many Options
  - WhitespaceTokenizer
  - StandardTokenizer
  - PatternTokenizer
  - More…

TokenFilters

- Process the tokens produced by the Tokenizer
- Can be many of them per field