SQL, NoSQL, MongoDB

CSE-291 (Cloud Computing) Fall 2016

Gregory Kesden

“SQL” Databases

- Really better called “Relational Databases”
- Key construct is the “Relation”, a.k.a. the table
  - Rows represent records
  - Columns represent attribute sets
- Find things within tables by brute force or indexes, e.g. B-Trees or Hash Tables
- Cross-reference tables via shared keys, basically an optimized cross-product, known as a “join”
  - Expensive operation

“SQL” Databases

- Backbone of modern apps
- Very, very high throughput can be achieved
- Scaling is challenging because there is no good way to partition tables while still achieving semantics
- Amazing work-arounds are possible – virtualize SANS to large storage devices, etc
- But, the model is what it is.

NoSQL Databases

- Any of the more modern databases that essentially give up the ability to do joins in order to be able to avoid huge monolith tables and scale
  - Key-Value (Dynamo or basic Cassandra)
  - Column-based (Hbase)
  - Document-based (MongoDB)
- Usually has more flexible scheme (no rigid tables means no rigid NxM structure)

MongoDB

- Document-based NoSQL database
  - Max 16MB per document
- *Documents* are rich BSON (Binary JSON) key-value documents
- *Collections* hold documents and can share indexes
  - Some like to suggest they are analogous to tables, but not all documents in a collection must have the same structure.
  - They just have some of the same keys
- Databases hold collections hold documents

MongoDB Document

- Note field:value tuples

https://docs.mongodb.com/manual/_images/crud-annotated-document.png

Embedded Documents

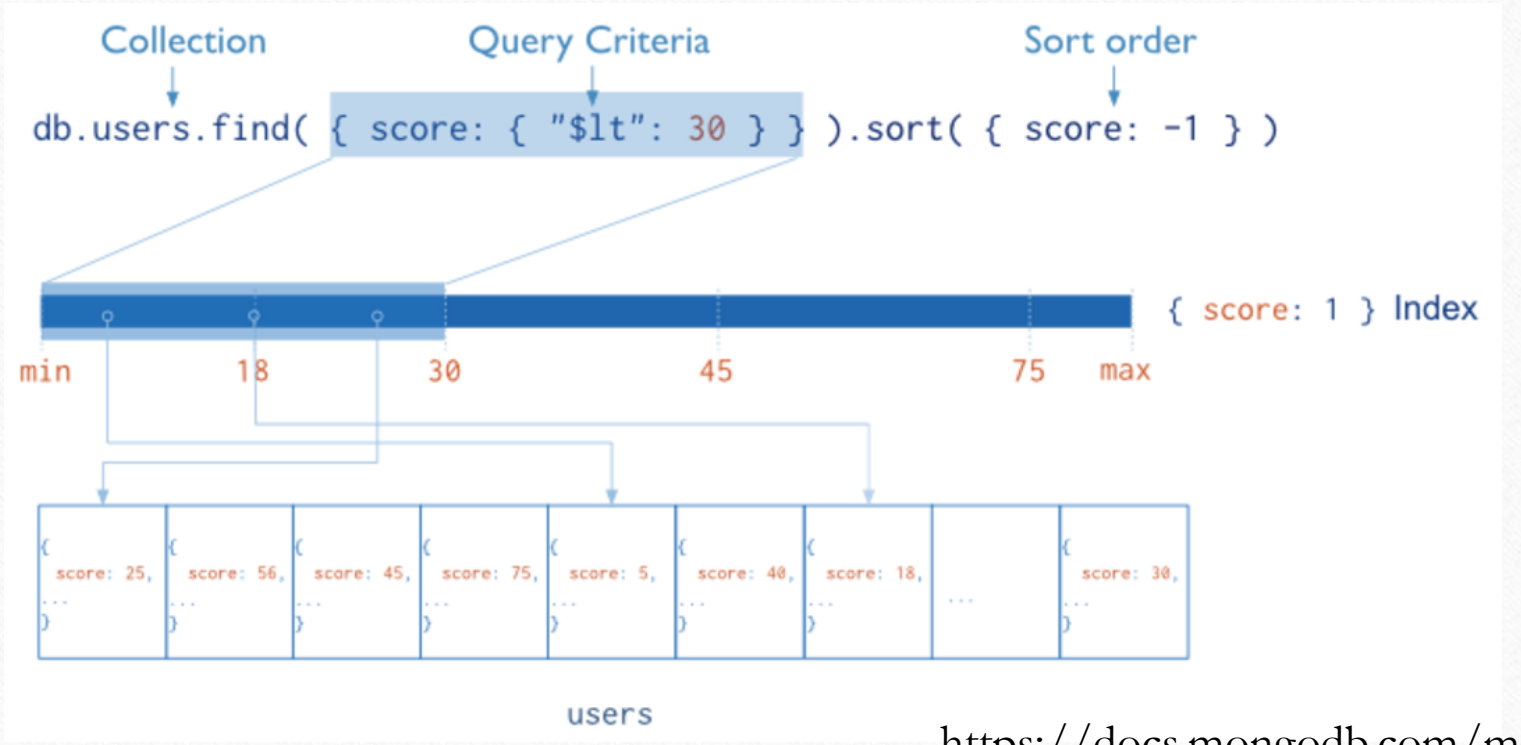
```
var mydoc = {
  _id: ObjectId(“1abcd45b123456754321abcd"),
  name: { first: “Gregory", last: “Kesden" },
  classes: [ “CSE-291", “CSE-110", “CSE-500" ],
  contact: { phone: { type: "cell", number: “412-818-7813" } },
}
```

Array access: classes.0
Embedded doc access: contact.phone.number

Join Operations?

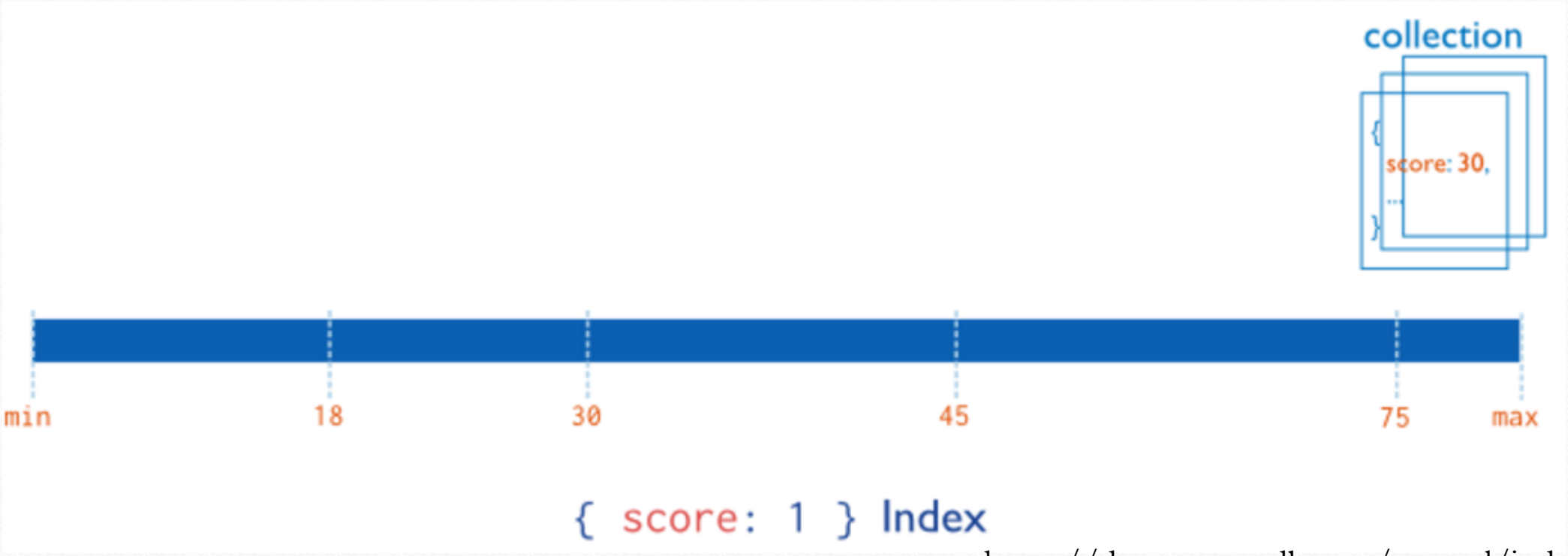
- In general, not a MongoDB thing
  - Get data from different places
  - Slow and expensive operation
- Much better to take advantage of denormalized structure to embed related things
- Can also “chase pointers” by chasing an id from one document into another document via another query. (More like using a foreigh key in SQL than a join)
- Worst case? Multiple passes using shared key.

Indexes (Much like any other DB)



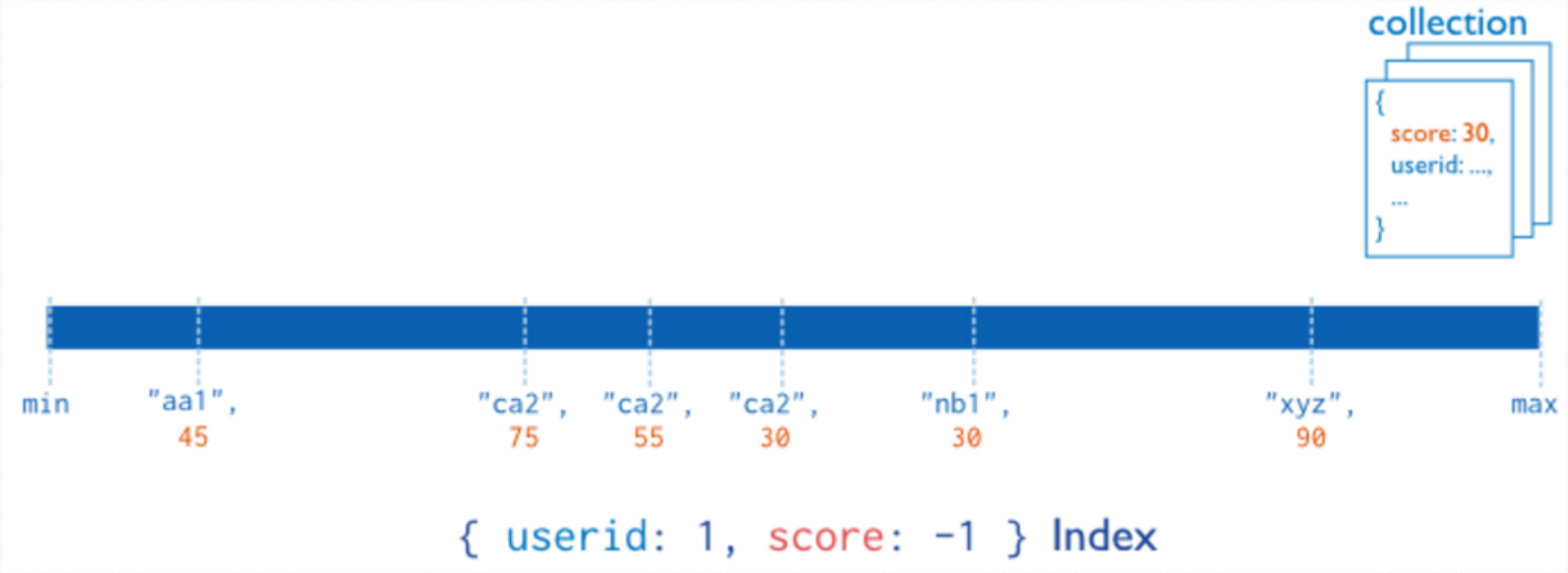
https://docs.mongodb.com/manual/indexes/

Single Field Indexes



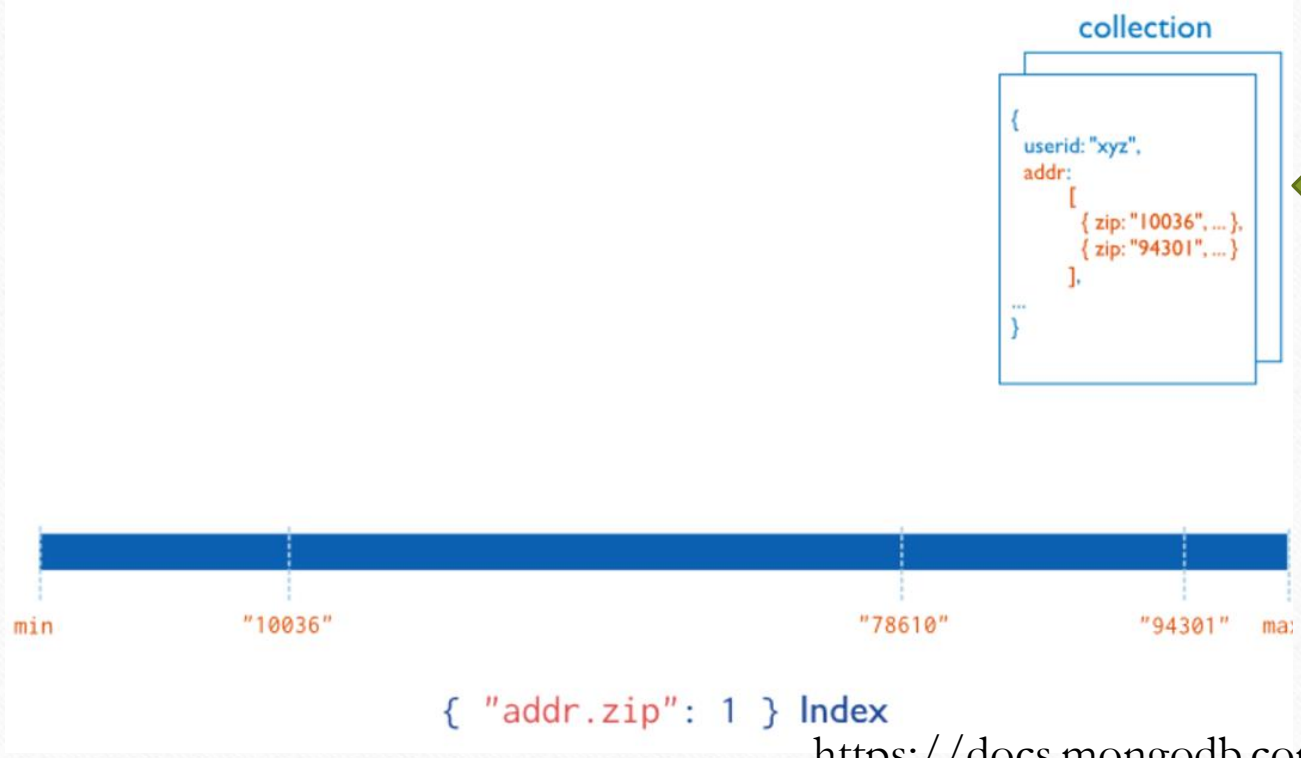
https://docs.mongodb.com/manual/indexes/

Compound Indexes



https://docs.mongodb.com/manual/indexes/

Multi-Key (Array Field) Indexes



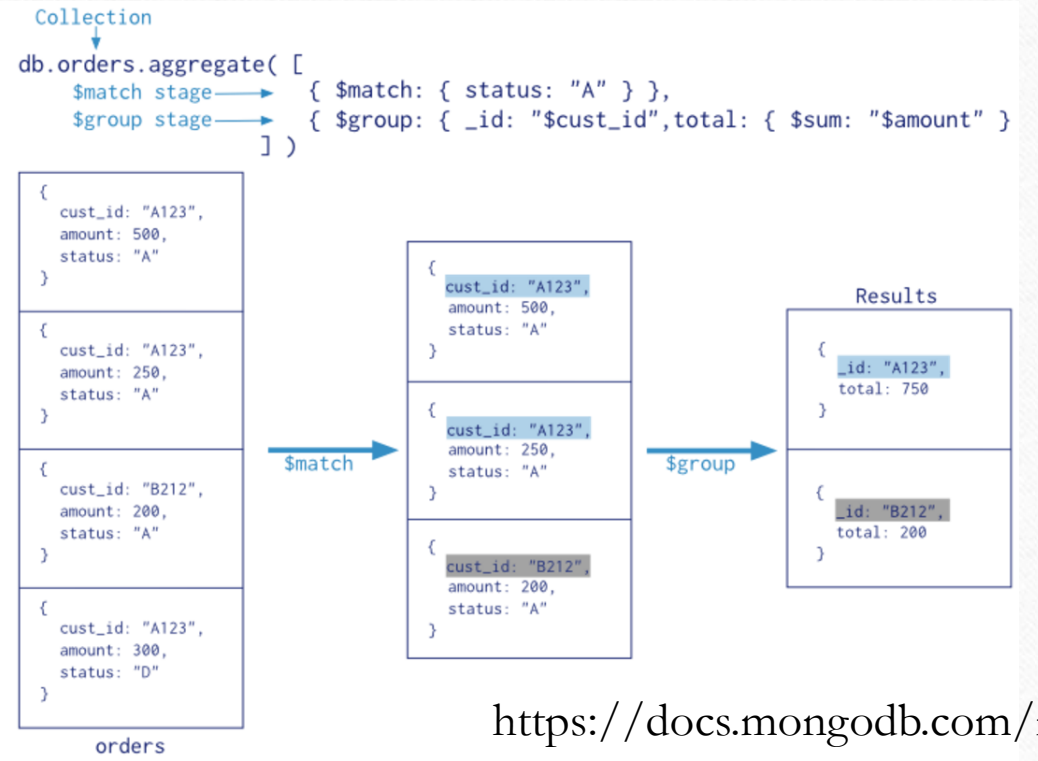
Note: One index for each element of the array

https://docs.mongodb.com/manual/core/index-multikey/

More About Indexing

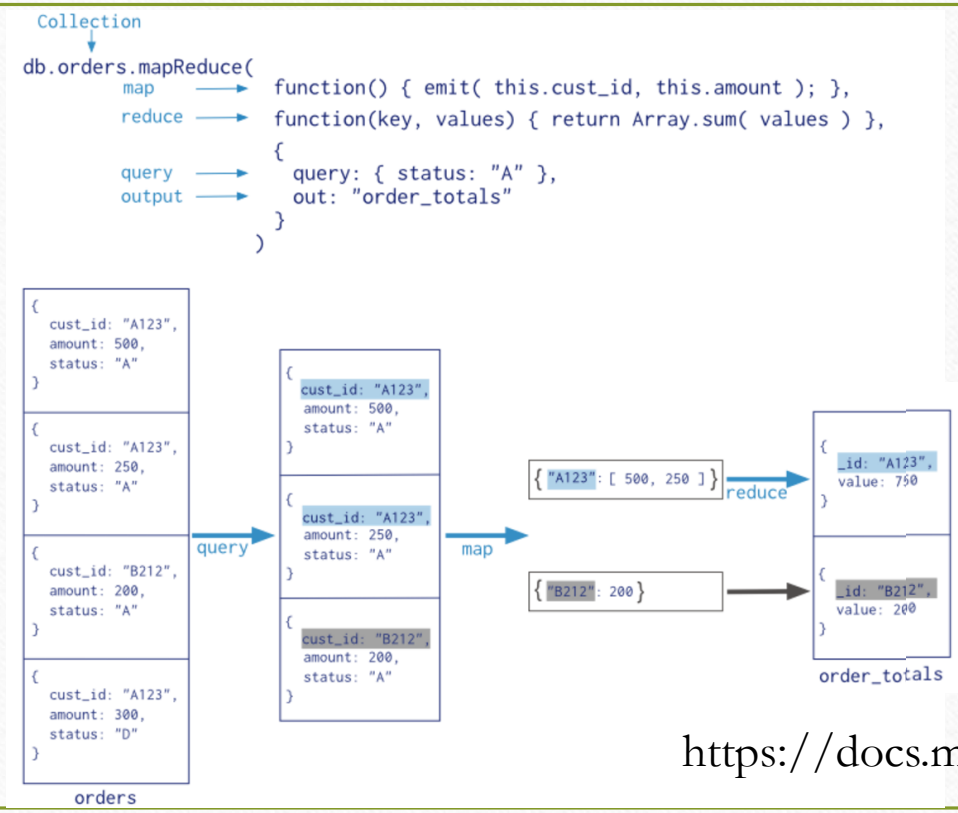
- Matches, Range-based results, etc
- Geospatial searches
- Text searches, language based, includes only meaningful words
- Partial indexes filter and only index matching documents
- TTL indexes, internally used to age out documents, where desired
- Covered queries are queries that can be answered directly from indexes, without scanning
- Intersection of indexes.

Aggregation Pipeline: Filter, Group, Sort, Ops (Average, Concatenation, etc)



https://docs.mongodb.com/manual/aggregation/

Map-Reduce



https://docs.mongodb.com/manual/aggregation/

Concurrency

- Multiple options, WiredTiger the default
- D*ocument-level* concurrency control for write operations. As a result, multiple clients can modify different documents of a collection at the same time.
- For most read and write operations, WiredTiger uses optimistic concurrency control. WiredTiger uses only intent locks at the global, database and collection levels. When the storage engine detects conflicts between two operations, one will incur a write conflict causing MongoDB to transparently retry that operation.
- Some global operations, typically short lived operations involving multiple databases, still require a global "instance-wide" lock. Some other operations, such as dropping a collection, still require an exclusive database lock.

https://docs.mongodb.com/manual/core/wiredtiger/

Snapshots and Checkpoints

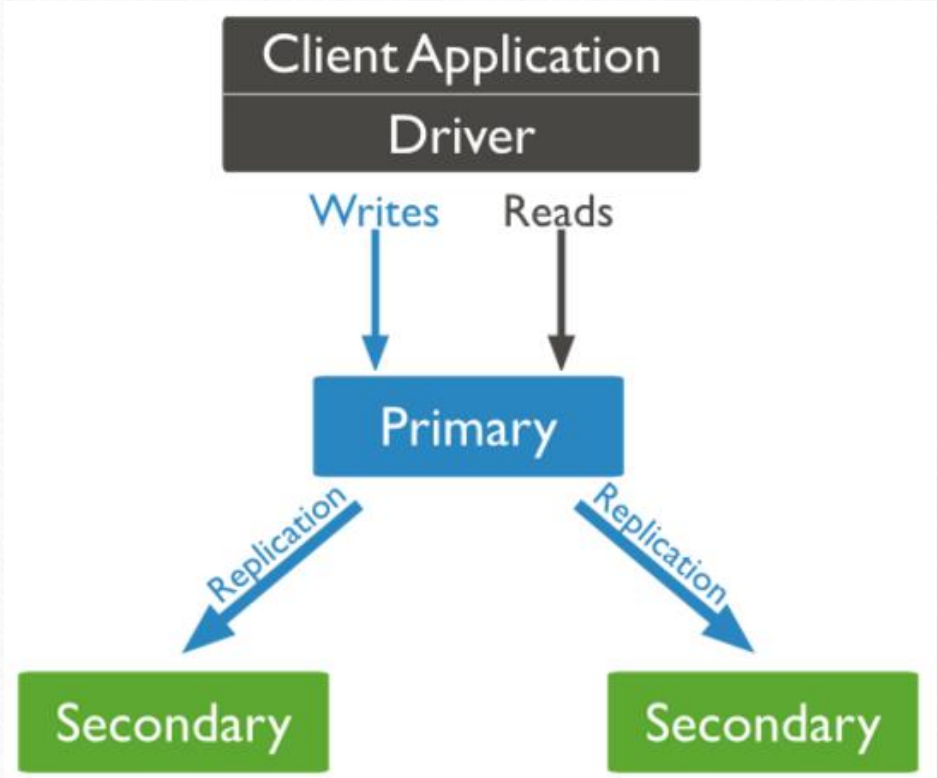
- At the start of an operation, WiredTiger provides a point-in-time snapshot of the data to the transaction. A snapshot presents a consistent view of the in-memory data.
- When writing to disk, WiredTiger writes all the data in a snapshot to disk in a consistent way across all data files. The now-durable data act as a *checkpoint* in the data files. The *checkpoint* ensures that the data files are consistent up to and including the last checkpoint; i.e. checkpoints can act as recovery points.
- MongoDB configures WiredTiger to create checkpoints at intervals of 60 seconds or 2 gigabytes of journal data.
- During the write of a new checkpoint, the previous checkpoint is still valid.
- The new checkpoint becomes accessible and permanent when WiredTiger's metadata table is atomically updated to reference the new checkpoint. Once the new checkpoint is accessible, WiredTiger frees pages from the old checkpoints.
- Journaling needed to recover changes ahead of checkpoints

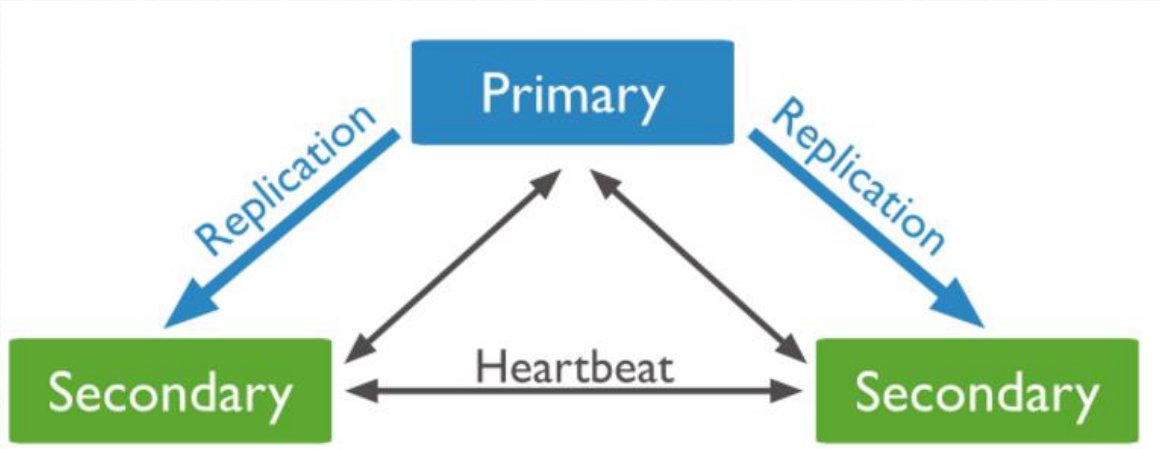
https://docs.mongodb.com/manual/core/wiredtiger/

Journaling

- Compressed write-ahead log (WAL)
- Used to recover state more recent than most recent checkpoint
- Buffered in memory, synced every 50ms
- Deleted upon clean shutdown
- Depending on file system, can preallocate log to avoid slow allocation

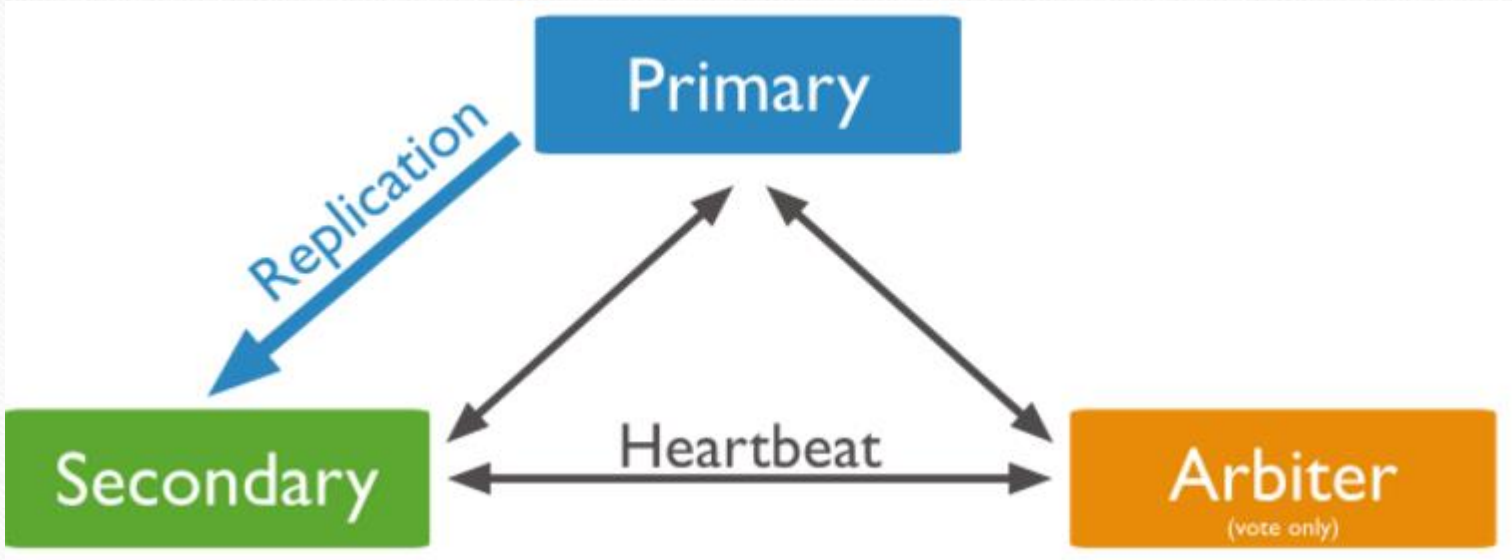
Replica Sets: Asynchronous Replication





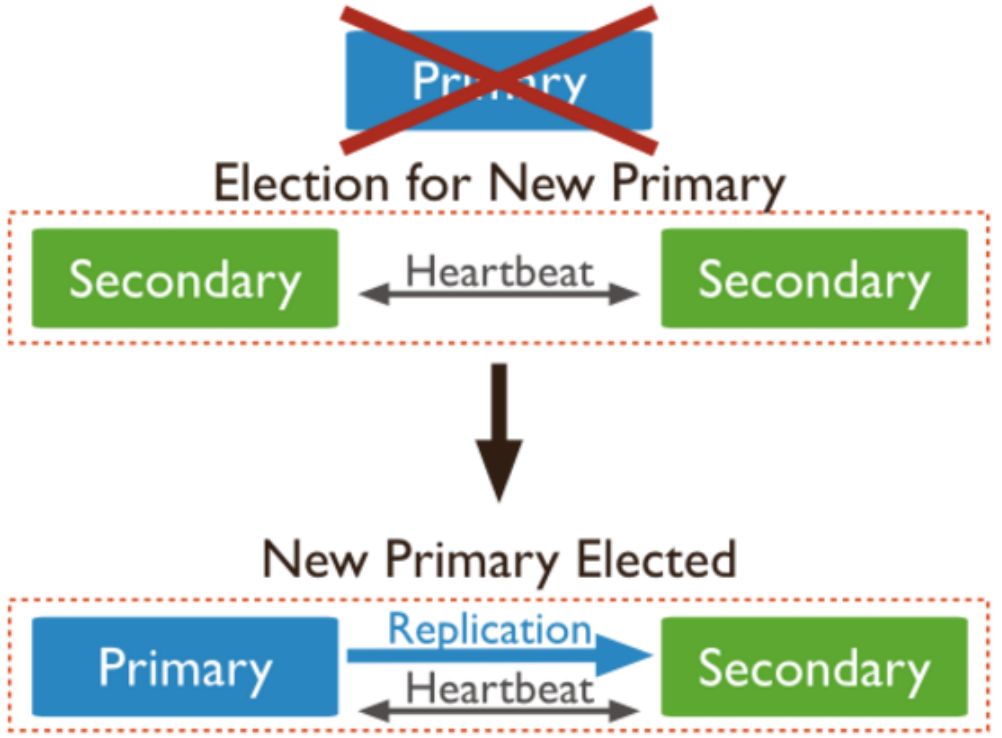
https://docs.mongodb.com/manual/replication/

Arbiters for Quorums: Real World Student-Like Move



https://docs.mongodb.com/manual/replication/

Automatic Failover



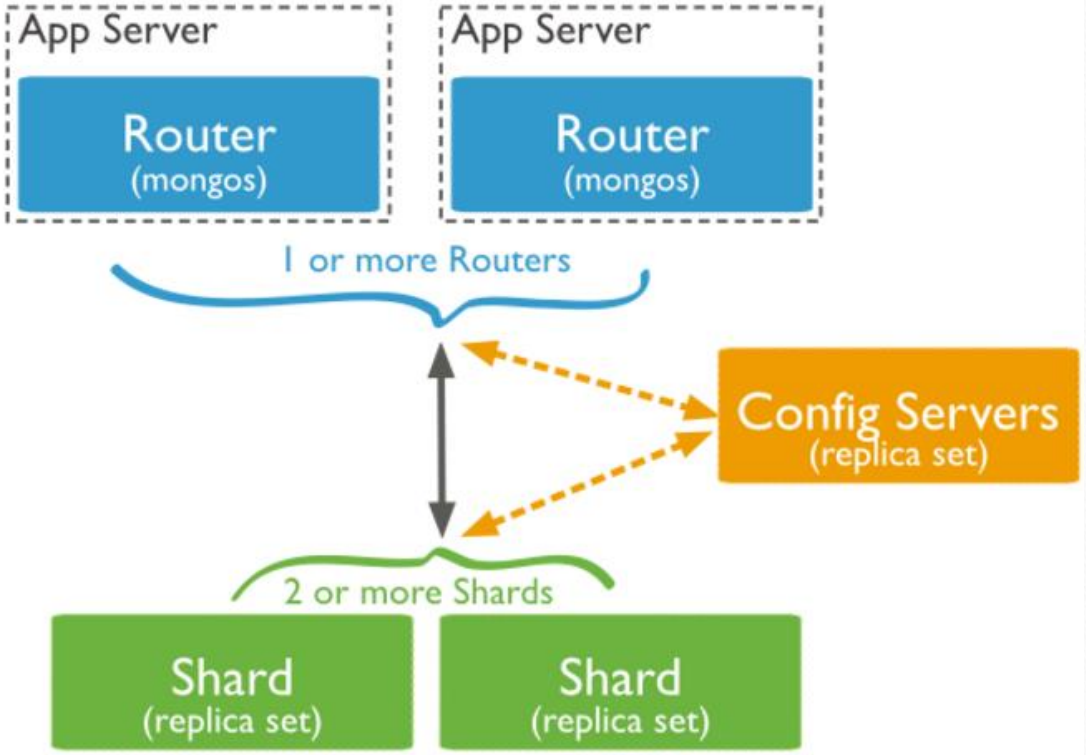
- Missing heartbeats for 10sec? Call election
- Secondary with most votes becomes new primary, temporarily
- But, uses bully-like primary to agree on top dog in the end
- Can be non-voting secondaries. Can be read, but not elected or voting.
- Read-only during election

https://docs.mongodb.com/manual/replication/

Supporting Scale

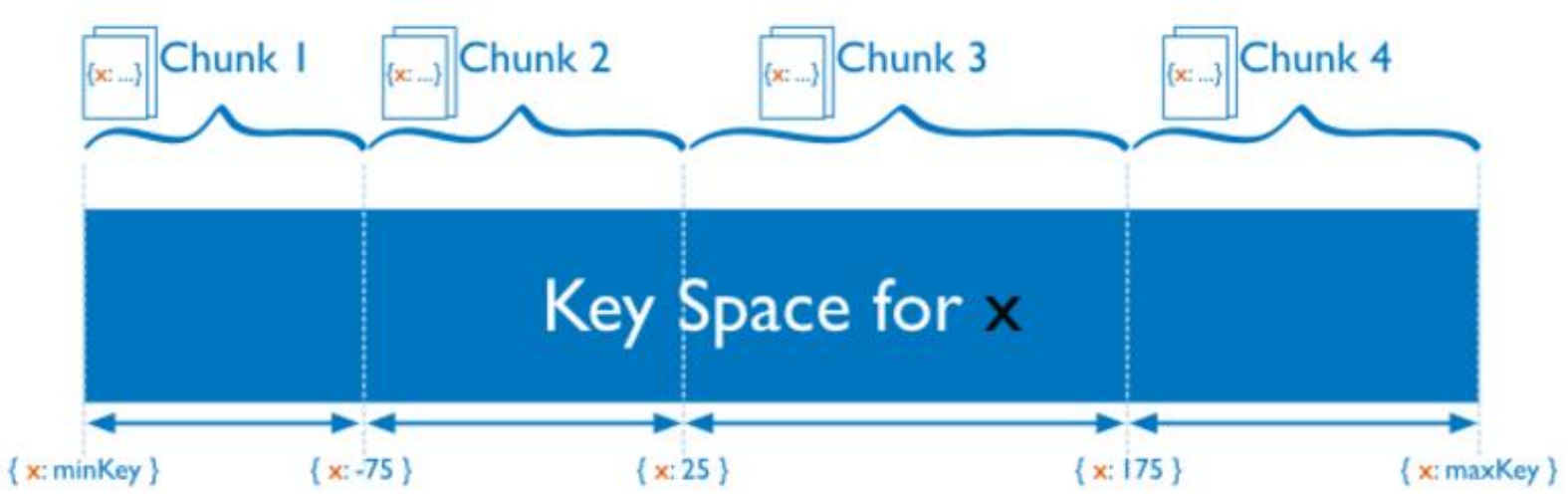
- Vertical – bigger host
- Horizontal -- Sharding
  - More hosts
  - Higher throughput
  - Greater capacity

Sharding



- Documents w/in sharded collection have shard key
  - Immutable, sued for sharding
  - Choice is very important, because key must be found in range by index. Can be bottleneck
- Collection partitioned by shard key range into chunks
- Chunks are distributed and replicated (replica sets)

Chunks



- Sharded into chunks by shard key
- Can be migrated manually or balancer
- Can be split if too large