NoSQL Overview & MongoDB Basics

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

February 8, 2021

Some slides were adopted from S. Davidson and Z. Ives with permission

Learning Objectives

After this lecture, we will:

- Introduce the NoSQL paradigm
- Discuss the trade-offs between relational and non-relational(NoSQL) databases
- Introduce MongoDB, a document-oriented database
- Learn MongoDB simple queries.

Why NoSQL?

- Databases are no longer one-size-fits-all
- The needs of modern applications do not always match what relational databases provide.
- Every large web platform (e.g. Google, Facebook, LinkedIn) has developed some sort of custom solution to scale.

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

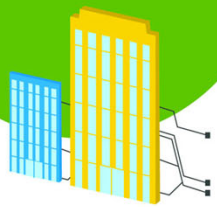
40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

6 BILLION PEOPLE
have cell phones

WORLD POPULATION: 7 BILLION

It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day

Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored



Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month

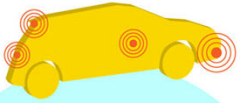
30 BILLION PIECES OF CONTENT
are shared on Facebook every month

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Velocity
ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure



By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Veracity
UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

"Big Data" is two problems

- The **analysis** problem
  - How to extract useful info, using modeling, ML and stats.
- The **storage** problem
  - How to store and manipulate huge amounts of data to facilitate fast queries and analysis
- Problems with traditional (relational) storage
  - Not flexible
  - Hard to partition, i.e. place different segments on different machines
- NoSQL solutions address these problems.

The analysis problem…

- So far, we've focused on the storage problem – flexible schemas. There is also the analysis problem, which requires scalability.
- Relational databases typically scale by getting bigger servers
  - Scaling across multiple servers is complicated
- NoSQL solutions are all about scaling across multiple servers (e.g. cloud instances)
  - Data can be automatically distributed across nodes/servers
  - "Map-reduce" spreads computation across a cluster

Types of NoSQL solutions

- **Key-value stores:**

- **Column-oriented:**

- **Document:**

- **Graph:** Neo4J

Relational-NoSQL Trade-offs

Fundamentally, there are several different trade-offs

- Schema vs. no schema
  - Schema ➔ performance, no schema ➔ flexibility but parse overhead (can have partial schemas like in XML)
- Replication, data partitioning
  - Replicas mean faster queries, slower (consistent) updates
- Level of abstraction
  - High-level queries – parsing, optimization, etc. – vs. low-level operations
- Consistency
  - What does the database do on concurrent updates, especially when distributed?

Outline

- ✓ NoSQL Introduction
- MongoDB
  - Model and simple queries

Overview of mongoDB

- MongoDB is an example of a document-oriented NoSQL solution
- The query language is limited, and oriented around "collection" (relation) at a time processing
- The power of the solution lies in the distributed, parallel nature of query processing
  - Replication and sharding

MondoDB Data Model

- A MongoDB deployment hosts several databases
  - A database holds a set of collections
    - A collection holds a set of documents
      - A document is a set of key-value pairs

RDBMS	MongoDB
Table	Collection
Row(s)	JSON Document
Index	Index
Join	Embedding & Linking
Partition	Shard
Partition Key	Shard Key

Basic data types

- Null
- Boolean
- Integer (32- and 64-bit)
- Floating point
- String
- Date
- ObjectId
- Code (JavaScript)
- Array
- Embedded document

Sample Document

```
mydoc =  {
   _id: 1,
   name: { first: "John", last: "Backus" },
   birthyear: 1924,
   contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
   awards: [ { award_id: "NMS001",
               year: 1975 },
             { award_id: "TA99",
               year: 1977} ]
   }
```

Always indexed, automatically assigned unless provided

Array of documents

Core MongoDB operations

- CRUD: *create*, *read*, *update*, and *delete*
- Insert
  - One at a time: db.people.insert(mydoc)
  - New (version 3.2): db.collection.insertOne(), db.collection.insertMany()
- Delete
  - Documents that match some predicate, e.g. to remove the document in the previous slide:
    db.people.deleteOne({"_id": 1})
    db.people.deleteMany({birthyear: 1924})
  - All documents in a collection: db.people.deleteMany()
    - The collection still remains, with indexes
  - Remove a collection (faster): db.people.drop()

Core MongoDB operations, cont.

- Update documents in a collection
  - db.collection.updateOne(), db.collection.updateMany()

```
db.people.updateMany( {birthyear: 1924}, {$set: {birthyear: 1925}})
db.people.updateMany( {birthyear: 1924}, {$set: {type: "Deceased"}})
```

  - $rename operator: change property name.

```
db.people.update({}, { $rename : {"birthyear": "birth" }})
```

Querying

- Use find( ) function and a query document
- Ranges, set inclusion, inequalities using $ conditionals
- Complex queries using $where clause
- Queries return a database cursor
- Meta-operations on cursor include skipping some number of results, limiting the number of results returned, sorting results.

Another sample document

```
d={
    _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "Kevin",
    date : new Date("February 2, 2012"),
    text : "About MongoDB...",
    birthyear:  1980,
    tags : [ "tech", "databases" ]
    }

> db.posts.insert(d)
```

Find

Return entire collection in posts:

```
db.posts.find( )
```

Return posts that match condition (conjunction):

```
db.posts.find({author: "Kevin", birthyear: 1980})
```

```
{ _id : ObjectId("4c4ba5c0672c685e5e8aabf3"), author : "Kevin",
date : Date("February 2, 2012"), birthyear:  1980,
 text : "About MongoDB...",  tags : [ "tech", "databases" ]}
```

"Pretty" format

- If you want to be able to read the result:

```
db.posts.find({author: "Kevin",  birthyear: 1980}).pretty()
```

```
{
    _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "Kevin",
    date : Date("February 2, 2012"),
    birthyear:  1980,
    text : "About MongoDB...",
    tags : [ "tech", "databases" ]
 }
```

Specifying which keys to return

```
db.people.find({}, {name:1, contribs:1})
```

```
{
   _id: 1,
   name: { first: "John", last: "Backus" },
   contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ]
}
```

```
db.people.find({}, {_id: 0, name:1})
```

```
{
   name: { first: "John", last: "Backus" }
}
```

Ranges, Negation, OR-clauses

- Comparison operators: $lt, $lte, $gt, $gte
  - db.posts.find({birthyear: {$gte: 1970, $lte: 1990}})
- Negation: $ne
  - db.posts.find({birthyear: {$ne: 1982}})
- Or queries: $in (single key), $or (different keys)
  - db.posts.find({birthyear: {$in: [1982, 1985]}})
  - db.posts.find({$or: [{birthyear: 1982}, {author:"John"}], name:"abdu"})

Arrays

- db.posts.find({tags: "tech"})
  - Print complete information about posts which are tagged "tech"
- db.posts.find({tags: {$all: ["tech", "databases"]}}, {author:1, tags:1})
  - Print author and tags of posts which are tagged with both "tech" and "databases" (among other things)
  - Contrast this with:

db.posts.find({tags: ["databases", "tech"]})

Querying Embedded Documents

- db.people.find({"name.first": "John"})
  - Finds all people with first name John
- db.people.find({"name.first": "John", "name.last": "Smith"})
  - Finds all people with first name John and last name Smith.
  - Contrast with

db.people.find({"name": {"first": "John", "last": "Smith"}})

Sample Document

```
mydoc =  {
    _id: 1,
   name: { first: “John”, last: “Backus” },
   birthyear: 1924,
  contribs: [ “Fortran”, “ALGOL”, “Backus-Naur Form”, “FP” ],
   awards: [ { award_id: “NMS001”,
                  year: 1975 },
                { award_id: “TA99”,
                  year: 1977} ]
                   }
```

Summary

- NoSQL solutions address the needs of modern applications
- MongoDB is an example of a document-oriented solution
  - The query language is oriented around "collection" at a time processing
- The power of many of these solutions lies in the distributed, parallel nature of query processing
  - Replication and sharding