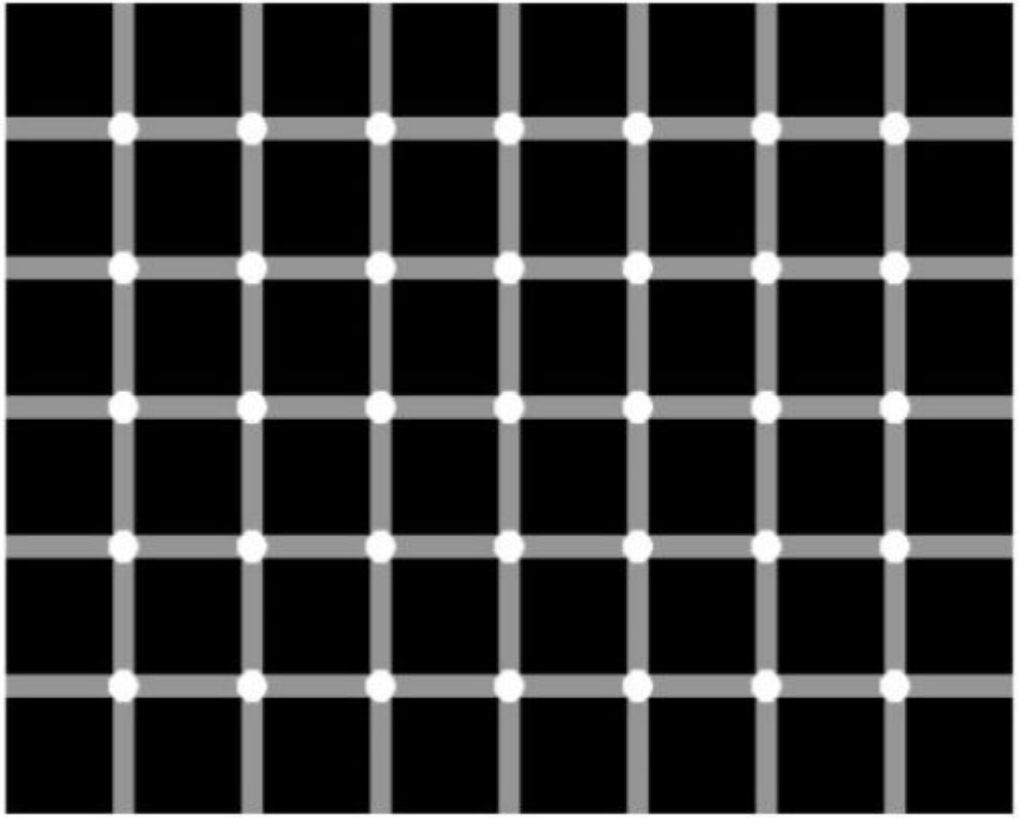
NoSQL Principles and Industrial Applications

Siddharth Srivastava



Agenda

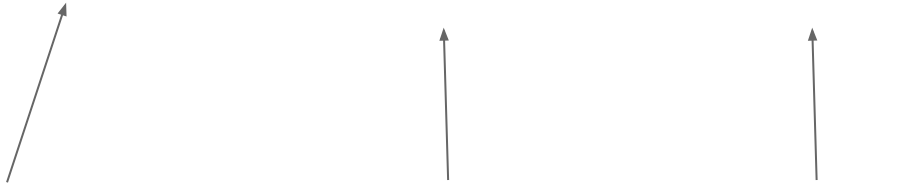
- Current trends in data management & computing
- Relational vs. NoSQL databases
  - the value of relational databases
  - new requirements
  - NoSQL features, strengths and challenges
- Types of NoSQL databases
  - key-value stores, document databases, column-family databases, graph databases
  - principles and examples

RDBMS vs NoSQL - Intuition

Query: Who is the faculty for DBMS at IIIT SriCity ?

RDBMS vs NoSQL: Intuition

Query: Who is the faculty for DBMS at IIIT SriCity



RDBMS vs NoSQL: Intuition

RDBMS Approach

- Look for the subject in Subject Table - DBMS
  - (Subject code, Subject Name)
- Get the subject code - DB001
- Get the location code from Location Table - L001
  - (Location Code, Location Name)
- Obtain the faculty code - F001
  - (Subject Code, Location Code, Faculty Code)
- Get Faculty Details - Dr. Prerana Mukherjee

RDBMS vs NoSQL: Intuition

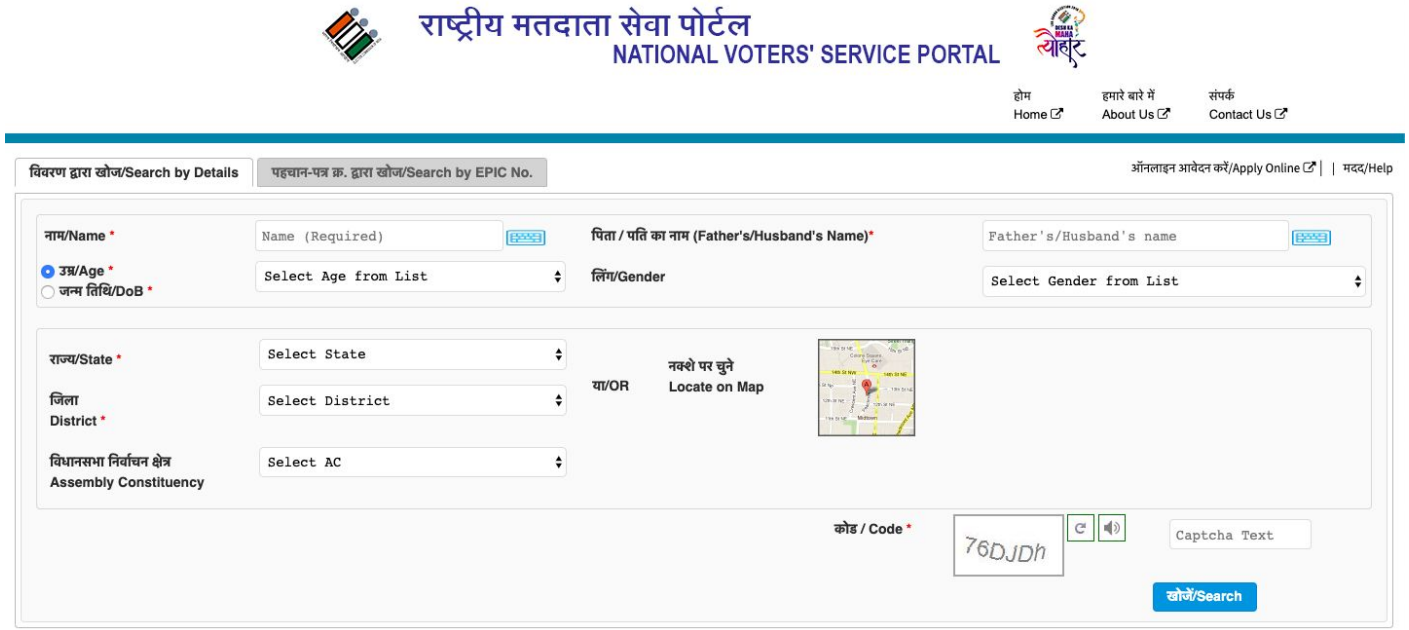
NoSQL Approach

- Look for best match from the available content where
  - Subject is ‘Database’
  - Location is ‘IIIT SriCity’
- Output: Dr. Prerana Mukherjee

(Imagine you are doing CTRL + F in a file)

RDBMS vs NoSQL: Intuition

Now let us scale up the problem - India's National Electoral Search



RDBMS vs NoSQL: Intuition

Challenges (National Electoral Search)

- 80 billion voters
- Each voter has nearly 20 attributes (multilingual name, age, dob, address, polling station, its coordinates etc.)
  - The names across the country are in English and regional languages
  - Lots of missing data
  - Age changes over time
  - Search support with multilingual inputs
  - Location based search - a nightmare !!
- One can update details as well

RDBMS vs NoSQL: Intuition

Challenges (National Electoral Search)

- Response time should be less than 50ms
- Number of concurrent users - about 50 lakhs
- Data size: 6 TeraBytes

RDBMS vs NoSQL: Intuition

How to construct with RDBMS (National Electoral Search)

- Create tables for voter details, constituency details
- Create indexes on all possible search combinations
  - size: 500 GB for one combination, in total 15 search combinations
  - RDBMS have trouble updating large indexes in real time (recall user can update their details)

RDBMS vs NoSQL: Intuition

How to construct with RDBMS (National Electoral Search)

- For response time, ensure that the information is available on multiple servers
  - RDBMS are, *in general*, not good at distributing data
- Location based search - name and his location has input, search within a radius
  - search is extremely slow

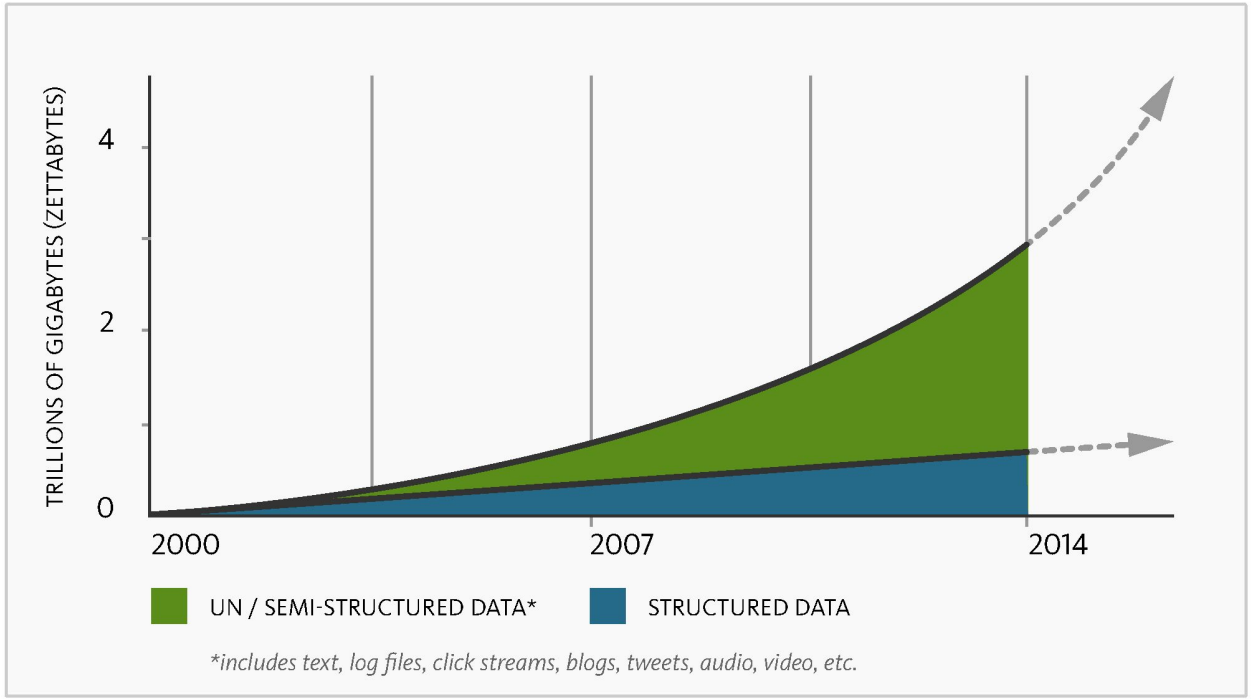
RDBMS vs NoSQL: Intuition

What are we looking for (National Electoral Search)

- A solution than can
  - Manage large indexes and update
  - Distribute and search the data on multiple servers
  - Has very low latency
  - Is reliable
  - Supports heterogeneous fields i.e. data is unstructured

NoSQL has all the above properties and many more...

Current Trends: Big Data



- Volume, Velocity and Variety of data

RDBMS for Big Data

- relational schema
  - data in tuples
  - a priori known schema
- schema normalization
  - data split into tables (3NF)
  - queries merge the data
- transaction support
  - trans. management with ACID
  - Atomicity, Consistency, Isolation, Durability
  - safety first

- but current data are naturally flexible
- inefficient for large data
- slow in distributed environment
- full transactions very inefficient in distributed envir.

NoSQL Databases

- What is “NoSQL”?
  - term used in late 90s for a different type of technology: Carlo Strozzi: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/
  - “Not Only SQL”?
    - but many RDBMS are also “not just SQL”

“NoSQL is an accidental term with no precise definition”

- first used at an informal meetup in 2009 in San Francisco (presentations from Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB, and MongoDB)

[Sadalage & Fowler: NoSQL Distilled, 2012]

Just Another Temporary Trend?

- There have been other trends here before
  - object databases, XML databases, etc.

- But NoSQL databases:
  - are answer to real practical problems big companies have
  - are often developed by the biggest players
  - outside academia but based on solid theoretical results
    - e.g. old results on distributed processing
  - widely used

The End of Relational Databases?

- Relational databases are not going away
  - are ideal for a lot of structured data, reliable, mature, etc.
- RDBMS became one option for data storage

Polyglot persistence – using different data stores in different circumstances [Sadalage & Fowler: NoSQL Distilled, 2012]

Two trends:

1. NoSQL databases implement standard RDBMS features

RDBMS vs NoSQL: At a glance

RDBMS	NoSQL
- Structured and organized data	- Stands for Not Only SQL
- Structured query language (SQL)	- No declarative query language
- Data and its relationships are stored in separate tables	- No predefined schema
- Data Manipulation Language, Data Definition Language	- Key-Value pair storage, Column Store, Document Store, Graph databases
- TightConsistency	- Eventual consistency rather ACID property
- ACID	- CAP Theorem

CAP Theorem

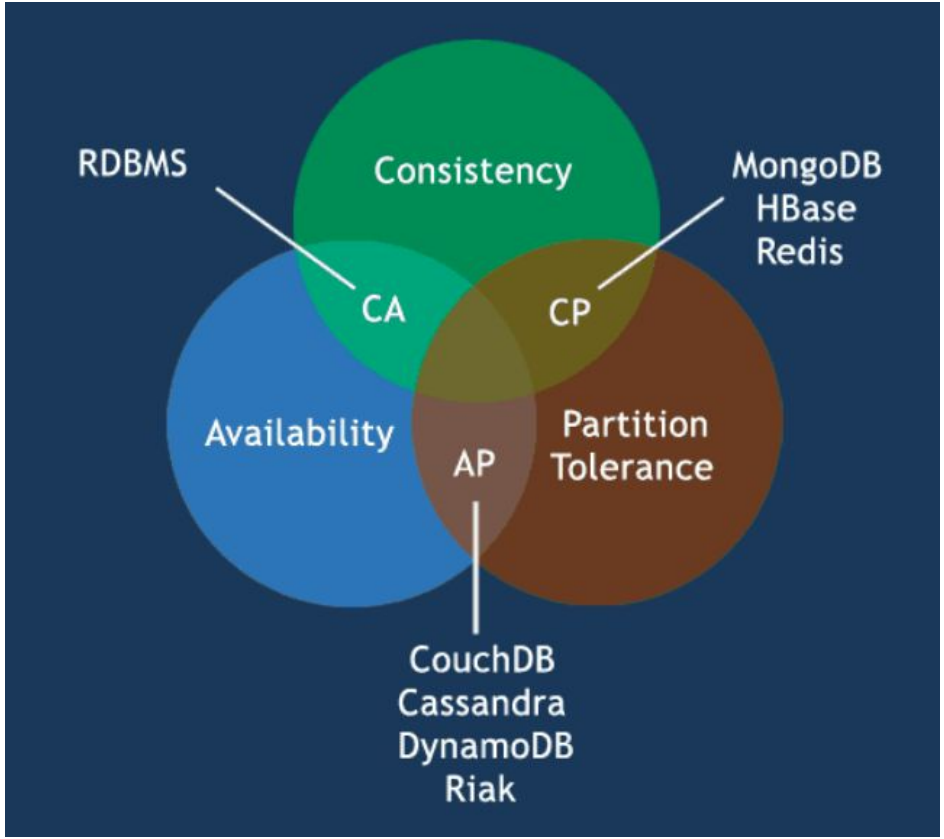
- **Consistency** : datain the database remains consistent after the execution of an operation e.g. after an update operation, all clients see the same data
  - Distributed System
- **Availability**: system is available, no downtime (emphasis in NoSQL)
- **Partition Tolerance**: system continues operation in case of failure (different nodes keep working)

CAP Theorem

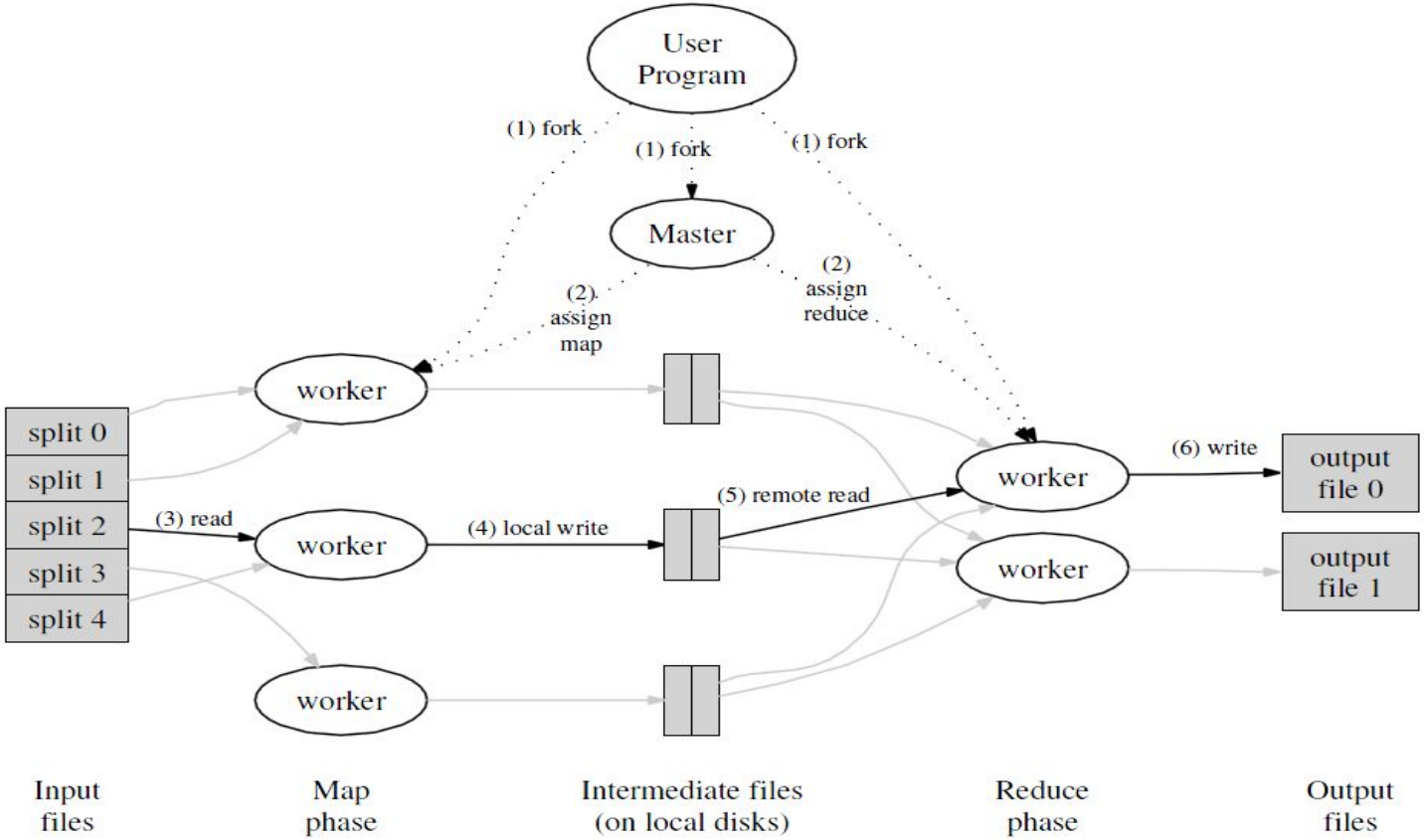
- **Consistency** : datain the database remains consistent after the execution of an operation e.g. after an update operation, all clients see the same data
  - Distributed System
- **Availability**: system is available, no downtime (emphasis in NoSQL)
- **Partition Tolerance**: system continues operation in case of failure (different nodes keep working)

Unlike ACID properties, it is very difficult to fulfill all three

CAP Theorem


RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

Behind the scenes: Map Reduce



source: Dean, J. & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters

Map Reduce: Features

- MapReduce is a generic approach for distributed processing of large data collections

- Requires a way to distribute the data
  - and to collect the results back after the processing

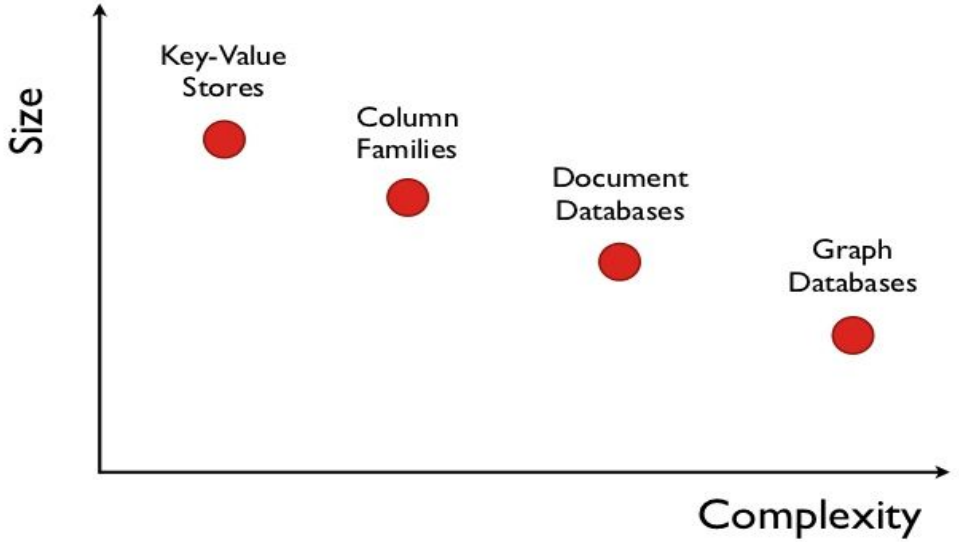
- The user must only specify two functions: **map** & **reduce**

Map Reduce: Implementation

Amazon Elastic
MapReduce

Types of NoSQL Databases

- Key-value stores
- Column-family stores
- Document databases
- Graph databases



* There is not a single solution that is better than the others

source: http://www.slideshare.net/emileifrem/nosql-east-a-nosql-overview-and-the-benefits-of-graph-databases

Key-Value Stores

- A simple hash table (map), primarily used when all accesses to the database are via primary key
  - key-value mapping
- In RDBMS world: A table with two columns:
  - `ID` column (`primary key`)
  - `DATA` column storing the value (unstructured `BLOB`)
- Basic operations:
  - Put a value for a key — `put(key, value)`
  - Get the value for the key — `value:= get(key)`
  - Delete a key-value — `delete(key)`

Key-Value Stores: Architecture

1. Embedded systems
    - the system is a library and the DB runs within your system

2. Large-scale Distributed stores Architecture often as a distributed hash table (DHT)

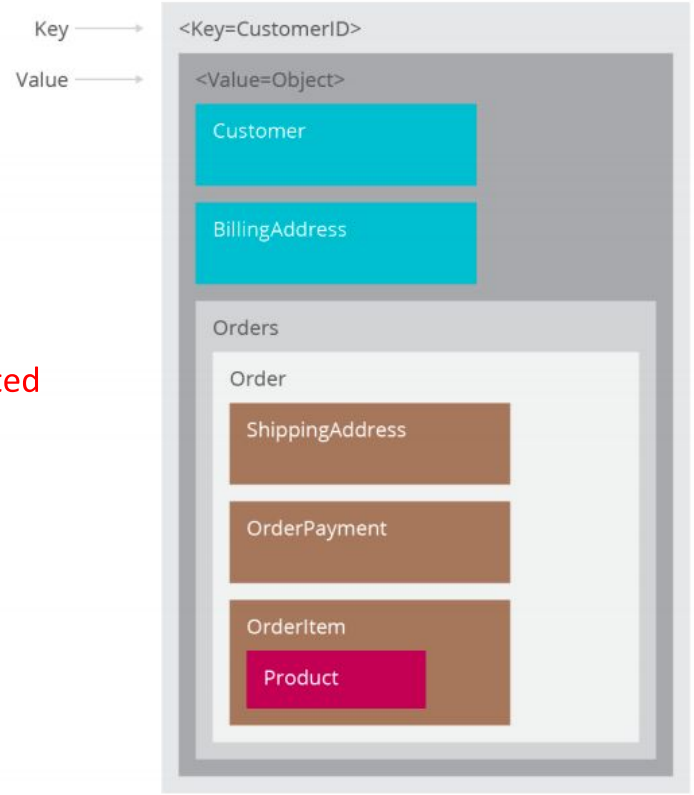
Features: it is simple

- great performance, easily scaled

Key Value Stores

- A key may be strings, hashes, lists, sets, sorted sets
- Key-value stores can be used as collections, dictionaries, associative arrays etc.
- Key-value stores follow the **Availability** and **Partition Tolerance** aspects of CAP theorem
- Suitable for: shopping cart contents, individual values like color schemes

Key Value Stores



Unstructured data associated with a key

Key-value Stores: Representatives

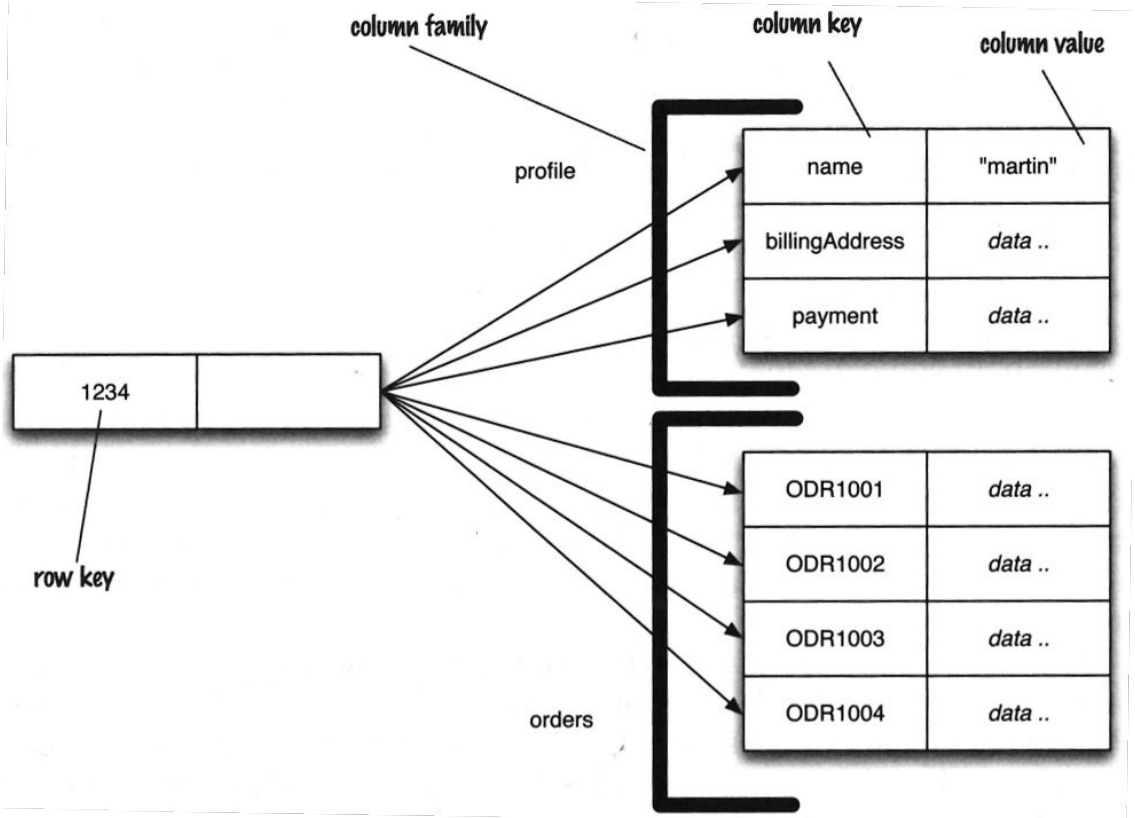
Project Voldemort

Ranked list: http://db-engines.com/en/ranking/key-value+store

Column-family Stores: Basics

- Data model: rows that have many columns associated with a row key
- Column families are groups of related data (columns) that are often accessed together
  - e.g., for a customer we typically access all profile information at the same time, but not customer's orders
- Best for _____ queries (MAX, MIN, COUNT, AVG etc.)

Column-family Stores: Example



source: Sadalage & Fowler: NoSQL Distilled, 2012

Column-family Stores: Representatives

Google BigTable

HYPERTABLE

Ranked list: http://db-engines.com/en/ranking/wide+column+store

Document Databases: Basics

- Basic concept of data: *Document*
- Documents are self-describing pieces of data
  - Hierarchical tree data structures
  - Nested associative arrays (maps), collections, scalars
  - XML, JSON (JavaScript Object Notation), BSON, …
- Documents in a collection should be “similar”
  - Their schema can differ
- Documents stored in the value part of key-value
  - Key-value stores where the values are examinable
  - Building search indexes on various keys/fields

Document Databases: Data Example

```
key=3 ->  { "personID": 3,
            "firstname": "Martin",
            "likes": [ "Biking","Photography" ],
            "lastcity": "Boston",
            "visited": [ "NYC", "Paris" ] }

key=5 ->  { "personID": 5,
            "firstname": "Pramod",
            "citiesvisited": [ "Chicago", "London","NYC" ],
            "addresses": [
               { "state": "AK",
                 "city": "DILLINGHAM" },
               { "state": "MH",
                 "city": "PUNE" }  ],
            "lastcity": "Chicago‟ }
```

source: Sadalage & Fowler: NoSQL Distilled, 2012

Document Databases: Queries

Example in MongoDB syntax

- Query language expressed via JSON
- clauses: where, sort, count, sum, etc.

SQL: `SELECT * FROM users`
MongoDB: `db.users.find()`

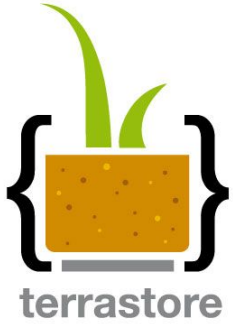
```
SELECT * FROM users WHERE personID = 3
db.users.find( { "personID": 3 } )

SELECT firstname, lastcity FROM users WHERE personID = 5
db.users.find( { "personID": 5}, {firstname:1, lastcity:1}
)
```

Relational vs Document Model

Relational Model	Document Model
Tables	Collections
Rows	Documents
Columns	Key-value pairs
Joins	N/A

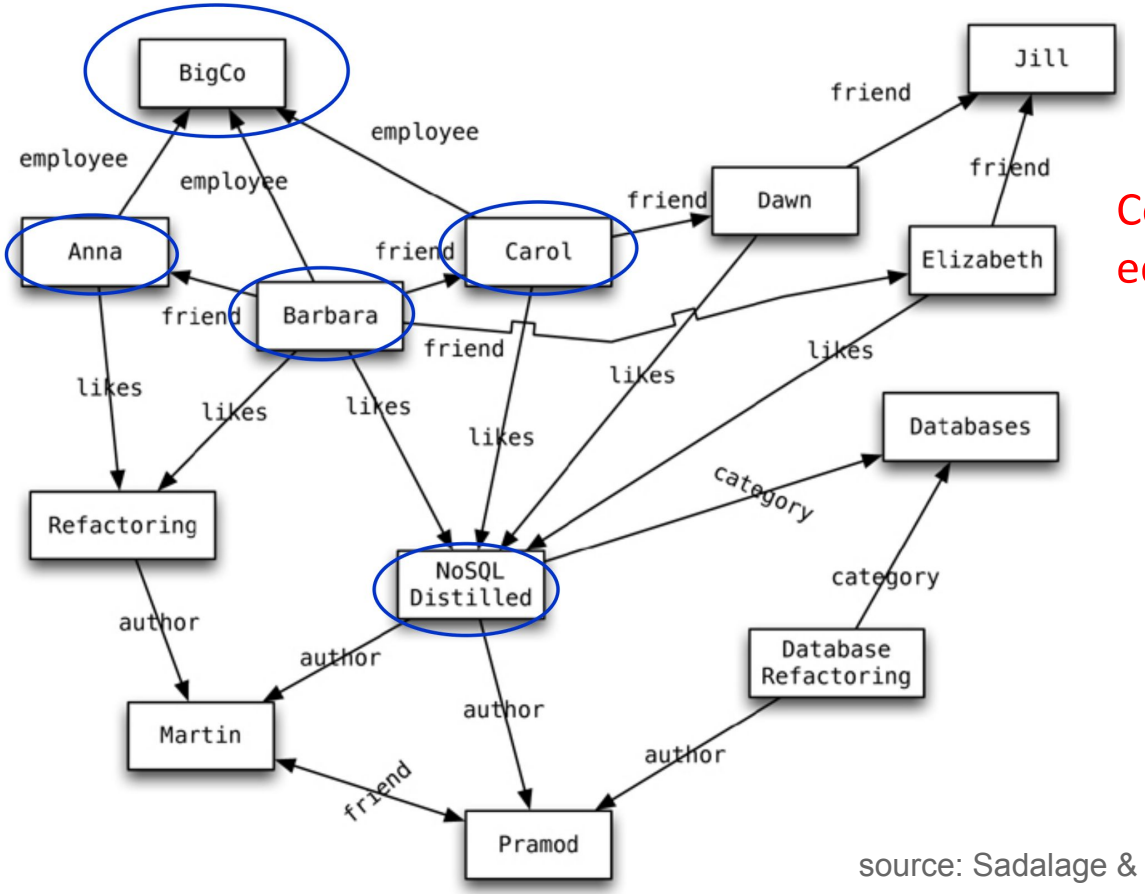
Document Databases: Representatives



MS Azure DocumentDB

Ranked list: http://db-engines.com/en/ranking/document+store

Graph Databases: Example



Collection of nodes, edges and attributes

source: Sadalage & Fowler: NoSQL Distilled, 2012

Graph Databases: Representatives

Ranked list: http://db-engines.com/en/ranking/graph+dbms

One Example: Facebook

Facebook statistics (2016)

- 1.86 billion monthly active users
- 4 million 'likes' per minute
- 250 billion stored photos (350 million uploaded daily)
- 300 PB of user data stored (2014)

2009: 10,000 servers

2010: 30,000 servers

2012: 180,000 servers (estimated)

source: http://expandedramblings.com/index.php/by-the-numbers-17-amazing-facebook-stats/
https://www.brandwatch.com/blog/47-facebook-statistics-2016/

Facebook: Database Tech. Behind

Apache Hadoop http://hadoop.apache.org/

- Hadoop File System (HDFS)
  - over 100 PB in a single HDFS cluster
- an open source implementation of MapReduce:
  - Enables efficient calculations on massive amounts of data

Apache Hive http://hive.apache.org/

- SQL-like access to Hadoop-stored data
- integration of MapReduce query evaluation

sources: http://goo.gl/SZ6jia http://royal.pingdom.com/2010/06/18/the-software-behind-facebook/

Facebook: Database Tech. Behind (2)

Apache HBase http://hbase.apache.org/

- a Hadoop column-family database
- used for e-mails, instant messaging and SMS
- replacement for MySQL and Cassandra

Memcached http://memcached.org/

- distributed key-value store
- used as a cache between web servers and MySQL servers in the beginning of FB

sources: http://goo.gl/SZ6jia http://royal.pingdom.com/2010/06/18/the-software-behind-facebook/

Facebook: Database Tech. Behind (3)

Apache Giraph http://giraph.apache.org/

- graph database
- facebook users and connections is one very large graph
- used since 2013 for various analytic tasks (trillion edges)

RocksDB http://rocksdb.org/

- high-performance key-value store
- developed internally in FB, now open-source

sources: https://code.facebook.com/posts/509727595776839/scaling-apache-giraph-to-a-trillion-edges/ http://goo.gl/XNtG6p

Sharding

What is database sharding ?

Sharding

- A database shard is a horizontal partition of data in a database or search engine.
- Each individual partition is referred to as a shard or database shard
- Each shard is held on a separate database server instance, to spread load.
- Very important for high availability

Applications at C-DAC

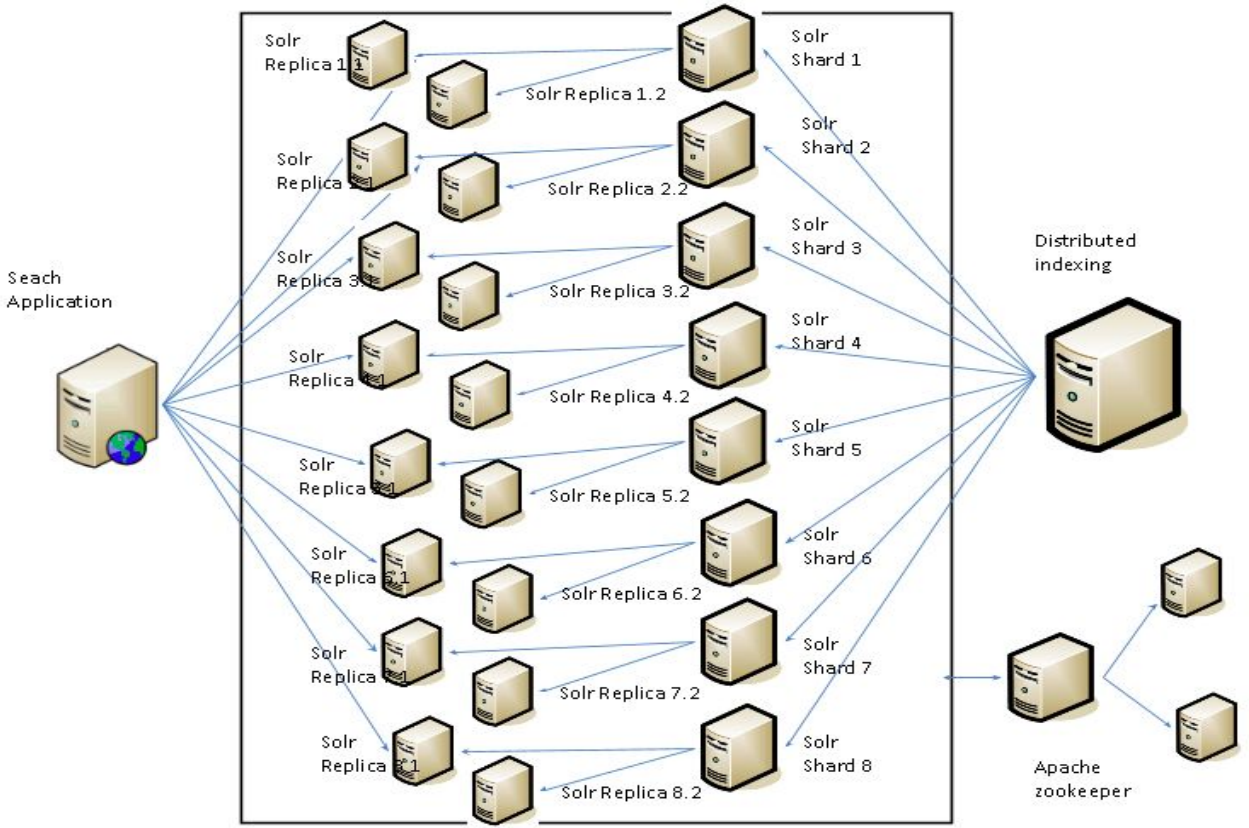
- National Electoral Search (election season !!!)
- e-RaktKosh - A nationwide initiative to reach citizens
- Hospital Management Information Systems (high availability report generation)

National Electoral Search

- We have already discussed the objectives
  - High Availability
  - < 50ms response time
  - > 50 lakh concurrent users
  - Multiple search criteria
  - ...

National Electoral Search

- 6 TB data + indexes + backup
- Envisaged infrastructure
  - 8 Servers
  - 64 GB RAM each



National Electoral Search

How did we end up doing it ?

- 2 servers (8GB RAM each - commodity hardwares)
- Lucene (not publicized as NoSQL)
- The voter data was partitioned as per the states
- Indexes were created on each state’s data
- For location mapping, a custom algorithm was developed

Hospital Management System

Task: Generate patient reports for various investigations and stats

Biochemistry

CR No	: 109111900354637	**Lab/Study No.**	: 0003	**Requisition Date**	: 01-Apr-2019 10:17
Patient Name	: Eeeemm	**Age/Sex**	: 34 Yr /M	**Coll./Study Date**	: 01-Apr-2019 10:17
Sample Type/No	: Serum/0003	**Ward/OPD**	: OPD	**Reporting Date**	: 01-Apr-2019 10:25
Dept/Unit	: General Medicine Casualty Unit				

Investigation	Result	Unit	Ref. Range
	Kidney Function Test		
Serum Urea	**122**	mg/dL	13 - 43
Creatinine	**122**	mg/dl	0.7 - 1.3
Uric Acid	**122**	mg/dl	3.5 - 7.2
Colour			
Serum Calcium	**122**	mg/dL	8.6 - 10
Chloride	**87**	meq/l	98 - 107
File Upload	-		

Comments:

********** END OF THE REPORT **********

Hospital Management System

Challenges

- Data from 300 hospitals for 1000 patients daily
- Each patient has 500 records (average)
- 150 million records each day
- At peak time, 1 million records each second
- Highly unstructured - each patient has different tests and values

Hospital Management System

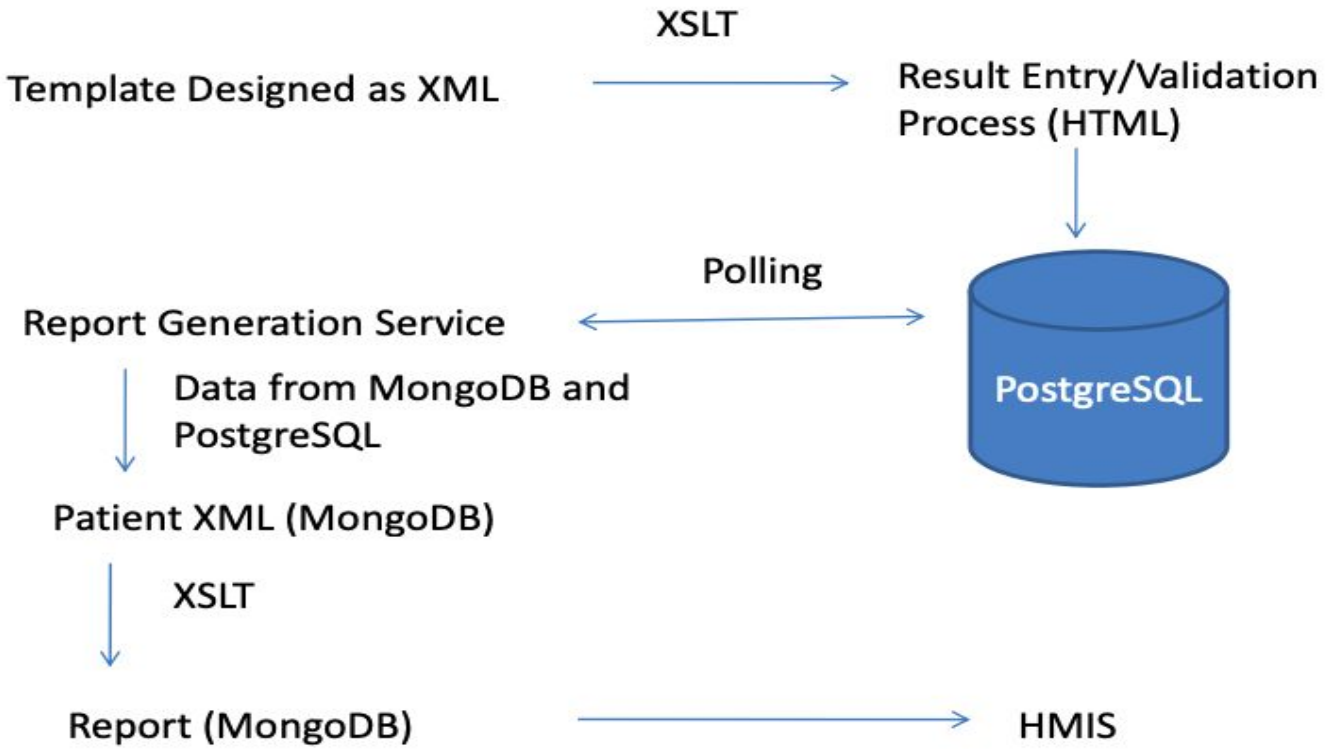
Solution

- Utilize strength of both RDBMS and NoSQL
- RDBMS
  - Good at mapping test and values
- NoSQL
  - Good at storing and retrieving data

Hospital Management System

- Utilize strength of both RDBMS and NoSQL
  - RDBMS
    - Good at mapping test and values
    - PostgreSQL
  - NoSQL
    - Good at storing and retrieving data
    - MongoDB GridFS
- The concept is being used in upcoming Distributed Health System (DHS) for entire country

Hospital Management System



e-RaktKosh (www.eraktkosh.in)

Use Case

- **Task: Find update frequency, who missed the update, outliers, discover hidden stats**

NoSQL Use Case

- 1800 Blood Banks across the country update blood stock daily
- Average update frequency is 4 times a day
- Each update inserts 120 records
  - 864000 records a day
  - 25 million records per month
- Need to be available to
  - Citizens (Portal, Mobile Apps, UMANG)
  - Ministers (CM Dashboards)
  - Blood banks (API)

NoSQL Use Case

- Expected Approach
  - For each blood bank we create a document
  - The document contains
    - Details of Blood Bank
    - Blood Stock
    - Updation date and other meta data
  - Traverse through the document with aggregation queries

NoSQL Use Case

- Expected Approach
  - For each blood bank we create a document
  - The document contains
    - Details of Blood Bank
    - Blood Stock
    - Updation date and other meta data
  - Traverse through the document with aggregation queries

Is it the optimal approach ?

NoSQL Use Case

- Recall NoSQL schema design is application specific
- If statistics are required, better arrange them in such a form (high performance use case)

NoSQL Use Case

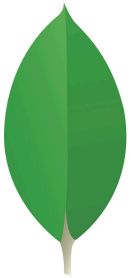
- Approach followed
  - Construct another document where on each insert aggregate statistics
  - Can be done in RDBMS also, but data is distributed so RDBMS approach is not optimal

Questions?

Please, any questions? Good question is a gift...

*MongoDB (from "hu**mongo**us") is a scalable, high-performance, open source, schema-free, document-oriented database.*

-- Mongodb.org

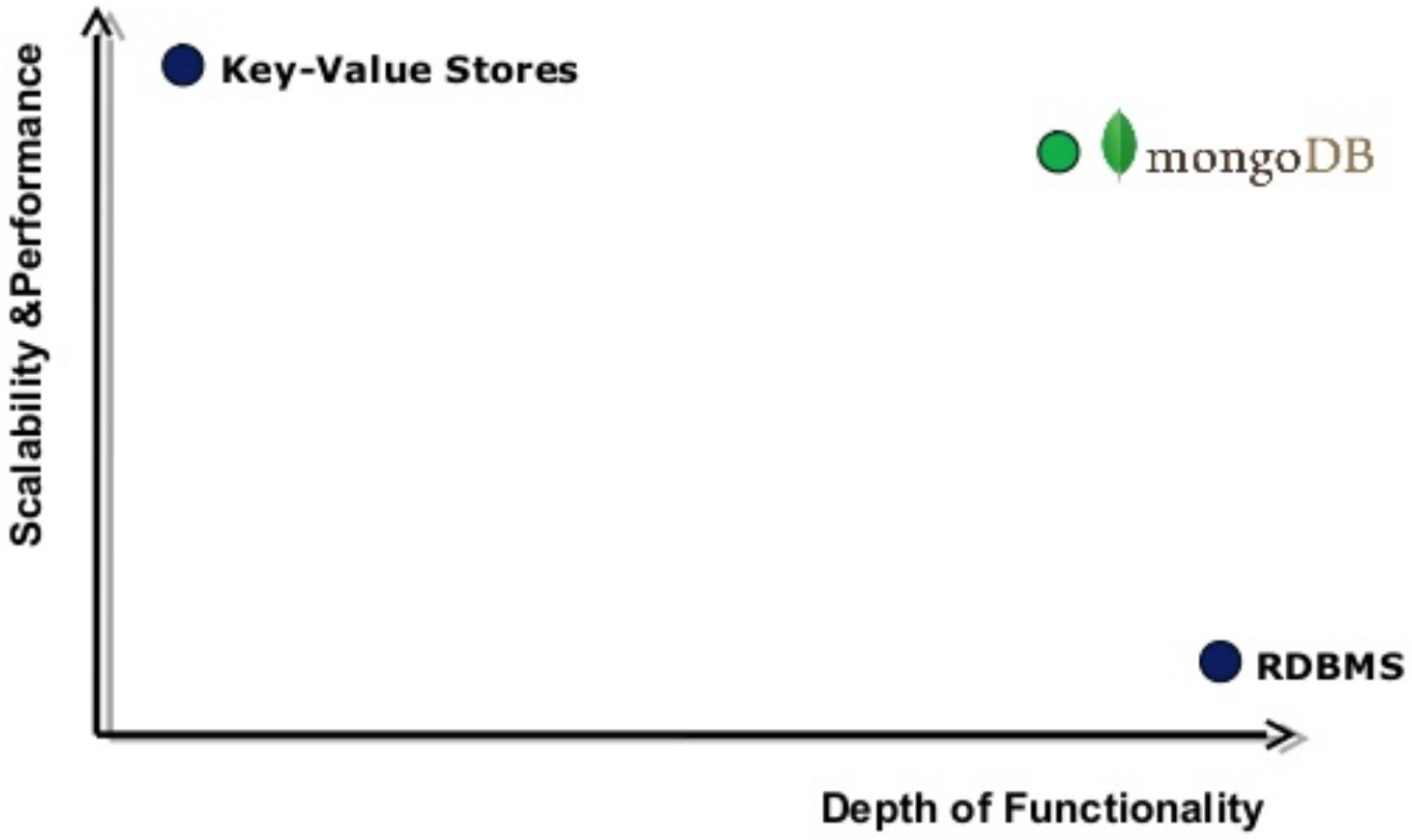


mongoDB

Background

- A NoSQL database of type document oriented
- Eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON)
- First developed by MongoDB Inc in October 2007
- Shifted to open source in 2009

Where MongoDB Stands?



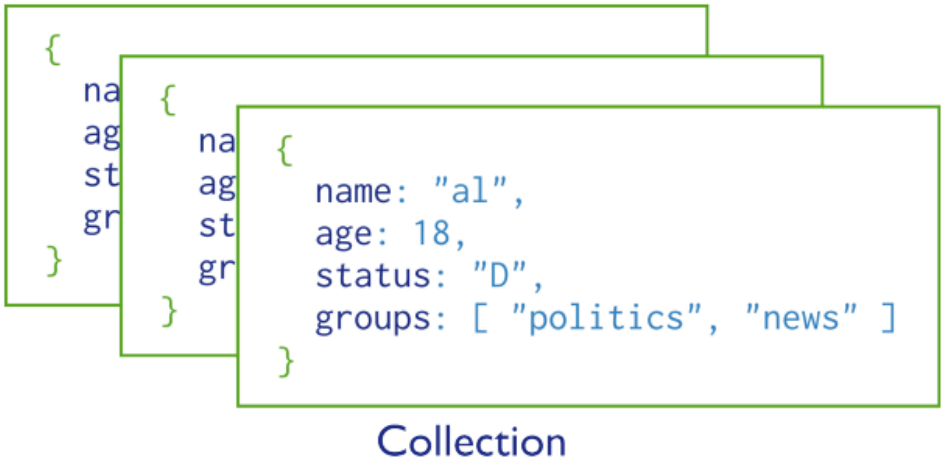
http://www.slideshare.net/cuelogic/introduction-to-mongodb-33683242

Examples of JSON Format


```
"filter_level":"medium",
"contributors":null,
"text":"Do you think neymar will score his first goal tonight ???",
"geo":{
        "type":"Point",
        "coordinates":[3[illegible],3[illegible]]
},
"retweeted":false,
"in_reply_to_screen_name":null,
"truncated":false,
"lang":"en",
"entities":{
        "symbols":[],
        "urls":[],
        "hashtags":[],
        "user_mentions":[]
},
"in_reply_to_status_id_str":null,
"id":363356918[illegible],
"source":"<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Android<\/a>",
"in_reply_to_user_id_str":null,
"favorited":false,
"in_reply_to_status_id":null,
"retweet_count":0,
"created_at":"Fri Aug 02 17:53:34 +0000 2013",
"in_reply_to_user_id":null,
"favorite_count":0,
"id_str":"36335691[illegible]",
"place":null,
"user":{
        "location":"",
        "default_profile":false,
        "profile_background_tile":true,
        "statuses_count":8100,
        "lang":"en",
        "profile_link_color":"0099B9",
        "profile_banner_url":"https://pbs.twimg.com/profile_banners/41433[illegible]",
        "id":4[illegible],
        "following":null,
        "protected":false,
        "favourites_count":855,
        "profile_text_color":"3C3940",
        "description":"My dream is all my life [illegible]",
```


http://www.clearpeaks.com/blog/oracle-endeca-information-discovery/twitter_and_oracle_endeca_information_discovery

MongoDB Basics (cont.)



Each document within a collection can have its own unique set of fields

```
{
  name: "sue",                    <-- field: value
  age: 26,                        <-- field: value
  status: "A",                    <-- field: value
  groups: [ "news", "sports" ]    <-- field: value
}
```

MongoDB CRUD

- MongoDB provides rich semantics for reading and manipulating data
- CRUD = Create, Read, Update, and Delete

CRUD: Create

In MongoDB

```
db.users.insert (
   {
     name: "sue",
     age: 26,
     status: "A"
   }
)
```

collection

field: value

field: value

field: value

document

In SQL

```
INSERT INTO users
            ( name, age, status )
VALUES      ( "sue", 26, "A" )
```

table

columns

values/row

CRUD: Read

In MongoDB

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

In SQL

```
SELECT _id, name, address
FROM   users
WHERE  age > 18
LIMIT  5
```

← projection
← table
← select criteria
← cursor modifier

CR**U**D: Update

In MongoDB

```
db.users.update(                          <-- collection
    { age: { $gt: 18 } },                 <-- update criteria
    { $set: { status: "A" } },            <-- update action
    { multi: true }                       <-- update option
)
```

In SQL

```
UPDATE users              <-- table
SET    status = 'A'       <-- update action
WHERE  age > 18           <-- update criteria
```

CRU**D**: Delete

In MongoDB

```
db.users.remove(
    { status: "D" }
)
```

collection

remove criteria

In SQL

```
DELETE FROM users
WHERE  status = 'D'
```

table

delete criteria

Using MongoDB

- You can either install MongoDB on your machine, or visit: http://www.tutorialspoint.com/mongodb_terminal_online.php

To Get Started…

- Global commands: `help`, `exit`, etc.
- Commands execute against the current database are executed against the db object, for example:
  - `db.help()`: returns a list of commands that you can use against db object
  - Note: `db.help` without `()` gives you the method body

Create Database

- To create a wonderland database:

```
use wonderland
```

  * creates the database and switches to it

- To get the collections in the current database:

```
db.getCollectionNames()
```

Insert Data

- To insert a document into the collection:

```
db.unicorns.insert(
    {name: 'Aurora',
    gender: 'f',
    weight: 450}
)
```

* Try out db.getCollectionNames() now, you'll see:

List Documents in a Collection

- Try out:

```
db.unicorns.find()
```

- One more field is added: `_id`
  - Every document must have a unique `_id` field
  - Can generate your own or have MongoDB generate automatically for you

Remove all data: db.unicorns.remove({}).
Get the data from: http://bit.ly/iiitsdbms

Query Selector (cont.)

- Use

```
{field: value}
```

  to select documents that match the condition.

- If matching multiple conditions is desired:

```
{field1: value1, field2: value2…}
```

  * This implies the _____ statement

Comparison Operators in MongoDB

- `$lt` less than
- `$lte` less than or equal to
- `$gt` greater than
- `$gte` greater than or equal to
- `$ne` not equal to

(Q1) Find the male unicorns weigh more than 700 pounds

Ans1:

```
db.unicorns.find({gender: 'm', weight: {$gt: 700}})
```

(Q2) Find the unicorns that have no vampire field

Ans 2:

```
db.unicorns.find({ vampires: {$exists: false}})
```

(Q3) Find the unicorns that like apples or oranges

Ans 3:

db.unicorns.find({ loves: {$in:
[‘apple','orange']}})

(Q4) Find the female unicorns that either love apples or weigh less than 500 pounds

Ans 4:

```
db.unicorns.find({gender: 'f', $or: [{loves: 'apple'}, {weight: {$lt: 500}}]})
```

CR**U**D: Update

- Intuitively, updating unicorn Roooooooodles' weight to 590 can be:

```
db.unicorns.update(
    {name: "Roooooooodles"},
    {weight: 590})
```

- But if you try:

```
db.unicorns.find({name: "Roooooooodles"})
```

the result will be: _____

CR**U**D: Update (cont.)

- The reason that no document was found was because the second parameter we supplied didn't have any update operators
- Therefore, the original document was replaced
- Try the following command to see:

```
db.unicorns.find({weight: 590})
```

CR**U**D: Update (cont.)

- To fix the problem, we should do:

```
db.unicorns.update({weight: 590},
 {$set: {name: “Roooooodles”,
         dob: new Date(1979, 7, 18, 18, 44),
         loves: [“apple”],
         gender: “m”,
         vampires: 99}})
```

CR**U**D: Update (cont.)

- The correct way to update at the beginning should therefore be:

```
db.unicorns.update({name: “Roooooodles”},
                   {$set: {weight: 590}})
```

More Update Operators

- `$inc`: increment a field by a certain positive or negative amount
- `$push`: add a value to the existing field

(Q5) Decrease unicorn Pilot’s number of vampires by 2

Ans 5:

```
db.unicorns.update({name: 'Pilot'}, {$inc: {vampires: -2}})
```

(Q6) Add “sugar” to the list of food unicorn Aurora loves to eat

Ans 6:

```
db.unicorns.update({name: 'Aurora'}, {$push: {loves: ‘sugar'}})
```

Projection

- find() can take a second argument, which is the project list
- Example:

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

  – The values following field names are boolean:
    - 1 means including the field
    - 0 means excluding the field
  – Note that except excluding _id, the list cannot have a mixture of exclusion and inclusion

Upserts

- An `upsert` updates the document if found or inserts it if not
- To enable upserting we pass a third parameter to update `{upsert: true}`

Upserts (cont.)

- This will not do anything:

```
db.unicorns.update({name: "Walala"},
        {$inc: {vampires: 1}})
```

- Instead, do this:

```
db.unicorns.update({name: "Walala"},
        {$inc: {vampires: 1}},
        {upsert: true})
```

Multiple Updates

- By default, update will only update a single document. Passing the third parameter `{multi: true}` will enable the multiple update

(Q7) Give all of the unicorns vaccine (set vaccinated to be true)

Ans 7:

```
db.unicorns.update({}, {$set: {vaccinated: true }},
{multi:true});

db.unicorns.find({vaccinated: true});
```

(Q8) Sort the unicorns based on weights decreasingly

Ans 8:

```
db.unicorns.find().sort({weight: -1})
```

(Q9) Sort the unicorns based on the names increasingly, then the number of vampires decreasingly

Ans 9:

```
db.unicorns.find().sort({name: 1, vampires: -1})
```

(Q10) Get the second and third heaviest unicorns

Ans: 10

```
db.unicorns.find() .sort({weight: -1}) .limit(2) .skip(1)
```

(Q11) Count the number of unicorns who have more than 50 vampires

Ans 11:

```
db.unicorns.count({vampires: {$gt: 50}})
```

References

- Karl Seguin, *The Little MongoDB Book*, http://openmymind.net/mongodb.pdf
- Kristina Chodorow, *MongoDB: The Definite Guide*, O'Reilly
- MongoDB CRUD Operations, https://docs.mongodb.org/master/MongoDB-crud-guide-master.pdf

MongoDb(No SQL Database)

Questions

Structure of file “restaurants.json”


```
{
 "address": {
    "building": "1007",
    "coord": [ -73.856077, 40.848447 ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
 },
 "borough": "Bronx",
 "cuisine": "Bakery",
 "grades": [
    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
 ],
 "name": "Morris Park Bake Shop",
 "restaurant_id": "30075445"
}
```


1. Write a MongoDB query to display all the documents in the collection restaurants

2. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.

3. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine, but exclude the field _id for all the documents in the collection restaurant.

4. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.

5. Write a MongoDB query to display all the restaurant which is in the borough Bronx.

6. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.

7.Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx.

8. Write a MongoDB query to find the restaurants who achieved a score more than 90.

9. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

10. Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.

11. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.

12. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168.

Note : Do this query without using $and operator.

13. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' not belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order.

14. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.

15. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'ces' as last three letters for its name.

16. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Reg' as three letters somewhere in its name.

17. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.

18. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronxor Brooklyn.

19. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronxor Brooklyn.

20. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10.

21. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.

22. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates.

23. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".

24. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52.

25. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

26. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

27. Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

28. Write a MongoDB query to know whether all the addresses contains the street or not.

29. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.

30. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.

31. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name.

32. Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.