MongoDB
Surviving the Graveyard

Robert J. Moore
President
Allanbank Consulting, Inc.

Robert.J.Moore@allanbank.com

Agenda

- Background
- MongoDB Performance
- Event Stream Processing
- Logs / Metrics

- https://github.com/allanbank/tob-nov-13-2012

Background

Background
Perspective

- Java Shop
- High Transaction Rate
- Multiple Sites
- A little data spillage/loss is acceptable

Background
Abuse Everything

- 3 Message Brokers
- 6 Metrics Architectures
- 4 Transactional Data Stores

Background
The Graveyard

- Presharding into Application Memory*
- Terracotta*
- HBase*
- ExtremeDB
- GemFire
- Voldemort

MongoDB Performance

MongoDB Performance

- Only 1 configuration you should use
  - Cluster of replica sets shards

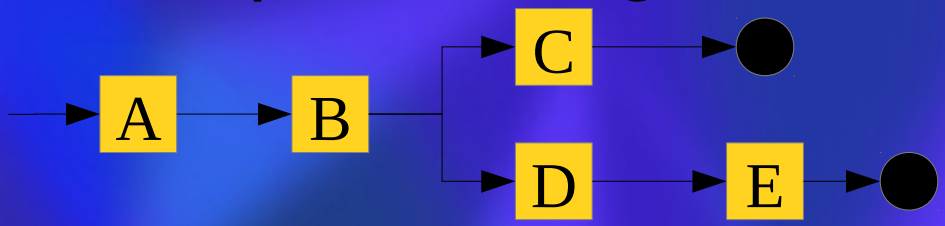
MongoDB Performance

- Must Pre-Split and Balance
- Other keys to going fast
  - Stay in memory
  - Don't grow documents
  - Spread the load
  - Don't wait for a write/journal

Event Stream Processing

Event Stream Processing
Logic

- Apply business rules to the set of events
- Emit the results of the business logic for down stream processing



- Example: Credit Card Fraud Detection
- FindAndModify pushing onto events array

Event Stream Processing
Sample Document

- Correlate on some domain id

```
{
    _id : {
        address: “123 Fake Drive”,
        city : “Annapolis Junction”,
        zip : “20701”
    },
    nextTimeOut : UTC(2012-11-13T14:20:00Z),
    events : [
        { store : “Amazon”, date : UTC(2012-11-13T01:00:00Z), amount : 123.00}
        { store : “Amazon”, date : UTC(2012-11-13T01:10:00Z), amount : 243.00}
        { store : “Amazon”, date : UTC(2012-11-13T01:20:00Z), amount : 399.00}
        ...
    ]
}
```

Event Stream Processing

Code

```
public List<O> handleEvent(E event) {
    byte[] bytes = factory.serialize(event);
    ObjectId id = factory.getId(event);

    DocumentBuilder update = BuilderFactory.start();
    update.push("$push").push("events")
            .add("event", bytes).add("ts", System.currentTimeMillis());
    try {
        FindAndModify command = new FindAndModify.Builder()
                .setQuery(where("_id").equals(id))
                .setUpdate(update).setReturnNew(true).setUpsert(true).build();

        Document doc = collection.findAndModify(command);
        List<E> events = deserialize(doc.get(ArrayElement.class, "events"));
        List<O> output = businessLogic.processEvents(events);

        return output;
    } catch (RuntimeException re) {
        // Handle the error for the 'event'.
    }
    return Collections.emptyList();
}
```

Event Stream Processing

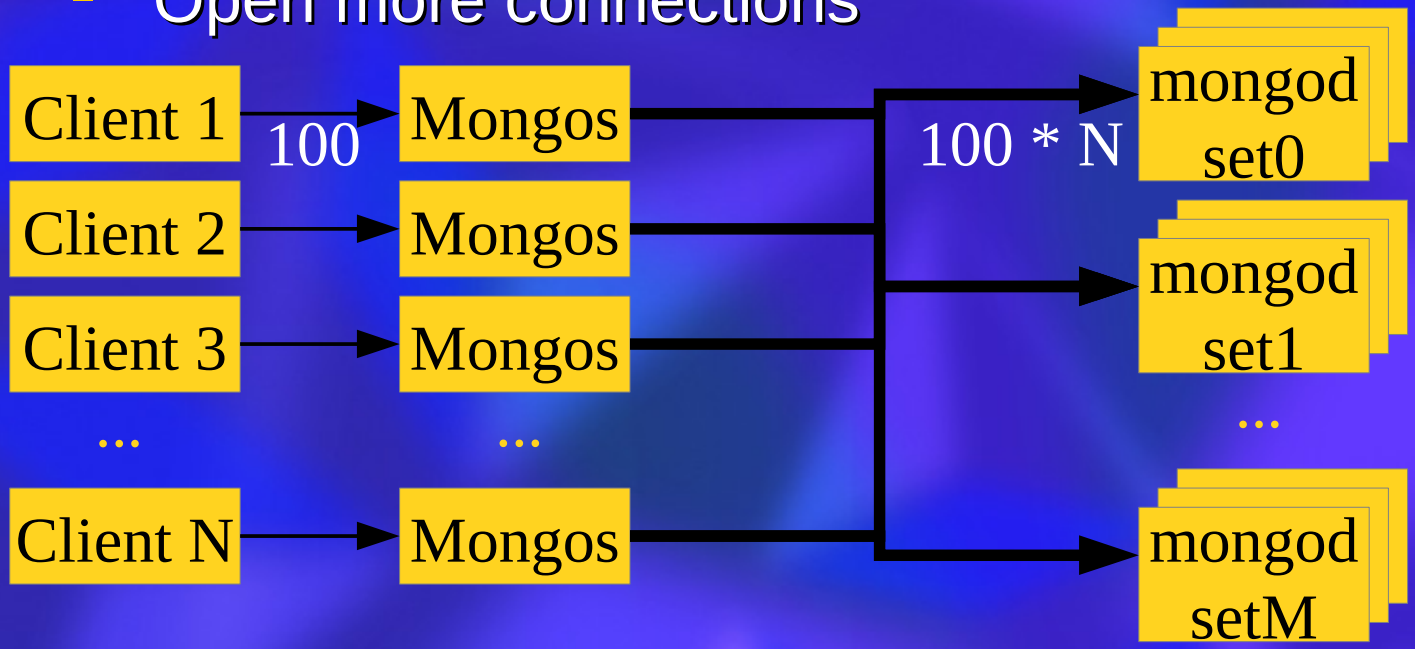
Analysis

- Documents grow over time
  - Causes moves on the server when they grow too big
  - MongoDB tries to adapt by automatically padding documents
- Documents get deleted
  - Fragmented space
    - Can “repair” to reclaim space
  - ~~2.2.0~~ 2.2.1 – Power of 2 Allocator
- Waiting...

Event Stream Processing

Stop Waiting

- 10gen driver blocks the caller waiting for a reply
  - “Blocked Threads Anti-pattern” - Release It, 2007
  - Open more connections



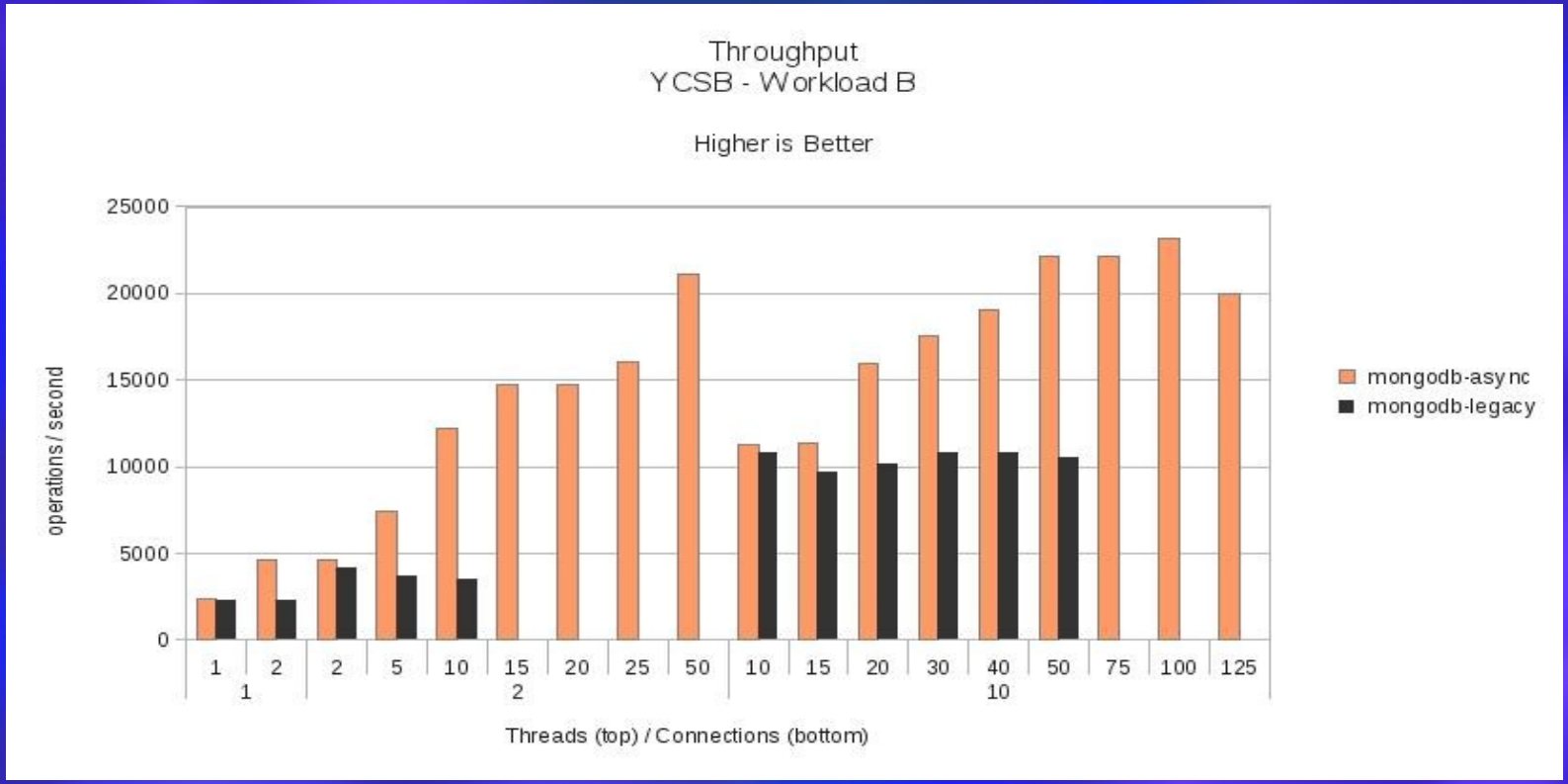
  - Induce Scheduler Problems

Event Stream Processing

Stop Waiting

- Asynchronous Driver
  - Focus on performance and usability
  - Lower latency, higher throughput even under heavy thread contention
  - Benefit when using the synchronous interface
  - Greater benefit using the asynchronous interface
  - http://www.allanbank.com/mongodb-async-driver

Event Stream Processing
Asynchronous Performance



http://www.allanbank.com/mongodb-async-driver/performance/ycsb.html

Event Stream Processing

Asynchronous code

```
public void handleEvent(final E event) {
    byte[] bytes = factory.serialize(event);
    ObjectId id = factory.getId(event);

    DocumentBuilder update = BuilderFactory.start();
    update.push("$push").push("events")
            .add("event", bytes).add("ts", System.currentTimeMillis());
    FindAndModify command = new FindAndModify.Builder()
            .setQuery(where("_id").equals(id))
            .setUpdate(update).setReturnNew(true).setUpsert(true).build();
    Callback<Document> findAndModifyCallback = new Callback<Document>() {
        @Override
        public void callback(Document doc) {
            List<E> events = deserialize(doc.get(ArrayElement.class, "events"));
            List<O> output = businessLogic.processEvents(events);
            for (O outputEvent : output) {
                gateway.send(outputEvent);
            }
        }
        @Override
        public void exception(Throwable thrown) {
            // Do something with the E 'event'.
        }
    };
    collection.findAndModifyAsync(findAndModifyCallback, command);
}
```

Logging / Metrics

Logging / Metrics

- Log record is semi-structured
  - Tag/value structure, not text blob
  - i.e., a JSON document
- Not really interested in the individual records, looking for patterns/aggregations

Logging / Metrics
Simpliest Thing That Could Possibly Work

- Given a map of keys/dimensions and values
  - Add to a single document with a timestamp
  - Probably wise to index the timestamp

```
public void write(Map<String, String> record) {
    DocumentBuilder document = BuilderFactory.start();
    document.addTimestamp("ts", System.currentTimeMillis());
    for (Map.Entry<String, String> element : record.entrySet()) {
        document.add(element.getKey(), element.getValue());
    }

    // Fire and forget!
    collection.insert(Durability.NONE, document);
}
```

Logging / Metrics
STTCPW: Document

```
{
    _id : ObjectId(xxx),
    ts: UTC( '2012-11-13T05:35:00' ),
    id: 'UA-33634837',
    url: '/mongodb-async-driver/',
    lang: 'en',
    location_country : 'US',
    location_city : 'Baltimore',
    browser: 'Chrome',
    os : 'Linux',
    network_provider : 'Verizon',
    search_provider : 'google',
    search_term : 'mongodb fastest java driver',
    user_id : '123456',
    ...
}
```

Logging / Metrics
STTCPW: Analysis

- Use of ObjectId for the _id causes clustered writes
  - Switch to a hash of the values in the document
- Have to perform some form if aggregation to “see” anything
  - Aggregation has to look at all of the records within the time range
  - Feeling very Hadoop sequence file-ish

Logging / Metrics
STTCPW: Getting results out.

```
protected void aggregate() throws ParseException {
    // Between 05:00 and 06:00 on 2012-11-13 for the Windows OS using Chrome
    // browser with a resolution of 1920x1080 from Austrailia?
    final SimpleDateFormat sdf = new SimpleDateFormat(
            "yyyy-MM-dd'T'HH:mm:ss");
    final Aggregate command = new Aggregate.Builder()
            .setReadPreference(ReadPreference.PREFER_SECONDARY)
            .match(where("ts")
                        .greaterThanOrEqualToTimestamp(
                            sdf.parse("2012-11-13T05:00:00").getTime())
                         .lessThanTimestamp(
                            sdf.parse("2012-11-13T06:00:00").getTime())
                    .and("os").equals("Windows")
                    .and("browser").equals("Chrome")
                    .and("screen_resolution").equals("1920x1080")
                    .and("location_country").equals("AU"))
            .group(constantId("count"), set("pageviews").count()).build();

    final List<Document> results = collection.aggregate(command);
    System.out.println(results);
}
```

Logging / Metrics
Version 2

- Observations
  - Most records have some kind of id for a related entity
  - Most record have a “type” or “name”
- Why not leverage that fact?

Logging / Metrics
Version 2

```
public void write(String id, String type, Map<String, String> record) {

    DocumentBuilder document = BuilderFactory.start();
    document.addTimestamp("ts", System.currentTimeMillis());
    for (Map.Entry<String, String> element : record.entrySet()) {
        document.add(element.getKey(), element.getValue());
    }

    DocumentBuilder update = BuilderFactory.start();
    update.push("$push").add(type, document);

    // Fire and forget!
    collection.update(where("_id").equals(id), update,
            false /* multiupdate */, true /* upsert */, Durability.NONE);
}
```

```
{
    _id : "0000-0000-0000-0001",
   trans : [ { store : “Amazon”, ts : UTC(2012-11-13T01:00:00Z), accepted : true } ],
   lost: [ { ts : UTC(2012-11-12T01:00:00Z) ],
}
```

Logging / Metrics
Version 2: Analysis

- Use of domain id could cause hot spots, again hash.
- Documents grow
  - Have to be careful to not grow without bounds
- Very nice for seeing what happened to a single entity
- Have made aggregation a little harder
  - Will want to unroll the arrays
  - Made map/reduce easier but should avoid since cannot run on secondaries

Logging / Metrics
Pre-aggregated / Tactical Metrics

- Observations
  - First version was really better suited for Hadoop
    - But Map/Reduce takes so long
    - Don't want to or can't wait
  - Version two does not perform/scale
  - The set of questions that can't wait may have a limited scope
    - Usually don't need the entire record
    - Usually just need counts, not each entity

Logging / Metrics
Tactical Metrics

- Aggregate the records on a subset of the fields/dimensions
- Truncate time to some reasonable resolution

```
// Can we answer: between 05:00 and 06:00 on 2012-11-13
// for the Windows OS using Chrome browser with a resolution
// of 1920x1080 from Austrailia? With this document?
{
     _id { ts: UTC( '2012-11-13T05:35:00' ),
           browser: 'Chrome',
           location_country : 'US',
           os : 'Linux',
           screen_resolution : '1920x1080', },
     count : 1
 }
```

Logging / Metrics
Tactical Metrics: Code

```
public void write(final Map<String, String> record) {

    final DocumentBuilder id = BuilderFactory.start();
    final DocumentBuilder idDoc = id.push("_id");
    id.addTimestamp("ts", currentTimePeriod());
    for (final String keyDimension : keyDimensions) {
        final String keyValue = record.get(keyDimension);
        if (keyValue != null) {
            idDoc.add(keyDimension, record.get(keyDimension));
        }
    }

    final DocumentBuilder update = BuilderFactory.start();
    update.push("$inc").add("count", 1);

    // Fire and forget!
    collection.update(where("_id").equals(id), update,
            false /* multiupdate */, true /* upsert */, Durability.NONE);
}
```

Logging / Metrics
Tactical Metrics: Aggregation

```
protected void aggregate() throws ParseException {
    // Between 05:00 and 06:00 on 2012-11-13 for the Windows OS using Chrome
    // browser with a resolution of 1920x1080 from Austrailia?
    final SimpleDateFormat sdf = new SimpleDateFormat(
            "yyyy-MM-dd'T'HH:mm:ss");
    final Aggregate command = new Aggregate.Builder()
            .setReadPreference(ReadPreference.PREFER_SECONDARY)
            .match(where("id.ts")
                        .greaterThanOrEqualToTimestamp(
                            sdf.parse("2012-11-13T05:00:00").getTime())
                        .lessThanTimestamp(
                            sdf.parse("2012-11-13T06:00:00").getTime())
                    .and("id.os").equals("Windows")
                    .and("id.browser").equals("Chrome")
                    .and("id.screen_resolution").equals("1920x1080")
                    .and("id.location_country").equals("AU"))
            .group(constantId("count"), set("pageviews").sum("count")).build();

    final List<Document> results = collection.aggregate(command);
    System.out.println(results);
}
```

Logging / Metrics
Tactical Metrics: Analysis

- Use of document for the id MAY cause hot spots
  - Again, we can hash, but keep _id as id
- Documents do not grow
- Cannot see what happened to a single entity
  - Can see what is happening across the system
- Aggregation is faster since the number of documents is smaller
- Nothing stops us from having multiple views
- Still doing an update per event

Logging / Metrics
Tactical Metrics: Version 2

- Why always increment by 1?
- Batch the updates in the log writing process
- Start to care about the batches
  - No more fire-and-forget

Logging / Metrics
Tactical Metrics V2: Code

```
private synchronized void commitBatch(final long now) {
    // ...
        final List<Future<Long>> updateResults = new ArrayList<Future<Long>>(batch.size());
        for (final Map.Entry<Document, ConcurrentMap<String, Long>> entry : batch
                .entrySet()) {
            final Document id = entry.getKey();
            final DocumentBuilder update = BuilderFactory.start();
            final DocumentBuilder inc = update.push("$inc");
            for (final Map.Entry<String, Long> value : entry.getValue().entrySet()) {
                inc.add(value.getKey(), value.getValue().longValue());
            }
            // Lot of information lost if the update fails. Lets make sure
            // it gets to the server and is handled.
            // ... but wait! Still no reason to wait for each update.
            updateResults.add(collection.updateAsync(where("_id").equals(id),
                    update, false /* multiupdate */, true /* upsert */, Durability.ACK));
        }

        for (final Future<Long> updateResult : updateResults) {
            updateResult.get();
        }
}
```

Logging / Metrics
Tactical Metrics V2: Analysis

- Everything from version 1 except
  - No more write per record
  - It scales

Graveyard Survival

- Performance
- Features
- Flexibility

Questions?

- Contact Information:

Robert.J.Moore@allanbank.com

Allanbank

CONSULTING