Sharing Life's Joy using MongoDB: A Shutterfly Case Study

MongoSV

Dec 2010

Kenny Gorman, Data Architect

Shutterfly Inc.

- Founded in December 1999
- Public company (NASDAQ: SFLY)
- Millions of customers have billions of pictures on Shutterfly
- Photo storage, photo books, sharing, prints, gifts
- Only photo sharing site that offers free, unlimited storage, down-sample, compress, or force delete photos
- > 6B photos

Existing Metadata Storage Architecture

- Metadata is persisted in RDBMS
- Images/media stored outside DB
- Java/Spring, C#,.Net
- Oracle™ RDBMS
- Sun™ servers and storage
- Vertically partitioned by function
- Hot Standbys used for availability
- > 20tb of RDBMS storage
- > 10000 ex/sec
- Extreme uptime requirements

Challenges

- Time to Market
- Cost
- Performance
- Scalability

New Data Persistence Architecture

- Data model matches use cases
- Rapid application development
- Scale out architecture
- Availability architecture
- Data locality
- Simple design
- Low cost

Enter MongoDB

- BSON/JSON data format, schema-less
- Best of RDBMS, yet not quite k,v store
- Data Access Layer: DAL (morphia for Java)
- Replica Sets
- Sharding
- Commercial support
- Active community, good adoption
- Less or no memcached needed?

Project Rollout

- Create persistence ‘platform’ for many projects.
- Phased rollout
- Java, C#
- XML projects good candidates
- Start with smaller projects
- Move to harder projects
- Safe rollout strategy
  - XML to BSON conversion using GridFS
  - Dual writes
- Introduce more features over time
  - Replica Sets
  - Durability
  - Sharding

Data Modeling

shutterfly

- XML w/o MongoDB

```
<?xml version=\"1.0\" encoding=\"utf-16\"?>\r\n
<votes>\r\n  <votes>\r\n
<voteItem user=\"099999999999\" vote=\"2\" />\r\n
<voteItem user=\"000011111111\" vote=\"1\" />\r\n
<voteItem user=\”434343434343\" vote=\"1\" />\r\n
</votes>\r\n</votes>”
```

- MongoDB with XML

```
{ "_id" : "site/the3colbys/3326/_votes",
  "V" : 0,
  "cD" : "Thu Sep 23 2010 20:38:54 GMT-0700 (PDT)",
  "wD" : "Thu Sep 23 2010 20:38:54 GMT-0700 (PDT)",
  "md5" : "71199d82ee730f271feface722a74d30",
  "data" : "<?xml version=\"1.0\" encoding=\"utf-16\"?>\r\n
            <votes>\r\n  <votes>\r\n
            <voteItem user=\"099999999999\" vote=\"2\" />\r\n
            <voteItem user=\”000011111111\" vote=\"1\" />\r\n
            <voteItem user=\”434343434343\" vote=\"1\" />\r\n
            </votes>\r\n</votes>”}
```

- MongoDB/BSON without XML

```
{ "_id" : "site/the3colbys/3326/_votes",
  "V" : 0,
  "cD" : "Thu Sep 23 2010 20:38:54 GMT-0700 (PDT)",
  "wD" : "Thu Sep 23 2010 20:38:54 GMT-0700 (PDT)",
  “votes” : {099999999999:2, 000011111111:1, 434343434343:1 }  }
```

Data Modeling

- Materialized Paths

```
// list all children by root
> db.nodes.find({path:/^1005/})
{ "_id" : ObjectId("4b58e8afdb07afba72000000"), "path" : "1005", "name" : "mystuff" }
{ "_id" : ObjectId("4b58e8afdb07afba72000001"), "path" : "1005.1", "name" : "events", "tags" : [ "outings", "events" ] }

// list all children by parent
> db.nodes.find({path:/^1005.2/})
{ "_id" : ObjectId("4b58e8afdb07afba72000002"), "path" : "1005.2", "name" : "family", "tags" : [ "gormans", "family and friends" ] }
{ "_id" : ObjectId("4b58e8afdb07afba72000003"), "path" : "1005.2.400", "name" : "beach", "tags" : [ "stinson", "beach" ] }
{ "_id" : ObjectId("4b58e8afdb07afba72000005"), "path" : "1005.2.442", "name" : "eala", "tags" : [ "daughter", "family", "baby" ] }

// move a node to a new parent atomically
> db.nodes.update({path:"1005.1.400"},{$set:{ path : "1005.2.400" } })
```

- Atomic Publish

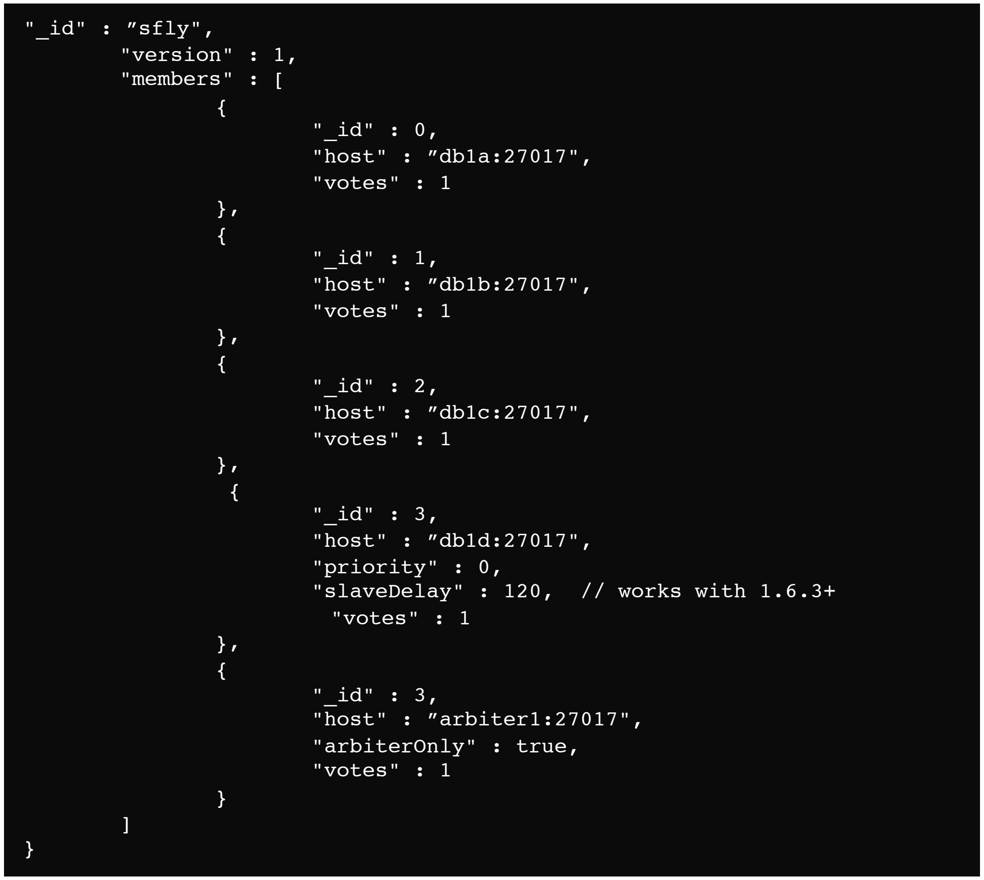
```
// show me all the data less or equal to the version I am at and show me anything that might become deleted
> db.t.find({"xmin":{$lte:0},"ttl":{$in:[0,0]}})
{ "_id" : ObjectId("4c16665e8ba2448137e45230"), "data" : "dogs", "xmin" : 0, "ttl" : [ 0 ] }

// By flipping the element of the ttl to the last version we want to see, it still shows up after we
// make the change until the application increments to the next version.
>db.t.update({"data":"chickens1"},{$set:{"ttl":[2]}},false,true)
{ "_id" : ObjectId("4c1669068ba2448137e45234"), "data" : "dogs", "xmin" : 0, "ttl" : [ 0 ] }
{ "_id" : ObjectId("4c166a1e8ba2448137e45238"), "data" : "chickens1", "xmin" : 2, "ttl" : [ 2 ] }
{ "_id" : ObjectId("4c166a2d8ba2448137e45239"), "data" : "chickens", "xmin" : 3, "ttl" : [ 0 ] }

// the application still doesn't see the update until:
> db.t.find({"xmin":{$lte:3},"ttl":{$in:[0,3]}})
{ "_id" : ObjectId("4c166a2d8ba2448137e45239"), "data" : "chickens", "xmin" : 3, "ttl" : [ 0 ] }
{ "_id" : ObjectId("4c16690e8ba2448137e45235"), "data" : "cats", "xmin" : 0, "ttl" : [ 0 ] }
{ "_id" : ObjectId("4c1669068ba2448137e45234"), "data" : "dogs", "xmin" : 0, "ttl" : [ 0 ] }
```

Typical MongoDB HW Configuration

- MongoDB ‘brick’
  - Single MongoDB instance per host
  - Not fancy
    - Dell R710 dual quad core
    - 48GB memory
    - SATA disk with BBU controller
    - 3TB usable, RAID 10
    - Centos, 2.6 kernel
    - ext3 file system
    - *W*+1 configuration
    - Gigabit Ethernet network interfaces

Replica Set Configuration

- 4 servers per set
- Set has *W*+1=3 active members
- 1 delayed slave
- *W*=2 (durability)
- One failure per set OK
- Odd # of votes per set
- Global arbiter
- Backups from slaves
- Gig-Ethernet between members



```
"_id" : ”sfly",
        "version" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "host" : ”db1a:27017",
                        "votes" : 1
                },
                {
                        "_id" : 1,
                        "host" : ”db1b:27017",
                        "votes" : 1
                },
                {
                        "_id" : 2,
                        "host" : ”db1c:27017",
                        "votes" : 1
                },
                 {
                        "_id" : 3,
                        "host" : ”db1d:27017",
                        "priority" : 0,
                        "slaveDelay" : 120,  // works with 1.6.3+
                         "votes" : 1
                },
                {
                        "_id" : 3,
                        "host" : ”arbiter1:27017",
                        "arbiterOnly" : true,
                        "votes" : 1
                }
        ]
}
```

MongoDB roll-out strategies

- Phase I: Simple use case
  - Primary and 2 replica DB’s, 1 ‘lagged’
  - Manual failover
  - MongoDB 1.4.2 (stable)
- Phase II: More complex use case
  - Replica sets for availability
  - *W*+1 configuration
  - From XML to BSON
  - MongoDB 1.6.3 (stable)
- Phase III: Full throttle
  - Replica sets for durability (*W*=2)
  - Shards for scale out
  - MongoDB ?

Rollout: Data Migration

- Write ‘on touch’
    - MongoDB as a cache
    - Write through all caches, write to cache on read miss.
    - Self populates when users sign in
- Background migration process
    - Batch job
    - Read/Write/Read/Check

So how did we do?

- Time to Market
  - Application developed in 1 sprint
- Cost
  - 500% improvement
- Performance
  - 900% improvement
  - 400ms to 2ms avg latency for inserts
- Scalability
  - Shard on demand

Lessons Learned

- Keep it simple
- Excellent developers make the difference!
- Write Locking/Concurrency
  - Protect your writers
  - Write efficient code with an eye towards design
- Data Modeling
  - Loose approach
  - Best practice patterns
- Walk before you run

Q&A

Questions?

Contact:

kg@kennygorman.com

http://www.kennygorman.com

twitter: @kennygorman

http://www.shutterfly.com

http://technologyblog.shutterfly.com/

kgorman@shutterfly.com