MongoDB Performance Tuning

MongoSF

May 2011

Kenny Gorman, Data Architect

Shutterfly Inc.

shutterfly

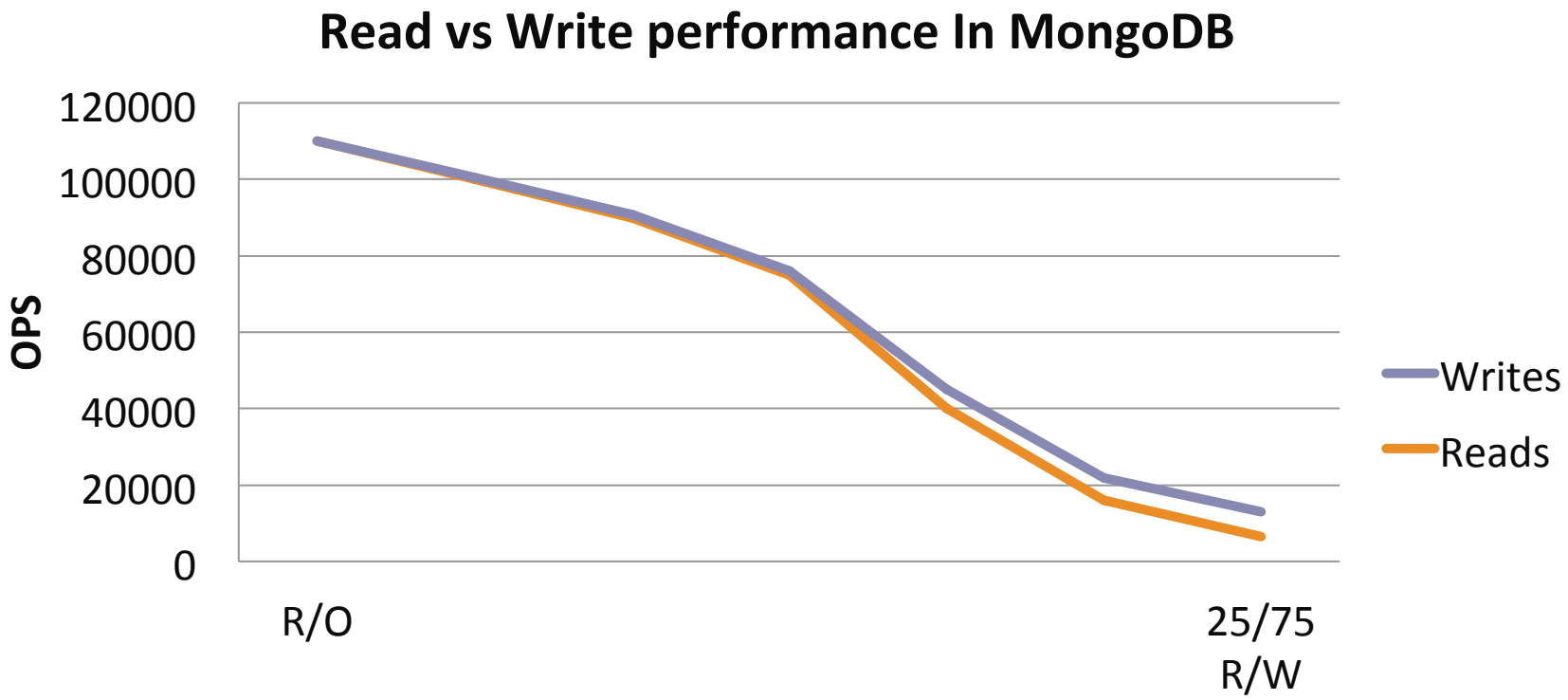
- Founded in December 1999
- Public company (NASDAQ: SFLY)
- > 6B photos
- Oracle, MongoDB, MySQL
- 6 total MongoDB projects, 4 currently in production
- No Cloud based services, our own datacenters.

MongoDB performance; high level

shutterfly

- Similar to traditional RDBMS environments
  - Many of the same patterns and old school tricks still apply
  - Data modeling matters
- Good single instance performance is a prerequisite to good scalability.
  - Tune your statements
  - Instance tuning
- General tuning order
  1. Modeling
  2. Statement tuning
  3. Instance tuning
  4. Hardware tuning
- Know when to stop tuning
  - When is it good enough?
- Build tuning into your SDLC; proactive vs reactive
  - QA testing
  - Application load testing
  - DB load testing
- YMMV
  - Test things with *your* workload

MongoDB Read vs Write performance



* 100 concurrent sessions

Statement Tuning; MongoDB Profiler

shutterfly

- DB level profiling system
- Writes to db.system.profile collection
- Enable it, leave it on. Low overhead.
  - db.setProfilingLevel(1,20);
- What to look for?
  - Full scans
    - nreturned vs nscanned
  - Updates
    - Fastmod (fastest)
    - Moved (exceeds reserved space in document)
    - Key updates (indexes need update)
  - Graph response times over time
- How to look?

```
Show profile
db.system.profile.find().sort({$natural:-1})
db.system.profile.find({millis:{$gt:20}})
```

Profiler Example

shutterfly

```
// need an index
> db.ptest.find({likes:1});
{ "_id" : ObjectId("4dd40b2e799c16bbf79b0c4f"), "userid" : 3404, "imageid" : 35, "img" :
    "www.kennygorman.com/foo.jpg", "title" : "This is a sample title", "data" :
    "38f6870cf48e067b69d172483d123aad", "likes" : 1 }
> db.system.profile.find({}).sort({$natural:-1});
{ "ts" : ISODate("2011-05-18T18:09:01.810Z"), "info" : "query test.ptest reslen:220 nscanned:100000
    \nquery: { likes: 1.0 }  nreturned:1 bytes:204 114ms", "millis" : 114 }

// document moves because it grows
> x=db.ptest.findOne({userid:10})
{
    "_id" : ObjectId("4dd40b37799c16bbf79c1571"),"userid" : 10,"imageid" : 62,
    "img" : www.kennygorman.com/foo.jpg, "title" : "This is a sample title",
   "data" : "c6de34f52a1cb91efb0d094653aae051"
}
> x.likes=10;
10
> db.ptest.save(x);
> db.system.profile.find({}).sort({$natural:-1});
{ "ts" : ISODate("2011-05-18T18:15:14.284Z"), "info" : "update test.ptest  query: { _id: ObjectId
    ('4dd40b37799c16bbf79c1571') } nscanned:1 moved  0ms", "millis" : 0 }
```

Profiler Example

shutterfly

```
// w/o fastmod
> x=db.ptest.findOne({userid:10})
{
     "_id" : ObjectId("4dd40b37799c16bbf79c1571"),
     "userid" : 10,
     "imageid" : 62,
     "img" : "www.kennygorman.com/foo.jpg",
     "title" : "This is a sample title",
     "data" : "c6de34f52a1cb91efb0d094653aae051",
     "likes" : 10
}
> x.likes=11;
11
> db.ptest.save(x);
> db.system.profile.find({}).sort({$natural:-1});
{ "ts" : ISODate("2011-05-18T18:26:17.960Z"), "info" : "update test.ptest  query: { _id: ObjectId
     ('4dd40b37799c16bbf79c1571') } nscanned:1 0ms", "millis" : 0 }

// with fastmod
> db.ptest.update({userid:10},{$inc:{likes:1}});
> db.system.profile.find({}).sort({$natural:-1});
{ "ts" : ISODate("2011-05-18T18:30:20.802Z"), "info" : "update test.ptest  query: { userid: 10.0 }
     nscanned:1 fastmod  0ms", "millis" : 0 }
```

Statement Tuning; Explain()

shutterfly

- Just like most RDBMS implementations
  - Use during development
  - Use when you find bad operations in profiler
  - db.foo.find().explain()
    - > Index usage; nscanned vs nreturned
    - > nYeilds
    - > Covered indexes
    - > Run twice for in memory speed

Explain Example

shutterfly

```
> db.ptest.find({likes:1}).explain()
{
    "cursor" : "BasicCursor",
    "nscanned" : 100000,
    "nscannedObjects" : 100000,
    "n" : 1,
    "millis" : 114,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {

    }
}
```

Explain Example

```
> db.ptest.find({userid:10}).explain()
{
    "cursor" : "BtreeCursor userid_-1",
    "nscanned" : 1,
    "nscannedObjects" : 1,
    "n" : 1,
    "millis" : 0,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : false,
    "indexBounds" : {
     "userid" : [
          [
               10,
               10
          ]
     ]
    }
}
>
```

```
> db.ptest.find({userid:10},{_id:0,userid:1}).explain()
{
    "cursor" : "BtreeCursor userid_-1",
    "nscanned" : 1,
    "nscannedObjects" : 1,
    "n" : 1,
    "millis" : 0,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "isMultiKey" : false,
    "indexOnly" : true,
    "indexBounds" : {
     "userid" : [
          [
               10,
               10
          ]
     ]
    }
}
```

High performance writes

shutterfly

- Single writer process, single DB wide lock scope in MongoDB (1.8.1)
- Total performance is a function of write performance
- All about lock %
  - Use mongostat and look at lock %
  - Graph lock %
- Tuning
  - Read-before-write
    - Spend your time in read and out of write lock scope
    - ~50% reduction in lock %
  - Profiler
    - Tune for fastmod's
      - Reduce moves
      - Evaluate indexes for keychanges
  - Architectural Changes
    - Split by collection
    - Shard
  - Hardware/Write caches
    - Configure RAID card for full write-cache
    - Make sure you have proper disk IOPS available
  - Kernel mods?

High performance reads

shutterfly

- Reads scale fairly easily if you have tuned writes
- Identify reads that can be off slaves
  - SlaveOK
  - Consideration for eventually consistent
- Cache to disk ratio
  - Try to have enough memory in system for your indexes
  - Mongostat faults column
  - Evaluate consistency requirements
    - Replicas
    - Shard
  - How to measure? Setup a test framework mirroring your environment
- Data Locality
  - Organize data for optimized I/O path. Minimize I/O per query.
  - Highly dependent on access patterns. Fetch a bunch of things by a key.
  - Huge gains (or could get worse)
  - How to keep it organized?

Data Locality Example


```
> db.disktest_noorg.find().sort({userid:-1})
{ "_id" : ObjectId("4dd2d82b6a2e502b3043ef33"), "userid" : 49999, "imageid" : 20, "img" : "www.kennygorman.com/foo.jpg", "title" : "This is a
    sample title", "data" : "79357fb65ba7b87f2632dfe8e098400c" }

> db.disktest_noorg.find({}, {'$diskLoc': 1,'userid':1}).sort({userid:-1}).limit(20).showDiskLoc()
{ "_id" : ObjectId("4dd2d82b6a2e502b3043efcd"), "userid" : 49995, "$diskLoc" : { "file" : 0, "offset" : 52953644 } }
{ "_id" : ObjectId("4dd2d82b6a2e502b3043efda"), "userid" : 49995, "$diskLoc" : { "file" : 0, "offset" : 52956088 } }
{ "_id" : ObjectId("4dd2d82c6a2e502b3043f2e5"), "userid" : 49995, "$diskLoc" : { "file" : 0, "offset" : 53102540 } }
{ "_id" : ObjectId("4dd2d82c6a2e502b3043f3e1"), "userid" : 49995, "$diskLoc" : { "file" : 0, "offset" : 53149916 } }
{ "_id" : ObjectId("4dd2d8316a2e502b3044747d"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 1204612 } }
{ "_id" : ObjectId("4dd2d8336a2e502b3044a6ff"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 3635452 } }
…

> var arr=db.disktest_noorg.find().sort({userid:-1})
> for(var i=0; i<arr.length(); i++) {
...     db.disktest_org.insert(arr[i]);
...  }
> db.disktest_org.find({}, {'$diskLoc': 1,'userid':1}).sort({userid:-1}).limit(20).showDiskLoc()
{ "_id" : ObjectId("4dd2d82b6a2e502b3043efcd"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 41684384 } }
{ "_id" : ObjectId("4dd2d82b6a2e502b3043efda"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 41684572 } }
{ "_id" : ObjectId("4dd2d82c6a2e502b3043f2e5"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 41684760 } }
{ "_id" : ObjectId("4dd2d82c6a2e502b3043f3e1"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 41684948 } }
{ "_id" : ObjectId("4dd2d8316a2e502b3044747d"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 41685136 } }
{ "_id" : ObjectId("4dd2d8336a2e502b3044a6ff"), "userid" : 49995, "$diskLoc" : { "file" : 1, "offset" : 41685324 } }
…
```

Data Modeling; optimizing for reads


```
container={
  _id:99,
  userID:100,
  folderName:”My Folder”,
  imageCount:29
}
image={
  _id:1001,
  folderID:99,
  userID:100,
  imageName:”My Image”,
  thumbnailURL:”http://foo/bar.jpg”
}

// write example
>db.container.update({_id:99},{$inc:{imageCount:1}});

// read optimized example
>db.image.find({folderID:99}).count().explain()

     ...
     "indexOnly" : true,
     ...
```

So…

shutterfly

1. Design an efficient schema
2. Tune your statements
3. If you still have performance problems then
   - High faults, high lock %
     - > Memory to disk ratio
     - > Tune writes
   - Low faults, high lock %
     - > Tune writes
   - High faults, low lock %
     - > Scale out reads
     - > More disk IOPS
4. Avoid trouble areas
   - Lots of writes
   - Lots of concurrent writes
   - Long duration writes
   - Lack of hardware resources

Tools

shutterfly

- mongostat
  - Aggregate instance level information
    - > Faults; cache misses
    - > Lock%; tune updates
- mtop
  - Good picture of current session level information
  - Picture of db.currentOp()
    - > Watch "waitingForLock" : true
- iostat
  - How much physical I/O are you doing?
- Home grown load test
  - Make it a priority to try different patterns, measure results.
- Historical data repository

Mongostat output

shutterfly

```
// w/o no miss, no locked
insert  query update delete getmore command flushes mapped  vsize    res faults locked % idx miss %     qr|qw  conn repl       time
     0     62      0      0       0      45       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:55:59
     0    120      0      0       1      55       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:56:00
     0    164      0      0       4      72       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:56:01
     0    158      0      0       0      72       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:56:02
     0    270      0      0       2      52       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:56:03
     0    116      0      0       4      46       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:56:04
     0    180      0      0       1      54       0   137g   160g  40.6g      0        0          0       0|0  2269    M   12:56:05

// r/w not too much miss, some inserts, not bad locked %
insert  query update delete getmore command flushes mapped  vsize    res faults locked % idx miss %     qr|qw  conn repl       time
    88     92     22      0     181     236       0  1542g  1559g    38g      7      2.9          0       0|0  1467    M   12:55:42
    93     93     15      0     170     218       0  1542g  1559g    38g     10      5.2          0       0|0  1467    M   12:55:43
    82    140      3      0     153     233       0  1542g  1559g    38g      4      1.5          0       0|0  1468    M   12:55:44
    94    134      5      0     169     251       0  1542g  1559g    38g      5      1.8          0       0|0  1468    M   12:55:45
    76    147     12      0     135     257       0  1542g  1559g    38g      6      2.5          0       0|0  1468    M   12:55:46
    77     78      9      0     133     173       0  1542g  1559g    38g      7      3.9          0       0|0  1468    M   12:55:47
    81     78      5      0     128     177       0  1542g  1559g    38g      7      6.1          0       0|0  1468    M   12:55:48
    71    133      7      0     125     212       0  1542g  1559g    38g      6      2.9          0       0|0  1468    M   12:55:49

// r/w, lots of update, higher miss, higher locked %
insert  query update delete getmore command flushes mapped  vsize    res faults locked % idx miss %     qr|qw  conn repl       time
     0     56      6      0      11       9       0   508g   517g    42g     70      9.2          0       0|0   798    M   12:55:24
     0     74     25      0      38      28       0   508g   517g    42g     59      6.2          0       0|0   798    M   12:55:25
     0     68      5      0       8       7       0   508g   517g    42g     22      2.2          0       3|1   798    M   12:55:26
     0     57      7      0      17      11       0   508g   517g    42g     62        3          0       0|0   798    M   12:55:27
     0    101     32      0      18      34       0   508g   517g    42g     38      8.6          0       4|0   798    M   12:55:28
     0    125     33      0      29      38       0   508g   517g    42g     44      8.1          0       0|0   798    M   12:55:29
     0    157     29      0      19      31       0   508g   517g    42g     85      7.8          0       1|0   798    M   12:55:30
     0    110     22      0      25      26       0   508g   517g    42g     54      8.5          0       1|0   798    M   12:55:31
     0    114     55      0      51      57       0   508g   517g    42g     80     16.7          0       0|0   798    M   12:55:32
```

Going nuts with Flashcache

shutterfly

- Needs serious IOPS?
- Facebook flashcache. Open source kernel module for linux that caches data on SSD
- Designed for MySQL/InnoDB.
- SSD in front of a disk exposed as a file system mount.
  - /mnt/mydb
- Only makes sense when you have lots of physical I/O.
- Especially good for MongoDB, reduces lock time (lock% goes down) even with high faults.
- We are engineering flashcache into our next gen MongoDB hosts.
- Easy speedup of 500%
  - High cache miss, needing lots of IOPS.
  - Read intensive, highly concurrent.
  - Shard less

Q&A

Questions?

Contact:

http://www.kennygorman.com

twitter: @kennygorman

http://www.shutterfly.com

kgorman@shutterfly.com