Lecture 11: MongoDB

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
What Is It?

• A document-oriented database
  – documents encapsulate and encode data (or information) in some standard formats or encodings

• NoSQL database
  – non-adherence to the widely used relational database
  – highly optimized for retrieve and append operations

• uses BSON format

• schema-less
  – No more configuring database columns with types

• No transactions

• No joins
The Basics

• 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
• A document may have one or more fields.
• MongoDB Indexes is much like their RDBMS counterparts.
The Basics

+ Simple queries
+ Makes sense with most web applications
+ Easier and faster integration of data
- Not well suited for heavy and complex transactions systems.
MongoDB Vs Relational DBMS

- Collection vs table
- Document vs row
- Field vs column
- schema-less vs schema-oriented
It Is Fast!

Anywhere from 2 to 10 times faster than MySQL
Example: Mongo Document

```javascript
user = {
    name: "Z",
    occupation: "A scientist",
    location: "New York"
}
```
Example: Mongo Document

```javascript
{
    name: 'Brad Steve',
    address:
    {
        street: 'Oak Terrace',
        city: 'Denton'
    }
}
```
Example: Mongo Collection

{ "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
  "Last Name": "DUMONT",
  "First Name": "Jean",
  "Date of Birth": "01-22-1963" },

{ "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
  "Last Name": "PELLERIN",
  "First Name": "Franck",
  "Date of Birth": "09-19-1983",
  "Address": "1 chemin des Loges",
  "City": "VERSAILLES" }
Example: A Blog

• A blog post has an author, some text, and many comments
• The comments are unique per post, but one author has many posts
• How would you design this in SQL?
Example: A Blog: Bad Design

Collections for posts, authors, and comments

References by manually created ID

```javascript
post = {
  id: 150,
  author: 100,
  text: 'This is a pretty awesome post.',
  comments: [100, 105, 112]
}
author = {
  id: 100,
  name: 'Michael Arrington'
  posts: [150]
}
comment = {
  id: 105,
  text: 'Whatever this sux.'
}
```
Example: A Blog: Better Design

Collection for posts

Embed comments, author name

post = {
    author: 'Michael Arrington',
    text: 'This is a pretty awesome post.',
    comments: [
        'Whatever this post sux.',
        'I agree, lame!'  
    ]
}

Why is this one better?
Benefits

• Embedded objects brought back in the same query as parent object
  – Only 1 trip to the DB server required
• Objects in the same collection are generally stored contiguously on disk
  – Spatial locality = faster
• If the document model matches your domain well, it can be much easier to comprehend than nasty joins
Queries in MongoDB

Query expression objects indicate a pattern to match

\[
\text{db.users.find( \{last\_name: 'Smith'\} )}
\]

Several query objects for advanced queries

\[
\text{db.users.find( \{age: \{$gte: 23\} \} )}
\]
\[
\text{db.users.find( \{age: \{$in: [23,25]\} \} )}
\]

Exact match an entire embedded object

\[
\text{db.users.find( \{address: \{street: 'Oak Terrace', city: 'Denton'\}\} )}
\]

Dot-notation for a partial match

\[
\text{db.users.find( \{"address.city": 'Denton'\} )}
\]
Indexing

• Indexes in MongoDB are similar to indexes in RDBMS.

• MongoDB supports indexes on any field or sub-field contained in documents.

• MongoDB defines indexes on a per-collection level.

• All MongoDB indexes use a B-tree data structure.
Building An Application

• Want to build an app where users can check in to a location
• Leave notes or comments about that location
• Requirements
  – Need to store locations (Offices, Restaurants etc)
    • Want to be able to store name, address and tags
    • Maybe User Generated Content, i.e. tips / small notes?
  – Want to be able to find other locations nearby
  – User should be able to ‘check in’ to a location
  – Want to be able to generate statistics
Building An Application

**Locations**

- loc1, loc2, loc3

**Users**

- User1, User2
location1 = {
  name: "10gen HQ",
  address: "17 West 18th Street 8th Floor",
  city: "New York",
  zip: "10011",
  latlong: [40.0,72.0],
  tags: ["business", "cool place"],
  tips: [
    {user:"nosh", time:6/26/2010, tip:"stop by for office hours on Wednesdays from 4-6pm"},
    {.....},
  ]
}

Example queries:
• db.locations.find({latlong:{$near:[40,70]}})
• db.locations.find({zip:"10011", tags:"business"})
• db.locations.find({zip:"10011")).limit(10)
Inserting and updating locations

Initial data load:  
db.locations.insert(place1)

Using update to Add tips:  
db.locations.update({name:"10gen HQ"},  
   {$push :{tips:  
      {user:"nosh", time:6/26/2010,  
       tip:"stop by for office hours on Wednesdays from 4-6"}}}})
user1 = {
    name: "nosh",
    email: "nosh@10gen.com",
    checkins: [{ location: "10gen HQ",
                 ts: 9/20/2010 10:12:00,
                 ...},
               ...
    ]
}
Simple Stats

db.users.find({'checkins.location': "10gen HQ"})

db.checkins.find({'checkins.location': "10gen HQ"})
  .sort({ts:-1}).limit(10)

db.checkins.find({'checkins.location': "10gen HQ",
  ts: {$gt: midnight}}).count()
Limitations of MongoDB

- No referential integrity
- High degree of denormalization means updating something in many places instead of one
- Lack of predefined schema is a double-edged sword
  - You must have a model in your app
  - Objects within a collection can be completely inconsistent in their fields
Conclusions

- MongoDB is fast
  - Very little CPU overhead
  - MongoDB is Implemented in C++ for best performance
- Very rapid development, open source
- useful when working with a huge quantity of data when the data's nature does not require a relational model
- used when what really matters is the ability to store and retrieve great quantities of data, not the relationships between the elements.
- Works on many platforms and there are many language drivers

http://www.mongodb.org/