Rearchitecting Kubernetes for the Edge
And the DC too!

Andrew Jeffery
andrew.jeffery@cl.cam.ac.uk

University of Cambridge

Thursday 11th March, 2021
About Me

- First year PhD student under Prof. Mortier
- BA in Computer Science from University of Cambridge
- Previously worked at The Hut Group
Work in Progress
Currently under submission to EdgeSys ’21
One thing to take away

Orchestration should not require strong consistency!
What we’ll cover

- The problem of strong consistency in orchestration
- Some edge case studies
- How does strong consistency impact Kubernetes
- Analysing etcd at scale
- Using CRDTs for eventual consistency
- Implications on deployment architectures
- Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
Where are we up to?

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
What is the problem?

Our services running at the edge should be:

- Performant
- Available
- Scalable

But they aren’t there yet...
Why is it important?

Removes our ability to react!
- To failures
- To changes in demand
- To reconfigurations
Where are we up to?

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
What do we mean by the edge?

Compared to a datacenter the edge has:

- Lots more sites, each being closer to end users
- Higher network latencies between sites and internally
- Lower bandwidth between sites and internally
- Less reliable components
Actual deployment case studies

Scale of communication is always increasing. Technology enables faster communication with more devices and more dynamic content:

- Next generation(s) of connectivity: 5G
- More devices: IoT
- Dynamic content: Elastic CDNs
Where are we up to?

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
What is Kubernetes?

*Production-Grade Container Orchestration*¹

- Scaling
- Healing
- Routing
- Extensibility

59 percent of large organizations use Kubernetes in production²

---

¹https://kubernetes.io/
²https://tanzu.vmware.com/content/blog/why-large-organizations-trust-kubernetes
Kubernetes Architecture

- Custom controllers
- Replicaset controller
- Scheduler
- Worker node
- API server
- Etcd node 1 (leader)
- Etcd node 2
- Etcd node 3
Etcd

A distributed, reliable key-value store for the most critical data of a distributed system\(^3\)

- Critical data here is Kubernetes state
- Supports transactions on data
- Also has concept of watches: notifications of changes

\(^3\)https://etcd.io/
Scheduling with centralised state

- Custom controllers
- Replicaset controller
- Scheduler
- API server
- Etdc node 1 (leader)
- Etdc node 2
- Etdc node 3
- Container registry
- Worker node

- 1, 6, 11, *
- 4, 9, 15, *
- 5, 10, 16, *
- 5, 10, 16, *
Key limitation

Centralised, strongly consistent state

Key Takeaway

Orchestration should not require strong consistency
Where are we up to?

- The problem of strong consistency in orchestration
- Some edge case studies
- How does strong consistency impact Kubernetes
- Analysing etcd at scale
- Using CRDTs for eventual consistency
- Implications on deployment architectures
- Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
Results of Etcd: Request breakdown

<table>
<thead>
<tr>
<th>Request type</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1542</td>
<td>52.3</td>
</tr>
<tr>
<td>T xn Range</td>
<td>476</td>
<td>16.1</td>
</tr>
<tr>
<td>T xn Put</td>
<td>866</td>
<td>29.3</td>
</tr>
<tr>
<td>Watch create</td>
<td>67</td>
<td>2.3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>2951</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table:** Etcd request counts averaged over 10 runs. Series of creating a deployment of 3 containers, scaling to 10, scaling to 5 and then deleting. Range requests are all linearisable. Requests with negligible count are omitted.
Results of Etcd: Latency

Cluster node count

Median latency (ms)

- put
- range_l
Results of Etcd: Throughput

Cluster node count

Requests per second (thousands)

- put
- range_l
Centralised, strongly consistent state
This is all etcd

**Key Takeaway**

Orchestration should not require strong consistency
Where are we up to?

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
Key limitations revisited

Decentralised, *eventually* consistent state

Key Takeaway

Orchestration should not require strong consistency
Changing the datastore

- We don’t want to change all of Kubernetes, that would be too much work
- We also want to use a strong mechanism for eventual consistency: Conflict-Free Replicated Datatypes (CRDTs)
What gets stored?

```json
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
selector:
  matchLabels:
    tier: frontend
template:
  metadata:
    labels:
      tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx
```
How out of date will this get?

Assuming we don’t want our state to be stale...

But maybe we can use staleness to our advantage?
Can Kubernetes even handle stale state?

Distributed systems are dynamic environments and at best we can only model them.

So does Kubernetes already act on stale information?
Stateless state: Services

- DNS could resolve to IP address which matches a dead container
- So, this could already be an issue
Stale state: ReplicaSets

- Under failures we could schedule more, as well as less
- So, again, this could already be an issue in our distributed system
- With CRDTs we can control over-replication rather than under-replication on merge
Scheduling with the decentralised state

Saved at least 3 critical path network hops!
Where are we up to?

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
Current edge architectures: Split DC and Edge

- etcd
- control-plane

Cloud

- worker nodes

Edge
Current edge architectures: All Edge

Cloud

etcdf
control-plane
worker nodes
Edge
Rearchitect with eventual consistency?

Our new datastore gives us more flexibility

- We can now scale across multiple edge sites
- We can run on slower links and still be efficient
- We can use fewer resources and maintain the same fault tolerance
Where are we up to?

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
What if we look back to the cloud?

Not everything will be running at the edge so maybe we can influence cloud deployments too?
Spreading out for availability
What we’ve covered

The problem of strong consistency in orchestration

Some edge case studies

How does strong consistency impact Kubernetes

Analysing etcd at scale

Using CRDTs for eventual consistency

Implications on deployment architectures

Looking back at the cloud, have we missed something?

Key Takeaway

Orchestration should not require strong consistency
Conclusion

Orchestration should not require strong consistency

By adhering to this we can

- Improve performance of our clusters and make them more reactive
- Make them more tolerant to failure
- Scale them across more sites without fear

Andrew Jeffery
andrew.jeffery@cl.cam.ac.uk
Bonus: Federation

- Rather than looking inside a cluster we can look between
- No room, run jobs in other cluster instead
- Kubernetes still wants strong consistency on this!
- Some researchers have a similar idea of using eventual consistency instead
Bonus: New system

- We could just create a new system from scratch
- But why waste the effort, adoption is hard enough for production systems
- Kubernetes is the industry standard now, easier to try things out and then maybe we can build afresh
Bonus: Adaptive consistency

- A useful option if some parts of state require more consistency
- Could also use etcd for less critical state and eventual consistency for high traffic items