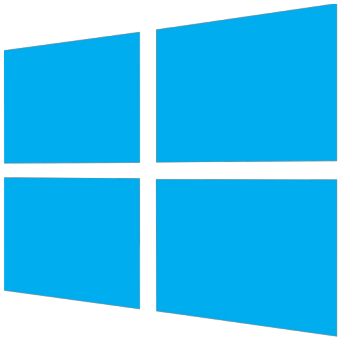
Progetto esame GAVI

Dblp Search-Engine

Sedoni Enrico
Mat: 108801

Tecnologie utilizzate

- Linguaggio di programmazione: python3.6
- Librerie esterne: whoosh (basato su lucene), sax
- Piattaforma di sviluppo: Windows 10
- Piattaforma di testing: Windows 10



Pro e Contro Whoosh

Pro:

Rapida prototipazione dell’applicazione

Ottime performance in ricerca dei documenti

Supporto al multithreading

Facile da usare

Supporto della community

Contro:

- Libreria poco documentata dallo sviluppatore
- Controllo non totale sulla struttura interna dell’applicazione

Linguaggio supportato

f-t-s : ([field:] search-pattern)
search-pattern : keyword | “phrase”
field: pub-search | venue-search
pub-search : pub-ele[.pub-field]
pub-ele: publication | article | incollection | inproceedings | phThesis | masterThesis
pub-field: author | title | year
venue-search: venue[.venue-field]
venue-field: title | publisher

Struttura xml

Spiegata nel file dtd…

- **Tag principali**: article, inproceedings, proceedings, book, incollection, phdthesis, mastersthesis
- Ogni tag contiene a sua volta altri campi in base alla tipologia di tag..

Es:

article → Title, author, year, pages…

Struttura xml

- Esempio del **tag article**:

```
<article key="journals/jifs/ELL19" mdate="2019-02-22">
   <author>Cheng-Guo E</author>
   <author>Quan-Lin Li</author>
   <author>Shiyong Li</author>
   <title>Cooperative game in parallel service systems with nonexponential service times.</title>
   <pages>127-137</pages>
   <year>2019</year>
   <volume>36</volume>
   <journal>Journal of Intelligent and Fuzzy Systems</journal>
   <number>1</number>
   <ee>https://doi.org/10.3233/JIFS-18101</ee>
   <url>db/journals/jifs/jifs36.html#ELL19</url>
 </article>
```

Struttura xml

- Alcuni tag appartengono a diverse categorie:
  - **Publication**: article, incollection, inproceedings, phdthesis, mastersthesis
  - **Venue**: proceedings, book, journal

Xml problematiche

1) Dimensione dei dati xml da indicizzare molto grande: 2,5GB
2) Scarse performance nell’indicizzazione dei documenti
3) Gestione dei journal

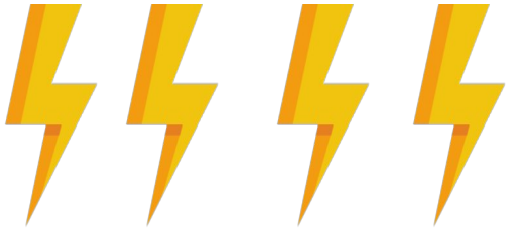
Problematica 1
(dimensione elevata xml)

- Fare il parsing di un file xml così grande non è semplice.
- Il parser più utilizzato per la sua semplicità (Dom) va a creare in memoria un albero di tutto l'xml. Ma si andrebbe ad occupare troppa memoria.

- Soluzione individuata:
  - Si è deciso di utilizzare il **parser xml sax**, che è un po' più complesso (funziona ad eventi), ma garantisce ottime performance e consumo ridotto della memoria

Problematica 2
(Scarse performance nell'indicizzazione)

- Le **performance** nell'indicizzare i dati erano molto **scarse** inizialmente
- Soluzione:
  - Si è scoperto il supporto al multithreading di whoosh, che permette in fase di indicizzazione dei documenti di utilizzare più thread e di dedicare ad ogni thread una certa quantità di memoria per l'indicizzazione



```
writer = ix.writer(procs=4, limitmb=512)
```

Problematica 3
(Gestione dei journal)

- Un caso particolare era quello di gestire query del tipo (query puramente di esempio):
  - **publication.author: "Dharmani Bhaveshkumar C" venue: science**

- Ovvero la ricerca di **tutte le pubblicazioni** di Dharmani Bhaveshkumar C nelle **riviste o conferenze** che contengono la parola: science
- Soluzione:
  - Le uniche pubblicazioni classificate come venue sono i journal (che sono **article contenenti il tag journal**)
  - Sfruttare il tag journal (che contiene il titolo vero e proprio del journal in cui è contenuto l'articolo)

Progettazione indice

- Whoosh permette di creare facilmente un inverted index. Basta creare l'indice e un writer dei documenti per l'indice. Per creare l'inverted index è necessario dire a whoosh quale schema usare (i campi dei documenti da scrivere)

```
schema = Schema(key=NUMERIC(stored=True), type=TEXT(stored=True),
author=TEXT(stored=True, phrase=True), title=TEXT(stored=True, phrase=True),
year=TEXT(stored=True), journal=TEXT(stored=True, phrase=True), ee=ID(stored=True),
publisher=TEXT(stored=True))
```

Progettazione indice

- Creazione e scrittura dell’indice:

```
ix = create_in("indexdir", schema)
writer = ix.writer(procs=nproc, limitmb=512)
Writer.add_document(…)
Writer.commit()
```

Problematiche Indice
(grandezza dell'indice)

- Se si scrive nell'inverted index tutti campi dell'xml questo verrebbe **troppo grande**.

  Soluzione:

  - Si è cercato di ridurre al **minimo i campi** memorizzati nell'inverted index per la ricerca. Alcuni campi selezionati sono:

"type","author","title","year", "journal","ee","publisher"

Nb: type è un campo aggiuntivo che indica il tipo di tag, es: article, book..

Progettazione indice

- Si è ottenuta una **dimensione** dei dati indicizzati di circa 4GB che è un valore abbastanza **accettabile** e gestibile bene dalla maggior parte dei sistemi moderni.

Parsing della query

- Whoosh implementa un query parser standard e si è deciso di utilizzare quello.
- Le query prima di essere passate al query parser vengono processate da un **“query manager”**, che va a creare a sua volta due liste: **fields** e **myquery**.
- La lista **fields** viene riempita solo se il query manager incontra delle parole del tipo: “word:”
- La lista **myquery** contiene solamente le parole della query, quindi non i fields.

  Es: article.author: pippo → fields: [“article.author”], myquery : [“pippo”]

Parsing della query

- Al query parser viene passato man mano **ogni elemento** della lista myquery fino alla fine
- Per ogni elemento di myquery:
  - In base alle liste **fields e myquery** si va a comporre la query i-esima, che verrà poi lanciata sull'inverted index
  - Se la lista fields è **vuota** allora significa che è una semplice keyword query o phrasal query, che vengono tranquillamente supportate dal query parser standard di whoosh, basta indicare nello **schema** i campi che devono supportare query di tipo frasale

Parsing della query

- Query particolari di tipo: “**publication.author: xxx venue: xxx**” vengono gestite in maniera un po’ differente, tra tutte le pubblicazioni dell’autore xxx vengono cercati i journal che contengono la parola xxx
- Query di tipo: “**venue.title: xxx**” vanno invece a recuperare tutti i documenti il cui titolo è xxx e poi vengono filtrati solo i venue, viceversa per le pubblicazioni
- I risultati verrano poi inseriti in **insiemi (set)** e gestiti in un momento successivo, per comodità

Problematiche
(velocità di ricerca)

- La ricerca attraverso l'inverted index avviene sempre in maniera abbastanza veloce, tuttavia si ha un calo di perfomance quando la query restituisce un numero molto elevato di documenti

- Soluzione:
  - Si è dovuto cercare un **compromesso**, il tempo per la risoluzione di alcune query sarà un po' più lungo.. whoosh permette di limitare il numero di documenti risultanti, tuttavia limitandolo troppo non si riuscirebbero a reperire alcuni documenti. Si è quindi scelto un numero intermedio che non limita troppo i risultati.

Problematiche

- Possibile **soluzione futura**:
  - Cercare un modo più efficiente che offre whoosh per recuperare i documenti data una query generica

Gestione dei documenti risultanti

- Una volta ottenuti i documenti dal query searcher questi possiedono anche uno **score**, che indica quanto il documento è pertinente rispetto alla query.
- I documenti vengono quindi ordinati per lo score e restituiti all'utente, in questo modo l'utente vedrà solo i primi **10 risultati** più pertinenti restituiti

Modelli di ranking

- Whoosh implementa di default già diversi **modelli di ranking**, principalmente di tipo probabilistico.
- E’ possibile cambiare il modello di ranking quando si va ad istanziare l’index searcher.
- I modelli selezionati sono: **BM25F** e **PL2**, due modelli che effettuano un ranking di tipo probabilistico

Fine

Grazie per l’attenzione