## CAP Theorem

CAP theorem maintains that any distributed system can obtain only 2 of the desired characteristics:

* Consistency
  + means nodes have same copies of replicated data available
  + every node returns the same, most recent and successful write
  + whenever data is written to one node, it must be instantly replicated to all other nodes for the transaction to be successful
* Availability
  + each read/write will be processed successfully or will receive a message that operation cannot be completed
  + every node must be able to respond in a reasonable amount of time
  + client making request receives the data even if some of the nodes are down
* Partition Tolerance
  + means system can continue to work even if few nodes get separated out from the network or in case of communication breakdowns
  + system can still be consistent in cases of network partitioning and heals when network is recovered
* MongoDB follows CP.
* Resolves network partitions by maintaining consistency and compromising on availablity.

## Documents, Collections and DBs

* Documents
  + single record, like a row in relational DBs
* Collection
  + collection of documents, like a table in relational Dbs
  + Types of collections
    - Capped collections – fixed size, similar to circular buffers
    - Clustered collections – created with clustered index, gives performance improvement
* DB
  + holds one or more collections
* Views
  + read-only object that can be queried
  + contents not persisted on disk, calculated on demand

## Schema Validation

* Lets you create validation rules for your fields
* Validations checked during updates and inserts
* 2 ways to specify validations
  + JSON schema validation
  + Validation with query operators
* 2 types of validation levels
  + strict – default, rules applied to all inserts and updates
  + moderate – rules applied to only existing valid documents
* 2 types of validation actions
  + error – default, inserts or updates that violate the rules are rejected
  + warn – operation succeeds, violation logged in MongoDB log
* For examples see the file attached to class 3

## CRUD Operations

* Create, Read, Update, Delete
* Insert operations – insert one, insert many, bulk writes
* Update – update one, update many, bulk writes
* Query – find one, find many, query nested documents, arrays, projections, return data as array or cursors
* Read Concerns and Write concerns during CRUD operations
* Retryable reads and writes
* For examples see the file attached to class 3

## Aggregation Pipeline

* Document processing pipeline having different stages
* Output for one stage is input to next
* Different optimization techniques avaliable for faster results
* For examples see the file attached to class 4

## GridFS

* Specification for storing files > 16MB which is the limit for a single document in MongoDB
* Each file is divided into chunks
* Defualt chunk size 255Kb
* Uses 2 collections
  + chunks – stores actual file chunks, document format:

{

"\_id" : <ObjectId>,

"files\_id" : <ObjectId>,

"n" : <num>,

"data" : <binary>

}

* + files – stores file’s metadata, document format:

{

"\_id" : <ObjectId>,

"length" : <num>,

"chunkSize" : <num>,

"uploadDate" : <timestamp>,

"md5" : <hash>,

"filename" : <string>,

"contentType" : <string>,

"aliases" : <string array>,

"metadata" : <any>,

}

* Indexes are automatically created on both the collections
* GridFS can be used with drivers or mongofiles
* Can be used to access portion of files without loading the whole file

## Indexes

* Special data structures used for faster retrieval of data
* Default \_id is automatically created by Mongo
* Use createIndex command to create indexes on collections
* Types:
  + Single Field
    - Index on 1 field
  + Compound
    - Index on multiple fields
  + Multi-key
    - When field is an array
  + Text Indexes
    - Supports searching for string content
  + Wildcard Indexes
    - Support queries against unknown/arbitrary fields
  + Hashed Indexes
    - Index on hashed value of filed
    - Created to support hashed sharding
  + Geo-spatial Indexes
    - Used for geo spatial data (location data)
    - Use planar and spherical geometry
* Index properties
  + TTL Indexes
    - Special single field indexes
    - Automatically remove documents from a collection after given interval of time or at specific time
  + Unique Indexes
    - Ensures that indexed fileds do not store duplicate values
  + Partial Indexes
    - Only index the documents that meet specific filter criteria
  + Case Insensitive Indexes
    - Supports queries that support string comparisons without considering case
  + Hidden Indexes
    - Not visible to query planner
    - Can be used to check the impact of dropping an index without actually deleting it
  + Sparse Indexes
    - Contains only those documents that have the indexed field (including null)
    - May not contain all the documents in the index
* ESR (Equality, Sort, Range) Rule
  + Helps in creating efficient indexes
  + Try to use this rule while creating indexes
  + Give preference to fields used in equality first, then to sort fileds and then fileds used in range based queries
* Use query explain plans to understand the useage of indexes
* For examples, see the file attached to class 4

# Diagnostics

* Database Profiling
  + captures information about different ops - read, write, cursor, commands
  + writes data in system.profile collection
  + off by default
  + levels
    - 0 - off, no profiling
    - 1 - slow queries or that match the filter
    - 2 - collects data for all ops - performance hit
  + Commands
    - getProfingStatus – to get the current profiling status
    - setProfilingLevel – to change the profiling level
* mongotop
  + tracks amount of time mongod spends reading and writing data
  + per collection level
  + by default - every second
  + Fields returned
    - ns - namespace <database>.<collection>
    - total - total time spend on this namespace
    - read - time spent on read operations
    - write - time spent on write operations
    - timestamp - timestamp for returned data
* mongostats
  + provides quick status overview
  + by default - every 1 second
  + fields returned
    - inserts - number of objects inserted per second in the db
    - query - number of query ops per second
    - update - number of updates per second
    - delete - number of delete ops per second
    - getmore - number of cursor batch ops per second
    - command - number of commands per second. for repl set in second local|replicated
    - flushes - number of checkpoints triggered per interval
    - dirty - % of WiredTiger cache with dirty bytes
    - used - % of WiredTiger cache currently in use
    - vsize - virtual mem used by process in MBs
    - res - amount of resident mem used by process
    - qr - length of queue of clients waiting to read
    - qw - length of queue of clients waiting to write
    - ar - number of active clients performing read ops
    - aw - number of active clients performing write ops
    - netIn - amount of network traffic received in bytes
    - netOut - amount of network traffic sent in bytes
    - conn - total open connections
    - set - name of replica set if applicable
    - repl - replication status of member
* For examples, see file attached to class 10

## Replica Sets

* Group of mongod servers that maintain the same data set
* Provides redundancy and high availability
* Types of nodes
  + Primary - supports reads and writes
  + Secondary - supports only reads
    - Priority - 0
      * Cannot become primary
      * Cannot trigger elections
      * they have data, accept reads and participate in voting
    - Hidden - Invisible to the client applications
      * must have priority 0
      * may vote in elections
      * No read operations
      * Use for backup and reporting
      * have data
    - Delayed - Contains earlier or delayed state
      * run historical snapshot of the data
      * useful for recovery due to human errors
      * must have priority 0
      * must be hidden
      * can vote in elections
    - Non-voting member
      * has data
      * allows reads
      * cannot vote
      * must have priority 0
  + Arbiter - no data
    - participates in elections
    - cannot become primary as does not have data
    - useful when you have even number of replica sets, can add arbiter to just participate in voting
    - should use only 1 arbiter in a set
* For details on how to create each type of node, see the file attached to class 6

## Sharding

* Sharding is an example of horizontal scaling
* Components
  + shard - contains subset of the data. each shard must be a replica set in itself
  + mongos - query router
  + config servers - store metadata and settings for the cluster. must be deployed as replica set
    - metadata like list of chunks and range for chunks
* Data sharded at collection level
* Shard keys - fields used to distribute data. Can be more than 1
* Shard key index - required for sharding. if new collection, its automatically created. If existing collection, create on shard key
* Sharding key strategy
  + Hashed Sharding
    - computing hash for shard key values
    - each chunk is assigned range based on hash
    - consecutive values may not get hashed to same chunk, better data distribution
    - range based queries may not target single shard
  + Range Sharding
    - key values directly divided into chunks
    - can support range queries
* Chunks - sharded data is partitioned into chunks. inclusive lower and exclusive upper range based on shard key
* Balancer - background process that balances chunks across shards for even distribution
* Advantages
  + Read/write workload is managed
  + storage capacity can be increased on the go by adding more shards and resharding the data
  + high availablity - if one shard goes down, partial data can still be accessed
* Once collection is sharded, cannot unsahrd it. Only reshard it.
* Db can have mixture of sharded and unsharded collections. Unsharded collections are stored on primary shard.
  + Primary shard has no relation to primary of replica set
  + While creating DB, shard with least data is picked as primary by mongos
* Each Db has its own primary shard
* Connect to mongos router to connect to sharded cluster. not to mongod
* For commands to create sharded cluster on local machine, see file attached to class 6

MongoDB Cloud

1. mongodb+srv:// connection string
2. Ops Manager
3. Synonym Search and Atlast text search