ADVANCED DATABASES
CIS 6930
Dr. Markus Schneider

Apache Solr

Group 21

Prarabdh Joshi
Himanshu Vyas
Mark Steele
Jiangjiang Zhu
What is Solr?

- Solr is an Open Source Search Platform, built on top of Lucene Java Search Library.
- It exposes the Lucene Java API as REST-Full Services
- Indexing in Solr can be done via XML, JSON, CSV or Binary over HTTP protocol.
- Solr provides essential configurations to make data extraction simple even from Rich Documents like pdfs, presentations, Doc files and spreadsheets.
- Queries are made using HTTP GET Method and the results are retrieved in XML, JSON, CSV or Binary Format.
History

• Solr was created by “Yonik Seeley” at CNET Networks in 2004.

• Basically Developed as a In-House project, aimed at adding Search Capabilities to the Company’s Website.

• It Initially had just a Master-Slave architecture, limiting it to small data sets with Scalability Issues.

• In 2006, CNET released it’s source code to Apache Software Foundation under the Lucene Top Level Project.

• In 2008, Solr 1.3 was released with added features including Distributed Search Capabilities.

• The latest version 6 of Solr was released in April 2016, adding support for executing parallel SQL queries and SolrCloud Collections.
Features in a Nutshell

• Advanced Full Text Search Capability
• Faceted Navigation through the Retrieved Data
• Optimization for High Value Web Traffic
• HTML administration interface
• Distributed Search through Sharding
• Auto Suggest and Auto Completion
More Features

- Automated Indexing of Distributed Documents
- JSON, XML, PHP, Ruby, Python and custom Java binary output formats over the HTTP protocol.
- Built-in security: Authentication, Authorization, SSL
- Near Real Time Search
- High Availability for Writes
- Auto Index Replication
- Extensive Plug In Architecture
Lucene/Solr Architecture

- Request Handlers:
  - /admin
  - /select
  - /spell

- Response Writers:
  - XML
  - Binary
  - JSON

- Update Handlers:
  - XML
  - CSV
  - binary

- Search Components:
  - Query
  - Spelling
  - Faceting
  - More like this
  - Distributed Search
  - Highlighting
  - Statistics
  - Debug
  - Clustering

- Core Search:
  - Core Search
  - IndexReader/Searcher

- Apache Lucene:
  - Apache Lucene
  - Text Analysis

- Indexing:
  - Indexing
  - IndexWriter

- Update Processors:
  - Signature
  - Logging
  - Indexing
  - Query Parsing
  - Analysis
  - Highlighting

- Extracting Request Handler (PDF/WORD)
- Apache Tika
- Data Import Handler (SQL/RSS)
- Index Replication
Solr Schema Hierarchy

- Solr Instance
  - Core/Index
    - Documents
      - Field
      - Field
    - Core/Index
    - Core/Index
  - Indexing & Querying
    - Schema.xml
Why Indexing?

• Indexing Collects, parses and stores Data for Information Retrieval
• It helps in optimizing Speed and Performance for relevant data search
• Without Indexing, Search Engines would scan every Document in the staple, requiring considerable time and computing
An Example for Tokenization:

“Group 21”

Tokenization

“Group”
Pos: 0
Offset: 0
Len: 5

Term Vector 1

“21”
Pos: 1
Offset: 6
Len: 2

Term vector 2
Writing to Index: The Lucene Way

Document

Field 1
Field 2
Field 3
Field 4

Analyzer

IndexWriter

Directory
Searching In Lucene

Expression → QueryParser → IndexSearcher

Analyzer

Query Object
NOSQL DATABASE EXAMPLES

- HBase
- CouchDB
- Redis
- Cassandra
- MongoDB
- RavenDB
- Membase
- Riak
- Elasticsearch
- Oracle Coherence
- Apache Solr
- Neo4j
Solr Data Model
Fields

- Can be compared to a RDBMS column
- Fields can contain different kinds of data.
- Field types tell Solr how to interpret data

```xml
<fields>
  <field name="id" type="string" indexed="true" stored="true" required="true" />
  <field name="name" type="text" indexed="true" stored="true" />
...
</fields>
```
FieldType

• Determines type of a field e.g. string, text etc.
• Associated with Lucene class
• Indexing rules are defined for FieldType

```xml
<fieldType name="text" class="solr.TextField">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StandardFilterFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPorterFilterFactory"/>
  </analyzer>
</fieldType>
```
The Document

- Represents basic and atomic unit of information in Solr
- Composed of fields
Similarities with RDBMS record

- A document can have a primary key
- A document has a structure consisting of one or more fields
Differences with RDBMS record

• Fields can be multivalued whereas a column in a database table can have only one value.
• Fields either have a value or don't exist at all. There's no notion of NULL value in Solr.
• Field names can be static or dynamic, but table columns in a database must be explicitly declared in advance.
The Inverted Index

• designed and optimized to allow fast searches at retrieval time
• consists of an ordered list of all the terms that appear in a set of documents
Inverted Index example

Let's consider 3 documents

{ 
	"id": 1, "title": "The Birthday Concert" 
},
{ "id": 2, "title": "Live in Italy" },
{ "id": 3, "title": "Live in Paderborn" } 
}
### Inverted Index example (contd.)

<table>
<thead>
<tr>
<th>Terms</th>
<th>Document Ids</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Birthday</td>
<td>X</td>
</tr>
<tr>
<td>Concert</td>
<td>X</td>
</tr>
<tr>
<td>Italy</td>
<td>X</td>
</tr>
<tr>
<td>Live</td>
<td>X</td>
</tr>
<tr>
<td>Paderborn</td>
<td></td>
</tr>
<tr>
<td>The</td>
<td>X</td>
</tr>
<tr>
<td>In</td>
<td></td>
</tr>
</tbody>
</table>
The Solr Core

• is a container for a specific inverted index

• The index configuration of a given Solr instance resides in a Solr core

• On the disk, Solr cores are directories, each of them with some configuration files that define features and characteristics of the core.

• A Solr application can have 0 or more cores
Text Analysis

• Three main concepts in analysis
  - Analyzers
  - Tokenizers
  - Filters
Analyzers

• Are used both during, when a document is indexed and at query time
• Same analysis process need not be used for both operations
• An analyzer examines the text of fields and generates a token stream
• May be a single class or may be composed of a series of tokenizer and filter class
 TokenType 

- The job of a tokenizer is to break up a stream of text into tokens/terms (TokenStream objects).
- Characters in the input stream may be discarded, such as whitespace or other delimiters.
Filters

- Examine a stream of tokens and decides whether to pass it along, replace it or discard it.
- Filters consume one TokenStream and produce a new TokenStream, they can be chained one after another indefinitely

```xml
<fieldType name="text" class="solr.TextField">
    <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.StandardFilterFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.EnglishPorterFilterFactory"/>
    </analyzer>
</fieldType>
```
Solr Query
Search Document

- q
- fq
- start
- row
- sort
- fl
- wt
Solr Query Syntax

- **Keyword Matching**
  - title: foo
  - title: “foo bar”
  - title: foo -title: bar

- **Wildcard Matching**
  - title: foo*
  - title: foo*bar

- **Range Search**
  - Mod_data:[20150101 TO 20160101]

- **Boosts**
  - (title:foo OR title:bar)^1.5 (body:foo OR body:bar)
Fuzzy & Proximity Search

- Fuzzy Search
  title: “computer”~0.5

- Proximity Search
  title: “foo bar”~2
    foo abc def bar
Faceting

- facet.query
- facet.field
- facet.mincount -> f.<field.name>.facet.mincount
- facet.limit -> f.<field.name>.facet.limit
- facet.offset -> f.<field.name>.facet.offset
- facet.sort count, facet.sort index
- tagging & excluding filter
- facet.range
- facet.range.start
- facet.range.finish
- facet.range.gap
Faceting
Highlighting

hl = true
simple.pre
simple.post
“highlighting” {
    “37477”: {
        “Name”: [“Apple <em>iPhone</em> 6s”]
    }
}
### Highlighting

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple iPhone 6S</td>
<td>2,549 TL</td>
<td>hepsiburada.com</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,599 TL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gittigidiyor.com</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,649 TL</td>
</tr>
<tr>
<td>Apple iPhone 5S</td>
<td>1,364 TL</td>
<td>hepsiburada.com</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,599 TL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gittigidiyor.com</td>
</tr>
</tbody>
</table>

**Specifications**

- **Apple iPhone 6S**
  - 3G - 4G - Wi-Fi
  - 12 MP
  - iOS 9 İşletim Sistemi

- **Apple iPhone 5S**
  - 8.0 MP
  - Akıllı Telefon
  - Eller Serbest - Titreşim - Ses Kaydı - Sesli Arama - Uçuş Modu - Qwerty Klavye
  - iOS 7 İşletim Sistemi
  - 3G - 4G - Wi-Fi - 4.5G
Other Query Features

- **spelling check**
  
  spellcheck.q=Keyword&spellcheck=on

- **grouping**

  group=true&group.field=year
Application & API

- post command -c coreName -p port
- Rest API
- SolrJ, Spring Data Solr, or other libraries
- DataImportHandler
Application & API
Scalability

• Designed to work under heavy search traffic
• Able to quickly find results with indexed searches
• Is very flexible depending on how many indexes you have
• Can be easily scaled to the user’s needs
• Can use a variety of scaling techniques (horizontal, vertical, replication, sharding, and cloud)
• Able to handle high query volume, and large index size
Single Server

- Best to maximize a single server before expanding horizontally or vertically
- Manage index through stop words and term frequencies
- Make use of cache and optimize it
Replication

- Used to handle high query volume
- Uses slaves to help search for indexes
- Used to scale horizontally
- Master takes snapshots and distributes new images
Sharding

- Used to handle a large amount of indexes
- Each system performing a search
- Suffers from excessive chatter
- Not ideal large scale scaling
- Ideal to balance requests per shard
Replication+Sharding

- Used when the index is too large for a machine, as a high query volume.
- Master shards do not communicate with each other
- Allows for fault tolerance using load balancing software
Solr Cloud

- Contains high fault tolerance
- High availability
- Central configuration for the entire cluster
- Automatic load balancing and fail-over for queries
- ZooKeeper integration for cluster coordination and configuration
- Flexible distributed search and indexing
Solr Cloud ZooKeeper

- Used to manage nodes for SolrCloud
- Keeps track of changes made
- Needs $2xF+1$ machines, to ensure requests can be served even on failure
Shards and Indexing Data in SolrCloud

• Automatic document distribution and indexing

• Can use the router to hash documents to shards, such as “q=solr&_route_=IBM!”

• Able to split shards even after the initial declaration of shards using CollectionAPI
Collection API Shard Splitting


```xml
<lst>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
  </lst>
  <str name="core">anotherCollection_shard1_1_replica1</str>
  <str name="status">EMPTY_BUFFER</str>
</lst>

<lst>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
  </lst>
  <str name="core">anotherCollection_shard1_0_replica1</str>
  <str name="status">EMPTY_BUFFER</str>
</lst>
```
Fault Tolerance

Write Tolerance
• Node uses leader to update shards
• Nodes keep track of updates with Transaction Log

Read Tolerance
• Only needs one available replica
• Can read partial results
Read Fault Tolerance

Fault Tolerance

```json
{
  "responseHeader": {
    "status": 0,
    "zKConnected": true,
    "QTime": 20,
    "params": {
      "q": "*:*"
    }
  },
  "response": {
    "numFound": 107,
    "start": 0,
    "docs": [ ... ]
  }
}
```

Partial Results

```json
{
  "responseHeader": {
    "status": 0,
    "zKConnected": true,
    "QTime": 20,
    "partialResults": true,
    "QTime": 20,
    "params": {
      "q": "*:*"
    }
  },
  "response": {
    "numFound": 77,
    "start": 0,
    "docs": [ ... ]
  }
}
```
References

- https://wiki.apache.org/solr/
- http://zookeeper.apache.org/
- http://www.slideshare.net/erikhatcher/solr-application-development-tutorial
- http://www.edureka.co/apache-solr-self-paced
Thank You