DBA MongoDB

Copyright ©2017 Mark Anderson

# M102: MongoDB for DBAs

Introduction/Course by Dwight Merriman, chairman and co-founder and Will Cross.

Objective; Operate, Administer, Set Up and maintain production MongoDB clusters.

Topics

* Schema Design
* Performance Tuning and Optimization
  + Set up Indexes and optimize
* MongoDB Administrative Shell
* Standard Tasks;
  + Backups
  + Restoring Backups
  + Security Configurations
* Building Large Scale Clusters and Systems
  + Replication
  + Sharding

# Course

* 7 weeks
  + Videos – 2hrs
  + Quizzes –not graded
  + Homework –graded -50% of grade
  + Final – 50%
    - Pass >=65%

# Chapter 1

* History/Purpose of MongoDB
* Basic Concepts
  + Scaling
  + Document Model
* Administrative Shell

## Concepts

* What it’s for/why
  + Started 5 years ago
    - [SCALE]Change of Hardware
      * World of lots of Parallelism
        + Cores Parallelism
        + Servers Parallelism
      * COMMODITY SERVERS
      * COMMODITY NETWORKS
      * Distributive computing
      * Cloud Computing
    - Scaling up Desire
      * Without big cost, ”Big Data” desire
    - [DEV] App Development
      * Easier and more elegant
    - [COMPLEX Data] Data Complex/unstructured or polymorphic
      * Varies, object by object – easily dealt with.
  + MongoDb to bridge this Gap and provide both speed an quality

### SCALE – Out problems

* Previously server – scale vertically; buy bigger box *(Vertically Scaling*)
* Horizontal scaling –want bigger – add more boxes (separate servers – so talk to each other over network)
* In both cases we want some sort of fail-over and high availability. Probability in horizontal scale higher of going down simply as there are more servers. Another concern is potential for failures occurring over network, nodes dropping.
* In mongo we ‘re thinking of running across LANs and WANs (Local area networks and Wide Area networks)
* Want to build large clusters that work on kind of commodity hardware and commodity servers and commodity networks such as Gigabyte Ethernet.
* Commodity – meaning not proprietary – not mainframes. Might consist of Cisco equipment running GigaBit Ethernet.
* Commodity servers are probably Intel based 64 bit servers and racks –each server may be $5k to $20 per server

Aim to create:

| a.

Scale and speed^ | c 80% features, with ability to scale out

|

| b

-------------------------------

Features-🡪

1. Memcashed data - Key value store, distributed, not persistent
2. Here we have super high feature such as RDBMS
3. Conceptually this is where MongoDB resides

noSQL db’s resides between a and b. so Mongo at the high end of functionality side (b).

We want to keep ad hoc queries, keep secondary indexes, keep some notion of transactions.

Example of clustered *Horizontal scaling servers;*

oooooooooo}

oooooooooo} -> clustered – appears as one machine from client /Client app perspective

ooooooooooo}

* *When scaling out in horizontaly (adding more servers to contain your data, problems arise as you go from 1 to say a few dozen*
* The servers must communicate with one another eating up network bandwidth
* The need for redundancy increases as the likelihood of some failure in the system per unit of time increases
* problem 1 here, return one set of data of machine from his cluster...not very efficient
* Another issue:

#### **Joins**

* Eg complex SQL statement with multiple joins, joining many rows of data to client
* Joins came from different servers in horizontal cluster, from multiple tables, potentially talk to 30 machines
* Could have one fat table then a few dimension tables (like a star schema in a relational warehouse) – dimension tables are small so we can keep a copy on every server – will work and some of the relational data warehouses do this –and hence can make these assumptions
* In this concept the time to send one message from one server to another is pretty significant compared to grabbing something from RAM out of own server, regardless fairly challenging to join

#### **Complex transactions**

* Imagine cluster is a thousand servers – with fairly large transactions – assume they touching 500 rows on 150 servers. Then need to commit that operation, we want isolation, atomicity of all work to be coordinated – together with a rollback at the end, together with concurrency control
* I we want to do distributive transaction and need to touch 2 servers- not too bad as we need a two phase commit – though does not scale up to this hundred server transaction with speed

MongoDB does not have solution to this in a distributive computing environment. Instead create something that does not have *Joins* or *Complex transactions* yet still useful. Implies a different data model. With power and fits with today’s development methodologies.

Imagine a Key Value Store – think of persistent hash map. {k->v…Get(k) Set(k,v) have to run lots of code ourselves.

MongoDb Approach;

* Document-oriented database – JSON JavaScript Object Notation
* Syntax of document in JS
* JSON is language-independent RFC (remote function call) a standard for JSON itself
* Maps nice to most languages in the way they store object-style data
* Still separating the data from the code
* Be able to read doc, without reading the program – we want data of the objects
* As we are not using joins, we can pre-join or not normalize that as the data in the first place alleviating some of the need for joins
* In Mongo a document is your basic unit of storage –equivalent of a record
* A Mongo document is stored as a unit – together not broken apart –normalization style.
* Transactional would be : *select\* from t,t2 where t.p = t\_z.p and p=”Q33*”
* Mongo: *db.parts.find({\_id:”Q33”};*
* Data is pre-joined – *embedding or containment –* think in terms of objects or entities, rather than rows
* In Mongo Queries are represented as JSON; db.parts.find(*{\_id:”Q33”});*
* In Mongo we store things in ***BSON*** (Binary JSON) (internally interesting implications)

* Philosophy
* Rationale

 