Hibernate Search

Googling your persistence domain model

Emmanuel Bernard
Doer
JBoss, a division of Red Hat

JBoss a division of Red Hat

Search: left over of today's applications

- Add search dimension to the domain model

"Frankly, search sucks on this project"

-- Anonymous' boss

Why? How to fix that? For cheap?

Integrate full-text search and persistent domain

- SQL Search vs Full-text Search
- Object model / Full-text Search mismatches
  - Demo
- Hibernate Search architecture
- Configuration and Mapping
- Full-text based object queries
- Some more features

SQL search limits

- Wildcard / word search
  - ‘%hibernate%’
- Approximation (or synonym)
  - ‘hybernat’
- Proximity
  - ‘Java’ close to ‘Persistence’
- Relevance or (result scoring)
- multi-”column” search

Full Text Search

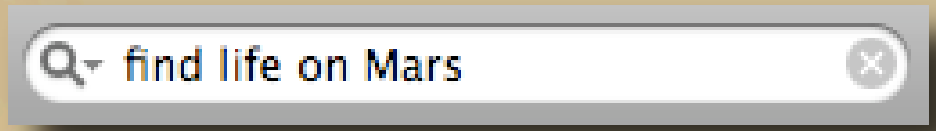
- Search information
  - by word
  - inverted indices (word frequency, position)
- In RDBMS engines
  - portability (proprietary add-on on top of SQL)
  - flexibility
- Standalone engine
  - Apache Lucene(tm) http://lucene.apache.org

Mismatches with a domain model

- Structural mismatch
  - full text index are text only
  - no reference/association between document
- Synchronization mismatch
  - keeping index and database up to date
- Retrieval mismatch
  - the index does not store objects
  - certainly not managed objects

Demo

Let's google our application!



Hibernate Search

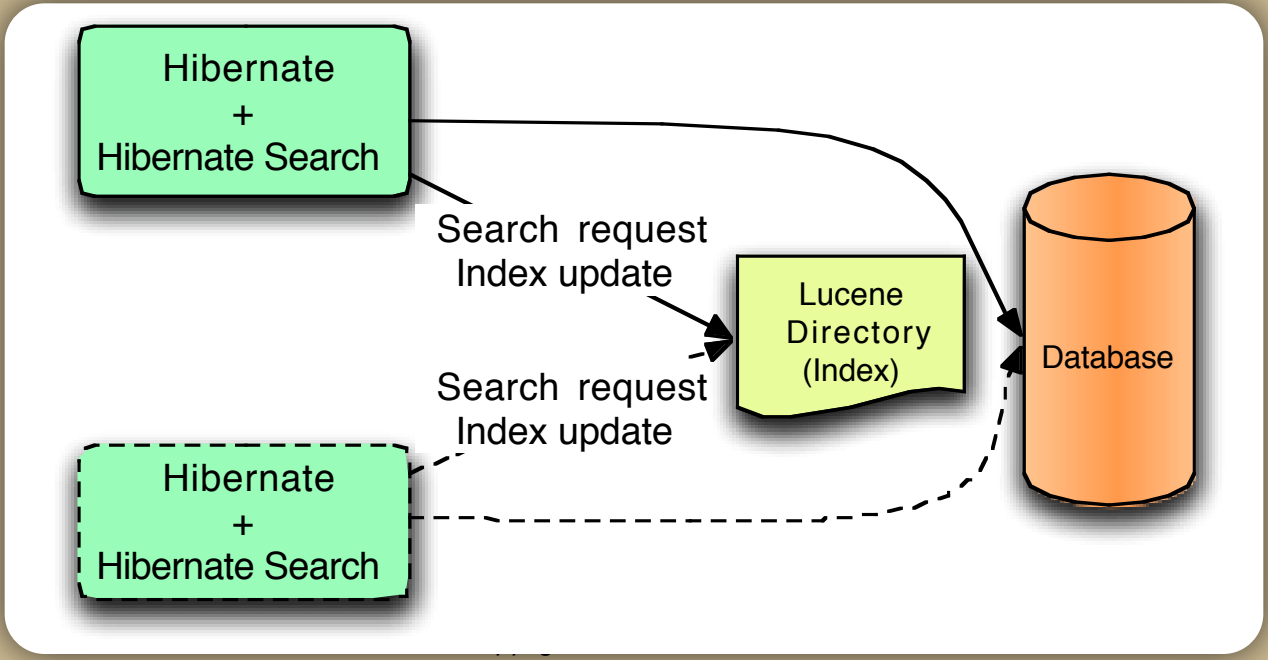
- Under the Hibernate platform
  - LGPL
- Built on top of Hibernate Core
- Use Apache Lucene(tm) under the hood
  - In top 10 downloaded at Apache
  - Very powerful but low level
  - easy to use it the “wrong” way
- Solve the mismatches

Architecture

- Transparent indexing through event system (JPA)
  - PERSIST / UPDATE / DELETE
- Convert the object structure into Index structure
- Operation batching per transaction
  - better Lucene performance
  - “ACID”-ity
  - (pluggable scope)
- Backend
  - synchronous / asynchronous mode
  - Lucene, JMS

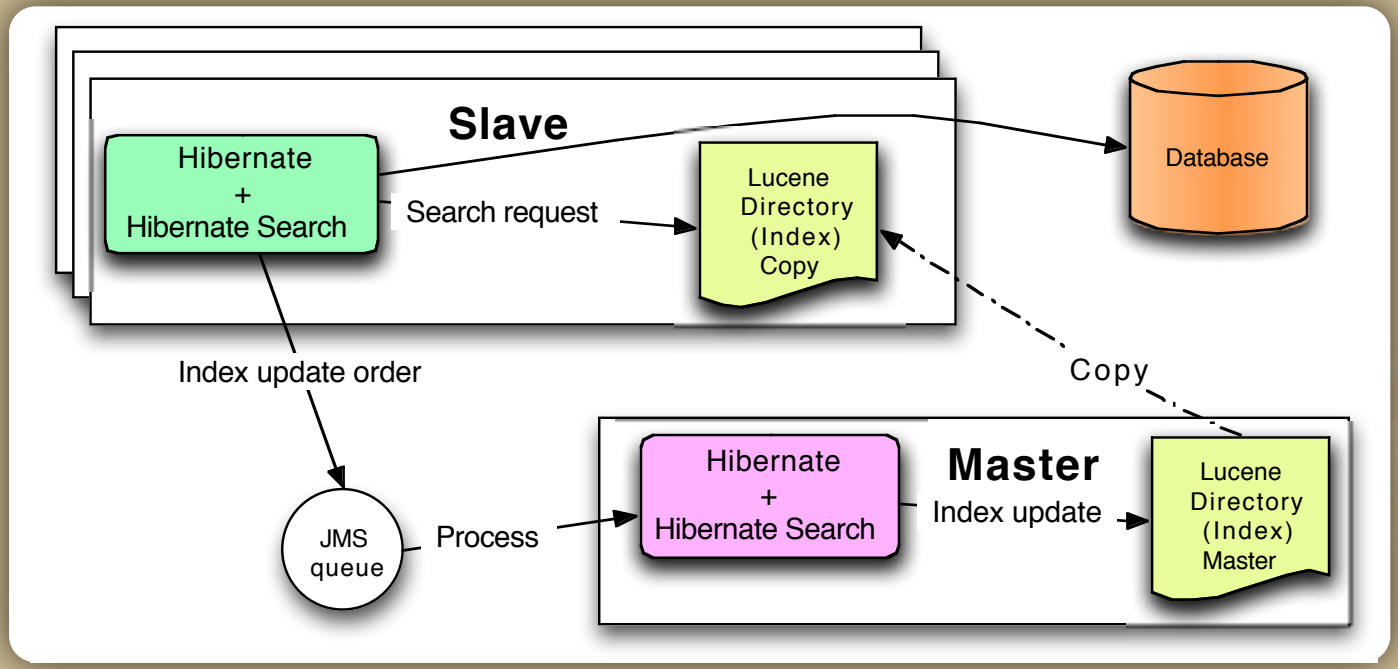
Architecture (Backend)

- Lucene Directory
  - standalone or symmetric cluster (limited scalability)
  - immediate visibility
  - can affect front end runtime



Architecture (Backend)

- JMS (Cluster)
  - Search processed locally / Change sent to master
  - asynchronous indexing (delay)
  - No front end extra cost / good scalability



Configuration and Mapping

- Configuration
  - event listener wiring
    - transparent in Hibernate Annotations
  - Backend configuration
- Mapping: annotation based
  - @Indexed
  - @Field(store, index)
  - @IndexedEmbedded
  - @FieldBridge

Query

- Retrieve objects, not documents
  - no boilerplate conversion code!
- Objects from the Persistence Context
  - same semantic as a JPA-QL or Criteria query
- Use org.hibernate.Query/javax.persistence.Query
  - common API for all your queries
- Query on correlated objects
  - “JOIN”-like query `"author.address.city:Atlanta"`

Query possibilities

- bring the “best” document first
- recover from typos
- recover from faulty orthography
- find from words with the same meaning
- find words from the same family
- find an exact phrase
- find similar documents

More query features

- Fine grained fetching strategy
- Sort by property rather than relevance
- Projection (both field and metadata)
  - metadata: SCORE, ID, BOOST etc
  - no DB / Persistence Context access
- Filters
  - security / temporal data / category
- Total number of results

Some more features

- Automatic index optimization
- Index sharding
- Manual indexing and purging
  - non event-based system
- Shared Lucene resources
- Native Lucene access

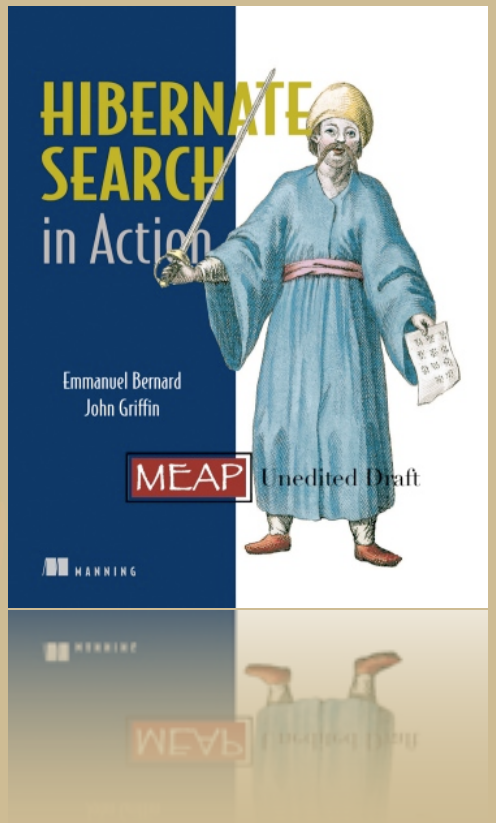
Full-Text search without the hassle

- Transparent index synchronization
- Automatic Structural conversion through Mapping
- No paradigm shift when retrieving data

- Clustering capability out of the box

- Easier / transparent optimized Lucene use

Q&A



For More Information

- Hibernate Search
  - http://search.hibernate.org
  - Hibernate Search in Action
- Apache Lucene
  - http://lucene.apache.org
  - Lucene In Action
- http://in.relation.to
- http://blog.emmanuelbernard.com