Présentation d'Apache Solr

Aurélien Pontacq

06/01/2009

Plan

1. Introduction
2. Principe de Solr
3. Indexation des documents
4. Recherche de documents
5. Schema.xml
6. Analyse
7. Caractéristiques intéressantes
8. Conclusion
9. Démonstration

1. Introduction - bref historique

- Désir de mettre en place une plateforme de recherche
  - Commerciales : licences chères
  - Open source : pas de solutions complètes

- CNET fait don du code de Solr à Apache début 2006
  - De nombreux sites web à lourd trafic l'utilisent : cnet.com, zvents.com, oodle.com, netflix.com, tsr.ch, …
  - Solr est basé sur Apache Lucene

1. Introduction - Apache Lucene

- Librairie de recherche « full-text »
  - Créée en 1999, don à Apache en 2001
- Libre, licence Apache
- 100% Java
  - Mais a été portée en divers langages (C/C++, .NET, Python, Ruby, …)
- Rapide, stable, performant, modulable, communauté très active
- Wikipédia l'utilise !

1. Introduction - Solr

- « Solr est un serveur de recherche pour entreprises, open source et basé sur la librairie Java de recherche Lucene, avec des APIs XML/HTTP, le principe de cache, de réplication, et une interface d'administration Web. »

- Tire profit des puissantes propriétés de Lucene
- Écrit en Java5 et déployable sous forme de webapp (.war).
- Grand nombre de supports
    - wiki, mailing list, …

1. Introduction - Solr nécessite ...

- Machine virtuelle Java5 ou plus récente
- Un conteneur de servlet
  - Jetty, Tomcat, …
- L'application Solr
  - Dernière version stable : 1.3.0 (septembre 2008)

2. Principe de Solr

- Interfaces XML/HTTP pour ajouter des documents (POST) et effectuer des recherches (GET).

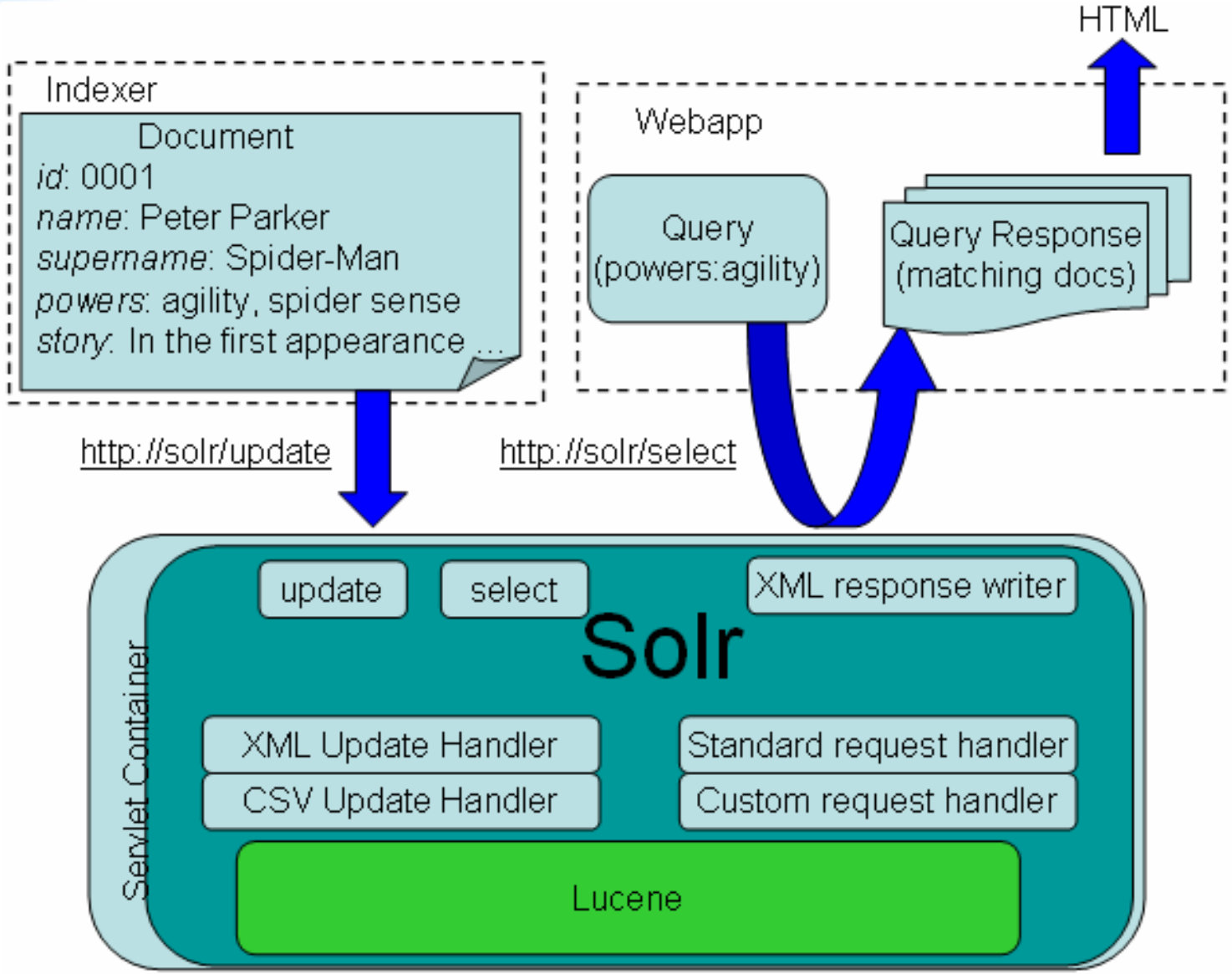
- 2 points d'entrée principaux
  - URL *update* => permet de maintenir les index

    http://localhost:8983/solr/update

  - URL *select* => utilisé pour les requêtes de recherche

    http://localhost:8983/solr/select

2. Principe de Solr



3. Indexation de documents - Principe

- Solr maintient dans son index une collection de Documents
- Un Document est un ensemble de champs (fields) auxquels sont associées des valeurs.
- Solr permet d'indexer par défaut les fichiers XML et CSV
  - Mais possibilité d'indexer des PDF, DOC, XLS ou PPT grâce à un handler de « Rich Documents ».
- Possibilité d'importer des données depuis une base de données, ou un flux RSS.
  - Grâce au « DataImportHandler »

3. Indexation de documents - Ajout

- HTTP POST sur http://localhost:8983/solr/update

```
<add><doc>
  <field name="id">0001</field>
  <field name="name">Peter Parker</field>
  <field name="supername">Spider-Man</field>
  <field name="powers">agility</field>
  <field name="powers">spider sense</field>
  <field name="story">In his first appearance, Peter Parker …
  </field>
</doc></add>
```

superheroes.xml

- Exemple avec *curl*
  - curl http://localhost:8983/solr/update -H "Content-type:text/xml" --data-binary @superheroes.xml

3. Indexation de documents - Suppression

- Suppression par l'ID

```
<delete><id>0001</id></delete>
```

- Suppression par une requête (plusieurs documents)

```
<delete>
  <query>powers:agility</query>
</delete>
```

3. Indexation de documents - Commit / optimize

- Commit : pour que toute modification d'index soit visible
  - Ferme le flux d'écriture d'index et supprime les doublons

```
<commit />
```

- Optimize : idem que Commit avec optimisation
  - Accélère les recherches
  - Réduit le nombre de segments d'index créé par Lucene

```
<optimize />
```

4. Recherche de documents - Requête

- http://localhost:8983/solr/select?q=**powers:agility**&start=0&rows=10&fl=name, supername

```
<response>
<lst name="responseHeader">
 <int name="status">0</int>
 <int name="QTime">0</int>
 <str name="q">powers:agility</str>
</lst>
<result name="response" numFound="1" start="0">
    <doc>
        <str name="name">Peter Parker</str>
        <str name="supername">Spider-Man</str>
    </doc>
</result>
</response>
```

4. Recherche de documents - Output

- Récupération de la réponse sous différents formats
  - XML (standard)
  - JSON (notation javascript)
  - Ruby
  - Python
  - PHP
  - XSLT

4. Recherche de documents - Scoring

- Facteurs de scoring
  - **tf** (term frequency) : Plus un terme apparaît de fois dans le document, plus le score est haut
  - **idf** (inverse document frequency) – nombre de documents total / nombre de documents contenant le terme
  - **lengthNorm** – privilégie les champs contenant le plus petit nombre de termes
  - **index-time boost et query-clause boost**

  => score = tf * idf * lengthNorm * boosts

- Scoring : ordre des résultats des requêtes, par défaut
  - Choix explicite de l'ordre : *select?q=powers:agility****;name asc;***

4. Recherche de documents - Syntaxe des requêtes

- Opérateur logique par défaut : OR
- name:Parker
- Parker, Peter Parker (≈ Peter OR Parker)
- name:"Peter Parker" AND supername:Spider-Man
- name:Peter^10 story:Peter
- +name:Parker +supername:Spider-Man –powers:flight
- id:[0 TO 1000]
- Par?er, Pet*r, Peter*, *:*

5. Schema.xml

- Contient les types des champs des documents indexés, ainsi que leur comportement
  - Types prédéfinis : int, float, string, date, boolean, … et types Solr : text, …
  - 2 modes possibles d'analyse : indexation / requête
    - Tokenizer : comment les mots sont découpés (split) ? Selon les espaces ? la ponctuation ?
    - Filters : sensibilité à la casse, stemming (racine), …
- Contient les champs eux-mêmes
- Définit une clé unique, le champ de recherche par défaut

5. Schema.xml

- Exemple : définition du type « text »

```
<fieldType name="text" class="solr.TextField" positionIncrementGap="100">
    <analyzer type="index">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.StopFilterFactory" ignoreCase="true"
            words="stopwords.txt"/>
    </analyzer>
    <analyzer type="query">
        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
        <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
            ignoreCase="true" expand="true"/>
        <filter class="solr.StopFilterFactory" ignoreCase="true"
            words="stopwords.txt"/>
    </analyzer>
</fieldType>
```

5. Schema.xml

- Fichiers .txt associés

stopwords.txt

```
#Standard english
#stop words

an
and
as
at
be
but
by
for
if
in
…
```

synonyms.txt

```
# Some synonym groups

GB,gib,gigabyte,gigabytes
MB,mib,megabyte,megabytes
Television, Televisions, TV, TVs

spider, arachnid

…
```

5. Schema.xml

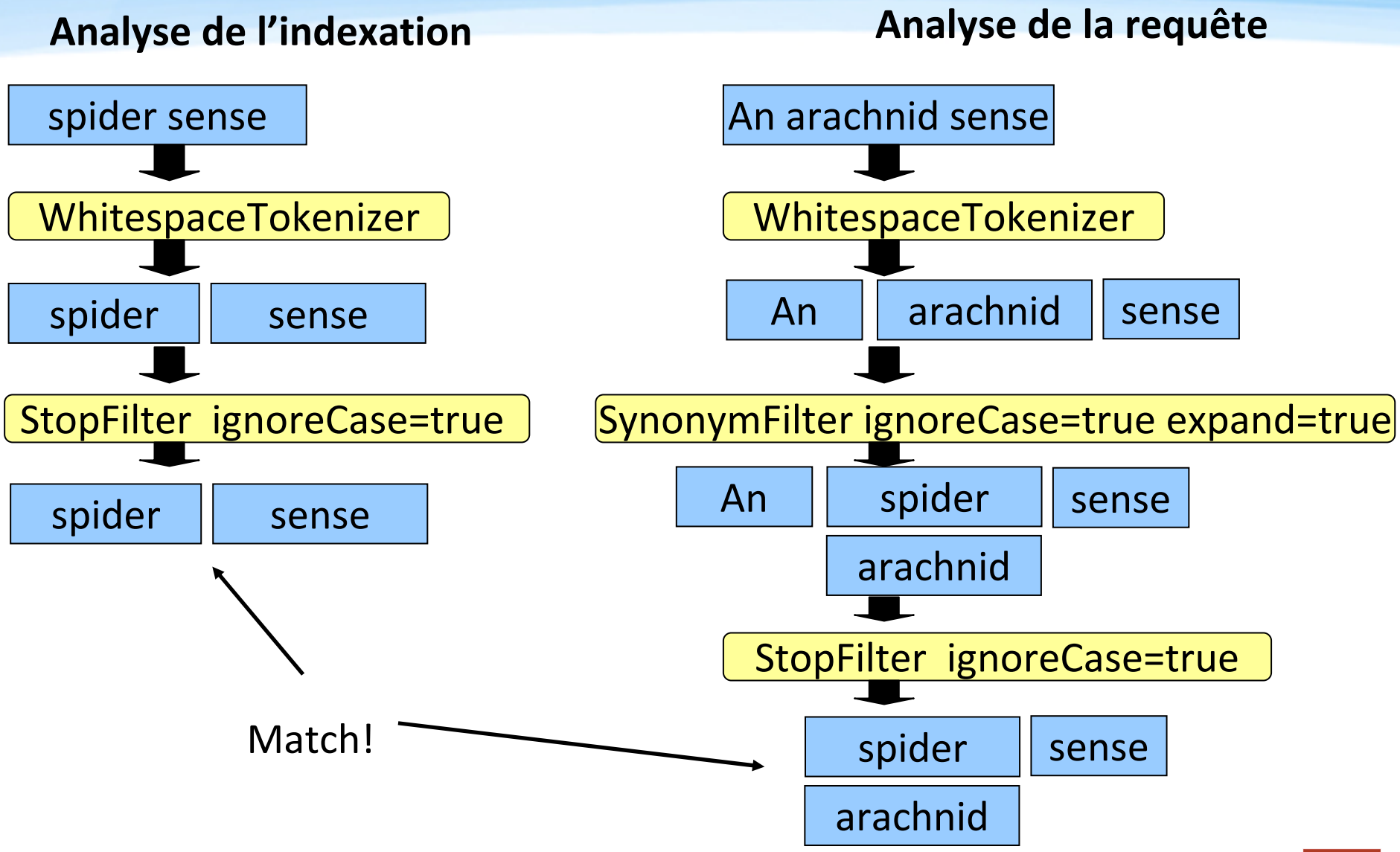
- Description des champs typés

```
<field name="id“ type="string“ indexed="true“ stored="true"/>
<field name=“name“ type=“text” indexed="true" stored="true"/>
<field name=“supername“ type=“string“ indexed="true“ stored="true"/>
<field name=“powers“ type=“text“ indexed="true“ stored="true“
      multiValued="true"/>
<field name=“story“ type=“text“ indexed="true“ stored=“true"/>

<uniqueKey>id</uniqueKey>
<defaultSearchField>name</defaultSearchField>
```

- indexed=true : champ sur lequel on peut effectuer des recherches
- stored=true : champ qui pourra être retourné dans une requête

6. Analyse


Analyse de l'indexation
spider sense
WhitespaceTokenizer
spider
sense
StopFilter ignoreCase=true
spider
sense
Analyse de la requête
An arachnid sense
WhitespaceTokenizer
An
arachnid
sense
SynonymFilter ignoreCase=true expand=true
An
spider
sense
arachnid
StopFilter ignoreCase=true
spider
sense
arachnid
Match!

7. Caractéristiques intéressantes

- Replication
  - 1 Solr Master, plusieurs Solr Searchers (rsync)
- Highlighting
  - Mise en surbrillance des termes
- Spellcheck
  - Principe des suggestions
- Facets

Conclusion

- Facile d’installation
- Un petit temps de prise en main
- Wiki un peu brouillon, en perpétuel changement
- Puissant, nombreuses fonctionnalités
- Bel avenir devant lui

Questions

?