Evaluating Durability of Distributed Databases

Theory and Empirical Studies of MongoDB

Kosta Dunn
Supervisor: Alan Fekete

Updated: 2018/11/12

THE UNIVERSITY OF SYDNEY

Background

Durability

Once a transaction has been committed, it will remain committed even in the case of a crash of 1 or more machines.

It is one of the ACID properties, which ensures that an operation which has been acknowledged *cannot* be lost. (Haerder and Reuter, 1983)

Durability is violated if an *acknowledged write is lost*.

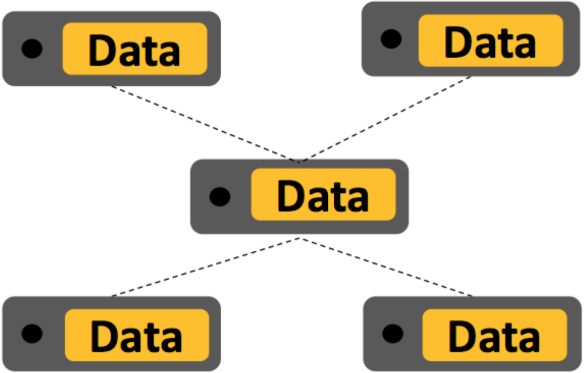
MongoDB

MongoDB is a widely used distributed database system. It is becoming one of the primary choices for storing critical user data.

- Document model
- Document = Row (in traditional database)
- No schema

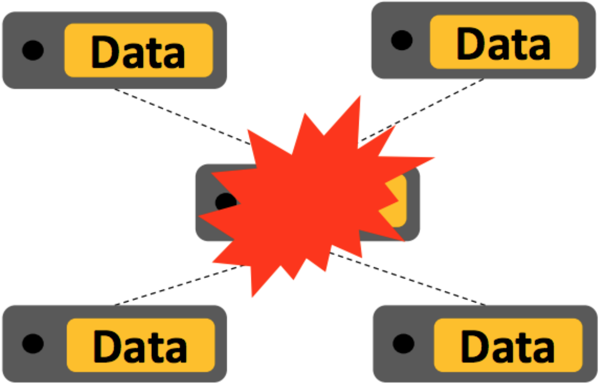
It provides high performance, availability and consistency. These properties are achieved via *replication*.

Replication



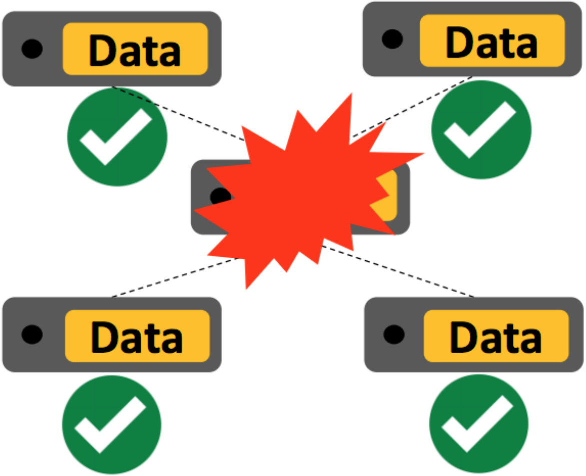
Create a *replica set* by copying data across multiple machines...

Replication



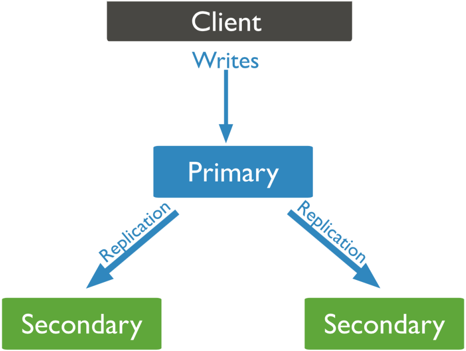
...So when one machine goes down...

Replication



... The data (and hence the service) are still available!

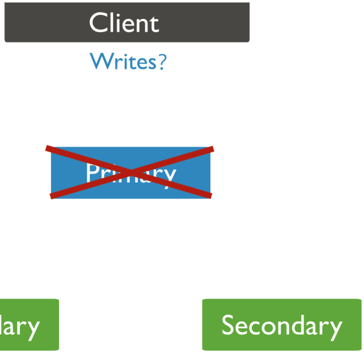
Replication in MongoDB



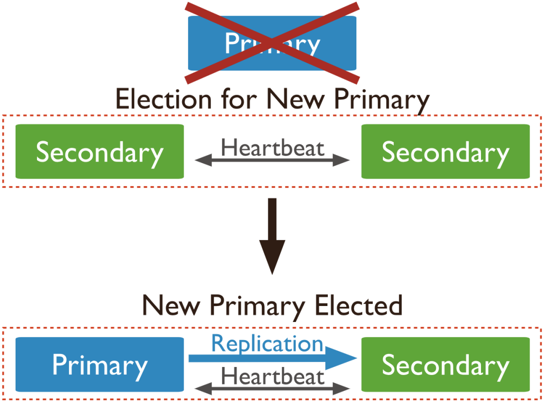
MongoDB uses a *Primary-Backup* strategy. **One** Primary, **the rest** are Secondaries.

All **write** operations must go through the Primary.

Replication in MongoDB

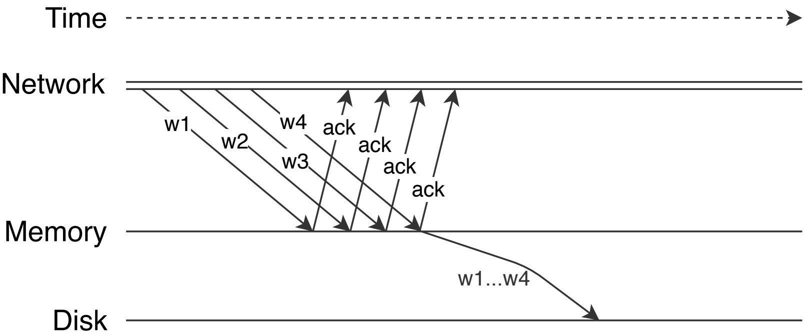


If a primary is down, **no write operations** can be acknowledged.



When the Primary fails, all working Secondaries perform an *election* to select a new primary.

MongoDB Journal



A journal is used to *recover* after failure by "replaying" the operations.

MongoDB with **Journaled** writes will **send acknowledgements** without them going to disk!

MongoDB Specifics

MongoDB allows users to configure the *strength* of their writes through the **write concern**.

Stronger = harder to lose, but longer to acknowledge.

MongoDB Specifics

MongoDB allows users to configure the *strength* of their writes through the **write concern**.

Stronger = harder to lose, but longer to acknowledge.

Primary Operation is applied to the data on the primary

Journaled Operation is applied to data and added to Journal on the primary

Majority Operation is applied and added to Journal on the *majority* of the replicas.

MongoDB Specifics

MongoDB allows users to configure the *strength* of their writes through the **write concern**.

Stronger = harder to lose, but longer to acknowledge.

Primary Operation is applied to the data on the primary

Journaled Operation is applied to data and added to Journal on the primary

Majority Operation is applied and added to Journal on the *majority* of the replicas.

Under **Majority** write concern, there should be *no write loss!*

What has been done

Brewer (2000) conjectured that databases cannot be simultaneously *Consistent, Available and Network failure tolerant*. Gilbert and Lynch (2002) proved this result.
$\Rightarrow$ There are limits to the capability of any database!

A tool "Jepsen" was developed to test effects of *Network failures* on databases. (Kingsbury, 2013, 2017; Patella, 2018)

And Alagappan et al. (2016) studied durability when *all* replicas fail, with a focus on file systems.
Note: probability of *all* replicas failing is **incredibly slim**.

There is **no work** on durability of distributed databases under single machine failures.

The Aim

To equip users and designers of distributed databases with the means to protect their systems from durability failures.

Thesis Contributions

- **Categorisation of scenarios that result in write loss.**
- **Design of an experiment capable of inducing write loss.**
- Algorithm to quantify the number of lost writes.
- **Empirical results to show that the experiment and algorithm work by detecting bugs in MongoDB 3.6-rc0.**
- Theoretical model for evaluating when a write becomes durable.
- **Estimation of when a write becomes durable on the Primary, using rudimentary client-accessible measurements.**
- **An empirical study of time-till-durability on the Primary for acknowledged writes.**

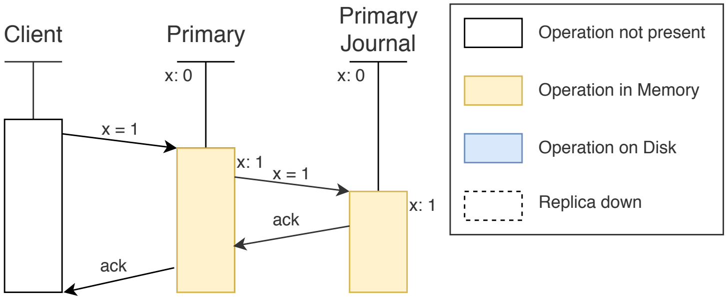
Detecting and Quantifying Durability Failures

Big picture

Can we create a scenario that *forces* MongoDB to lose a write?

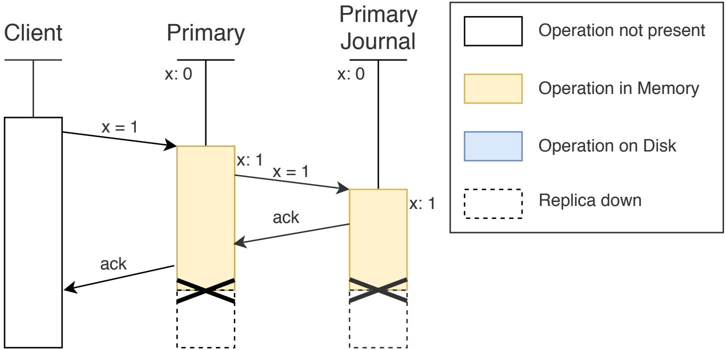
How big is the impact of these scenarios?

Write Loss scenario



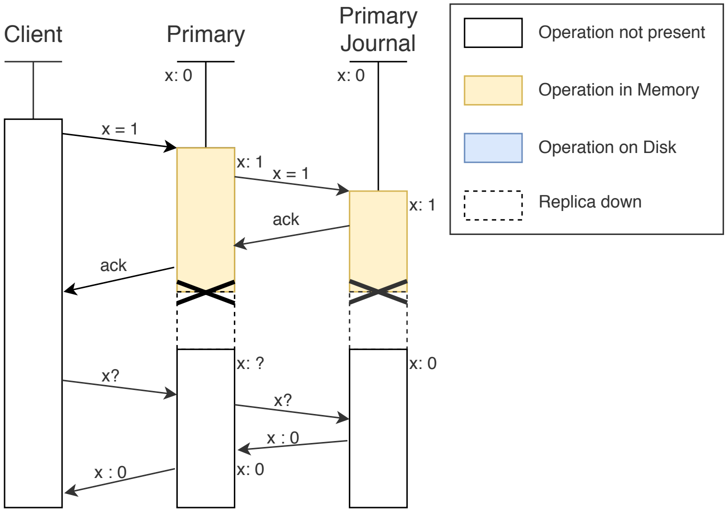
We issue a **Journaled** write and get an *acknowledgement*.

Write Loss scenario



The Primary *crashes*.

Write Loss scenario



The Primary *loses* any writes still *in-memory*.

How do we do this empirically?

We created a tool that:

- Configures a replica set
- Stresses it with reads and writes
- Crashes the Primary **100 seconds** into experiment.
- Recovers the (old) Primary **200 seconds** into experiment.
- Observes *incorrect values* in queries.

Results - Write Loss: Mongo 4.0

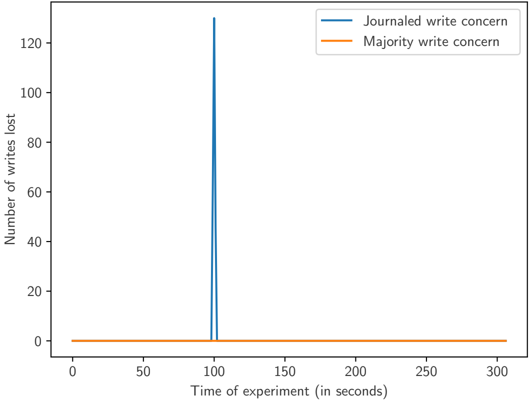


Figure: Distribution of write loss for every second in MongoDB 4.0.

Results - Write Loss: Mongo 3.6-rc0

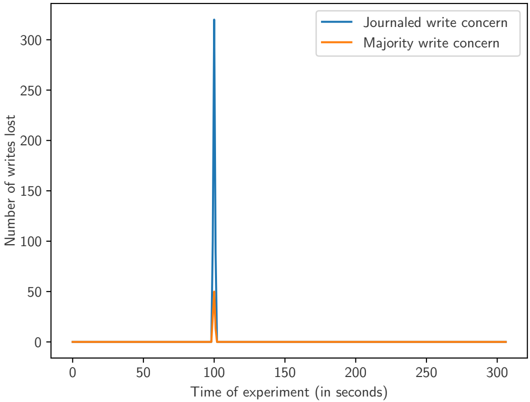


Figure: Distribution of write loss for every second in MongoDB 3.6-rc0.

Results - Summary

MongoDB 4.0 performs as predicted by our theory. Found losses where we expect based on our theory.

MongoDB 3.6-rc0 loses writes where they *shouldn't be lost*.

$\Rightarrow$ Our tool succeeded in detecting bugs in MongoDB 3.6-rc0.

Conclusion

Our tool works!

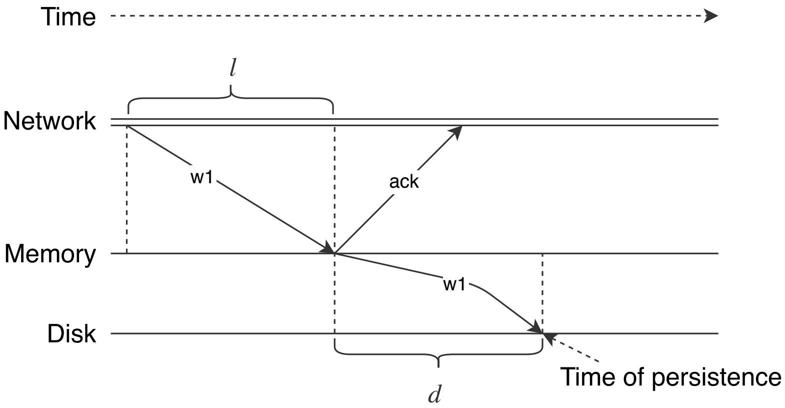
Estimating when Writes Become Durable

Big Picture

Write loss is more common the closer the write is to the failure.

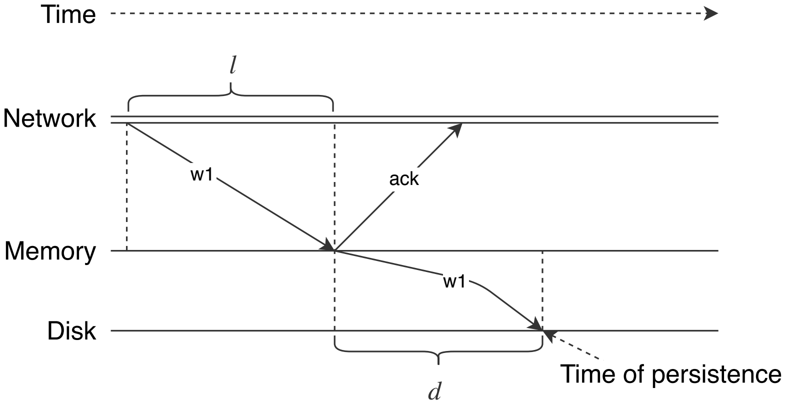
Can we estimate when our writes becomes durable, at least on the *Primary* replica?

When does a write become durable?



A write becomes *durable* when it gets *persisted to disk*.
In other words $l + d$.

Problem



We can't measure d.

What can we measure?

- Latency / (ping)
- Response time of write operations...

What can we measure?

- Latency / (ping)
- Response time of write operations... With different **write concerns**

Estimating d from Write Concerns

Which **write concern** persists a write to *disk*?

Estimating *d* from Write Concerns

Which **write concern** persists a write to *disk*?

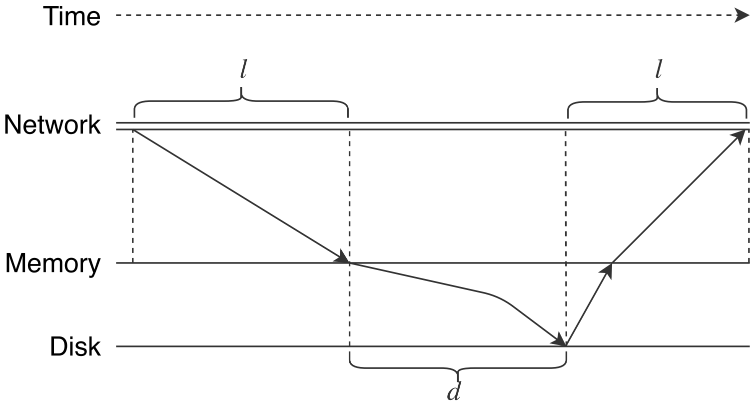
Journaled!

Journaled Write Concern

A **Journaled** write will be *applied to memory* and *added to the journal* before being acknowledged.

Since the journal does not always hit the disk, we have only an *approximation*.

Modelling Journaled Write Concern



Behaviour of a **Journaled** write.

Modelling Journaled Write Concern

That means, a **Journaled** write is:

$$j = l + d_{est} + l$$

Estimating durability

We want to estimate $t = l + d$ (time to durability).

We know l and $j = l + d_{est} + l$.

We then define the **estimate** as $t_{est} = j - l$:

$$\begin{aligned} t_{est} &= j - l \\ &= l + d_{est} \\ &\approx l + d \end{aligned}$$

Results - latency distribution

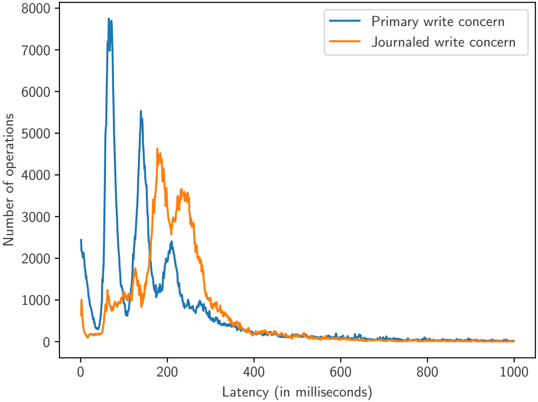


Figure: A frequency graph of the number of write operations acknowledged at latencies of 1-1000ms.

Results - cumulative distribution

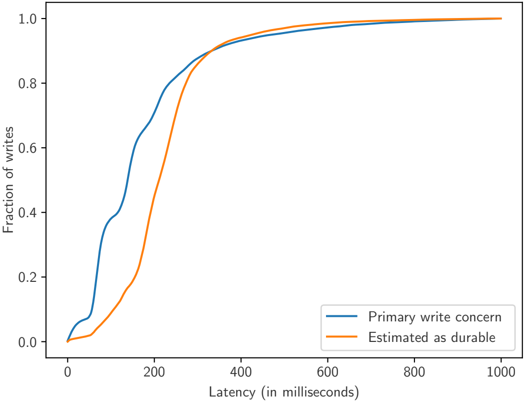


Figure: A cumulative frequency graph of the writes which become acknowledged and 1-durable at latencies 1-1000ms.

Results - Summary

We can *estimate* when a write becomes durable on the primary by looking *how long* a **Journaled** write takes to come back.

In our case, 90% of writes are durable by 300ms.

Conclusion

Thesis Findings

- Identified and categorised scenarios which cause writes to be lost.
- Created a tool capable of inducing write loss.
- Designed an algorithm to quantify write loss.
- Showed that our tool works by finding bugs in MongoDB.
- Derived a formula for when a write becomes durable on any number of replicas.
- Developed an estimation of when a write becomes durable on the *Primary*, using rudimentary client-accessible measurements.
- Found that 90% of writes are durable by 300ms after submission.

Limitations

- Induced only **one** failure per experiment, only on the Primary replica.
- Only used client-accessible measurements.
- Focused on durability only on the Primary.

Future Work

- Induce multiple failures, on any replica
- Investigate MongoDB's own logs
- Explore estimating durability on any number of replicas

Thesis Contributions

This presentation focused on a subset of our thesis contributions. Here is the complete list:

- Categorisation of *scenarios* that result in write loss.
- Design of an experiment capable of inducing write loss.
- Algorithm to quantify the number of lost writes.
- Empirical results to show that the experiment and algorithm work by detecting bugs in MongoDB 3.6-rc0.
- Theoretical model for evaluating when a write becomes durable.
- Estimation of when a write becomes durable on the *Primary*, using rudimentary client-accessible measurements.
- An empirical study of time-till-durability on the *Primary* for acknowledged writes.

Ramnatthan Alagappan, Aishwarya Ganesan, Yuvraj Patel, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Correlated crash vulnerabilities. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 151–167, Berkeley, CA, USA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL `http://dl.acm.org/citation.cfm?id=3026877.3026890`.

Eric A Brewer. Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 7—-, New York, NY, USA, 2000. ACM. ISBN 1-58113-183-6. doi: 10.1145/343477.343502. URL `http://doi.acm.org.ezproxy1.library.usyd.edu.au/10.1145/343477.343502`.

Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51, jun 2002. ISSN 01635700. doi: 10.1145/564585.564601. URL `http://portal.acm.org/citation.cfm?doid=564585.564601`.

References III

Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, dec 1983. ISSN 03600300. doi: 10.1145/289.291. URL `http://portal.acm.org/citation.cfm?doid=289.291`.

Kyle Kingsbury. Jepsen: Mongodb https://aphyr.com/posts/284-call-me-maybe-mongodb, 2013. URL `https://aphyr.com/posts/284-call-me-maybe-mongodb`.

Kyle Kingsbury. Mongodb 3.4.0-rc3 https://jepsen.io/analyses/mongodb-3-4-0-rc3, 2017. URL `https://jepsen.io/analyses/mongodb-3-4-0-rc3`.

References IV

Kit Patella. Mongodb 3.6.4 https://jepsen.io/analyses/mongodb-3-6-4, 2018. URL `https://jepsen.io/analyses/mongodb-3-6-4`.