COORDINATING DISTRIBUTED SYSTEM CONFIGURATION CHANGES WITH HUMMING CONSENSUS

Scott Lystig Fritchie, Basho Japan

PaPOC 2016, London

2016-04-18

町

Machi

“village” or “town”

Motivation

- Building a distributed, fault-tolerant blob/file store: Machi.
- Support eventual consistency (EC) … we are Basho.
- Support strong consistency (SC) … sometimes you want it.
    - Not both modes at the same time.
- Use the **same** configuration manager for EC & SC modes.

Motivation

- SC management system & framework smorgasbord!
    - ZooKeeper, etcd, Raft+framework, Paxos+framework, …
- **The availability of a distributed system is limited by the availability of its manager.**
- Failure of majority of nodes will cripple SC managers.
- We want EC Machi to be available even with 1 node alive.

Motivation

- EC managers are far less common.
- Riak Core is an obvious choice but has too many Riak-style assumptions for use by Machi.
  - Power-of-2 ring partitioning
  - Preference list calculation method

Managing System Configuration

```
dd if=/dev/random bs=4k \
    of=/etc/myapp.conf
```

Managing System Configuration

```
dd if=/dev/random bs=4k \
    of=/etc/myapp.conf
```

NO!

- Valid configurations are not random
- Config metadata can include:
  - Type of service (Riak, MySQL, HTTP reverse proxy)
  - Network use (IP addresses & ports, protocols spoken)
  - Static group membership (defined by sysadmin)
  - Dynamic group membership (defined by runtime behavior)

WHAT IS CHAIN REPLICATION?

Neil Conway
@neil_conway

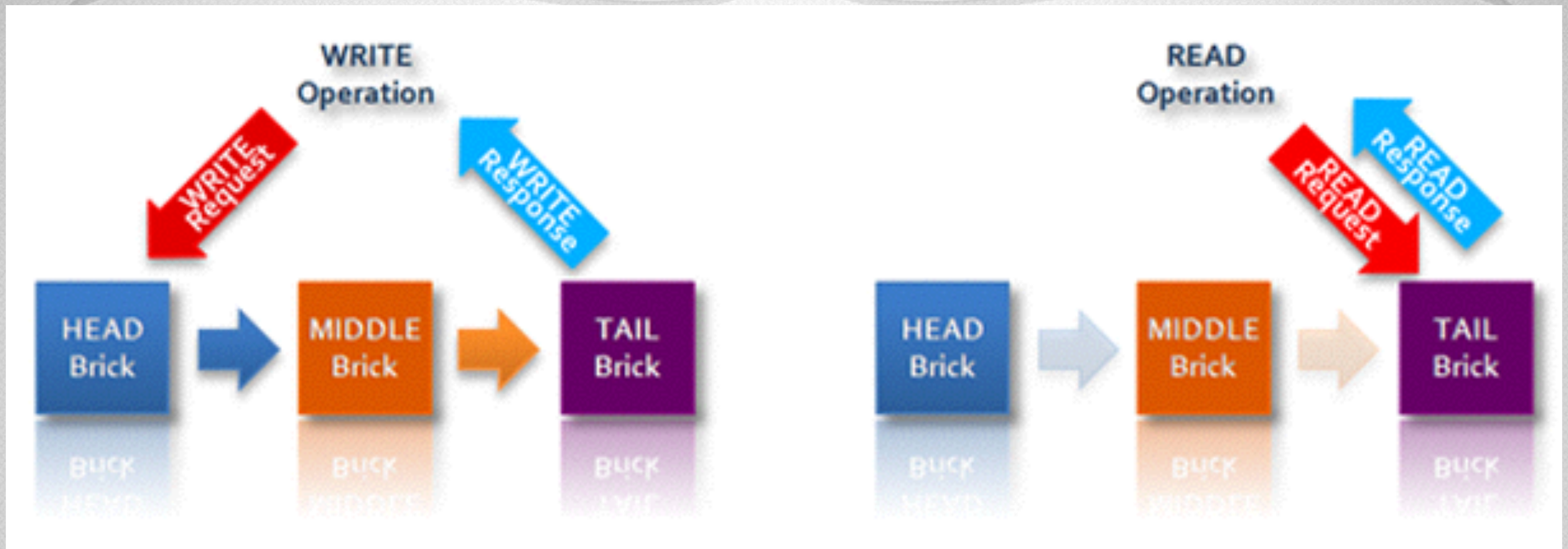
Following

Chain replication: strange that it is so well-known among academics and yet seemingly obscure to practitioners.

RETWEETS	FAVORITES
5	13

3:08 PM - 21 Oct 2015

Chain Replication On One Slide



- Variant of primary/secondary replication: strict chain order!
- Sequential read @ tail. Linearizable read @ all.
  Dirty read @ head or middle.

Managing Chain Replication

- **Screw up chain order -> screw up consistency**
- Today's managers assume SC only environments
  - What about Machi in EC mode?

Machi's Configuration Metadata

- Chain name
- Consistency mode: EC, SC
- Static membership: Servers permitted to replicate this chain
- Dynamic membership: Who's running? Who's dead?
- Chain order
- Coordinating chain repair
    - Data re-sync when server reboots/newly-added.

CONSENSUS AND HUMMING IN THE IETF

RFC 7282

To reinforce that we do not vote, we have also adopted the tradition of “humming”: When, for example, we have face-to-face meetings and the chair of the working group wants to get a “sense of the room”, instead of a show of hands, sometimes the chair will ask for each side to hum on a particular question, either “for” or “against”.

INSTEAD OF MEASURING HUMMING VOLUME, WHAT IF WE MEASURE PITCH?

Once Upon A Time, There Were Some Distributed Music Composers

INSTEAD OF MEASURING HUMMING VOLUME, WHAT IF MEASURED PITCH?

- I choose B-flat.
- I hum B-flat.
- I listen.
- I hear unison B-flat.
- **The answer is B-flat.**

INSTEAD OF MEASURING HUMMING VOLUME, WHAT IF MEASURED PITCH?

- I choose B-flat.
- I hum B-flat.
- I listen.
- I hear B-flat, D, and E: discord!
- **Not unanimous. Try again.**

What Could Go Wrong?

Our Model

- “Fail recovery”: crash & restart a finite number of times.
- Message omission permitted.
- Messages can be dropped or reordered.
- Message corruption is detectable via checksum verification.
- Failure detection is eventually accurate.
- No Byzantine misbehavior.
- Each participant is independent, uses same rules & invariants.

Epoch Register Store

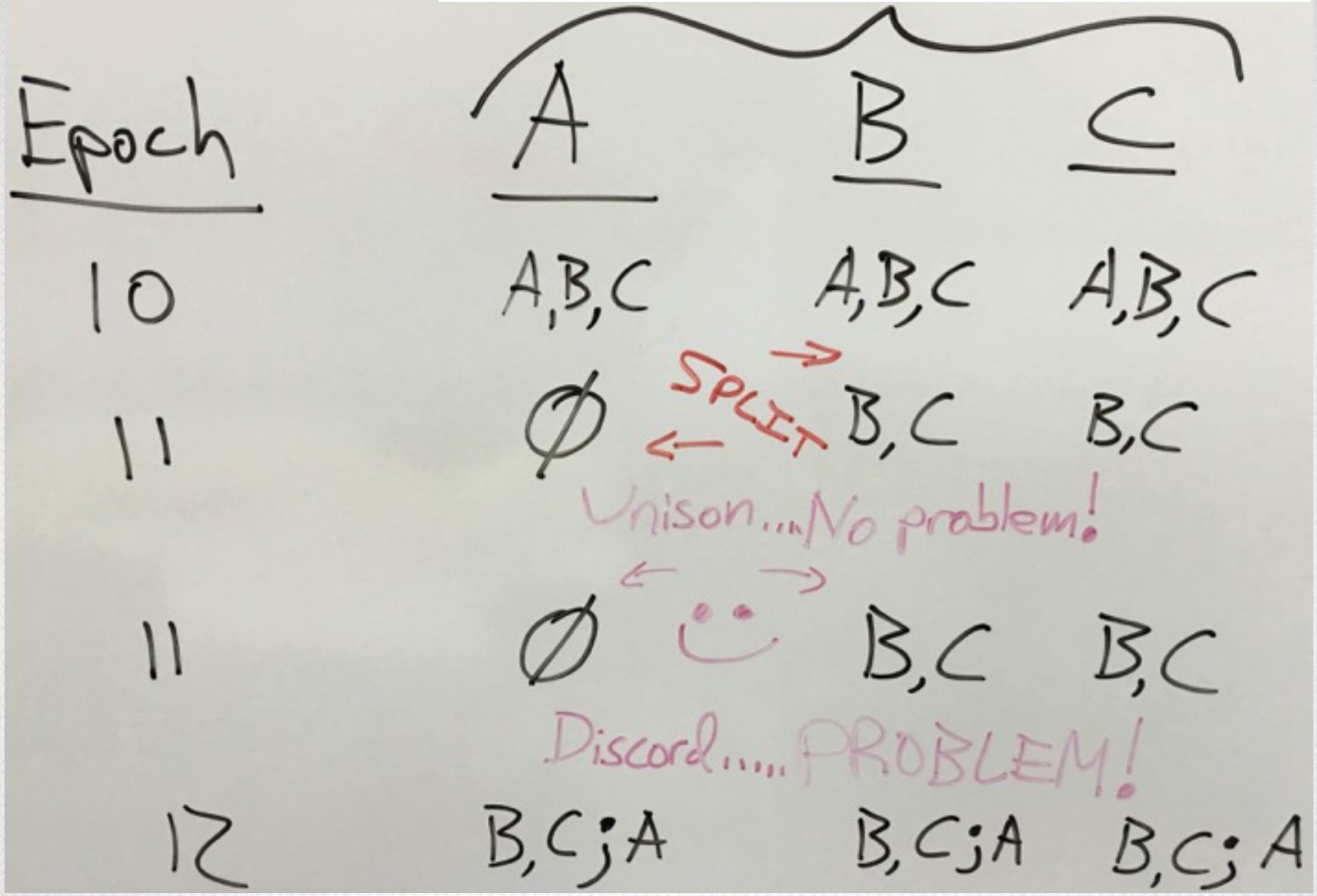
- Modeled as a map:
  - Key = epoch #
  - Value = write once register, blob of configuration (app-specific)
- Each participant has an epoch register store, accessible to all.
- All communication between HC participants is solely via the epoch register stores.

Humming Consensus On A Slide

1. Read config with largest epoch number from all available epoch register stores.

2. If minimum # of servers are available and all found copies of latest epoch # are unanimous/equal:
    1. If current config = latest config, **stop**.
    2. If transition current -> latest is safe, **use latest & stop**.
    3. **Else we ignore the latest epoch's value!**
3. Calculate a new config with new & bigger epoch number, blindly write it to all epoch register stores. Goto step 1.

Epoch Register Store



SC mode: No conflict at epoch 11 … until the net-split heals

TODAY’S STATUS

No Formal Proofs Yet

Today's Humming Consensus

- Fully implemented in Erlang
  - Works well in network partition simulator
  - **Property-based testing has been invaluable,** with & without using QuickCheck
- Hasn't seen The Real World yet!
- Source & docs: https://github.com/basho/machi

Network Partition Simulator

- Map: simulate uni-directional message drops between actors
  - Example: **A->B** drop messages but **B->A** is OK
- Partition map may change at random intervals
- Partition map may remain frozen/stable
- Asymmetric partitions cause more chatter & churn, but HC copes well enough today, still room for improvement.
- Today's practical size: 7 or 9 actors (livelock struggle)

HC’s biggest problem: flapping

- Bickering children: I’m right, you’re wrong, no compromise!
- Example: Assume that current chain order is [A,B,C].
  - Messages from A->B fail but all other combinations are ok
  - A believes that B is down, next config suggestion = [A,C]
  - B believes that A is down, next config suggestion = [B,C]
  - C believes nobody is down, next config suggestion = [A,B,C]

Detecting Flapping

- Very easy method … in hindsight.
- If I suggest the exact same config *R* times in a row, then I am flapping.
- *R*'s value set as a heuristic … 4 or 5 works well.

Mitigating Flapping

- Machi uses simple method: fall back to simplest safe chain
  - EC mode: chain of length 1: [Myself]
  - SC mode: chain of length 0: []
    - I.e., withdraw myself from service
- Existing repair & merge logic acts to fix the chain.
- Future improvement possible to reduce churn.

Insight In Hindsight

- It’s OK to ignore a configuration written to the epoch store!
    - Valid configuration state change space is small.
    - Independent actors can select a valid config transition.
    - If a configuration transition looks insane, then write another one.

Thank You!

Questions?

Eventual Consistency + C.R.

- **WAT?** Chain replication w/o strong consistency is crazy!
- Machi's file data is CRDT'ish: merge any write in any order
    - How? Write-once registers plus file namespace tricks
- CR's value to Machi
    - Cheaper than quorum replication: *f+1* to survive *f* failures
    - Entropy management: If server *X* fails, what is my risk of data loss?

Different Modes Of Operation

- Strong consistency: Chain length >= majority quorum size
  - CP mode minimum length prevents split brain syndrome.
  - **2f+1** servers to tolerate **f** failures.
- Eventual consistency: Chain length = 1 is OK!
  - Machi files are write-once registers at byte level, all Machi file ops are CRDT-like, always mergeable.
  - Humming Consensus can chain repair and chain merge after network partition.

Chain State Transition Invariants

- Strict separation: “in sync” prefix, “repairing/out of sync” suffix
- Never re-order “in sync” portion of chain
- Move “in sync” -> “down” at any time
- Move “down” -> “repairing” at any time
- Move “repairing” -> “in sync” only after repair effort is OK
- Move “repairing” -> “in sync” **only to end of in sync list**

Cheating The 2F+1 Chain Length

- Avoiding split brain: 2f+1 of “real” servers + “witness” servers
  - A, B, and C are real servers: humming consensus + file service; W1 & W2 are “witness servers” (humming consensus only + quick epoch number check on read/write)
- Zero real server failures: **A -> B -> C**, 5 of 5 in h.c., 3 real
- One real server failure: **W1 -> B -> C**, 4 of 5 in h.c., 2 real
- Two real server failures: **W1 -> W2 -> C**, 3 of 5 in h.c., 1 real

Corfu-Style Epoch Management

- All client ops tagged with current epoch # E
- If client op **E < E_current**, then **server refuses op**
- Any hosed client is OOS until newer epoch is found.
  - ... by reading from servers' private projection stores
- If client op **E > E_current**, then **server wedges self**
- Any wedged server is OOS until newer epoch is chosen
  - ... by humming consensus

REFERENCES AND CREDITS

For More Information

- Source code repo: https://github.com/basho/machi/
- Docs: https://github.com/basho/machi/tree/master/doc
- Chain replication and CORFU: section 11 of https://github.com/basho/machi/blob/95437c2f0b6ce2eec9824a44708217a266e880b6/doc/high-level-machi.pdf also, that paper's bibliography
- On Consensus and Humming in the IETF: https://www.ietf.org/rfc/rfc7282.txt
- NFS v2 RFC: https://www.ietf.org/rfc/rfc1149.txt
- Elastic Replication: https://www.cs.cornell.edu/projects/quicksilver/public_pdfs/er-socc.pdf
- The Part-time Parliament: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.2111&rank=1

For More Information

- HDFS: https://en.wikipedia.org/wiki/Apache_Hadoop#HDFS
- QFS: https://en.wikipedia.org/wiki/Quantcast_File_System
- WTF: http://arxiv.org/abs/1509.07821
  - Preprint of "The Design and Implementation of the Wave Transactional Filesystem"
- SeaweedFS: https://github.com/chrislusf/seaweedfs
- The original allegory: http://www.snookles.com/slf-blog/2015/03/01/on-humming-consensus-an-allegory/

Image Credits

- Composers: http://blog.mymusictheory.com/wp-content/uploads/2012/12/composers-mix-529x300.jpg
- Neil Conway: https://twitter.com/neil_conway/status/656713576422379520
- Mark Callaghan: https://twitter.com/markcallaghan/status/656810474365841410
- Chain replication diagram: https://github.com/hibari/hibari-doc
- Beethoven: https://upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Beethoven.jpg/399px-Beethoven.jpg
- Monty Python: http://images4.static-bluray.com/movies/covers/23375_front.jpg
- Under construction: https://github.com/h5bp/lazyweb-requests/issues/99
- Heinlein book+modification: Orb Books cover, 1997 (?)
- Scott's photo library