ICS 321 Fall 2011

Other Data Models: Unstructured, Graph, Key-Value Pairs

Asst. Prof. Lipyew Lim
Information & Computer Science Department
University of Hawaii at Manoa
Outline

Unstructured Data and Inverted Indexes

Web Search Engines

RDF & Linking Open Data

Big Table, CouchDB, & Cassandra
Unstructured Data

• What are some examples of unstructured data?
• How do we model unstructured data?
• How do we query unstructured data?
• How do we process queries on unstructured data?
• How do we index unstructured data?
Unstructured Text Data

• Field of “Information Retrieval”
• Data Model
  – Collection of documents
  – Each document is a bag of words (aka terms)
• Query Model
  – Keyword + Boolean Combinations
  – Eg. DBMS and SQL and tutorial
• Details:
  – Not all words are equal. “Stop words” (eg. “the”, “a”, “his” ...) are ignored.
  – Stemming: convert words to their basic form. Eg. “Surfing”, “surfied” becomes “surf”
Inverted Indexes

• Recall: an index is a mapping of search key to data entries
  – What is the search key?
  – What is the data entry?
• Inverted Index:
  – For each term store a list of postings
  – A posting consists of <docid,position> pairs

<table>
<thead>
<tr>
<th>Term</th>
<th>Posting lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS</td>
<td>doc01 10 18 20 doc02 5 38 doc01 13</td>
</tr>
<tr>
<td>SQL</td>
<td>doc06 1 12 doc09 4 9 doc20 12</td>
</tr>
<tr>
<td>trigger</td>
<td>doc01 12 15 doc09 14 21 25 doc10 11 55</td>
</tr>
</tbody>
</table>
Lookups using Inverted Indexes

- **Given a single keyword query “k” (eg. SQL)**
  - Find k in the lexicon
  - Retrieve the posting list for k
  - Scan posting list for document IDs [and positions]

- **What if the query is “k1 and k2” ?**
  - Retrieve document IDs for k1 and k2
  - Perform intersection

<table>
<thead>
<tr>
<th>lexic</th>
<th>Posting lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS</td>
<td>doc01 10 18 20</td>
</tr>
<tr>
<td>SQL</td>
<td>doc06 1 12</td>
</tr>
<tr>
<td>trigger</td>
<td>doc01 12 15</td>
</tr>
</tbody>
</table>
Too Many Matching Documents

• Rank the results by “relevance”!
• Vector-Space Model
  – Documents are vectors in high-dimensional space
  – Each dimension in the vector represents a term
  – Queries are represented as vectors similarly
  – Vector distance (dot product) between query vector and document vector gives ranking criteria
  – Weights can be used to tweak relevance
• PageRank (later)
Internet Search Engines

World Wide Web

Web Crawler

Web Page Repository

Search Engine

Web Server

Inverted Index

Keyword Query

Ranked Results

Snippets

Query

Doc IDs

Postings etc

Indexer

Lipyeow Lim -- University of Hawaii at Manoa
Ranking Web Pages

• Google’s PageRank
  – Links in web pages provide clues to how important a webpage is.

• Take a random walk
  – Start at some webpage p
  – Randomly pick one of the links and go to that webpage
  – Repeat for all eternity

• The number of times the walker visits a page is an indication of how important the page is.

Vertices represent web pages. Edges represent web links.
### Resource Description Framework (RDF)

#### Books

<table>
<thead>
<tr>
<th>ID</th>
<th>Author</th>
<th>Title</th>
<th>Publisher</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isbn0-00-651409-X</td>
<td>Id_xyz</td>
<td>The glass palace</td>
<td>Id_qpr</td>
<td>2000</td>
</tr>
</tbody>
</table>

#### Authors

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Homepage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_xyz</td>
<td>Ghosh, Amitav</td>
<td><a href="http://www.amitavghosh.com">http://www.amitavghosh.com</a></td>
</tr>
</tbody>
</table>

#### Publishers

<table>
<thead>
<tr>
<th>ID</th>
<th>Publisher Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_qpr</td>
<td>Ghosh, Amitav</td>
<td>London</td>
</tr>
</tbody>
</table>
RDF Graph Data Model

Nodes can be literals

Nodes can also represent an entity

Edges represent relationships or properties
More formally

• An RDF graph consists of a set of RDF triples
• An RDF triple \((s, p, o)\)
  – “s”, “p” are URI-s, ie, resources on the Web;
  – “o” is a URI or a literal
  – “s”, “p”, and “o” stand for “subject”, “property” (aka “predicate”), and “object”
  – here is the complete triple: (<http://...isbn...6682>, <http://..//original>, <http://...isbn...409X>)
• RDF is a general model for such triples
• RDF can be serialized to machine readable formats:
  – RDF/XML, Turtle, N3 etc
<rdf:Description rdf:about="http://.../isbn/2020386682">
  <f:titre xml:lang="fr">Le palais des mirroirs</f:titre>
  <f:original rdf:resource="http://.../isbn/000651409X"/>
</rdf:Description>
Querying RDF using SPARQL

- The fundamental idea: use graph patterns
- the pattern contains unbound symbols
- by binding the symbols, subgraphs of the RDF graph are selected
- if there is such a selection, the query returns bound resources

**SELECT ?p ?o**
**WHERE** {subject ?p ?o}
Example: SPARQL

```sparql
SELECT ?isbn ?price ?currency  # note: not ?x!
WHERE {?isbn a:price ?x.
  ?x p:currency ?currency.}
```
Linking Open Data

• Goal: “expose” open datasets in RDF
  – Set RDF links among the data items from different datasets
  – Set up, if possible, query endpoints
• Example: DBpedia is a community effort to
  – extract structured (“infobox”) information from Wikipedia
  – provide a query endpoint to the dataset
  – interlink the DBpedia dataset with other datasets on the Web
@prefix dbpedia <http://dbpedia.org/resource/>.
@prefix dbterm <http://dbpedia.org/property/>.

dbpedia:Amsterdam
  dbterm:officialName "Amsterdam" ;
  dbterm:longd "4" ;
  dbterm:longm "53" ;
  dbterm:longs "32" ;
  dbterm:leaderName dbpedia:Job_Cohen ;
  ...
  dbterm:areaTotalKm "219" ;
  ...

dbpedia:ABN_AMRO
  dbterm:location dbpedia:Amsterdam ;
  ...

<table>
<thead>
<tr>
<th>Country</th>
<th>Netherlands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Province</td>
<td>North Holland</td>
</tr>
<tr>
<td>Government</td>
<td>Municipality</td>
</tr>
<tr>
<td>- Type</td>
<td></td>
</tr>
<tr>
<td>- Mayor</td>
<td>Job Cohen</td>
</tr>
<tr>
<td>- Alderman</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lodewijk Asscher</td>
</tr>
<tr>
<td></td>
<td>Carolien Gehrels</td>
</tr>
<tr>
<td></td>
<td>Tjaard Herringe</td>
</tr>
<tr>
<td></td>
<td>Maarten van Poelgeest</td>
</tr>
<tr>
<td></td>
<td>Marijke Vos</td>
</tr>
<tr>
<td>- Secretary</td>
<td>Erik Gentsien</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Area</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- City</td>
<td>2.19 km² (84.6 sq mi)</td>
</tr>
<tr>
<td>- Land</td>
<td>1.06 km² (41 sq mi)</td>
</tr>
<tr>
<td>- Water</td>
<td>5.73 km² (22.3 sq mi)</td>
</tr>
<tr>
<td>- Urban</td>
<td>1.029 km² (397.3 sq mi)</td>
</tr>
<tr>
<td>- Metro</td>
<td>1.815 km² (700.8 sq mi)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elevation</th>
<th>2 m (7 ft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population</td>
<td>765,265</td>
</tr>
<tr>
<td></td>
<td>4,404/km² (11,648/sq mi)</td>
</tr>
<tr>
<td></td>
<td>1,384,422</td>
</tr>
<tr>
<td></td>
<td>2,158,372</td>
</tr>
<tr>
<td></td>
<td>Amsterdam</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time zone</th>
<th>CET (UTC+1)</th>
<th>CEST (UTC+2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Postcodes</th>
<th>1011 – 1109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area code(s)</td>
<td>020</td>
</tr>
</tbody>
</table>

Website: [www.amsterdam.nl](http://www.amsterdam.nl)
Linking the Data

<http://dbpedia.org/resource/Amsterdam>  
owl:sameAs <http://rdf.freebase.com/ns/...> ;  
owl:sameAs <http://sws.geonames.org/2759793> ;  
...

<http://sws.geonames.org/2759793>  
owl:sameAs <http://dbpedia.org/resource/Amsterdam>  
wgs84_pos:lat "52.3666667" ;  
wgs84_pos:long "4.8833333";  
geo:inCountry <http://www.geonames.org/countries/#NL> ;  
...
Google’s Bigtable

“Bigtable is a sparse, distributed, persistent multidimensional sorted map”

• It is a type key-value store:
  – Key: (row key, column key, timestamp)
  – Value: uninterpreted array of bytes

• Read & write for data associated with a row key is atomic

• Data ordered by row key and range partition into “tablets”

• Column keys are organized into column families:
  – A column key then is specified using <family:qualifier>

• Timestamp is a 64 bit integer timestamp in microseconds
Example: Webpages using Bigtable

- Row key = reversed string of a webpage’s URL
- Column keys:
  - contents:
  - anchor:cnnsi.com
  - anchor:my.look.ca
- Timestamps: t3, t5, t6, t8, t9
CouchDB

• A distributed document database server
  – Accessible via a RESTful JSON API.
  – Ad-hoc and schema-free
  – robust, incremental replication
  – Query-able and index-able

• A couchDB document is a set of key-value pairs
  – Each document has a unique ID
  – Keys: strings
  – Values: strings, numbers, dates, or even ordered lists and associative maps
Example: couchDB Document

"Subject": "I like Plankton"
"Author": "Rusty"
"PostedDate": "5/23/2006"
"Tags": ["plankton", "baseball", "decisions"]
"Body": "I decided today that I don't like baseball. I like plankton."

• CouchDB enables views to be defined on the documents.
  – Views retain the same document schema
  – Views can be materialized or computed on the fly
  – Views need to be programmed in javascript
Cassandra

• Another distributed, fault tolerant, persistent key-value store

• Hierarchical key-value pairs (like hash/maps in perl/python)
  – Basic unit of data stored in a “column”: (Name, Value, Timestamp)

• A column family is a map of columns: a set of name:column pairs. “Super” column families allow nesting of column families

• A row key is associated with a set of column families and is the unit of atomicity (like bigtable).

• No explicit indexing support – need to think about sort order carefully!
Example: Cassandra

- **mccv**
  - Users
  - emailAddress
    - name: emailAddress
    - value: foo@bar.com
  - webSite
    - name: webSite
    - value: http://bar.com
  - visits
    - name: visits
    - value: 243

- **user2**
  - Users
  - emailAddress
    - name: emailAddress
    - value: user2@bar.com
  - twitter
    - name: twitter
    - value: user2