APAN PS5400: Managing Data

Week 9: NoSQL Databases, MongoDB

Lecturer: Shekhar Pradhan

COLUMBIA UNIVERSITY
School of Professional Studies

Recap of last week

- NoSQL data bases
- CAP theorem
- Cassandra database system
- Downloading and installing Cassandra

What are MongoDB databases?

- Another type of NoSQL database
- huMONGOus database
- Developed by 10Gen, a NYC company
- Document-oriented
  - CouchDB is another document-oriented model
- Schema less (schema optional)
- Sharding
- Replication
- Horizontal scaling

Document oriented

- The fundamental data storage unit is a document, containing zero or more fields
  - A document in MongoDB corresponds to a row in relational databases
    - The fields corresponds to columns in relational databases
- A Collection is a set of documents
  - Corresponds to a Table in relational databases
- The documents in a collection do not all have to have the same set of fields
  - It is in this sense that MongoDB databases are schema less
  - Documents are stored as binary versions of JSON objects (BSON)

MongoDB document example

```
{
_id: “456”,
name: “Brad”,
sign: “Gemini”.
email: “pitt@pittstop.com”
}
```

Note: No NULL value for home_town field

```
{
_id: “389”,
name: “Angelina”,
email: “jolie@GoToHellBrad.com”,
home_town: [“LA”, “Barcelona”]
}
```

Note: No NULL value for sign field

What if we wanted to indicate which is the permanent home town and which is temporary?

A document in MongoDB is stored as a binary serialization of a JSON type document.

Namespaces

- A namespace is like a database in relational database systems

  -
    ```
    > use tutorial
    Switched to db tutorial
    ```
  - Creates a namespace called ‘tutorial’ and makes it available for use
    - So no need to explicitly create a namespace before using it
  - All subsequent commands are assumed to pertain to this namespace unless a command to use another namespace is executed
- A namespace consists of a collection of documents

Collections

- Collections are sets of documents
  - These are like tables in relational database systems
- Documents with very different fields can be stored in a collection.
  - But it makes sense from the point of efficient search to keep more or less similarly structured documents, about related topics be stored in the same collection.
- > db.students.insert({name: “John Doe”})
- Notice: we created the collection students just by telling MongoDB to insert a document into it—no need to explicitly create the collection.
  - Similarly, the document inserted doesn’t have to be created prior to insertion.

Finding documents

```
> db.students.find() //select all documents in collection ‘students’
{“_id”:ObjectId(“5c806c41839f4c0f6a40767”), “name”: “John Doe”}
```

- The find() command returns a JSON object
  - The “_id” key and its value are added automatically to the document at the time of insert
  - The primary index of the document is the _id field.

Find with query predicate

```
> db.students.insert({name: “Jane Smith”})
```

This command adds another document to students collection

```
> db.students.find()
{ "_id" : ObjectId("5c806c41839f4c0f6a407676"), "name" : “John Doe" }
{ "_id" : ObjectId("5c80738a839f4c0f6a407677"), "name" : "Jane Smith" }
> db.students.find({name: “Jane Smith”})
```

//the field ‘name’ and the value ‘Jane Smith is a query predicate Asking for all documents with ‘Jane Smith’ in the ‘name’ field. It returns the following:

```
{ "_id" : ObjectId("5c80738a839f4c0f6a407677"), "name" : "Jane Smith" }
```

Updating documents—Operator Update

```
> db.students.update({name: “John Doe”}, {$set: {country: “USA”}})
//Update the {name: “John Doe”} document by adding the field ‘country’ with the values ‘USA’
>db.students.find({name: “John Doe”})
{ "_id" : ObjectId("5c806c41839f4c0f6a40767"), "name" : “John Doe”, “country”: “USA” }

>db.students.update({name: “John Doe”}, {$set: {name: “Johnny”}})
//change the name to ‘Johnny’
```

Updating documents—Replacement Update

To replace one document with another document in the collection use replacement update.

```
>db.students.update({name: “Johnny”}, {status: “new”})
```

That is, without the $set operator

```
>db.students.find()
{ "_id" : ObjectId("5c806c41839f4c0f6a407677"), "name" : “Jane Smith" }
{ "_id" : ObjectId("5c80738a839f4c0f6a407676"), "status" : "new" }
```

Deletes

To delete all the documents from a collection (w/o deleting the collection)

```
> db.students.remove()
```

To delete just a specific document

```
> db.students.remove({name: “Johnny”})
```

To delete the collection

```
>db.students.drop()
```

Efficient Find of Documents

- If a MongoDB has hundreds of documents, how can it efficiently find documents?
  - Suppose you want to find all documents with a certain id or a certain name
    - “Find me all documents about Brad Pitt”
- It would be very inefficient looking through all the documents to see which of them might be about Brad Pitt?
- How does a search engine like Google do that?
- What advantage might MongoDB have over the web documents searched by Google?
- Can the fact that documents in MongoDB are in JSON (or BSON format) provide any advantage over searching web documents?

Indexing

- MongoDB databases are automatically indexed on the _id field (primary index)
- You can also create indices on other fields—secondary indices
    - Up to 64 per collection allowed
- Indices enable efficient search
- In this respect, it is like an RDBMS

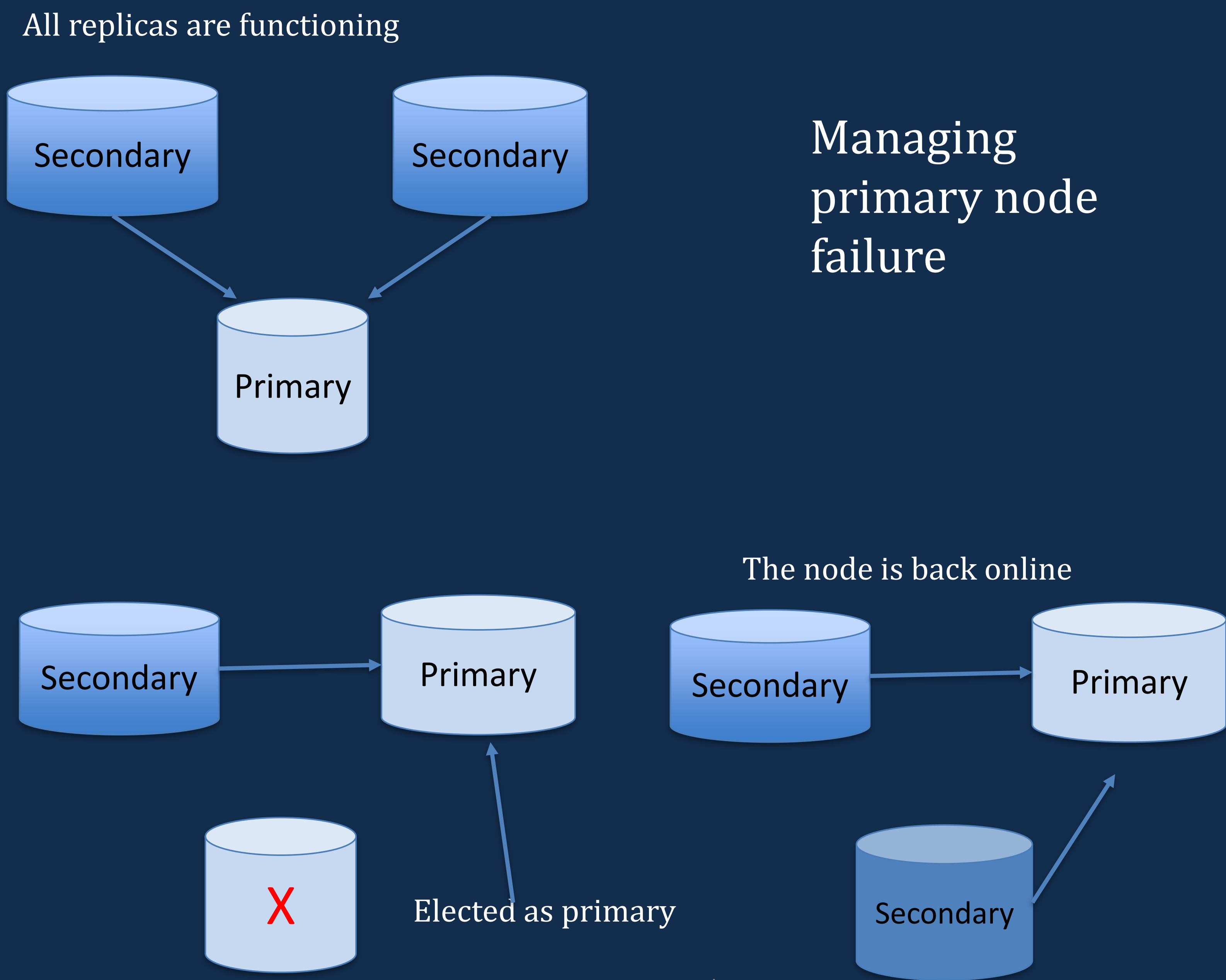
– But it is schema less

– Has sharding and replication to make it faster than RDBMS

Replicas

- Each MongoDB database (or each shard) is replicated in a set of replicas (‘replica set’)
- Each replica set has a primary replica (‘master’) and the other replicas are secondary (‘slaves’)
  - Unlike in Cassandra in which has no master replicas
  - This difference is crucial to understanding their different ways of handling consistency and availability

Managing primary node failure



Primary node fails

Consistency

- All writes are to the primary replica, which are then propagated to the secondary replicas
- What if the primary replica fails?
  - The secondary replicas vote for one of them to be primary, and the write is directed to that one.
  - When the failed replica comes back online, the write is propagated to it
    - Thus, ensuring consistency (eventual consistency)
- Reads are to the primary (strong consistency)
  - Optionally to the secondary (the new primary), if the primary fails (eventual consistency in that case)

Availability

- By distributing the members of replica sets on different servers, racks, clusters, etc.
- Through a mechanism for electing a primary replica among those replicas that are not down
  - So the primary going down does not make the data in the replica set unavailable for read or write.

In the remaining time we will discuss installing and running MongoDB