Document Stores
Knowledge Objectives

1. Explain the main difference between key-value and document stores
2. Justify why indexing is a first-class citizen for document-stores and it is not for key-value stores
Application Objectives

1. Given an application layout and a small query workload, design a document-store providing optimal support according to a given set of criteria
Structuring the Value

- Essentially, they are key-value stores
  - Same design and architectural features
- The value is a document
  - XML (e.g., eXist)
  - JSON (e.g., MongoDB and CouchDB)
- Tightly related to the Web
  - Normally, they provide RESTful HTTP APIs
- So... what is the benefit of having documents?
  - New data model (collections and documents)
    - New atom: from rows to documents
  - Indexing
Designing Document Stores

- Follow one basic rule: 1 fetch for the whole data set at hand
  - Aggregate data model: check the data needed by your application simultaneously
    - Do not think relational-wise!
  - Use indexes to identify finer data granularities

- Consequences:
  - Independent documents
    - Avoid pointing FKs (i.e., pointing at other docs)
  - Massive denormalization
  - A change in the application layout might be dramatic
    - It may entail a massive rearrangement of the database documents
JSON Document-Stores

- JSON-like documents
  - MongoDB
  - CouchDB

- JSON is a lightweight data interchange format
  - Brackets ([ ]) represent ordered lists
  - Curly braces ({} ) represent key-value dictionaries
    - Keys must be strings, delimited by quotes ("")
    - Values can be strings, numbers, booleans, lists, or key-value dictionaries

- Natively compatible with JavaScript
  - Web browsers are natural clients

http://www.json.org/index.html
JSON Example

- **Definition:** A document is an object represented with an unbounded nesting of array and object constructs.

```json
{
    "title": "The Social network",
    "year": "2010",
    "genre": "drama",
    "summary": "On a fall night in 2003, Harvard undergrad and computer programming genius Mark Zuckerberg sits down at his computer and heatedly begins working on a new idea. In a fury of blogging and programming, what begins in his dorm room soon becomes a global social network and a revolution in communication. A mere six years and 500 million friends later, Mark Zuckerberg is the youngest billionaire in history... but for this entrepreneur, success leads to both personal and legal complications.",
    "country": "USA",
    "director": {
        "last_name": "Fincher",
        "first_name": "David",
        "birth_date": "1962"
    },
    "actors": [
        {
            "first_name": "Jesse",
            "last_name": "Eisenberg",
            "birth_date": "1983",
            "role": "Mark Zuckerberg"
        },
        {
            "first_name": "Rooney",
            "last_name": "Mara",
            "birth_date": "1985",
            "role": "Erica Albright"
        },
        {
            "first_name": "Andrew",
            "last_name": "Garfield",
            "birth_date": "1983",
            "role": "Eduardo Saverin"
        },
        {
            "first_name": "Justin",
            "last_name": "Timberlake",
            "birth_date": "1981",
            "role": "Sean Parker"
        }
    ]
}
```
MongoDB: Data Model

- **Collections**
  - *Definition*: A grouping of MongoDB documents
    - A collection exists within a single database
    - Collections do not enforce a schema
  - **MongoDB Namespace**: `database.collection`

- **Documents**
  - *Definition*: JSON documents (serialized as BSON)
    - Basic atom
    - Identified by `_id` (user or system generated)
    - Aggregated view of data
    - May contain
      - References *(NOT FKs!)* and
      - Embedded documents
MongoDB: Document Example

```
user document
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
contact document
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

```
access document
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```
MongoDB: Document Example

```json
{
    _id: <ObjectId1>,
    username: "123xyz",
    contact: {
        phone: "123-456-7890",
        email: "xyz@example.com"
    },
    access: {
        level: 5,
        group: "dev"
    }
}
```
Activity

- **Objective:** Learn how to model documents
- **Tasks:**
  1. (15’) Model the TPC-H using documents
  2. (5’) Discussion

```sql
SELECT l_orderkey,
       sum(l_extendedprice*(1-l_discount)) as revenue,
       o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]' AND c_custkey = o_custkey AND l_orderkey = o_orderkey AND o_orderdate < '[DATE]' AND l_shipdate > '[DATE]'
GROUP BY l_orderkey,
        o_orderdate, o_shippriority
ORDER BY revenue desc,
        o_orderdate;
```
MongoDB Shell

- Show dbs
- Use `<database>`
- Show collections
- Show users
- `coll = db.<collection>`
- `Find(criteria, projection)`
- `Insert(document)`
- `Update(query, update, options)`
- `Save(document)`
- `Remove(query, justOne)`
- `Drop()`
- `EnsureIndex(keys, options)`

**Notes:**
- `db` refers to the current database
- `query` is a document (query-by-example)

MongoDB: Querying

- Find and findOne methods
  
  ```javascript
  database.collection.find()
  database.collection.find( { qty: { $gt: 25 } } )
  database.collection.find( { field: { $gt: value1, $lt: value2 } } )
  ```

- Aggregation Framework

  - An aggregation pipeline
  - Documents enter a multi-stage pipeline that transforms them into an aggregated result
    - *Filters* that operate like queries
    - *Document transformations* that modify the form of the output
    - Grouping
    - Sorting
    - Other operations

- MapReduce
MongoDB: The Aggregation Framework

Collection

```javascript
db.orders.aggregate(
  $match phase: { $match: { status: "A" } },
  $group phase: { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
)
```

- `{ 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

$match

$group

Results

- `{ _id: "A123", total: 750 }
- `{ _id: "B212", total: 200 }`
MongoDB: Architecture

Shard Server (primary)  Shard Server (secondary)  Shard Server (primary)  Shard Server (secondary)

Key space
- INF  chunk1  chunk2  chunk3  chunk4  chunk5  + INF

Keep a map between key range and chunk
Keep a map between chunk and shard

Config Server (mongod)  Route Server (balancer)  Route Server (mongos)

Keep a map between key range and chunk
Keep a map between chunk and shard

Monitor the write request and statistically estimate whether the chunk reaching its max capacity. Initiate split request if necessary

For update, route request based on doc shard key
For query, route request if find criteria contain shard key. Otherwise, do scatter and gather
For sort by shard key, route request sequentially
For sort by non-shard key, scatter the sort request and merge/sort the result from different servers

Periodically poll chunk distribution among shards
Trigger migration when unbalance is detected

Client Lib
App


March 2014
Oscar Romero
MongoDB: Storage

MongoDB: Fragmentation

- Based on *sharding* (horizontal fragmentation)
  - Shard key: a mandatory field in all documents of the collection
  - Chunk: Hash-based or range-based horizontal fragmentation according to the shard key
    - Range-based: MongoDB determines the chunks by range
      - Adequate for range queries
    - Hash-based: Consistent hashing (a hash function determines the chunks)
      - A Hash-indexed field is required

![Diagram of chunk distribution and hash function]
MongoDB: Notation

- Shard clusters
  - Shards: Nodes containing data
    - A shard may contain several chunks
  - Config Servers: Nodes containing the global catalog (e.g., hash directory)
  - Query routers: Nodes containing a copy of the hash directory to redirect queries
MongoDB: Shard Clusters Management

- Query routers are replicas of the config servers
  - Secondary versioning (config servers)
  - Eager replication (to both config servers and query routers)
    - 2PCP (potential distributed deadlocks!)

- Config Servers
  - The hash directory is mandatorily replicated to avoid single-point failures
    - MongoDB asks for 3 config servers
  - Writes happen if:
    - A shard splits
    - A chunk migrates between servers (e.g., adding servers)

- Query routers
  - Read from config servers
    - When they start (on restart)
    - Every time a split / migration happens
Document Stores

MongoDB: Splitting/Migrating Chunks

- Default chunk size: 64MB
- The query router (mongos) asks a shard to split
  - Inserts and updates trigger splits
- Shards rearrange the data (data migration)
  - During the migration requests to that chunk address the origin shard
  - Changes made during the migration are afterwards applied in the destination shard
  - Finally, changes in the hash directory are made in the config servers
    - Query routers eagerly synchronized
- A balancer avoids uneven distributions
MongoDB: Replication

- Each *shard* (in a *shard cluster*) is a *replica set*
  - Maps to a *mongod* instance (with its config servers)
- Replica Set: Master versioning with lazy replication
  - One master
    - Write / Update / Delete
  - Several replicas
    - Reads
- Replica Set management
  - The master has recovery system
    - Writes / Updates / Deletes and index modifications are kept in memory (@master)
    - Specific recovery system (*journaling*): WAL redo logging
    - When the journal (i.e., log) is flushed to disk it is deleted
  - Members interconnected by heartbeats
  - If the master fails, voting phase to decide a new master
    - PAXOS (arbiters allowed)
  - If a replica fails, it catches up with the master once back
MongoDB: Well-Known Limitations

- Architectural Issues
  - Thumb rule: 70% of the database must fit in memory
  - Be careful with updates! (padding)
    - Holes caused by reallocation
    - Compact the database from time to time

- Document Issues
  - The resulting document of an aggregation pipeline cannot exceed the maximum document size (16Mb)
    - GridFS for larger documents
  - Attribute names are kept as they are (no catalog)

- Querying Issues
  - No transactions
    - Consistency only guaranteed at document level
    - Strong / loose consistency parametrizable
  - Thumb rule: A query must attack a single collection
  - The aggregation capabilities are still rather immature
  - No optimizer!
Summary

- Document-stores
  - Semi-structured value
  - Indexing
- Designing document-stores
- MongoDB
Bibliography

- D. Battre et al. *Nephele/PACTs: A Programming Model and Execution Framework for Web Scale Analytical Processing*. SoCC’10
Resources

- https://www.mongodb.org
- http://exist-db.org