MongoDB Overview

MongoDB is an Open-source database.

Developed by 10gen, for a wide variety of applications. Now it is named as MongoDB.

It is an agile database that allows schemas to change quickly as applications evolve.

Scalability, High Performance and Availability.

By leveraging in-memory computing.

MongoDB’s native replication and automated failover enable enterprise-grade reliability and operational flexibility.
Indexes provide high performance read operations for frequently used queries.

This section introduces indexes in MongoDB, describes the types and configuration options for indexes, and describes special types of indexing MongoDB supports.

The section also provides tutorials detailing procedures and operational concerns, and providing information on how applications may use indexes.
What is MongoDB?

- Developed by 10gen
- Founded in 2007
- A document-oriented, NoSQL database
- Hash-based, schema-less database
- No Data Definition Language
- In practice, this means you can store hashes with any keys and values that you choose
- Keys are a basic data type but in reality stored as strings
- Document Identifiers (_id) will be created for each document, field name reserved by system
- Application tracks the schema and mapping
- Uses BSON format
- Based on JSON – B stands for Binary
- Written in C++
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang
What is MongoDB?

- Open Source
- Document Storage
- Object Oriented
- Written in C++
- Easy to Use
- Full Index Support
What is MongoDB? (Cont....)
Why Should We Use MongoDB?

▷ Document Oriented Storage: Data is stored in the form of JSON style documents
▷ Index on any attribute
▷ Replication & High Availability
▷ Auto-Sharding
▷ Rich Queries
▷ Fast In-Place Updates
▷ Map Reduce functions
▷ Professional Support By MongoDB
Database

• Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Document

• At the heart of MongoDB is the *document: an ordered set of keys with associated values.*

• The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary. In JavaScript, for example, documents are represented as objects

Collection

• Collection is a group of MongoDB documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

• Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.
What is a Document?

- This simple document contains a single key, "_id", with a value of "123"
- Most documents will be more complex than this simple one and often will contain multiple key/value pairs:
  
  ```
  { 
    _id: "123",
    title: "MongoDB: The Definitive Guide ",
    authors: [ 
      { _id: "kchodorow", name: "Kristina Chodorow" },
      { _id: "mdirold", name: "Mike Dirolf" } 
    ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher: { 
      name: "O’Reilly Media", founded: "1980", location: "CA"
    }
  }
  ```
Functionality of MongoDB

- Dynamic schema
  - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
  - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based
  - Partitioning of data (sharding)
- No joins nor transactions
Why use MongoDB?

- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems
Focus on Consistency and Partition tolerance

- **Consistency**
  - All replicas contain the same version of the data

- **Availability**
  - System remains operational on failing nodes

- **Partition tolarance**
  - Multiple entry points
  - System remains operational on
  - System split

**CAP Theorem:**

satisfying all three at the same time is impossible
• A MongoDB instance may have zero or more ‘databases’
• A database may have zero or more ‘collections’.
• A collection may have zero or more ‘documents’.
• A document may have one or more ‘fields’.
• MongoDB ‘Indexes’ function much like their RDBMS counterparts.
Mongod – Database instance

Mongos - Sharding processes
• Analogous to a database router.
• Processes all requests
• Decides how many and which mongods should receive the query
• Mongos collates the results, and sends it back to the client.

Mongo – an interactive shell (a client)
• Fully functional JavaScript environment for use with a MongoDB

You can have one mongos for the whole system no matter how many mongods you have

OR you can have one local mongos for every client if you wanted to minimize network latency.
Choices made for Design of MongoDB

- Scale horizontally over commodity hardware
  - Lots of relatively inexpensive servers

- Keep the functionality that works well in RDBMSs
  - Ad hoc queries
  - Fully featured indexes
  - Secondary indexes

- What doesn’t distribute well in RDB?
  - Long running multi-row transactions
  - Joins
  - Both artifacts of the relational data model (row x column)
JSON stands for JavaScript Object Notation, lightweight data-interchange format and language independent. It is "self-describing and easy to understand.

An example product in this API is:

```json
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```
Data is in name / value pairs

- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”

- Data is separated by commas:
  - Example: “name”: “R2-D2”, race: “Droid”

Curly braces hold objects

- Example: {“name”: “R2-D2”, race: “Droid”, affiliation: “rebels”}

An array is stored in brackets[

- Example: [ “rebels”], {“name”: “R2-D2”, race: “Droid”, affiliation: “rebels”}

{“name”: “Yoda”, affiliation: “rebels”} ]
Binary-encoded serialization of JSON-like documents

Zero or more key/value pairs are stored as a single entity

Each entry consists of a field name, a data type, and a value

Large elements in a BSON document are prefixed with a length field to facilitate scanning

\x16\x00\x00\x00\x02hello
\x00\x06\x00\x00\x00world
\x00\x00
JSON-style Documents Represented as BSON

```
{"hello": "world"} ()
```

Flexible “Schemas”

```
{  "author": "mike",
    "text": "...
} 

{  "author": "eliot",
    "text": "...
    "tags": ["mongodb"]
} 
```
Limited BNF of a BSON Document

<table>
<thead>
<tr>
<th></th>
<th>::=</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td><code>int32 e_list &quot;\x00&quot;</code></td>
<td>BSON Document</td>
</tr>
<tr>
<td>e_list</td>
<td><code>element e_list</code></td>
<td>Sequence of elements</td>
</tr>
<tr>
<td>element</td>
<td><code>&quot;\x01&quot; e_name data type</code></td>
<td>Specific data type</td>
</tr>
<tr>
<td>e_name</td>
<td><code>cstring</code></td>
<td>Key name</td>
</tr>
<tr>
<td>string</td>
<td><code>int32 (byte*) &quot;\x00&quot;</code></td>
<td>String</td>
</tr>
<tr>
<td>cstring</td>
<td><code>(byte*) &quot;\x00&quot;</code></td>
<td>CSTRING</td>
</tr>
<tr>
<td>binary</td>
<td><code>int32 subtype (byte*)</code></td>
<td>Binary</td>
</tr>
<tr>
<td>subtype</td>
<td><code>&quot;\x00&quot;</code></td>
<td>Binary / Generic</td>
</tr>
<tr>
<td></td>
<td><code>&quot;\x01&quot;</code></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td><code>&quot;\x02&quot;</code></td>
<td>Binary (Old)</td>
</tr>
<tr>
<td></td>
<td><code>&quot;\x03&quot;</code></td>
<td>UUID (Old)</td>
</tr>
<tr>
<td></td>
<td><code>&quot;\x04&quot;</code></td>
<td>UUID</td>
</tr>
<tr>
<td></td>
<td><code>&quot;\x05&quot;</code></td>
<td>MD5</td>
</tr>
<tr>
<td></td>
<td><code>&quot;\x80&quot;</code></td>
<td>User defined</td>
</tr>
<tr>
<td>code_w_s</td>
<td><code>int32 string document</code></td>
<td>Code w/ scope</td>
</tr>
</tbody>
</table>
MongoDB does not need any pre-defined data schema
Every document in a collection could have different data
• Addresses NULL data fields

```
{name: “will”,
  eyes: “blue”,
  birthplace: “NY”,
  aliases: [“bill”, “la ciacco”],
  loc: [32.7, 63.4],
  boss: ”ben”}
```

```
{name: “jeff”,
  eyes: “blue”,
  loc: [40.7, 73.4],
  boss: “ben”}
```

```
{name: “brendan”,
  aliases: [“el diablo”]}
```

```
{name: “matt”,
  pizza: “DiGiorno”,
  height: 72,
  loc: [44.6, 71.3]}
```

```
{name: “ben”,
  hat: ”yes”}
```
MongoDB Features

- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce functionality

Scalable

Agile
Index Functionality

- B+ tree indexes
- An index is automatically created on the _id field (the primary key)
- Users can create other indexes to improve query performance or to enforce Unique values for a particular field
- Supports single field index as well as Compound index
  - Like SQL order of the fields in a compound index matters
  - If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array
- Sparse property of an index ensures that the index only contain entries for documents that have the indexed field. (so ignore records that do not have the field defined)
- If an index is both unique and sparse – then the system will reject records that have a duplicate key value but allow records that do not have the indexed field defined
Collection
BSON document
a set of fields
key-value pair
key: a name (string)
val: any basic type
string, int, float, date, binary, array, document, ...

Collection
BSON document
db.users
db.products
db.barbieDollsYouHideFromYourFriends
Few MongoDB Clients

- Rangespan
- MetLife
- Craigslist
- Foursquare
- GOV.UK
- McAfee
- CMS
- MTV
- Man | AHL
- The National Archives
- Lombard Odier
- RMS
- National Cancer Institute
- Met Office
- Otto
- US Department of Veterans Affairs
- FCC
- GAP
- Dillard's
- US Department of Energy’s Berkley Lab and MIT
Metlife uses MongoDB for “The Wall” an innovative customer service application provides a 360-degree consolidated view of MetLife customers, including policy details and transactions across lines of business.

eBay has a number of projects running on MongoDB for search suggestions, metadata storage, cloud management and merchandizing categorization.

MongoDB is a repository that powers MTV Networks’ next generation CMS, which is used to manage and distribute content for all of MTV Networks major websites.

MongoDB is used for back-end storage on the SourceForge front pages projects pages, and download pages for all projects.

Craigslist uses MongoDB to archive billions of records.

ADP uses MongoDB for its high performance, scalability, reliability and its ability to preserve the data manipulation capabilities of traditional relational databases.
Industry Domain Where MongoDB is Used

- Government
- Financial Services
- Healthcare
- Media and Entertainment
- Tele-communications
- Retail
Financial Services

- Risk Analytics and Reporting
- Reference Data Management
- Market Data Management
- Portfolio Management
- Order Capture
- Time Series Data
Government

- Surveillance Data Aggregation
- Crime Data Management and Analytics
- Citizen Engagement Platform
- Program Data Management
- Healthcare Record Management
Health Care

- 360-Degree Patient View
- Population Management for At-Risk Demographic
- Lab Data Management and Analytics
- Mobile Apps for Doctors and Nurses
- Electronics Healthcare Records (HER)
Media and Entertainment

- Content Management and Delivery
- User Data Management
- Digital Asset Management
- Mobile and Social Apps
- Content Archiving
Rich Product Catalogs
Customer Data Management
New Services
Digital Coupons
Real-Time Price Optimization
Telecommunication

- Consumer Cloud
- Product Catalog
- Customer Service Improvement
- Machine-to-Machine (M2M) Platform
- Real-Time Network Analysis and Optimization
CRUD operations

- Create
  - `db.collection.insert(<document>)`
  - `db.collection.save(<document>)`
  - `db.collection.update(<query>, <update>, { upsert: true })`

- Read
  - `db.collection.find(<query>, <projection>)`
  - `db.collection.findOne(<query>, <projection>)`

- Update
  - `db.collection.update(<query>, <update>, <options>)`

- Delete
  - `db.collection.remove(<query>, <justOne>)`

Collection specifies the collection or the ‘table’ to store the document
CRUD examples

> db.user.insert(
  
    {first: "John",
     last: "Doe",
     age: 39
    }

> db.user.find ()

{ 
   "_id" : ObjectId("51"),
   "first" : "John",
   "last" : "Doe",
   "age" : 39
}

> db.user.update(

  { "_id" : ObjectId("51")},

  { 
    $set: {
      age: 40,
      salary: 7000
    }
  }

> db.user.remove(

 {}

> db.user.remove(

  { "first": /^J/
  })

© 2015 BlueCamphor Technologies (P) Ltd.
Db.collection specifies the collection or the ‘table’ to store the document

- `db.collection_name.insert(<document>)`
  - Omit the `_id` field to have MongoDB generate a unique key
  - Example `db.parts.insert({type: “screwdriver”, quantity: 15 })`
  - `db.parts.insert({_id: 10, type: “hammer”, quantity: 1 })`

- `db.collection_name.update(<query>, <update>, { upsert: true })`
  - Will update 1 or more records in a collection satisfying query

- `db.collection_name.save(<document>)`
  - Updates an existing record or creates a new record
Read Operations

- `db.collection.find(<query>, <projection>).cursorModified`
  - Provides functionality similar to the SELECT command `<query>` where condition, `<projection>` fields in result set
  - Has cursors to handle a result set
  - Can modify the query to impose limits, skips, and sort orders.
  - Can specify to return the ‘top’ number of records from the result set

Example: `var PartsCursor = db.parts.find({parts: "hammer"}).limit(5)`

- `db.collection.findOne(<query>, <projection>)`
## Query Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$eq</td>
<td>Matches value that are equal to a specified value</td>
</tr>
<tr>
<td>$gt, $gte</td>
<td>Matches values that are greater than (or equal to) a specified value</td>
</tr>
<tr>
<td>$lt, $lte</td>
<td>Matches values less than or (equal to) a specified value</td>
</tr>
<tr>
<td>$ne</td>
<td>Matches values that are not equal to a specified value</td>
</tr>
<tr>
<td>$in</td>
<td>Matches any of the values specified in an array</td>
</tr>
<tr>
<td>$nin</td>
<td>Matches none of the values specified in an array</td>
</tr>
<tr>
<td>$or</td>
<td>Joins query clauses with a logical OR returns all</td>
</tr>
<tr>
<td>$and</td>
<td>Join query clauses with a logical AND</td>
</tr>
<tr>
<td>$not</td>
<td>Inverts the effect of a query expression</td>
</tr>
<tr>
<td>$nor</td>
<td>Join query clauses with a logical NOR</td>
</tr>
<tr>
<td>$exists</td>
<td>Matches documents that have a specified field</td>
</tr>
</tbody>
</table>
Update Operations

- `db.collection_name.insert(<document>)`
  - Omit the `_id` field to have MongoDB generate a unique key
  - Example `db.parts.insert({type: "screwdriver", quantity: 15 })`
  - `db.parts.insert({_id: 10, type: "hammer", quantity: 1 })`

- `db.collection_name.save(<document>)`
  - Updates an existing record or creates a new record

- `db.collection_name.update(<query>, <update>, { upsert: true })`
  - Will update 1 or more records in a collection satisfying query

- `db.collection_name.findAndModify(<query>, <sort>, <update>, <new>, <fields>, <upsert>)`
  - Modify existing record(s) – retrieve old or new version of the record
Delete Operations

- `db.collection_name.remove(<query>, <justone>)`
  - Delete all records from a collection or matching a criterion
  - `<justone>` - specifies to delete only 1 record matching the criterion
  - Example: `db.parts.remove(type: /^h/ } )` - remove all parts starting with h
  - `Db.parts.remove()` – delete all documents in the parts collections
Aggregation framework provides SQL-like aggregation functionality

- Pipeline documents from a collection pass through an aggregation pipeline, which transforms these objects as they pass through
- Expressions produce output documents based on calculations performed on input documents
- Example: total quantity

```
db.parts.aggregate ( {$group: { _id: type, : { $sum: quantity} } } )
```
Map reduce functionality

- Performs complex aggregator functions given a collection of keys, value pairs
- Must provide at least a map function, reduction function and a name of the result set
- More description of map reduce next lecture
Indexes: High performance Read

- Typically used for frequently used queries

- Necessary when the total size of the documents exceeds the amount of available RAM.

- Defined on the collection level
  - Can be defined on 1 or more fields
    - Composite index (SQL)  Compound index (MongoDB)

- B-tree index

- Only 1 index can be used by the query optimizer when retrieving data

- Index covers a query - match the query conditions and return the results using only the index;
  - Use index to provide the results.
Replication of Data

- Ensures redundancy, backup, and automatic failover
  - Recovery manager in the RDMS

- Replication occurs through groups of servers known as replica sets
  - Primary set – set of servers that client tasks direct updates to
  - Secondary set – set of servers used for duplication of data
  - At most can have 12 replica sets
    - Many different properties can be associated with a secondary set i.e. secondary-only, hidden delayed, arbiters, non-voting

- If the primary set fails the secondary sets ‘vote’ to elect the new primary set
Consistency of Data

All read operations issued to the primary of a replica set are consistent with the last write operation

- Reads to a primary have **strict consistency**
  - Reads reflect the latest changes to the data
- Reads to a secondary have **eventual consistency**
  - Updates propagate gradually
- If clients permit reads from secondary sets – then client may read a previous state of the database
- Failure occurs before the secondary nodes are updated
  - System identifies when a rollback needs to occur
  - Users are responsible for manually applying rollback changes
A memory-mapped file is a segment of virtual memory which has been assigned a direct byte-for-byte correlation with some portion of a file or file-like resource.

**mmap()**
Other Additional Features

Supports geospatial data of type

- Spherical
  - Provides longitude and latitude
- Flat
  - 2 dimensional points on a plane
- Geospatial indexes
NoSQL built to address a distributed database system

- Sharding
- Replica sets of data

CAP Theorem: consistency, availability and partition tolerant

MongoDB

- Document oriented data, schema-less database, supports secondary indexes, provides a query language, consistent reads on primary sets
- Lacks transactions, joins
thank you!