Introduction to MongoDB – a noSQL Database System

NoSQL Systems

Alternative to traditional relational DBMS

+ Flexible schema
+ Quicker/cheaper to set up
+ Massive scalability
+ Relaxed consistency $\rightarrow$ higher performance & availability

– No declarative query language $\rightarrow$ more programming
– Relaxed consistency $\rightarrow$ fewer guarantees

– Examples: MongoDB, Hadoop/Hive, Cassandra, Apache SPARK

NoSQL Systems

- Drivers:
  - Massive growth of data
  - Joins are expensive across multiple machines
  - Licensing fees of traditional databases

Scalability

- Traditional databases and NoSQL databases take different approaches:
  - Traditional databases use vertical scaling: more CPUs, more RAM, and more storage
  - NoSQL databases use horizontal scaling or *sharding*
    - Decompose data across multiple computers and do distributed queries

NoSQL: The Name

- "SQL" = Traditional relational DBMS
- Recognition over past decade or so:
  Not every data management/analysis problem is best solved using a traditional relational DBMS
- "NoSQL" = "No SQL" =
  Not using traditional relational DBMS
- "No SQL" ≠ Don't use SQL language

NoSQL: The Name

- "SQL" = Traditional relational DBMS
- Recognition over past decade or so:
  Not every data management/analysis problem is best solved using a traditional relational DBMS
- "NoSQL" = "No SQL" =
  Not using traditional relational DBMS
- "No SQL" ≠ Don't use SQL language

✳ "NoSQL" = "Not Only SQL"

NoSQL Systems*

Several incarnations

- Key-value stores
- Document stores
- Graph database systems
- MapReduce framework
- Column stores

* For a more comprehensive survey, refer to

http://web.eecs.umich.edu/~mozafari/winter2014/eecs684/papers/nosql-book.pdf

Key-Value Stores

Extremely simple interface

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)

Implementation: efficiency, scalability, fault-tolerance

- Records distributed to nodes based on key
- Replication
- Weak (no?) transactional semantics (to achieve high availability)
  - No atomicity: Single-record atomicity only. No notion of grouping multiple updates into one transaction.
  - Weak durability: (In MongoDB) Writes are only “durable” after 100 ms by default. Thus, a crash could lose updates that appear to be "committed"
  - Weak consistency across replicas: Replicas not guaranteed to remain in sync between updates and reads. Update at replica 1 is not atomically propagated. Subsequent reads at other replicas may get old values!

Key-Value Stores

Extremely simple interface

- Data model: (key, value) pairs
- Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)
- Some allow (non-uniform) columns within value
- Some allow Fetch on range of keys

Example systems

- Google BigTable, Amazon Dynamo, Cassandra, Voldemort, HBase, …

Case Study: MongoDB

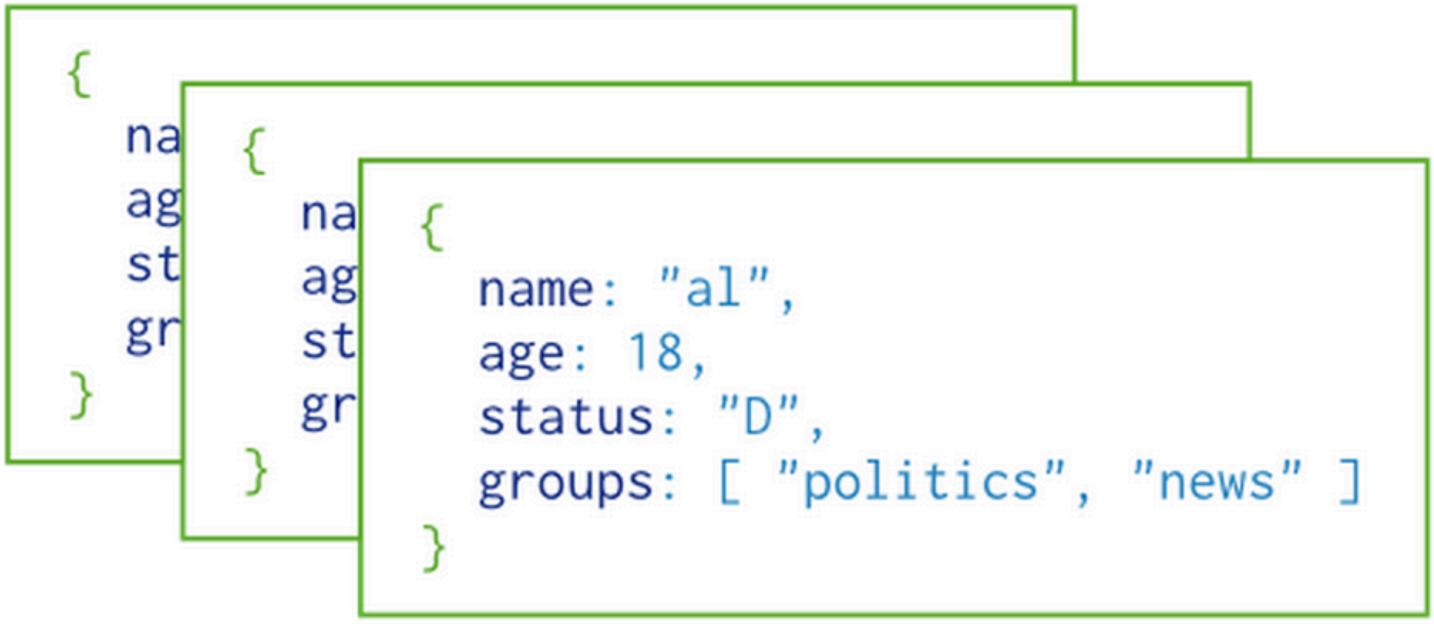
- Document corresponds to a tuple in a relation

  (source: MongoDB tutorial)

- Documents are simply JSON objects in JavaScript syntax, consisting of field:value pairs.
- Documents can be hierarchical – a value can be a JSON object or a list.

Collections

- A collection corresponds to a table in relational databases. It is a set of documents (common structure among documents in the same collection is **NOT** enforced!)



Key commands

- db.collectionname.insert(json_object)
- db.collectionname.find(predicate)
- db.collectionname.aggregate(operators)

- The best way to learn is to simply follow the tutorial. Install Community Edition as in the tutorial.
  - http://docs.mongodb.org/manual/tutorial/getting-started/

Example

After installing the community edition:

- Import sample.json from Canvas -> Files into your local instance of mongo.
- % mongoimport --collection users --file sample.json --jsonArray
- To pretty-print the json file, look for online pretty-printer on Google.

More generally, to import into a mongo DB at a remote server with userid and password:

% mongoimport <dbname> --host <hostname> -u <userid> -p <password> --collection <collectionname> -- file <filename> --jsonArray

Example queries

- Type “mongo” to connect to local mongo.
- Or: mongo <dbname> --host eecs484.eecs.umich.edu -u <userid> -p <password>
- “SELECT * from users”: Mongo equivalent:

  > db.users.find();

  - It returns a cursor object and prints10 tuples at a time.
  - Type “it” to see additional tuples.
- Using Javascript variables:

  > var x = db.users.find();

Iterate over a cursor

```
var mycursor  = db.users.find();
while (mycursor.hasNext()) {
    var w = mycursor.next(); // next document
    print(w.user_id, w.DOB); // print fields
}
// mycursor now points to the end.
```

Selections: Predicates in Find

Find users born on 21st Nov.

- var mycursor = db.users.find({"DOB" : 21, "MOB" : 11});

Find users born on 21st Nov. in state ”Rohan”:

> var mycursor = db.users.find({"DOB" : 21, "MOB" : 11, "hometown.state" : "Rohan"});

Note that the structure of the document is:

{user_id: …,

DOB: … ,

MOB : ... ,

hometown : {city : ... state : ..., country : ...},

...

}

find

- See the Getting Started guide for:
  - Predicates with greater than, less than, etc.
- And conditions
- Or conditions
- Sorting (equivalent to ORDER BY in SQL)

Projections

- Find can also include projections.
- Find first_name and last_name of users born in state Rohan on Nov. 21st:

```
var mycursor = db.users.find({"DOB" : 21, "MOB" : 11, "hometown.state" : "Rohan"},
{first_name : 1, last_name : 1});
```

- > mycursor
- { "_id" : ObjectId("5664e69d270b10887550707d"), "first_name" : "Isabel", "last_name" : "THOMAS" }
- { "_id" : ObjectId("5664e69d270b1088755071d0"), "first_name" : "Gimli", "last_name" : "ANDERSON" }
- { "_id" : ObjectId("5664f005270b108875507398"), "first_name" : "Isabel", "last_name" : "THOMAS" }
- { "_id" : ObjectId("5664f005270b1088755074ea"), "first_name" : "Gimli", "last_name" : "ANDERSON" }
- >

Projections

- _id is a special value, which serves as a key.
- Mongo automatically creates an _id value for inserted values in a collection. Dropping it in a projection requires an explicit projection to 0 for _id:

> var mycursor = db.users.find({"DOB" : 21, "MOB" : 11, "hometown.state" : "Rohan"}, {first_name : 1, last_name : 1, **_id : 0**});

> mycursor

{ "first_name" : "Isabel", "last_name" : "THOMAS" }

{ "first_name" : "Gimli", "last_name" : "ANDERSON" }

{ "first_name" : "Isabel", "last_name" : "THOMAS" }

{ ”first_name" : "Gimli", "last_name" : "ANDERSON" }

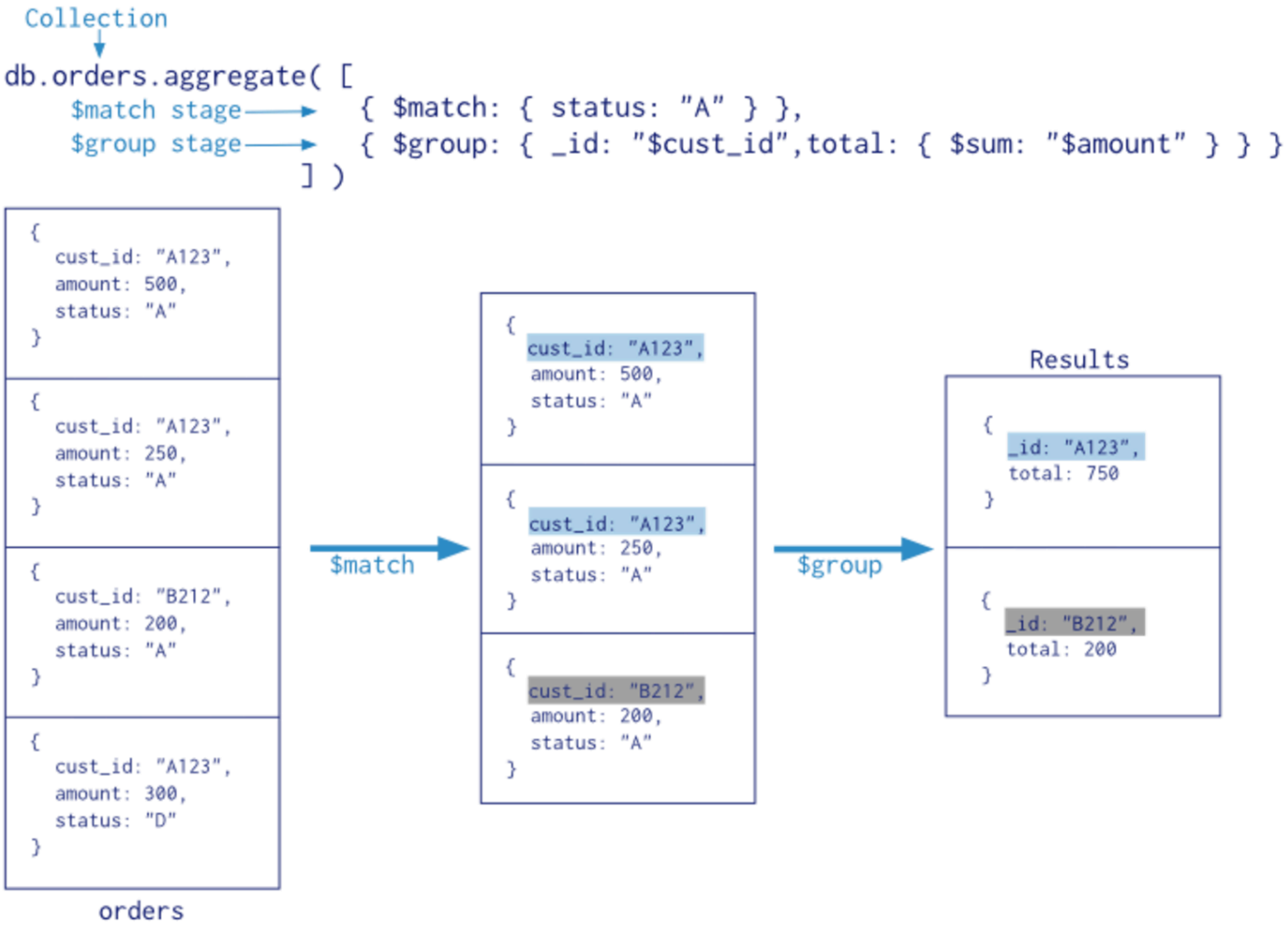
Counting

- Counting tuples: apply count() to results from find()

```
> var mycursor = db.users.find({"DOB" : 21, "MOB" : 11, "hometown.state" : "Rohan"}, {first_name : 1, last_name : 1});
> mycursor.count()
4
```

- The find command returned 4 documents.

Aggregations -- Pipeline



Other Aggregate stages

- $group: similar to GROUP BY
- $sort : for sorting data
- $unwind <arrayfield>: flattens arrays. See docs.
- $out <out_collection>: to put the result into a output collection.

See documentation for other aggregate stages

Trying it out

- Go to
  - http://docs.mongodb.org/manual/tutorial/getting-started/

If you have installed mongodb, run “mongo”. Else, click on “Try it out” at the above URL and follow the tutorial along.

Joins

- Joins are not available in current NoSQL databases for “scalability” reasons
  - Expensive to do when a table is split across multiple nodes
- What should a developer do?
  - Denormalize the database so that all the data is in one collection
    - Problem: updates can be difficult
  - Or, write procedural code to simulate the join across multiple collections

So, why mongodb?

- **Lower licensing fees (especially compared to commercial relational databases)**
- **For huge datasets. Sharding is well supported to allow horizontal scaling**
- Richer document model may simplify programming
- Less of a need for a DBA perhaps (??)

Why not mongodb (or NoSQL in general)?

- **Poor transactional semantics**
- **Procedural language**
- **No language standards**
- Traditional databases are starting to introduce sharding:
  - See postgres_fdw extension in 9.3 version
- Traditional databases are catching up on performance, e.g., see following:
  - //blogs.enterprisedb.com/2014/09/24/postgres-outperforms-mongodb-and-ushers-in-new-developer-reality/

Overall…

- One perspective: Why you should never use a NoSQL database (e.g., mongoDB):

http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/

- Another perspective: Mongo enables applications that are difficult to achieve on traditional DBs

http://docs.mongodb.org/ecosystem/use-cases/