Full-text Search with NoSQL Technologies
NoSQL Search Roadshow 2013, Berlin

Kai Spichale
About me

- Kai Spichale
- Software Engineer at adesso AG
- NoSQL, Full-text searching, Spring, Java EE

- adesso is among Germany’s top IT service providers
- Consulting and software development focus
- More than 1,000 members of staff
- Some of the most important customers are Allianz, Hannover Rück, Westdeutsche Lotterie, Zurich Versicherung, DEVK, and DAK
<table>
<thead>
<tr>
<th>NoSQL</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Exponential data growth</td>
<td>- Shift in data access:</td>
</tr>
<tr>
<td>- Semi-structured data</td>
<td>&gt; More full-text search</td>
</tr>
<tr>
<td>- More connections</td>
<td>&gt; Higher user expectations</td>
</tr>
<tr>
<td>- 80 percent of business-relevant</td>
<td>&gt; Keyword search and link directories</td>
</tr>
<tr>
<td>information is in unstructured form</td>
<td>become impractical</td>
</tr>
</tbody>
</table>

- 80 percent of business-relevant information is in unstructured form.
Agenda

- Lucene full-text search
- NoSQL:
  - Architectural drivers
  - MongoDB
  - Neo4j
  - Apache Cassandra
  - Apache Hadoop
- Summary
Techniques for searching documents in collections

grep-like naive approach:
  > Serial scanning is slow
  > No negation
  > No distinction between phrase and keyword search

Build inverted index
  > Term $\rightarrow$ Document
  > Contains references to documents for each token
Apache Lucene

- Java lib for full-text searches
- De facto standard for open source software

- Attributes:
  - Application-agnostic
  - Scalable, high performance

- Features:
  - Ranked searching
  - Multiple query types, faceting
  - Sorting
  - Multi-Index searching
Text Analysis

Documents

Extraction Parsing

Character Filter

Tokenizer

Token Filter

Inverted Index

**de.GermanAnalyzer:**
StandardTokenizer > StandardFilter
> LowerCaseFilter > StopFilter > GermanStemFilter
Eat your own dog food.

First come, first served.

The exception proves the rule.

Stop word List

<table>
<thead>
<tr>
<th>ID</th>
<th>Term</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>come</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>dog</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>eat</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>exception</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>first</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>food</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>own</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>prove</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>rule</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>serve</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>your</td>
<td>1</td>
</tr>
<tr>
<td>Type</td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>+adesso –italy</td>
<td></td>
</tr>
<tr>
<td>(MUST, MUST_NOT, SHOULD)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phrase</td>
<td>„foo bar“</td>
<td></td>
</tr>
<tr>
<td>Wildcard</td>
<td>fo*a?</td>
<td></td>
</tr>
<tr>
<td>Fuzzy</td>
<td>fobar~</td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>[A TO Z]</td>
<td></td>
</tr>
</tbody>
</table>
Agenda

- Lucene full-text search
- **NoSQL:**
  - Architectural drivers
  - MongoDB
  - Neo4j
  - Apache Cassandra
  - Apache Hadoop
- Summary
One size fits all approach

- Which NoSQL store satisfies our requirements best?
- Is full-text search supported?
Let’s take a closer look on:

- MongoDB
- Neo4j
- Apache Cassandra
- Apache Hadoop
Document-oriented Databases

- Designed for storing and retrieving documents
- Semi-structured content such as BSON documents

```json
{
   "_id" : ObjectId("42"),
   "firstname" : "John",
   "lastname" : "Lennon",
   "address" : {
      "city" : "Liverpool",
      "street" : "251 Menlove Avenue"
   }
}
```
MongoDB

- Supports ad-hoc CRUD operations
  
  ```
  db.things.find({firstname:"John"})
  ```

- Server-side execution of JavaScript
- Aggregations, MapReduce

- Simple keyword search with multikey indexes:
  - Index array content as separate entries

```javascript
{
  article : "some long text",
  _keywords : [ "some" , "long" , "text"]
}
```
Version 2.4 supports text indexes
Language-specific stemming based on Snowball

```
db.foo.runCommand("text", {search: "adesso -italy", language: "english"})
```

Still a beta feature
MongoDB

- Mongo Connector integrates MongoDB with another system (backup MongoDB cluster, Solr, elasticsearch)
- System architecture with separate search engine possible
## Choosing the Right Approach

<table>
<thead>
<tr>
<th></th>
<th>MongoDB</th>
<th>MongoDB + Search Engine</th>
<th>Search Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ No result set merging</td>
<td>✓ Full-text search with faceting</td>
<td>✓ No result set merging</td>
<td>✓ Full-text search with faceting</td>
</tr>
<tr>
<td>✓ Complex queries with aggregations</td>
<td>✓ Complex queries with aggregations</td>
<td>✓ No result set merging</td>
<td>✓ Full-text search with faceting</td>
</tr>
<tr>
<td>❑ Simple text search (but experimental text index)</td>
<td>❑ Result set merging ❑ Increased complexity (ops, dev)</td>
<td>❑ Backup? ❑ Aggregations?</td>
<td></td>
</tr>
</tbody>
</table>
Graph Databases

- Stored data is represented as graph structures
  - Nodes
  - Edges (Relationships)
  - Properties

- Universal datamodel

- Traversing

- Example: Neo4j

```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>George</td>
</tr>
<tr>
<td>3</td>
<td>Paul</td>
</tr>
</tbody>
</table>

friend

id=1
name="John"

id=2
name="George"

id=3
name="Paul"

friend

friend
Traversing

- Visiting nodes by following relationships
- Breadth- and depth-first traversing
- Gremlin, Cypher

```plaintext
START john=node:peoplesearch(name='John')
MATCH john<-[:friend]->afriend RETURN afriend
```

Result = George
Database itself is a natural index consisting of its edges and nodes

Example: „name“, „city“

```
personRepository.findByPropertyValue("name", "John");
```

Auto indexing keeps track of property changes
The default separate index engine used is Apache Lucene

```java
@NodeEntity
class Person {
    @Indexed(indexName="peoplesearch", indexType=IndexType.FULLTEXT)
    private String name;
    ..
}

Index<PropertyContainer> index = template.getIndex("peoplesearch");
index.query("name", "Jo*");
```
Wide Column Store

- Google BigTable: "a sparse, distributed multi-dimensional sorted map"
- Data is organized in rows, column families, and columns
- Ideal for sharding (horizontal partitioning)

**Different columns per row**

```
unique row keys

pmccart | address | name
---------|---------|------
"Liverpool .." | "McCartney"

j Lennon | address | name | state
---------|---------|------|-------
"Liverpool .." | "Lennon" | "UK"

gharris | name
---------|------
"Harrison"
```
Apache Cassandra

- BigTable clone
- Distributed Hash Table (Amazon Dynamo)
- Eventual consistency (configurable levels)

- Cassandra Query Language (CQL) = SQL dialect without joins

```
SELECT name FROM user WHERE firstname="John";
```

- Hadoop integration
Solandra = Solr using Cassandra as backend

DataStax Enterprise Search
- One local Solr instance per Cassandra node
- Integration is based on secondary index API
- CQL supports Solr Queries

```
SELECT title FROM solr WHERE solr_query='name:jo*';
```

- Cassandra’s ring information is used to construct Solr distributed search queries
Agenda

- Lucene full-text search
- NoSQL:
  - Architectural drivers
  - MongoDB
  - Neo4j
  - Apache Cassandra
  - Apache Hadoop
- Summary
Apache Hadoop

- **Hadoop:**
  - Framework for distributed processing of large data sets in computer clusters
  - Distributed filesystem + MapReduce implementation

- Scalable and reliable platform of a comprehensive data analysis ecosystem
Hadoop MapReduce

- **Map Phase:**
  - Records are processed by map function

- **Shuffle Phase:**
  - Distributed sort and grouping

- **Reduce Phase:**
  - Intermediate results are processed by reduce function
Data is processed by mappers and reducers

map\((k, v)\) \(\rightarrow\) [\((K1, V1), (K2, V2), \ldots\) ]

reduce\((Kn, [V_i, V_j, \ldots])\) \(\rightarrow\) \((Km, R)\)
What kind of problems does MapReduce solve?

- Problems processed without reducer
  - Searching
  - File converting
  - Sorting
  - Map-side join

- Problem processed with reducer
  - Grouping and aggregation
  - Reduce-side join

- More complex problems:
  - Solved by combinations of multiple MapReduce jobs
Search document including “A”

Mapper emits only documents that fit the searching criteria

Documents

1: A,B,C
2: D,E
3: B,E
4: A,D
5: A,C,E

Result = 1, 4, 5
**Hadoop MapReduce: Indexing**

- **HDFS:**
  - Stores raw data

- **Mapper:**
  - Extracts text (creates e.g. SolrInputDocument)
  - Calls Lucene for indexing (calls e.g. StreamingUpdateSolrServer)
Hadoop MapReduce: Indexing

@Override
public void map(
    LongWritable key, Text val,
    OutputCollector<NullWritable, NullWritable> output,
    Reporter reporter)
    throws IOException {

    st = new StringTokenizer(val.toString());
    lineCounter = 0;

    while (st.hasMoreTokens()) {
        doc = new SolrInputDocument();

        doc.addField("id", fileName + key.toString() + lineCounter++);

        doc.addField("txt", st.nextToken());

        try {
            server.add(doc);
            server.add(doc);
        } catch (Exception exp) {
            ...}
    }
}
Apache Tika

- Extracts metadata and structured text content
  - HTML, MS Office documents, PDF, etc.
- Stream parser can process large files
Lucene is only a library, not a standalone search engine

Complete search engines:
- Solr
- ElasticSearch
Apache Flume

- Distributed service for collecting, aggregating and moving large amounts of data (e.g. log data)
- Streaming techniques
- Fault tolerant

Web Server, Applikations → Flume → HDFS → Tika → MapReduce Job → Elasticsearch → Lucene → Lucene Index
Alternatives

- Nutch Crawler creates one entry in CrawlDB per URL
- Hadoop DistCp copies data within and between hadoop systems
- Apache Sqoop transfers bulk data between Hadoop and RDMBS
Fundamental mismatch:
- MapReduce for batch processing
- Lucene for interactive searching
- MapReduce for indexing large datasets
- Basis for (offline) BigData solutions
Summary

- More semi-structured data
- Increasing relevance of full-text searching

- Combination of NoSQL and Lucene:
  - MongoDB: integration via MongoDB Connector
  - Neo4j: native Lucene integration
  - Cassandra: Datastax‘s Solr integration
  - Hadoop: indexing large datasets with MapReduce

- Alternative: search engine as document-oriented database
Thank you for your attention!