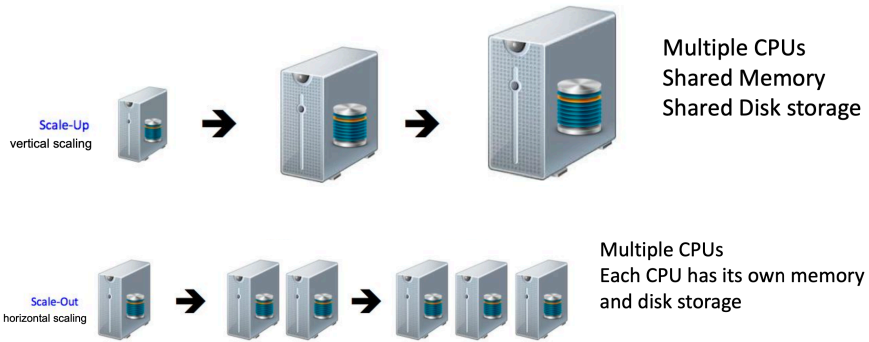
Distributed Databases with NoSQL

CST 363

Agenda

- Distributed Databases
  - Server scale out
  - Partitioned (also called “sharded”) tables
  - Replicated tables
- NoSQL databases and Key-Value storage
- MongoDB architecture
- JavaScript basics

Server Scaling Economics



- With current technology, scale-out is more economical provided workload can be divided into parts that can run in parallel, and the workload can be distributed across multiple computers.
- Shared memory and disk architecture have scaling limitations on the number of CPUs.

Cloud Data Center

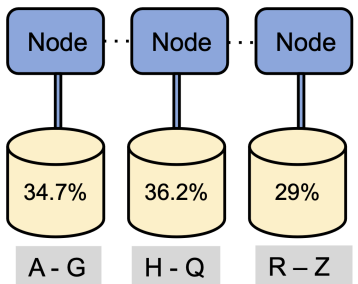
Sharding

- Sharding: partition data across multiple databases
- Partitioning usually done on some partitioning attributes (also known as partitioning keys or shard keys e.g. user ID
  - E.g., records with key values from 1 to 100,000 on database 1, records with key values from 100,001 to 200,000 on database 2, etc.
- Application must track which records are on which database and send queries/updates to that database

Sharding – by key range, by hash

Key Range

last name

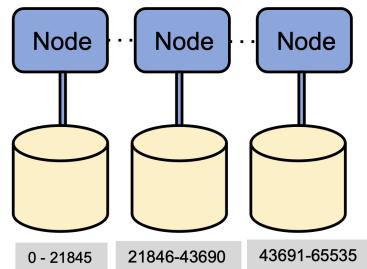



```
name LIKE 'M%'
```

- search only node 2 - distribution may be uneven

Hash

last name → 2 byte integer




```
name LIKE 'M%'
```

- search all nodes - more even distribution

Sharding – by key range, by hash (cont.)

Keeping the partitions balanced in size and workload is a major problem

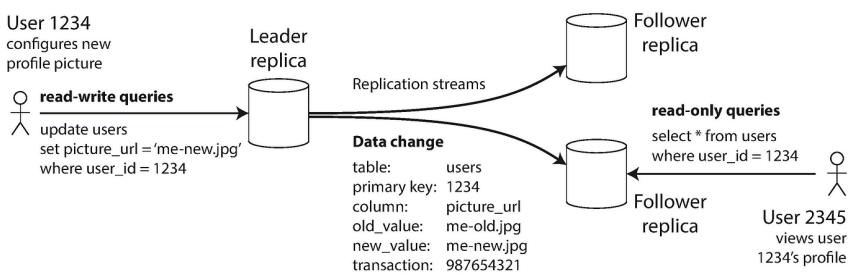
- MongoDB uses “chunks”. Larger chunks are split, small chunks are merged.
- Chunks can be moved from one server to another for workload balancing.
- Splitting, merging and moving can be done with the database online.

Sharding Pros and Cons

- Positives: scales well, easy to implement for a single large table.
- Drawbacks:
  - Not transparent: application has to deal with routing of queries
  - Queries over non-key columns must be done on all servers and the results merged together. (How to do aggregations, joins, sub-selects?)
  - When a database is overloaded, moving part of its load to another server is not easy
  - Chance of failure more with more servers
    - need to keep replicas to ensure availability, which is more work for application

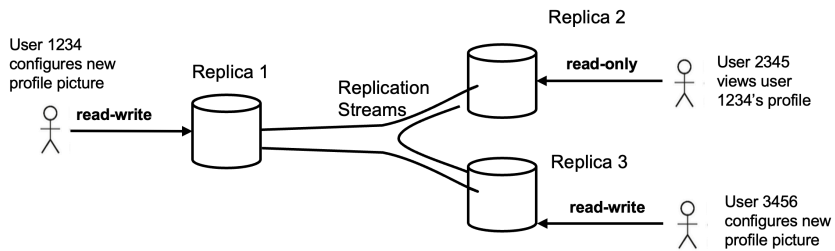
Leader-Follower replication

1. One replica is designated the leader
   - all write requests are sent to leader; leader updates its local store
2. Other replicas are followers
   - leader sends data change to followers
   - followers update their local copy, and acknowledge the leader
3. Read requests can be sent to any replica
   - For consistency, send read requests to leader
   - Reading from a follower may result in stale data but distributes the read workload



Peer replication

- Any replica can receive write request.
- Client sends write request to nearest replica.
- Replica updates its database and sends data change to all other replicas



Replication Issues

Issue	Leader-Follower	Peer
Consistency • Does a read get the most current data? • Does a process see its own writes?	Read from follower may get stale data. For consistency, read from master.	Read may get stale data.
Consistency • When is a write request considered complete?	After master is updated. After master and 1 follower? Or after all followers? Master could crash before sending updates to followers.	Same.
Consistency • Eventual consistency	Eventually, all followers will be in sync with the master. Eventual consistency.	Eventual consistency, but simultaneous write requests to different replicas cause conflict.
Availability	If master goes down, elect a new master that is the most up to date.	
Performance	Write load on master may be too high.	Write load can be more evenly balanced.

Motivation for NoSQL databases

- Very large volumes of data being collected
  - Driven by growth of web, social media, and more recently internet-of-things
  - Web logs were an early source of data
  - Analytics on web logs has great value for advertisements, web site structuring, what posts to show to a user, etc
- Big Data: differentiated from data handled by earlier generation databases
  - Volume: much larger amounts of data stored
  - Velocity: much higher rates of insertions
  - Variety: many types of data, beyond relational data

Motivation for NoSQL databases (cont.)

- Transaction processing systems that need very high scalability
  - Many applications willing to sacrifice ACID properties and other database features, if they can get very high scalability
- Query processing systems that
  - Need very high scalability, and
  - Need to support non-relation data

Schema flexibility

- Relational schema requires all rows to have same schema
- Ability to specify schema at insert time

Key Value Storage Systems

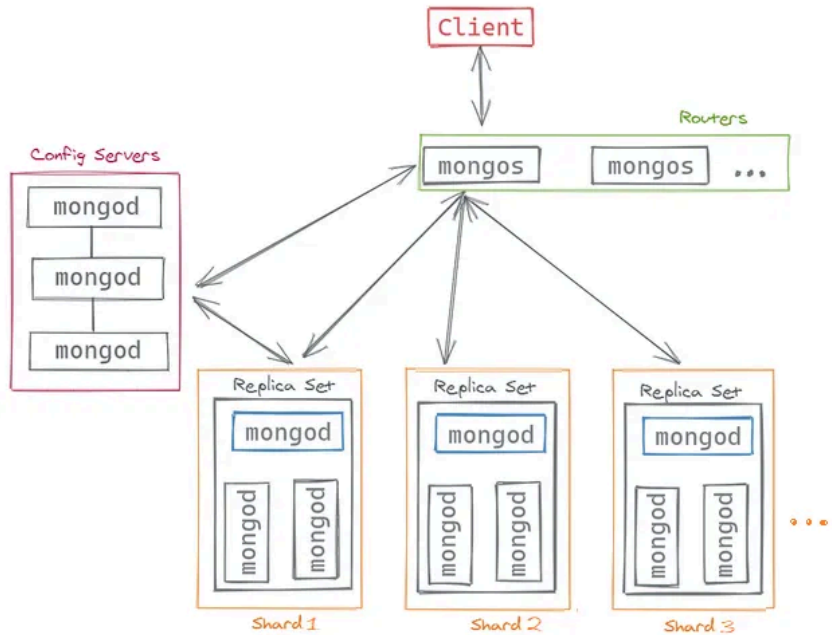
- Key-value storage systems store large numbers (billions or even more) of small (KB-MB) sized records
- Records are partitioned across multiple machines and
- Queries are routed by the system to appropriate machine
- Records are also replicated across multiple machines, to ensure availability even if a machine fails
  - Key-value stores ensure that updates are applied to all replicas, to ensure that their values are consistent
- Document stores store semi-structured data, typically JSON
  - MongoDB

MongoDB

- Document model
  - JSON format
  - No schema that is checked on a write.
  - All documents have an `_id` field. Max document size 16 MB.
- Documents are organized into collections
- Collections are organized into databases
- Mongo interactive shell and scripts use JavaScript

MongoDB Architecture

Diagram shows example of 3 shards, each with 3 replicas for reliability



Architecture Details

- Clients connect to a mongos router which does routing of the request for the client
- A request that involves a single _id key is routed to the node that stored that document by the mongos router
- Config server is critical
  - Given `_id` value, config server knows which shard stores the document
  - 3 replicas of config server for reliability
- A request that does not specify the _id key, is broadcast to all nodes and the results are consolidated by the router
- MongoDB uses **leader-follower replication**
  - Default is reads go to leader
  - Can be changed using “read concern“
  - A “write concern” can be used to wait for the write to be done by replicas

JSON

JSON stands for JavaScript Object Notation

- A document (aka object) consists of attribute names and values.
- The value can be simple, a list, another document, or a list of documents.

JavaScript shell programming

```
$ mongosh
> let doc = {a : 3.14, b : true,  c : "John",  d : null,
             e : { f : 72,  g : "Hello"  },
             h : [  4, 9, -2,  3  ],
             w : [ { city: "London",  temp: 50.6  },
                   { city: "Denver",  temp: 28    },
                   { city: "Chicago", temp: 62.1 }  ]  }
> doc.c                          // John
> doc.e.g                        // Hello
> doc.w[1].temp                  // 28
> doc.w[2].temp = 74.0           // assignment
> doc.w[2]                       // {city: "Chicago", temp:74.0}
> doc.x = [45, 42, 35, -12, 7];  // adds a new multi-valued attribute “x” to doc
```

3 different ways to print a list of values

```
doc = {a : 3.14, b : true,  c : "John",  d : null,
          e : { f : 72,  g : "Hello"  },
          h : [  4, 9, -2,  3 ],
          w : [ { city: "London",  temp: 50.6  },
                { city: "Denver",  temp: 28    },
                { city: "Chicago", temp: 62.1 }  ]  }
```

```
for (let k=0; k<doc.h.length; k++) {
    print(doc.h[k]);     // prints 4, 9, -2, 3
}
```

```
for (let value of doc.h) {
    print(value);    // prints 4, 9, -2, 3
}
```

```
doc.h.forEach( x ⇒ print(x) ); // prints 4, 9, -2, 3
```

Printing all attribute names and values in a document

```
doc = {a : 3.14, b : true,  c : "John",  d : null,
        e : { f : 72,  g : "Hello"  },
        h : [  4, 9, -2,  3  ],
        w : [ { city: "London",  temp: 50.6  },
              { city: "Denver",  temp: 28    },
              { city: "Chicago", temp: 62.1 }  ]  }

for (let x in doc) {     // print name of each attribute
  print(x);              // a  b  c  d  e  h  w
  print(doc[x]);         // print attribute value
}

Object.keys(doc).forEach(x ⇒ { print(x); print(doc[x];})   // same result
```

Recap – Partitioning

- Partitioning
  - The database of documents is partitioned (sharded) across nodes but a single document is stored on one node config server keeps list of nodes with key ranges for each node
  - A rebalancer moves documents between nodes to keep the system balanced

Recap – Replication

- Replication
  - replication uses master – follower
    - by default, all reads/writes go to master. A program will always see its own modifications.
    - a write generally does not wait for all followers to be updated.
    - an insert request can specify a “write concern” that will wait until propagation occurs to some number of replicas in the cluster
    - `db.collection.insert( document, {w:2});`
  - a read request can contain a “read concern” which specifies whether data returned must be current or can come from a follower node that is not in sync with master node.
    - `db.collection.find().readConcern(“local”);`
- if leader fails, a quorum of followers elects a new leader.

Appendix – Mongo statements and commands

- `mongosh` – mongo interactive shell
- `exit` – terminate shell
- `show collections`
- `use <database>` switches to a database
- `db` returns name of current database
- `find`, `findOne` – retrieve operations
- `replaceOne`, `deleteOne`, `insertOne`, `insertMany` – document operations
- `drop` – removes the collection