Scalable Lucene with Hibernate Search, Infinispan

Sanne Grinovero
Team Hibernate, Red Hat inc.
Who I am?

- R&D, Software Engineer at Red Hat
  - Hibernate team – focus on Search
  - Infinispan team – focus on Search and Query integrations
  - Hibernate OGM... soon

- Other open source contributions:
  - Apache Lucene/Solr
  - JGroups
  - Seam
  - Linux...
Agenda

• The search problem: intro to Lucene
• Lucene with JPA applications: Hibernate Search
• New features of Search
• Lucene on Infinispan (demo)
• Coming soon
  • Hibernate OGM
The Search problem
The Search problem

- Who searches, doesn't know what he is searching for:
  - Please, don't ask the user to give up the primary key
  - Doesn't know the exact content of the document either
Hibernate model example

```java
@Entity
public class Actor {
    @Id @GeneratedValue
    private Integer id;
    private String name;
}

@Entity
public class Product {
    @Id @GeneratedValue
    private Integer id;
    private String title;
    @ManyToMany
    private Set<Actor> actors = new HashSet<Actor>();
}
```
The stupid search engine

```java
String title = "\%" + titleParam + "\%";
String actor = "\%" + actorParam + "\%";

em.createQuery(
    "select distinct from Product p JOIN p.actors a" + 
    "where p.title like :title and a.name like :actor" + 
    "order by p.title" )
.setParameter( "title", title )
.setParameter( "actor", actor );
```
The dumb search engine

String title = (title == null) ? "" : "%" + title.toLowerCase() + "%";
String actor = (actor == null) ? "" : "%" + actor.toLowerCase() + "%";

em.createQuery("select distinct p from Product p JOIN p.actors a " +
  "where lower(p.title) like :title " +
  "and lower(a.name) LIKE :actor order by p.title")
 .setParameter("title", title)
 .setParameter("actor", actor));
Does it work? How about these:

```java
String author = "Fabrizio De André"
String title = "Nuvole barocche"
List<Product> list = s.createQuery("...").list();
```

```java
String author = "De André, Fabrizio"
String title = "Nuvole barocche"
List<Product> list = s.createQuery("...").list();
```

```java
String author = "De Andre, Fabrizio"
String title = "Nuvole barocche"
List<Product> list = s.createQuery("...").list();
```
More requirements

- Unique search input
  - Might contain either/both author, title names
  - Both entities might be composed of multiple terms

- Relevance
  - Products matching both should be listed on top
  - Exact word matches should be scored better
    - So you need approximate word matches?
    - How about typos?
List<Prodotti> list = s.createQuery("...").setParameter("F. de André nuvole barocche").list();

• Mixed case, accents
• Relative order of terms, distance
• Abbreviations, typos
• Match on multiple fields
• 18,800 results in 0.41 seconds
More useful stuff:

- Similarity:
  - *hibernate* ~ *hybernat*

- Proximity, synonyms, abbreviations:
  - 'JPA' or 'Java Persistence API'

- Boosting, field adjustments:
  - A match in the title is more “worth” than in the text content?

- Stemming (is language specific!)
Assuming Google returned results in alphabetical order, would you like it?

“hibernate search”
About 3.580.000 results (0.04 seconds)
So what..

- The database is not a good fit
  - SQL is not appropriate for the task
  - Still SQL is very handy for other tasks
- Need to use the best tool for each task:
  - Relational databases
  - filesystems
  - Fulltext search engines
  - key-value stores
- Need to integrate them, keep data integrity and consistency
Please welcome the savior: Apache

- Open source Apache™ top level project,
  - In the “top 10” of most downloaded and active projects
- Very advanced implementation
- Main language is Java, some ports exists in other languages
- Rich open source environment around it
- Many products embed it
Product examples using Apache Lucene
Apache Lucene

- Similarity
- Sinonyms
- Stemming
- Stopwords
- TermVectors
- MoreLikeThis
- Faceted Search
- Speed!
Similarity

- N-Grams (edit distance)
- Phonetic (Soundex™)
- Any custom...

Cagliari ~ càliari
Cagliari ~ cag agl gli lia ari
Cagliari ~ CGRI
Lucene: Sinonyms (or close)

- Can be applied at “index time”
- at “query time”
- Requires a vocabulary
  - WordNet

newspaper ~ daily ~ journal
Journal ??~ newspaper
Lucene: Stemming

continuait ~ continu
continuation ~ continu
continué ~ continu
continuelle ~ continu\textit{uel}
Lucene: Stopwords

- Removes terms which are so frequently used that they are not suited as search keywords – might depend on your domain!

a, able, about, across, after, all, almost, also, am, among, an, and, any, are, as, at, be, because, been, but, by, can, cannot, could, dear, did, do, does, either, else, ever, every, for, from, get, got, had, has, have, he, her, hers, him, his, how, however, i, if, in, into, is, it, its, just, least, let, like, likely, may, me, might, most, must, my, neither, no, nor, not, of, off, of ten, on, only, or, other, our, own, rather, said, say, says, she, should, since, so, some, than, that, the, their, them, then, there, these, they, this, tis, to, too, twas, us, wants, was, we, were, what, when, where, which, while, who, whom, why, will, with, would, yet, you, your
Apache Lucene: Index

- It requires an Index
  - On filesystem
  - In memory
  - ...
- Made of immutable segments
  - Optimized for search speed, not for updates
- A world of strings and frequencies
So, about integrating with my database..

- The index structure is deeply different than a relational database – not all is possible
- You need to keep the data in sync
  - In case they are not, which one should be trusted more?
- How do queries look like?
- What do queries return?
Different worlds

- A Lucene Document, is unstructured (schemaless), something close to
  
  $Map<String, String>$
  
- An Hibernate model is structured to be functional as representation of your business model

- Entities returned by an EntityManager or Hibernate Session are managed, to keep the database in sync

- A bridge is needed
The data mismatch

Object world

- Film
  - title: The Godfather
  - <<actors>>

- Actor
  - name: Al Pacino

- Actor
  - name: Marlon Brando

Index world

- Film Document
  - title: The Godfather
  - actor: Al Pacino
  - actor: Marlon Brando
The architectural mismatch

Diagram:
- Application Framework
  - ORM
    - DB
  - Lucene
    - Fulltext index
- Notification of changes via Session or EntityManager
Quickstart Hibernate Search

• Add hibernate-search dependency:

```xml
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-search</artifactId>
  <version>3.3.0.Final</version>
</dependency>
```
Quickstart Hibernate Search

- All the rest of configuration is optional:
  - Where we store indexes
  - Extension modules, custom analyzers
  - Performance tuning
  - Advanced mapping
  - Clustering
    - JGroups
    - Infinispan
    - JMS
Quickstart Hibernate Search

@Entity
public class Essay {
    @Id
    public Long getId() { return id; }

    public String getSummary() { return summary; }
    @Lob
    public String getText() { return text; }
    @ManyToOne
    public Author getAuthor() { return author; }

    ...
}
@Entity @Indexed
public class Essay {
    @Id
    public Long getId() { return id; }

    public String getSummary() { return summary; }
    @Lob
    public String getText() { return text; }
    @ManyToOne
    public Author getAuthor() { return author; }

    ...
}
Quickstart Hibernate Search

@Entity  @Indexed
public class Essay {
    @Id
    public Long getId() { return id; }
    @Field
    public String getSummary() { return summary; }
    @Lob
    public String getText() { return text; }
    @ManyToOne
    public Author getAuthor() { return author; }
    ...

Quickstart Hibernate Search

@Entity  @Indexed
public class Essay {
    @Id
    public Long getId() { return id; }
    @Field
    public String getSummary() { return summary; }
    @Lob @Field @Boost(0.8)
    public String getText() { return text; }
    @ManyToOne
    public Author getAuthor() { return author; }
    ...

Quickstart Hibernate Search

@Entity @Indexed
public class Essay {
    @Id
    public Long getId() { return id; }
    @Field
    public String getSummary() { return summary; }
    @Lob @Field @Boost(0.8)
    public String getText() { return text; }
    @ManyToOne @IndexedEmbedded
    public Author getAuthor() { return author; }
...

Another example

```java
@Entity
class Author {
    @Id @GeneratedValue
    private Integer id;
    private String name;
    @OneToMany
    private Set<Book> books;
}

@Entity
class Book {
    private Integer id;
    private String title;
}
```
Index structure

@Entity @Indexed
public class Author {
    @Id @GeneratedValue
    private Integer id;
    @Field(store=Store.YES)
    private String name;
    @OneToMany
    @IndexedEmbedded
    private Set<Book> books;
}

@Entity
public class Book {
    private Integer id;
    @Field(store=Store.YES)
    private String title;
}

<table>
<thead>
<tr>
<th>Field</th>
<th>ITSVopfOLBC</th>
<th>Norm</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_hibernate_class</code></td>
<td>I-S-----O----</td>
<td>---</td>
<td>com.sourcesense.Author</td>
</tr>
<tr>
<td><code>books.title</code></td>
<td>ITS---------</td>
<td>1.0</td>
<td>Blaze</td>
</tr>
<tr>
<td><code>books.title</code></td>
<td>ITS---------</td>
<td>1.0</td>
<td>IT</td>
</tr>
<tr>
<td><code>id</code></td>
<td>I-S-----O----</td>
<td>---</td>
<td>1</td>
</tr>
<tr>
<td><code>name</code></td>
<td>ITS---------</td>
<td>0.625</td>
<td>Stephen King</td>
</tr>
</tbody>
</table>
String[] productFields = {"summary", "author.name"};

Query luceneQuery = // query builder or any Lucene Query

FullTextEntityManager ftEm =
        Search.getFullTextEntityManager(entityManager);

FullTextQuery query =
        ftEm.createFullTextQuery( luceneQuery, Product.class );

List<Product> items =
        query.setMaxResults(100).getResultList();

int totalNbrOfResults = query.getResultSize();

TotalNbrOfResults= 8,320,000
(0.20 seconds)
Creating a Lucene Query with the DSL

```java
QueryBuilder productQb = fullTextSession
    .getSearchFactory()
    .buildQueryBuilderFactory()
    .forEntity( Product.class )
    .get();

Query fullTextQuery = productQb
    .keyword()
    .onField( "name" ).andField( "author" )
    .matching( userInput )
    .createQuery();
```
Results

- Managed Pojo: updates are applied to both Lucene and database
- JPA pagination, known APIs:
  - `.setMaxResults(20).setFirstResult(100);`
- Type restrictions, polymorphic fulltext queries:
  - `.createQuery(luceneQuery, A.class, B.class, ..);`
- Projection
- Result mapping
Filters

```java
FullTextQuery ftQuery = s // s is a FullTextSession
    .createFullTextQuery( query, Product.class )
    .enableFullTextFilter( "minorsFilter" )
    .enableFullTextFilter( "specialDayOffers" )
      .setParameter( "day", "20110218" )
    .enableFullTextFilter( "inStockAt" )
      .setParameter( "location", "sophia-antipolis" );
List<Product> results = ftQuery.list();
```
Filters

Raw results

Filter 1

Filter 2

Final Results

Doc 1
Doc 2
Doc 3
Doc 4
Doc 5
Doc 6
Doc 7
Doc 8

Doc 1
Doc 2
Doc 4
Doc 5
Doc 6
Doc 8

Doc 1
Doc 4
Advanced text analysis

@Entity  @Indexed

@AnalyzerDef(name = "frenchAnalyzer", tokenizer =
    @TokenizerDef(factory=StandardTokenizerFactory.class), filters = {
        @TokenFilterDef(factory = LowerCaseFilterFactory.class),
        @TokenFilterDef(factory = SnowballPorterFilterFactory.class, params = {
            @Parameter(name = "language", value = "French")
        })
    })

public class Book {

    @Field(index=Index.TOKENIZED, store=Store.NO)
    @Analyzer(definition = "frenchAnalyzer")
    private String title;

More...

- @Boost e @DynamicBoost
- @AnalyzerDiscriminator
- @DateBridge(resolution=Resolution.MINUTE)
- @ClassBridge e @FieldBridge
- @Similarity
- Automatic Index optimization
- Sharding, sharding filters
Clustering using a queue

Diagram showing the process of clustering using a queue, with Lucene readers (slaves) indexing work and a Lucene writer (master) managing the master index. Each slave copies from the source database.
New features

- Query DSL
- Improved performance
  - Lucene 3
  - NumericField
  - FieldCache
- Programmatic mapping
- Faceting
- Infinispan clustering
Infinispan

- Open source highly scalable data grid platform
  - Distribution or Replication
  - Sync or Async
  - Transactional
  - Persists contents using a CacheLoader
    - Write-through or write-behind
    - Shared or per cluster node
  - Hibernate second-level cache
  - State of the art eviction strategies
Infinispan on clouds

- Cloud-friendly autodiscovery strategies
  - S3_PING, JDBC_PING, TCP_PING
- Fully dynamic clustering
  - Add or kill nodes on the fly
  - Using a CacheLoader, scale back to zero nodes without data loss
- Very high performance
  - Configurable reliability
  - Configurable consistency
Shared memory, shared index
Demo: cluster autodiscovery
Per-node storage
Shared storage

Infinispan distributed index

Lucene+App

Server

Lucene+App

Server

Lucene+App

Server

Lucene+App

Server

DB
Let's see this in action: now storing an Index in the database, write-behind configuration
Hibernate Search with Infinispan

- Infinispan provided components:
  - Query
  - Lucene Directory
  - Map/Reduce API
- Hibernate:
  - Infinispan 2\textsuperscript{nd} level cache
- Hibernate Search:
  - Infinispan based DirectoryProvider
  - Queues (JMS, JGroups) for clustered writing
What is coming next?

- True write distribution
- Distributed Query via Infinispan's Map/Reduce API
- Totally independent from Hibernate core – reusable engine
- Hibernate OGM
Merci!
Questions?

http://search.hibernate.org
  • Hibernate Search in Action
    [Manning]

http://lucene.apache.org
  • Lucene In Action (2°ed)
    [Manning]

http://www.infinispan.org

http://in.relation.to

http://forum.hibernate.org

twitter.com/SanneGrinovero
twitter.com/Hibernate
twitter.com/Infinispan