CouchDB 2.0

The awkward bits

Mike Wallace
29th September 2015

Hello!

Legal Disclaimer

<3 CouchDB

define: awkward

adjective

1. **causing difficulty; hard to do or deal with.**

    synonyms: difficult, tricky;

2. **causing or feeling uneasy embarrassment or inconvenience.**

    synonyms: embarrassing, uncomfortable, unpleasant, delicate, ticklish, tricky, sensitive, problematic, problematical, troublesome, perplexing, thorny, vexatious;

Source: https://www.google.co.uk/?q=define%3Aawkward#safe=active&q=define:awkward

All Software is Awkward

(in at least some aspects)

We know this because it pages us at 3am to tell us

But we deal with it.
And we learn.
And we become better operators.

But why do we have to wait until things go wrong?

What if product descriptions were written for system operators?

What if, instead of…

“Elasticsearch clusters are resilient — they will detect new or failed nodes, and reorganize and rebalance data automatically, to ensure that your data is safe and accessible.”

Source: https://www.elastic.co/products/elasticsearch

We saw something like…

“Elasticsearch loses data even when the partitions are total. You don’t need a bridge node to make replicas diverge.”

Source: https://aphyr.com/posts/317-call-me-maybe-elasticsearch

And where today we see…

“Enhanced features ensure data is always available and durable throughout any situation.”

Source: https://cloudant.com/

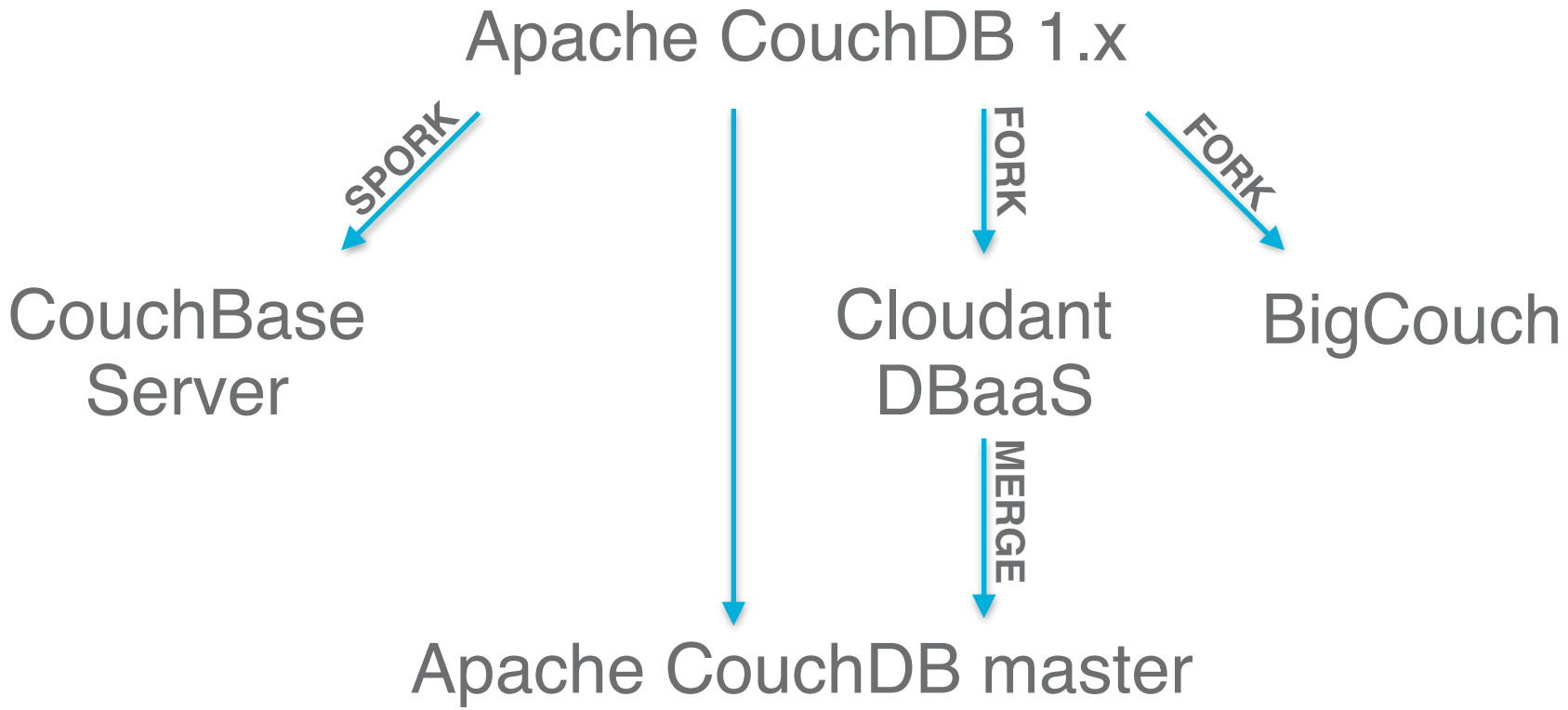
We saw…

“Unbounded conflicting revisions of a document can increase write latency for all documents on a shard until those conflicts are resolved.”

Source: Me, just now

Talking about the limitations of your system is a Good Thing™

A brief history of CouchDB


Apache CouchDB 1.x
SPORK
FORK
FORK
CouchBase Server
Cloudant DBaaS
BigCouch
MERGE
Apache CouchDB master

Things I had to leave out
(because they’re no longer awkward)

Cluster setup

Used to require manual creation of shard maps as JSON documents

Error prone and time consuming

Fixed:
https://github.com/apache/couchdb-setup

Compaction

Operations used to take all the available IO

Interactive requests couldn’t compete, resulting in significantly increased latencies

Fixed:
https://github.com/apache/couchdb-ioq

Awkward bit #1

Compaction

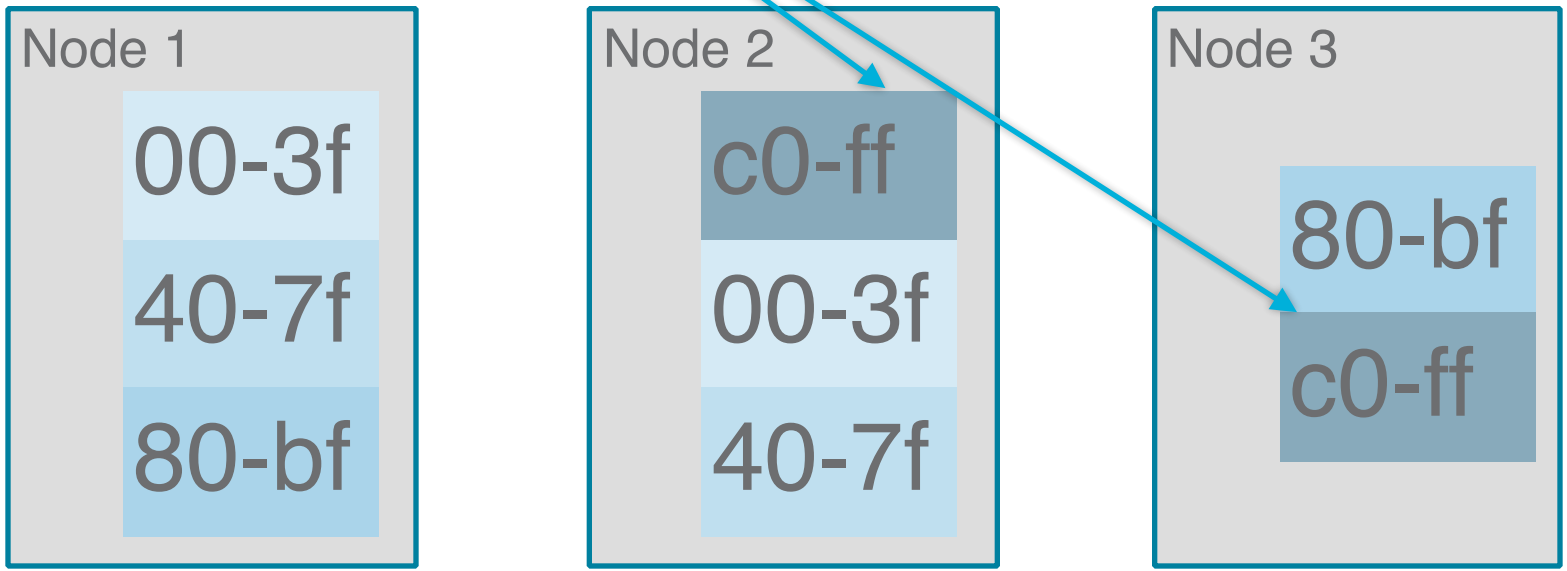
Didn’t you just say compaction wasn’t awkward any more?

Compaction is currently a shard operation, not a DB operation

A shard is a CouchDB database

{“_id”:”foo”}

hash(“foo”) = c4



Compaction at the cluster level requires co-ordinating logic.

This is not yet written (a partial implementation exists).

Options:

Write a script to compact the shards for each DB.

Use the compaction daemon and let each node handle its own compaction.

“Compaction cannot be triggered for a whole database. You need to either script the compaction of each shard for a DB, or let each node run the compaction daemon.”

Source: Me, just now

Talking about the limitations of your system is a Good Thing™

Awkward bit #2

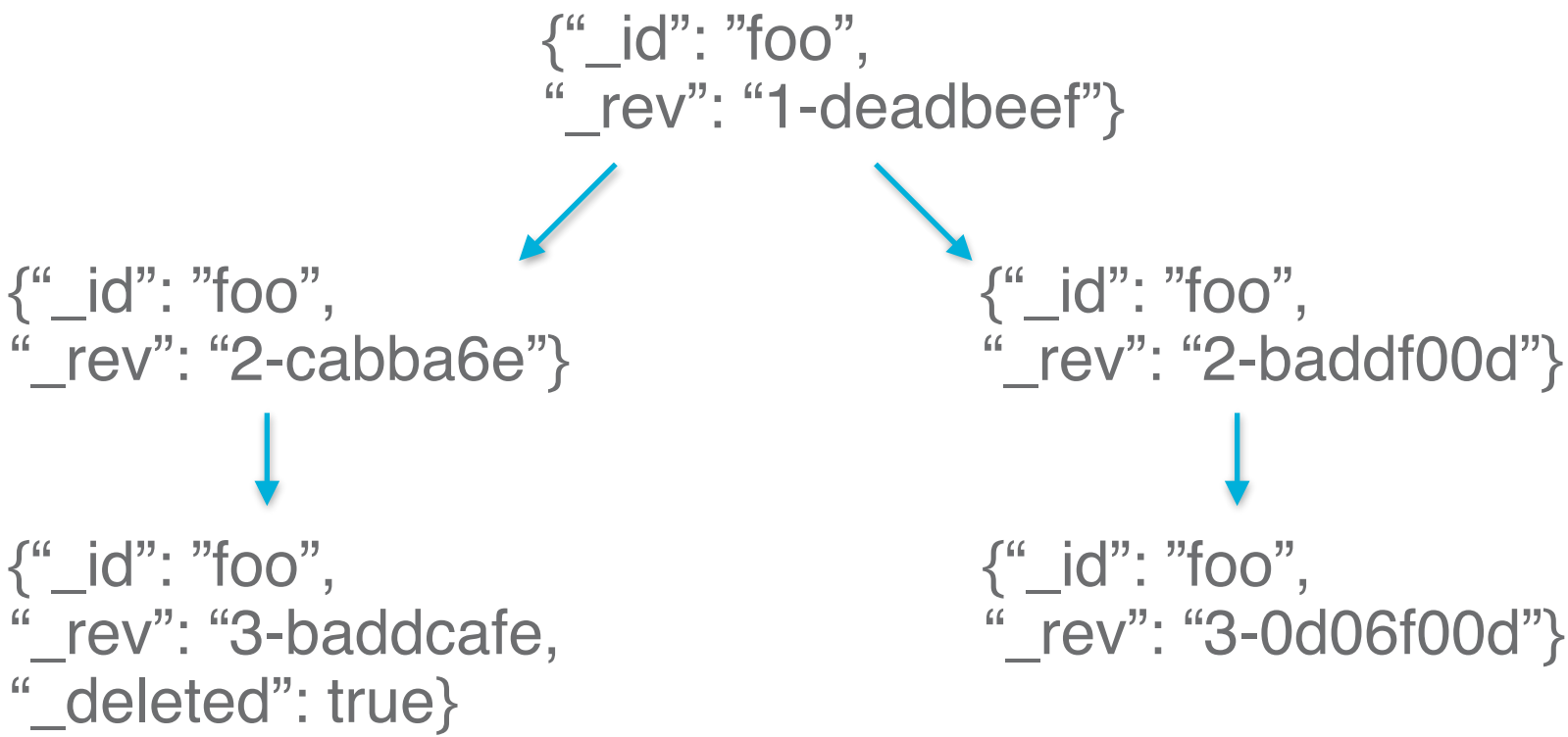
Purging

Purging is currently a shard operation, not a DB operation.

So is it just a case of figuring out which shards to purge?

No.

A document is a tree of revisions


{“_id”: ”foo”,
“_rev”: “1-deadbeef”}
{“_id”: ”foo”,
“_rev”: “2-cabba6e”}
{“_id”: ”foo”,
“_rev”: “2-baddf00d”}
{“_id”: ”foo”,
“_rev”: “3-baddcafe,
“_deleted”: true}
{“_id”: ”foo”,
“_rev”: “3-0d06f00d”}

Purging removes branches from your revision tree.

This completely breaks MVCC and eventual consistency.

Any ongoing replications cannot resolve that inconsistency.

The *exact* same set of revisions *must* be *successfully* purged on each shard copy.

Guaranteeing this in the presence of network partitions and partial failure is hard.

So it is deferred to the operator.

“Purging cannot be triggered for a whole database. Purging at the shard level can break eventual consistency. Replication to a new DB with a validation function to block unwanted revisions is preferable.”

Source: Me, just now

Talking about the limitations of your system is a Good Thing™

Awkward bit #3

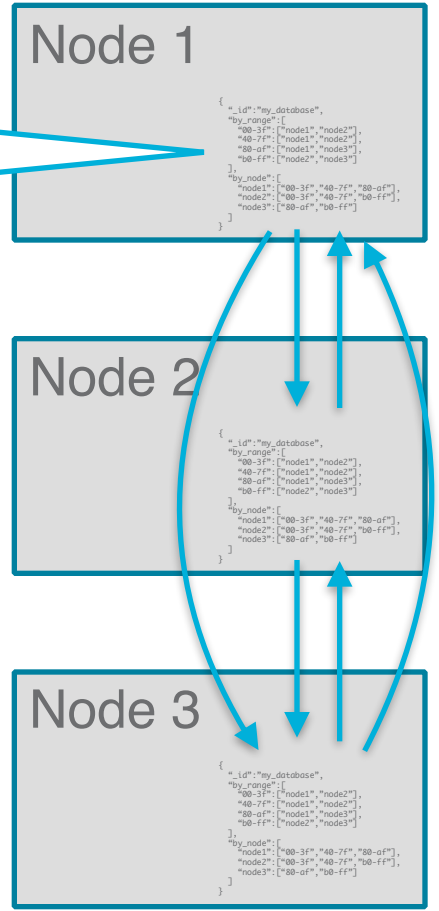
DB creation

Database creation requires two operations for each shard

Create shard map

Create shard files

```
{
  “_id”:”my_database”,
  “by_range”:[
    “00-3f”:[”node1”,”node2”],
    “40-7f”:[”node1”,”node2”],
    “80-af”:[”node1”,”node3”],
    “b0-ff”:[”node2”,”node3”]
  ],
  “by_node”:[
    “node1”:[“00-3f”,”40-7f”,”80-af”],
    “node2”:[“00-3f”,”40-7f”,”b0-ff”],
    “node3”:[“80-af”,”b0-ff”]
  ]
}
```



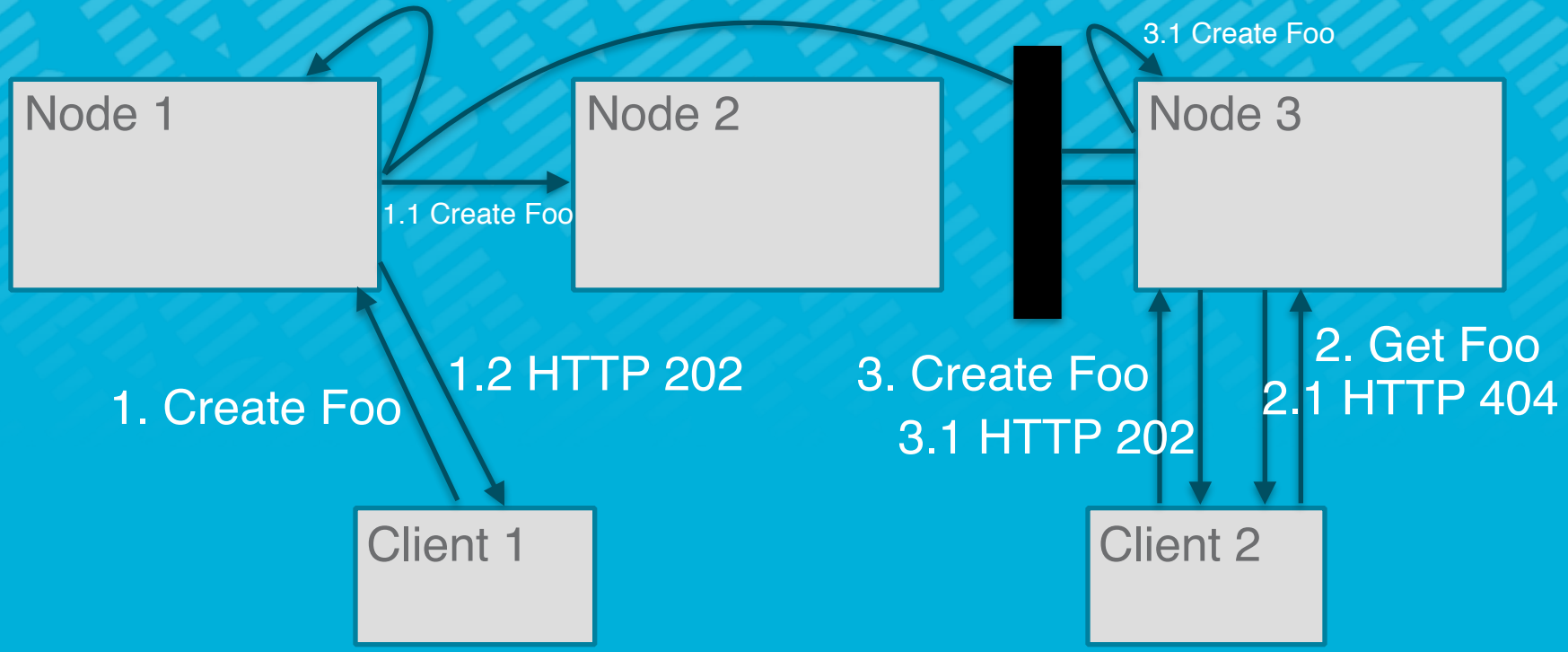
The co-ordinating node will wait for responses from all workers before returning a response.

But the operation is not atomic, so if there is a partial failure we don’t roll anything back.

What happens if one node is down?

HTTP 202 (accepted)

What about network partitions?



What happens when this partition heals?

One shard map becomes the “winner” and the live DB. The data on the other shards disappears.

Fortunately, the shards still exist on disk, so you can replicate the data to the live shards.

What if all nodes are up but one node is not responding?

An error is returned after a timeout.

But… the DB is still created on the responding nodes.

There are things we could do here…

Make database creation a CP operation?

Be more flexible in binding data to nodes?

Do away with static shard maps?

“Be wary when creating databases in the event of a network partition and treat HTTP 500 responses with suspicion. Try and avoid creating databases regularly (not always possible).”

Source: Me, just now

Talking about the limitations of your system is a Good Thing™

Awkward bit #4

Impaired nodes

Fault tolerance dynamo-style

N = number of replicas
Q = number of ranges
W = write quorum
R = read quorum

$$W + R > N$$

Popular values: N=3, W=2, R=2

These are sometimes referred to as “tuneable consistency”.

This is untrue.

The only consistency is eventual.

Scenario: Three node cluster, one node is down.

Q: You make a write with W=3. What happens?

Scenario: Three node cluster, one node is down.

A: The write lands on two nodes and you get an HTTP 202 response.

Scenario: Three node cluster, one node is down.

Q: You make a read with R=3. What happens?

Scenario: Three node cluster, one node is down.

A: The read gets a response from two nodes and an HTTP 200 response is returned.

Scenario: Three node cluster, one node is impaired; it reports itself as up but RPC calls are timing out.

Q: You make a write with W=3. What happens?

Scenario: Three node cluster, one node is impaired; it reports itself as up but RPC calls are timing out.

A: The co-ordinating node gets two acks and waits the duration of fabric/request_timeout (default 60 seconds) before giving up and returning an HTTP 202 accepted.

Scenario: Three node cluster, one node is impaired; it reports itself as up but RPC calls are timing out.

Q: You make a read with R=3. What happens?

Scenario: Three node cluster, one node is impaired; it reports itself as up but RPC calls are timing out.

A: The co-ordinating node gets two results and waits for the duration of fabric/request_timeout and then... throws the two values away and returns an HTTP 500 internal server error.

Other problems

We keep hammering unresponsive nodes with even more requests.

We can’t vote a node out of the cluster. As long as it can report itself as up, it is considered up.

IMO we can improve things here:

Nodes could monitor the health of other nodes.
Nodes could vote out unresponsive nodes.
Nodes could back off if requests to another node aren't being serviced.

Getting these things right is challenging.

“R=N or W=N doesn’t really buy you anything but it does reduce your fault tolerance.”

Source: Me, just now

“An impaired cluster node can increase request latency by several orders of magnitude for certain requests. Use low W and R values and monitor request latency obsessively.”

Source: Me, just now

Talking about the limitations of your system is a Good Thing™

Conclusions

Compaction can be per node or per shard, but not per DB.

Purging is dangerous and should be avoided at all costs.

Be wary about DB creation during partitions and be suspicious of HTTP 500 responses.

Impaired nodes can really harm your system. Monitor, monitor and then monitor.

You can’t have strong consistency in an eventually-consistent data store, no matter how much you might want it.

There are some really interesting places where *you* could contribute to CouchDB.

Talking about the limitations of your system is a Good Thing™

Don’t forget…

A system is more than the sum of its awkward bits.

The code which adds the dynamo clustering to CouchDB has been running in production for five years.

It's backing thousands of business applications right now.

They're all meeting their SLAs (well, hopefully).

And finally…

Go and try CouchDB 2.0 (developer preview)!

https://git-wip-us.apache.org/repos/asf?p=couchdb.git

https://github.com/apache/couchdb.git

./configure && make && dev/run

Thanks for staying with me!