Choosing a fault-tolerant database to support cloud infrastructure

Alexandre Hardy Omkar Tilak

Yet another Conference, 1 October 2012, Moscow

nimbula

Overview

Background
- Cloud OS
- Requirements
- Contenders

Cassandra
- Features
- Architecture

MongoDB
- Features
- Architecture
- Replication
- Sharding

Evaluation
- Test data
- Tests performed
- Test platform
- Our settings

Results
- Footprint
- Performance
- Resiliency
- Further testing

Conclusion
- Conclusion

Overview

Background

Cloud OS

Requirements

Contenders

Cassandra

MongoDB

Evaluation

Results

Conclusion

Background

Overview
Background
Cloud OS
Requirements
Contenders
Cassandra
MongoDB
Evaluation
Results
Conclusion

- Nimbula Director is a self-service Cloud OS.
- It provides facilities for creating and managing:
  - Virtual machines.
  - Virtual networks.
  - Virtual storage attachments.
  - ... and more.

Nimbula Director: A Cloud OS

Overview
Background
Cloud OS
Requirements
Contenders
Cassandra
MongoDB
Evaluation
Results
Conclusion

- Supports multiple tenants with full isolation between tenants (or allows them to cooperate).
- Automatic failure detection, and recovery from single node failures.
- Needs a resilient database to support self-healing.
- Please visit http://www.nimbula.com for details ...
- ... and come visit our stand!

Overview
Background
Cloud OS
Requirements
Contenders
Cassandra
MongoDB
Evaluation
Results
Conclusion

What we need to support a Cloud OS:

- The ability to scale out the database:
  - Based on the number of nodes and the objects a deployment can create in the cloud.
  - For improved performance.
  - To increase capacity of the database.
- Reliability:

Overview
Background
Cloud OS
Requirements
Contenders
Cassandra
MongoDB
Evaluation
Results
Conclusion

What we need to support a Cloud OS:

- The ability to scale out the database:
- Reliability:
  - Sufficient replicas to recover from failure. (We want 3)
  - Efficient mechanism to keep sufficient replicas of a data item (i.e. don't re-replicate everything).
  - Handle split brain (network) scenarios effectively, while maintaining good data durability.

Why not a conventional relational database?

Overview
Background
Cloud OS
Requirements
Contenders
Cassandra
MongoDB
Evaluation
Results
Conclusion

- Compatible licensing.
- Limited replication. Not always easy to set replication guarantees on transactions.
- Need configurable consistency across replicas.

Overview
Background
Cloud OS
Requirements
Contenders
Cassandra
MongoDB
Evaluation
Results
Conclusion

There are many projects that could potentially provide a solution. We selected two based on a prior study of the alternatives, and proceeded to evaluate these two in depth:

- Cassandra 1.0.0
- MongoDB 2.0.0

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

Cassandra

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Proven – Cassandra is in use at Netflix, Twitter, Urban Airship, Constant Contact, Reddit, Cisco, OpenX, Digg, CloudKick, Ooyala, and more companies that have large, active data sets. The largest known Cassandra cluster has over 300 TB of data in over 400 machines.
- Fault Tolerant
- Decentralized
- You're in Control

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Proven
- Fault Tolerant – Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no down time.
- Decentralized
- You’re in Control

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Proven
- Fault Tolerant
- Decentralized – There are no single points of failure. There are no network bottlenecks. Every node in the cluster is identical.
- You're in Control

Cassandra features

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Proven
- Fault Tolerant
- Decentralized
- You're in Control – Choose between synchronous or asynchronous replication for each update. Highly available asynchronous operations are optimized with features like Hinted Handoff and Read Repair.

Cassandra features

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Rich Data Model – Allows efficient use for many applications beyond simple key/value.
- Elastic
- Durable
- Professionally Supported

nimbula

Cassandra features

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Rich Data Model
- Elastic – Read and write throughput both increase linearly as new machines are added, with no down time or interruption to applications.
- Durable
- Professionally Supported

Cassandra features

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Rich Data Model
- Elastic
- Durable – Cassandra is suitable for applications that can't afford to lose data, even when an entire data center goes down.
- Professionally Supported

Cassandra features

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Rich Data Model
- Elastic
- Durable
- Professionally Supported – Cassandra support contracts and services are available from third parties.

Cassandra features

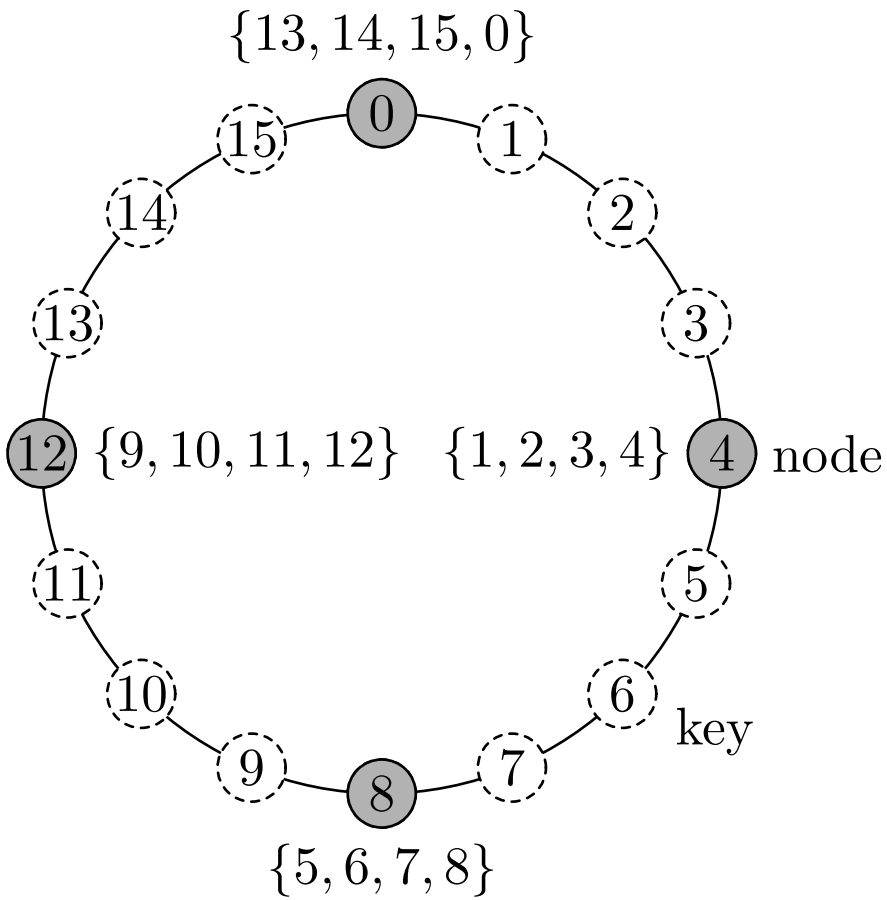
Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

- Rich Data Model
- Elastic
- Durable
- Professionally Supported

Sounds perfect!

Overview
Background
Cassandra
Features
Architecture
MongoDB
Evaluation
Results
Conclusion

Distributed hash table:



Overview
Background
Cassandra
MongoDB
Features
Architecture
Replication
Sharding
Evaluation
Results
Conclusion

MongoDB

MongoDB features

Overview
Background
Cassandra
MongoDB
Features
Architecture
Replication
Sharding
Evaluation
Results
Conclusion

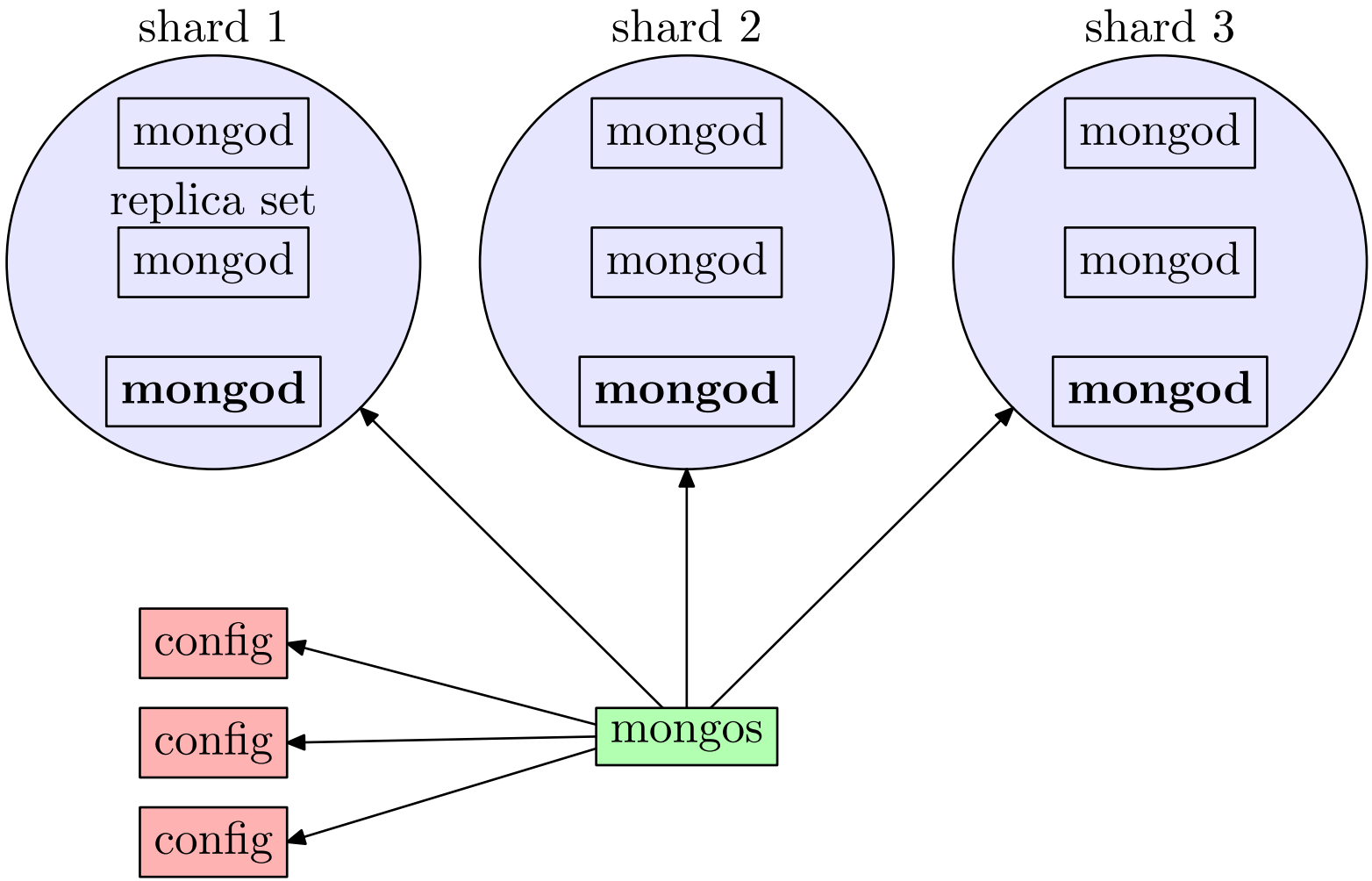
- Document-oriented storage – JSON-style documents with dynamic schemas offer simplicity and power.
- Full Index Support – Index on any attribute, just like you're used to.
- Replication & High Availability – Mirror across LANs and WANs for scale and peace of mind.
- Auto-Sharding – Scale horizontally without compromising functionality.
- Querying – Rich, document-based queries.

MongoDB features

Overview
Background
Cassandra
MongoDB
Features
Architecture
Replication
Sharding
Evaluation
Results
Conclusion

- Fast In-Place Updates – Atomic modifiers for contention-free performance.
- Map/Reduce – Flexible aggregation and data processing.
- GridFS – Store files of any size without complicating your stack.
- Commercial Support – Enterprise class support, training, and consulting available.

Overview
Background
Cassandra
MongoDB
Features
Architecture
Replication
Sharding
Evaluation
Results
Conclusion



MongoDB: Replication

Overview
Background
Cassandra
MongoDB
Features
Architecture
Replication
Sharding
Evaluation
Results
Conclusion

- Master-slave replication, configurable number of slaves.
- Journal for greater reliability.
- Oplog for replication (i.e. A slave doesn't have to download the full dataset, it can recover by replaying the Oplog).
- Configurable persistence (majority of replicas, sync to disk).
- Master election which handles split-brain scenarios.

Overview
Background
Cassandra
MongoDB
Features
Architecture
Replication
Sharding
Evaluation
Results
Conclusion

- Auto-sharding
- Config servers act as "routers" for shard data. Store metadata.
- Mongo shard server handles multiple requests communicating with shards.
- Each shard is a replica-set with a replication factor.
- Disadvantage: Not as easy to disperse data across many nodes.

Overview

Background

Cassandra

MongoDB

Evaluation

Test data

Tests performed

Test platform

Our settings

Results

Conclusion

Evaluation

Test data

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

- 160 Customers
- 8000 Users
- 1600 Groups
- 960 Image lists

91040 Objects

- 4800 Machine images
- 8000 Security lists
- 16000 Security list rules
- 51520 Permissions

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

Performance tests

- Read performance in a realistic setting: API listing of users, with authorization checks (graph traversal).
- Write performance (create several customers, users and other objects in the system) with authorization checks etc.
- Pure read performance through our python transport.
- Pure write performance through our python transport.
- Pure delete performance through our python transport.

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

Resiliency tests

- Reboot nodes serially.
- Reboot nodes simultaneously.
- Reinstall nodes serially (with recovery time).
- Reinstall nodes serially (without recovery time).

After each failure, we wait for the system to get back into a minimal "healthy" state, assisting where necessary (assuming the assistance can be automated).

Tests not-performed

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

We performed split-brain tests on MongoDB at a later stage, but the tests were not performed with Cassandra, so we do not report on that in this presentation.

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

4 SuperMicro Nodes:

- RAM: 8GB
- CPU: Intel Core i5 650 3.2GHz
- HDD: 1 x 500GB , Seagate ST3500418AS 7200RPM

Baseline for comparison:

- PostgreSQL replicated with DRBD.

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

Prototype:

- Use timestamp to implement "transaction" semantics (newest operation wins).
- Reads and writes are specified to require QUORUM, with 3 replicas.
- i.e. 2 successful writes are required.

Overview
Background
Cassandra
MongoDB
Evaluation
Test data
Tests performed
Test platform
Our settings
Results
Conclusion

Prototype:

- Single connection to database.
- 3 replicas.
- No sharding for resiliency tests.
- We select a "majority" for writing.
- i.e. 2 successful writes are required.

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

Results

Disk Footprint

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

	Cassandra	MongoDB[1]	PostgreSQL
per customer	42Mb	192Mb	41Mb
per node (3 nodes)	6.8Gb	37 Gb	6.4Gb

[1] MongoDB preallocates files. It begins by allocating 64Mb, and when that is near full, it preallocates another 128Mb etc. This is per database, and in Nimbula Director we had 10 databases.

Memory Footprint

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

	Cassandra[1]	**Mongo**[2]
Memory	370Mb	500Mb

[1] Cassandra was configured to use 256Mb heap size.

[2] MongoDB uses memory mapped files, so the actual footprint depends on the OS.

Read performance

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

	Cassandra	MongoDB	PostgreSQL
List 8222 users: API	10.558s	6.768s	10.03s
List 8222 users: Raw DB	2.923s	0.879s	2.592s
1000 Reads: Raw DB	0.9565s	0.771s	0.859s

Write performance

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

	Cassandra	MongoDB	PostgreSQL
160 customers[1]	5 hours[2]	340 minutes[3]	1693 minutes
1000 objects: (direct)	39.84s	1.519s 143s[4] 8.98s[5]	15.662s

[1] 91040 Objects created, with authorization checks (graph traversals) and multiple reads.

[2] Only 31 customers. Test did not complete.

[3] This is without a guarantee that the data is written to disk, only a guarantee that the data is written to at least two nodes.

nimbula

Write performance

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

	Cassandra	MongoDB	PostgreSQL
160 customers[1]	5 hours[2]	340 minutes[3]	1693 minutes
1000 objects: (direct)	39.84s	1.519s 143s[4] 8.98s[5]	15.662s

[4] If we turn on fsync, then MongoDB waits until there is sufficient data, or a fixed interval before committing to disk.

[5] We increase the write throughput with 20 connections to the database per application.

Delete performance

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

	Cassandra	MongoDB	PostgreSQL
1000 objects: Raw DB	32.92s	1.328s	15.465s

Reboot nodes – Serially

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

- Cassandra: Pass
- MongoDB: Pass

Reboot nodes – Simultaneously

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

- Cassandra: Pass
- MongoDB: Pass

Re-install nodes

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

- Cassandra: Pass, with assistance. We had to run `nodetool -repair` and correct the hash ring. Neglecting this step resulted in data loss which Cassandra did not detect.
- MongoDB: Pass, with assistance. We had to remove nodes that no longer existed from the replica set configuration (we later automated this step).

Re-install nodes (no recovery time)

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

We failed a node in the middle of replication, but ensured that all data remained available in the cluster.

- Cassandra: We experienced issues failing `nodetool -repair`. We were unable to complete repair after failing a single repair.
- MongoDB: We did not have to perform any extra recovery steps for this scenario. MongoDB handled this situation almost identically to the reinstall of nodes with a completely healthy record set.

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

- We implement additional indexes by making use of columns.
- Some notion of transactions are required to implement indexes.
- Have to use ZooKeeper, or timestamps.
- Timestamps require an additional global clock to be reliable.
- Customizable consistency and replication.
- Bootstrap in an automated environment can be a bit challenging: Cassandra doesn’t handle schema changes on multiple nodes very well.

Further testing: MongoDB sharding

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Footprint
Performance
Resiliency
Further testing
Conclusion

- The meta-data used for auto-sharding is not as resilient as the rest of the data in MongoDB.
- We managed to handle several failures, but still managed to arrive in a state which required manual intervention. This manual intervention sometimes involved manual reconstruction of all the sharding meta-data.
- We decided not to ship Nimbula Director with MongoDB auto sharding.

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Conclusion
Conclusion

Conclusion

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Conclusion
Conclusion

- No database is perfect: need to select the best option for the requirements.
- Cassandra did not meet our resiliency requirements, although it did meet several other requirements.
- MongoDB performed exceptionally well during failure tests, and satisfied our other requirements.
- Nimbula Director 2.0 was shipped with MongoDB on 31 May 2012.

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Conclusion
Conclusion

- MongoDB has proved to be extremely resilient in all our subsequent tests.
- We are currently implementing our own sharding solution for MongoDB to improve scalability.

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Conclusion

Questions?

Contact details

Overview
Background
Cassandra
MongoDB
Evaluation
Results
Conclusion

Presenter: Alexandre Hardy
e-mail: ah@nimbula.com
Telephone: 1-650-257-1300 ext. 7003
Twitter: @nimbula

nimbula