"Event Streaming & Message Queueing with MongoDB",

"event": "MongoDB Berlin 2013",
"name": "Adam Warski",
"company": "SoftwareMill",
"e-mail": "adam@warski.org",
"twitter": "@adamwarski"
About me

• CTO at SoftwareMill, [http://softwaremill.com](http://softwaremill.com)
• Programmer
• Engaged in some open-source projects
  o Hibernate Envers
  o ElasticMQ
  o Veripacks
  o SoftwareMill Bootstrap
• Blog @ [http://www.warski.org/blog](http://www.warski.org/blog)
• Current tech stack:
  o Scala
  o Akka
  o Spray
  o MongoDB
Plan

• Project background
  o High-performance messaging system
  o Accept requests from users
  o Process asynchronously
  o Provide reporting

• Using Mongo as a queue

• Using Mongo for Event Streaming in Java
Mongo Message Queue
The requirements

- **Persistent** messaging
  - Short-lived messages
  - Some may linger for a longer time
  - Messages shouldn’t be lost

- **Fast**
  - But not insanely fast
  - Currently we need 1000s msgs / second

- If needed, possible to scale-up & **scale-out**
Queue interface

- **Send** a message (a String)
- **Receive** a message, blocking it for x seconds
- **Delete** a message
Sounds familiar?

- Amazon SQS semantics
- At-least-once delivery guarantee
- Also check out ElasticMQ, [http://elasticmq.org](http://elasticmq.org)
How to implement?

- **Mongo document structure:**
  - `_id`
  - Message content
  - Next delivery (timestamp)

- **Message send:**
  - Insert into collection
  - Next delivery := now
  - Return `_id` (message id)

- **Message delete:**
  - Delete document from collection
How to implement? (2)

• **Message receive:**
  - Find-and-modify
  - Find: next delivery must be <= now
  - Modify: next delivery := now + 10 seconds

• **Why does this work?**
  - Find-and-modify is **crucial**
  - Atomic operation
Meeting the requirements

- Replication OOTB
  - Replica Sets

- Scaling out
  - Starling/Kestrel model
  - Setup 2 identical replica sets (e.g. 2x3 servers)
  - Send/receive from a random server
Good/bad sides

• **Good sides:**
  o Easy to implement
  o Simple interface
  o Replication

• **Bad sides:**
  o Active polling
  o No batching
Write concerns

• Can we tolerate lost messages?

• Different write concerns during send
  - SAFE
  - REPLICA_SAFE
Mongo Event Streaming
General idea

- System generates a **series of events**
- Other components follow the stream
- Similar to Event Sourcing/CQRS
- Reading and writing of the events is **decoupled**
- Any following component may die & **catch up**
- Bursts of event activity don’t cause an overall slowdown
The requirements

• Fast event writing
  o again, 1000s per second
• **Main source of truth** in the system
• Stream the events
  o as they are written
  o in batches
  o write reports to SQL DB
• Replicate data
• Store data up to Y GB
  o prevent lack of disk space
The collection

• Capped collection
  o By definition, size-constrained
  o We get a **circular buffer** for events

• Replicated
  o Hence an index on `_id` is mandatory
  o Until 2.2, capped collections didn’t have an `_id` index by default
Writing events

- Insert

- Write concerns – how tolerant we are of event loss

- Events should be immutable
  - Nice (Java) code
  - Event sinks wouldn’t know when events get updated
  - Changing document size – moving blocks on disk
  - Not possible in a capped collection
Reading events

- There may be multiple readers

- We want to get new events as they come in
  - But without active polling, if possible

- **Tailable cursors** are the answer
  - Need to provide a starting point – last read event
  - Will optionally block if no data is available
  - Can’t be a TTL collection

- The reader must store the **last read event id**
  - Transactions can be useful here
DBObject query = lastReceivedEventId.isPresent() ? BasicDBObjectBuilder.start("_id", BasicDBObjectBuilder.start("$gte", lookFrom(lastReceivedEventId.get())).get()).get() : null;

DBObject sortBy = BasicDBObjectBuilder.start("$natural", 1).get();

DBCollection collection = ... // must be a capped collection
DBCursor cursor = collection.find(query).sort(sortBy).addOption(Bytes.QUERYOPTION_TAILABLE).addOption(Bytes.QUERYOPTION_AWAITDATA);
Reading events (Java)

- Note the `$gte`
  - Skip events until the last received event is found
  - Looking from ... e.g. 10 minutes before the last received event
  - Cannot query for “documents created after a given document”

- We get a Java **Iterator**

- Data from Mongo is received in batches
  - Implemented by the Java driver
  - Only some calls to `hasNext()`/`next()` will cause network I/O

- The potentially blocking call is `hasNext()`
Intermediate queue

• To get events in batches without delays, we need an intermediate queue

• Two threads
  o One reading from Mongo
  o Second reading from the queue and processing

• E.g. a `LinkedBlockingQueue`
  o Has a size limit
  o When reading, first we do a blocking `poll()`
  o Then drain the queue
Intermediate queue
Use dedicated components?

• Maybe, in the future
  o If performance requirements rise
• The components are very easy to replace
  o If you write nice code, that is ;)
• Simplified setup & deployment
  o Both local, and on production
• Fewer external components
• Focus on the business problem, not on the infrastructure
  o As always, a question of balance
Thank you!

Blog: http://www.warski.org/blog

E-mail: adam@warski.org

Twitter: @adamwarski