Document Database and MapReduce

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Lecture topics

- CAP Theorem ACID vs BASE
- Document Databases
- MongoDB
- Map-Reduce
- Summery

NoSQL database and CAP theorem

Motivation

- As we mentioned before relational database systems are designed to run on a single server
- RDBMS satisfy the ACID rules to provide consistency and availability of the data for the users
- But how do NoSQL databases deal with the data in their implementation?

NoSQL database and CAP theorem

CAP theorem

- States that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:
  - **Consistency** every read receives the most recent write or an error
  - **Availability** every request receives a response, without guarantee that it contains the most recent version of the information
  - **Partition** tolerance (the system continues to operate despite arbitrary partitioning due to network failures

NoSQL database and CAP theorem

ACID vs BASE

- **B**asically **A**vailable, **S**oft State and Eventual **C**onsistent
- Because of this characteristic the query language must be able to process data saved locally and in a cluster (to be discussed in another slide)

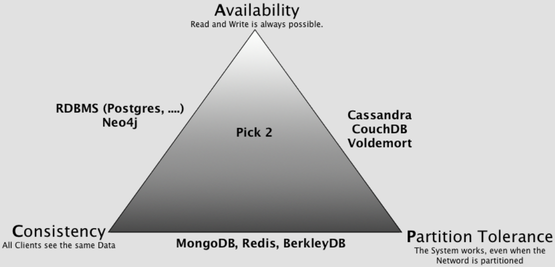

Availability
Read and Write is always possible.
RDBMS (Postgres, ...)
Neo4j
Pick 2
Cassandra
CouchDB
Voldemort
Consistency
All Clients see the same Data
MongoDB, Redis, BerkleyDB
Partition Tolerance
The System works, even when the Netword is partitioned

Figure: CAP Theorem

HOGESCHOOL ROTTERDAM

RDBMS vs Document database

ACID vs BASE

- Relational databases are considered to be structural
- Document databases uses semi-structured formats
- text files such as logs are unstructured

Document Database

Introduction

- A document database is a nonrelational database that stores data as semi-structured documents such as in XML or JSON formats
- Document databases are free to implement ACID transactions or other characteristics of a traditional RDBMS
- A document database allows some form of data description without enforcing a schema
- The alignment with web-development programming practices has resulted in JSON and document databases/storage

Document Database

Introduction

- Let us see what are those formats and how they are used!
- We will start with eXtensible Markup Language
- Then we will look at JavaScript Object Notation and it's Binary version

Document Database

eXtensible Markup Language (XML)

- Defined by the WWW Consortium (W3C)
- Extensible, unlike HTML, users can add new tags, and separately specify how the tag should be handled for display
- XML has become the basis for all new generation data interchange formats. For instance bank transfers and secure document exchange
- Documents have tags giving extra information about sections of the document. Those tags can also be nested

HTML as XML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="UTF-8"/>
    <title>Polyglot (X)HTML Template</title>
  </head>
  <body>
    <p>content....</p>
  </body>
</html>
```

Document Database

XML

- Each XML based standard defines what are valid elements, using XML type specification languages to specify the syntax
- DTD (Document Type Descriptors): describes the structure of an XML document
- XML Schema (newer than DTD): a special type of XML document that describes the elements that may be present
- Sample implementation database BaseX (basex.org)

Document Type Descriptors

```
<! ELEMENT department(dept_name, building, budget)>
<! ELEMENT dept_name (#PCDATA)>
<! ELEMENT budget (#PCDATA)>
<! ELEMENT university ( ( department | course |
    instructor | teaches )+)>
```

```
Notation:
        |       : alternatives
        +       :  1 or more occurrences
        *       :  0 or more occurrences
        #PCDATA :      Parsed charachter data i.e.
            parsed string
```

Document Database

XML Processing

- XPath: A syntax for retrieving specific elements from an XML document using wildcards.
- XQuery: A query language provides mechanisms for modifying a document. XQuery is sometimes referred to as “the SQL of XML”.
- XSLT (Extensible Stylesheet Language Transformations): A language for transforming XML documents into alternative formats, including non-XML formats such as HTML.
- DOM (Document Object Model): An object-oriented API that programs can use to interact with XML, XHTML, and similarly structured documents.

Document Database

Tree Model of XML Data

- Query and transformation languages are based on a tree model of XML data
- An XML document is modeled as a tree, with nodes corresponding to elements and attributes

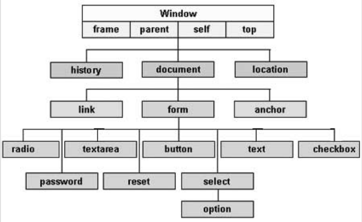


Figure: DOM sample of an HTML document

Document Database

XML Processing: XPath

- XPath: is used to address (select) parts of documents using path expressions
  - The initial denotes root of the document (above the top-level tag)
  - Think of file names in a directory hierarchy
  - Selection predicates may follow any step in a path, in [ ]
  - It is possible to apply selection criteria on the values using comparison operators [a]

[a] Demonstrate an example

Document Database

XML Processing: XQuery and XPath

- XQuery is derived from the Quilt query language, which itself borrows from SQL.
- XQuery uses a: for . . . let . . . where . . . order by . . . result . . . [a] [b]
  - for = from
  - where = where
  - order by = order by
  - result = select

[a] **let**: allows temporary variables, and has no equivalent in SQL

[b] Demonstrate an example

Document Database

JavaScript Object Notation JSON

- JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs
- JSON Schema is based on the concepts from XML Schema, but is JSON-based
- Document databases use JSON documents in order to store records, just as tables and rows store records in a relational database

Document Database

BSON

- BSON: binary-encoded format used in MongoDB instead of JSON
- BSON extends the JSON model to provide additional data types such as integer and float to be efficient for encoding and decoding within different languages.
- BSON implementation supports embedding objects and arrays within other objects and arrays

Document Database

MongoDB

- A MongoDB instance may have zero or more databases
- A database may have zero or more collections.
  - Can be thought of as the relation (table) in DBMS, but with many differences.
- A collection may have zero or more documents.

Document Database

MongoDB

- A MongoDB instance may have zero or more databases
- A database may have zero or more collections.
  - Can be thought of as the relation (table) in DBMS, but with many differences.
- A collection may have zero or more documents.
  - Docs in the same collection don't even need to have the same fields
  - Docs are the records in RDBMS
  - Docs can embed other documents
  - Documents are addressed in the database via a unique key differences.

Document Database

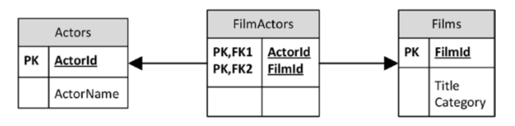
MongoDB

- A MongoDB instance may have zero or more databases
- A database may have zero or more collections.
  - Can be thought of as the relation (table) in DBMS, but with many differences.
- A collection may have zero or more documents.
  - Docs in the same collection don't even need to have the same fields
  - Docs are the records in RDBMS
  - Docs can embed other documents
  - Documents are addressed in the database via a unique key differences.
- A document may have one or more fields.
- query language is JavaScript
- Threre is no join provided in MongoDB. You have to implement it manually.

Document Data-Model

Relational Model

- Suppose you have the following entities and their relationships



Actors

PK **ActorId**

ActorName

FilmActors

PK,FK1 **ActorId**

PK,FK2 **FilmId**

Films

PK **FilmId**

Title

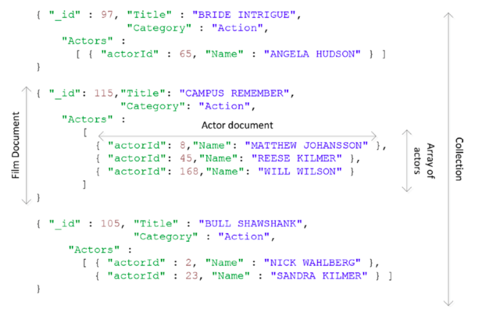
Category

- How would we model this in a document structure?

Document Data-Model

Embedded

- First mapping possibility is to map to one embedded collection.



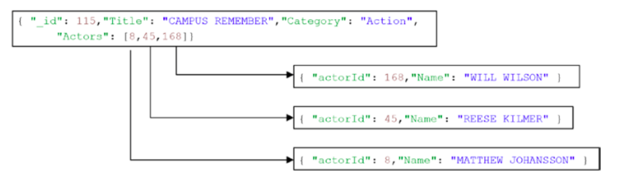
```
{ "_id" : 97, "Title" : "BRIDE INTRIGUE",
              "Category" : "Action",
    "Actors" :
      [ { "actorId" : 65, "Name" : "ANGELA HUDSON" } ]
}

{ "_id": 115,"Title": "CAMPUS REMEMBER",
             "Category": "Action",
    "Actors" :
        [
          { "actorId": 8,"Name": "MATTHEW JOHANSSON" },
          { "actorId": 45,"Name": "REESE KILMER" },
          { "actorId": 168,"Name": "WILL WILSON" }
        ]
}

{ "_id" : 105, "Title" : "BULL SHAWSHANK",
               "Category" : "Action",
     "Actors" :
       [ { "actorId" : 2, "Name" : "NICK WAHLBERG" },
         { "actorId" : 23, "Name" : "SANDRA KILMER" } ]
}
```

- document database are not designed to be normalized and data repetition is accepted, but could have side effects.

Document Data-Model

Linking using object _id

- Second mapping possibility is to map to different collections and link the documents



```
{ "_id": 115,"Title": "CAMPUS REMEMBER","Category": "Action",
     "Actors": [8,45,168]}
```

```
{ "actorId": 168,"Name": "WILL WILSON" }
```

```
{ "actorId": 45,"Name": "REESE KILMER" }
```

```
{ "actorId": 8,"Name": "MATTHEW JOHANSSON" }
```

- This approach is less suited for document databases since the binary data of those collections are not stored as a continuous stream.
- Another disadvantage of this approach is the lack of join query

Querying collections and objects

- Select queries in MongoDB

```
\\SQL
SELECT * FROM actors

SELECT * FROM actors WHERE age = 23

SELECT * FROM actors WHERE age = 23 ORDER BY name

\\Mongo
db.actors.find()

db.actors.find({age: 23})

db.actors.find({age: 23}).sort({name:1})
```

Querying collections and objects

- Insert queries in MongoDB
- Suppose we have a relationship between actor entity and address entity based on actor_id

```
\\SQL
INSERT INTO actors(actor_id,name,age)
        VALUES(3, "actor name", 45)
INSERT INTO address(addressid, street, city,
    actor_id)
        VALUES(5, "Wijnhaven 66", "Rotterdam", 3)

\\Mongo
db.actors.insert({name:"actor name", age: 23,
                address:{street:"Wijnhaven 66",
                    city:"Rotterdam"}
        })
```

Querying collections and objects

- Update queries in MongoDB

```
\\SQL
UPDATE actors
        SET name = "New name"
        WHERE actor_id = "a1"

\\Mongo
db.actors.update(
   { _id: "a1" },
 \\& sould be dollar
   { &set : {
                name: "New Name"}
        })
```

MapReduce

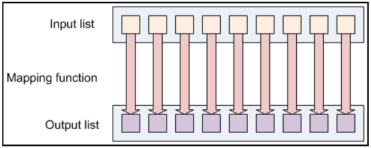
Introduction

- MapReduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.
- Map- and Reduce functions are commonly used in functional programming
- In INFDEV02-2 and INDEV02-3 we already introduced HOFs
- MapReduce rely on the concept of higher order functions HOFs are very powerful in the context of NoSQL databases.
- The following functions will be further discussed : FlatMap, Map and Reduce

Map Function

- Apply the function f to each element of list x
- map(f, x[0...n-1])
- in Python:

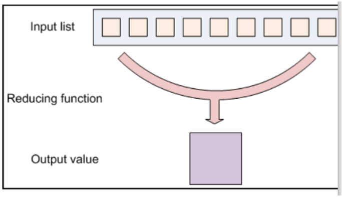
```
def square(x):
      return x * x

map(square, [1, 2, 3, 4])  #would return [1, 4, 9,
   16]
```



Reduce Function

- Repeatedly apply binary function f to pairs of items in x, replacing the pair of items with the result until only one item remains
- reduce(f, x[0...n-1])
- in Python:

```
def add(x,y):
        return x+y
reduce(add, [1,2,3,4]) #would result in a 10
```



FlatMap Function

- Repeatedly apply function f to items in a sublist, then removing the sublist structure with the result of one dimensional list
- FlatMap(f, x[[0..n-1],[0..m-1]])
- in pseudo Python: suppose the f function returns the element without any change

```
listOfLists = [[1, 2],[3, 4, 5], [6]]
for l in  listOfLists:
        map(f, l)
reduce(list.__add__, listOfLists)
#would result in a flatten list [1, 2, 3, 4, 5, 6]
```

MapReduce

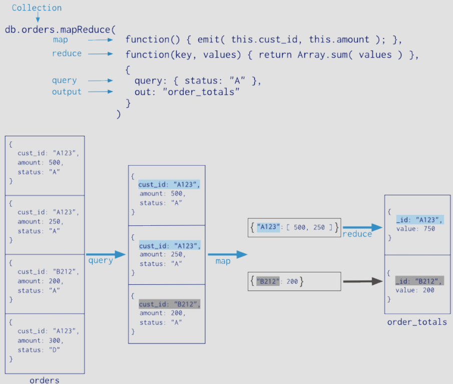
SQL and MapReduce

- We have seen so far what map en reduce functions are
- But why do we need them in document sturcture?
- What are the similarities between MapReduce and SQL?

MapReduce

MapReduce Function in MongoDB

- Data in mongoDB are saved in documents
- The MapReduce function first queries the collection, then maps the result documents to emit key-value pairs which is then reduced based on the keys that have multiple values.



```
db.orders.mapReduce(
                     function() { emit( this.cust_id, this.amount ); },
                     function(key, values) { return Array.sum( values ) },
                     {
                       query: { status: "A" },
                       out: "order_totals"
                     }
                   )
```

Collection
map
reduce
query
output
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{ "A123": [500, 250] }
{ "B212": 200 }
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals

MapReduce

Map and Filter functions vs SQL

- Relational databases use the map, filter and reduce paradigm (where it is called project, select, aggregate).
- SELECT MAX(pixels) FROM cameras WHERE brand = 'Nikon'
  - cameras is a sequence (a list of rows, where each row has the data for one camera)
  - WHERE brand = 'Nikon' is a **filter**
  - pixels is a **map** (extracting just the pixels field from the row)
  - MAX is a **reduce**
- Demo!

MapReduce

Join as MapReduce in MongoDB

- MongoDb does not provide explicit join queries to join two collections
- To implement a join you need
  - a mapper for each collection to retrieve key and values for each collection
  - A reducer function to reduce values for each key
- Demo!

MapReduce

MapReduce Function in a Cluster

- How does CAP theorem effect the implementation of MapReduce?
- Generally speaking It depends on the execution of MapReduce whether local or in a cluster
- We have seen how MapReduce is executed locally in document database
- What about clusters?

MapReduce

MapReduce Function in a Cluster

- The distributed MapReduce idea is similar to (but not the same as!): reduce(f2, map(f1, x))
- Key idea: "data-centric" architecture – Send function f1 directly to the data : Execute it concurrently
- Then merge results with reduce: Also concurrently

MapReduce

End

- Thank you and the best of luck!