Bases de données documentaires et distribuées, http://b3d.bdpedia.fr

Moteurs de recherche

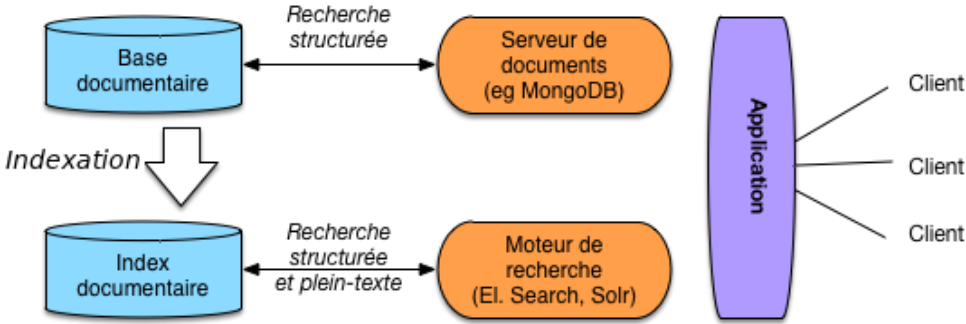
Moteurs libres

- Apache Solr (Lucene)
- ElasticSearch (Lucene)
- Sphinx
- Xapian

Moteurs commerciaux

- Google Search Appliance
- Exalead, Qwant
- Amazon CloudSearch
- Microsoft Azure Search, Bing

Bases documentaires et moteur de recherche



Bases documentaires et moteur de recherche

Un scénario typique :

- une recherche par mot-clé dans un site
- Les données du site sont gérées classiquement par une base documentaire (MySQL, Postgres, MongoDB)
- On extrait de la base tous les textes à **indexer** et on en fait des documents JSON, transmis au moteur de recherche
- Ce dernier se charge alors de répondre quand un utilisateur emploie un champ de recherche.

Bases documentaires et moteur de recherche

Un système comme ElasticSearch est consacré à la **recherche**, c'est-à-dire la lecture la plus efficace possible de documents.

Il s'appuie pour cela sur des structures compactes, compressées, optimisées (les **index inversés**)

Ce n'est pas (pas forcément) un très bon outil pour les autres fonctionnalités d'une base de données :

- Le stockage par exemple n'est ni aussi robuste ni aussi stable.
- Mises à jour coûteuses et asynchrones

Présentation d'Elasticsearch

Elasticsearch est un moteur de recherche basé sur Lucene

- grande communauté d'utilisateurs
- open source, profite de la recherche dans le domaine sur Lucene
- utilisé par de grands opérateurs du Web sur des collections immenses

En fait, plusieurs composants dont :

- Logstash, l'ETL, pour extraire, transformer et charger les données
- ElasticSearch, le moteur lui-même
- Kibana, pour produire des tableaux de bords de surveillance

Interrogation

- ElasticSearch s'appuie sur le système d'indexation **Lucene**, dont le rôle est essentiellement de créer les index inversés, et d'implanter les algorithmes de parcours brièvement introduits dans la session précédente.
- Lucene propose un langage de recherche basé sur des combinaisons de mot-clés, langage étendu et raffiné par ElasticSearch (cf plus tard)
- La première méthode pour transmettre des recherches est de passer une expression en paramètre à l'URL :

```
http://localhost:9200//nfe204/movies/_search?q=alien
http://localhost:9200/nfe204/movies/_search?q=alien,coppola
http://localhost:9200/nfe204/movies/_search?q=alien,coppola,1994
# utilisable dans l'interface ElasticVue, onglet Rest
```

la réponse d'ES


```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.558217,
    "hits" : [ {
      "_index" : "nfe204",
      "_type" : "movies",
      "_id" : "movie:2",
      "_score" : 0.558217,
      "_source":{ "title": "Alien", "year": 1979,
                  "genre": "Science-fiction",
                   "summary": "... ",
                   "country": "USA",
                   "director": "Ridley Scott",
                   "actors": [ ["Sigourney Weaver"]   ] }
    } ]
  }
}
```

Termes

- Notion de base : le **terme**
- c'est un mot au sens usuel
- ou une séquence de mots entre apostrophes

On peut interroger un index avec :

```
Neo apprend
```

Puis :

```
"Neo apprend"
```

Première recherche : documents avec "Neo", "apprend" ou les deux

Termes (suite)

- la recherche d'un terme s'effectue toujours sur un champ.
- La syntaxe complète pour associer le champ et le terme est :

```
champ:terme
```

- si non précisé, c'est le champ par défaut qui est utilisé
- pratique courante : concaténer toutes les chaînes de caractères en un champ "_all" général, défini par défaut
- Nos requêtes deviennent :

```
_all:Neo _all:apprend
```

- et

```
_all:"Neo apprend"
```

Termes (suite)

- Les valeurs des termes (dans la requête) et le texte indexé sont tous deux soumis à des transformations spécifiées dans le schéma.
- Une transformation simple est de tout transcrire en minuscules.

```
_all:"Neo APPREND"
```

- Les transformations appliquées à la requête ET au texte indexé doivent être cohérentes : si les termes sont transformés en majuscules, et le texte indexé en minuscules, on n'aura jamais de résultat !

Termes (suite)

On peut spécifier des termes (pas des séquences) incomplets

- le ' ?' indique un caractère inconnu ; "opti ?al" désigne "optimal", "optical", etc.
- le '*' indique n'importe quelle séquence de caractères ; "opti*" pour toute chaîne commençant par "opti"
- Approximation avec $\sim$: rechercher "optimal" et "optimal$\sim$" (0 et 1 résultat, "optical")
  La similarité est évaluée par une distance d'édition (nb opérations pour passer de "optimal" à "optical")
- Les intervalles sont exprimés avec [] (bornes comprises) ou { } bornes exclues

```
year:[1984 TO 2010]
```

Requêtes Booléennes

- Les critères peuvent être combinés avec des **opérateurs Booléens** : **AND**, **OR** et **NOT**
- Attention : majuscules

```
year:[1990 TO 2005] OR title:M*
year:[1990 TO 2005] AND NOT title:M*
```

- Par défaut, c'est un **OR** qui est appliqué
- Recherche sur plusieurs critères ramène l'union des résultats sur chaque critère pris individuellement

Requêtes structurées

On fournit un document décrivant la requête en la *postant* (méthode `POST`) :

```
{
    "query": {
                "match": {
                  "title": "Star Wars"
                }
            }
    }
```

Un premier pas vers une interrogation plus structurée.

Choix des champs et du résultat

On peut ajouter des spécifications dans la requête.

```
{
    "query": {
              "match": {
                "title": "Star Wars"
              }
          },
          "fields": ["title"],
          "_source": false
   }
```

Effectuer des recherches approchées

```
{
  "query": {
     "fuzzy": {
               "title": {
                 "value": "matrice"
               }
      }
          },
   "fields": ["title"],
 "_source": false
}
```

Recherches par intervalle

```
{
        "query": {
          "range": {
            "year": {"gte": 2010, "lte": 2020}
          }
        },
        "fields": ["title", "year"],
        "_source": false
}
```

En conclusion

Ce qui précère constitue l'essentiel des opérations d'interrogation avec un moteur de recherche.

- On est loin de la finesse d'interrogation d'un langage comme SQL.
- On fait peu mais on fait bien : systèmes extrêmement efficaces, même pour de très grands volumes.

Surtout, deux différences essentielles :

- Le système permet des recherches **approchées** : on trouve tous les documents "proches de" la requête
- Le résultat peut être **ordonné** par pertinence décroissante.

Comme ça marche ? Voir la suite...