Cloud-Hydra: A Cloud Native Multi-Cloud Defensive Load Balancing Framework

Josh Stern, Rachid Tak Tak, Julian Trinh, Filip Vukelic

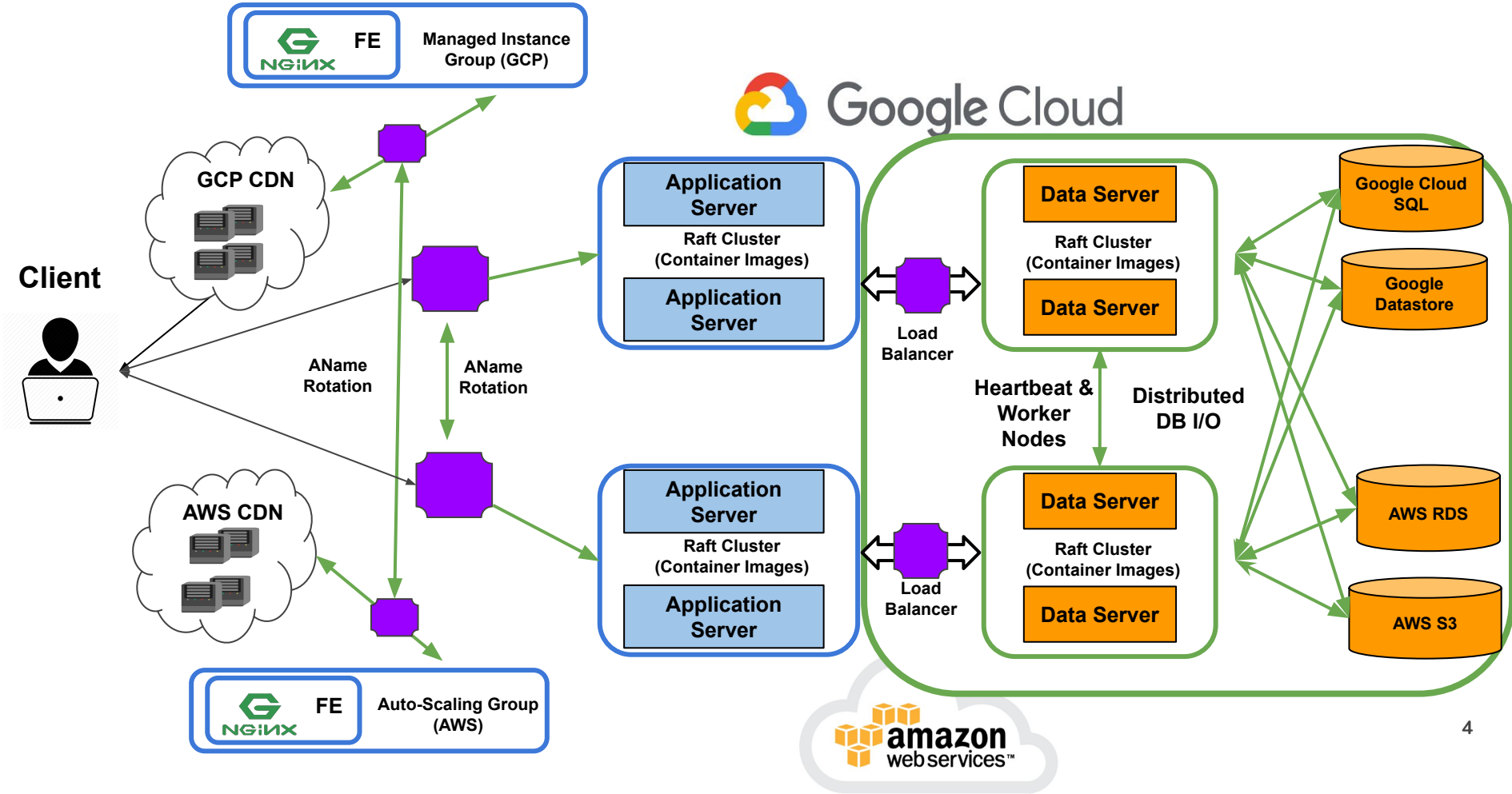
SPRINT 5

Sprint History

- **Sprint 1- Built fullstack web app in GCP and AWS**
  - Setup GCP and AWS infrastructure
- **Sprint 2 - Unified Data Layer**
  - Raft consensus
  - Cross-cloud consensus, single cloud load balancing
- **Sprint 3 - Cross cloud forwarding and recovery**
  - Leader Forwarding
  - Data layer Load Balancing
  - Database Recovery (single cloud)
- **Sprint 4 - Stitching together both clouds**
  - Synchronize cross cloud infra
  - Multi-Cloud DB Recovery
  - Stateless application layer

Goals of Sprint 5

- DNS load balancing
  - Front end
  - Application layer
- End-to-end testing of the entire system via UI
- Wallet Crushing: benchmarking throughput and latency


FE
Managed Instance Group (GCP)
Google Cloud
GCP CDN
Client
AName Rotation
AName Rotation
AWS CDN
Application Server
Raft Cluster (Container Images)
Application Server
Application Server
Raft Cluster (Container Images)
Application Server
Load Balancer
Load Balancer
Data Server
Raft Cluster (Container Images)
Data Server
Heartbeat & Worker Nodes
Distributed DB I/O
Data Server
Raft Cluster (Container Images)
Data Server
Google Cloud SQL
Google Datastore
AWS RDS
AWS S3
FE
Auto-Scaling Group (AWS)
amazon web services

End-to-end Interconnection

- Remaking our infrastructure to new GCP and AWS emails and accounts
- Front end DNS ANAME rotation
- Since application layer is stateless, also applied DNS ANAME rotation
- Fully interconnecting the entire infrastructure

Cloud Hardware Specs

- GCP load balancer
  - 2 vCPUs (2GHz)
  - 7.5 GB RAM
- GCP Raft Nodes
  - 1 vCPU (2GHz)
  - 3 GB RAM
- GCP SQL
  - 1 vCPU
  - 3.75 GB

- AWS load balancer
  - 1 vCPU
  - 1 GB RAM
- AWS Raft Nodes
  - 1 vCPU
  - 1 GB RAM
- AWS RDS
  - 1 vCPU
  - 1 GB

Test Suite

- Script sends 200 requests in a loop
  - send next request only after previous one receives response
- Read payload 65 Bytes
- Write payload 36 Bytes

Latency - Hitting Multi-Cloud DBs vs Single

Latency(ms)	AWS	GCP	Both
Reads	47.03	39.29	44.41
Writes	73.65	68.19	76.92

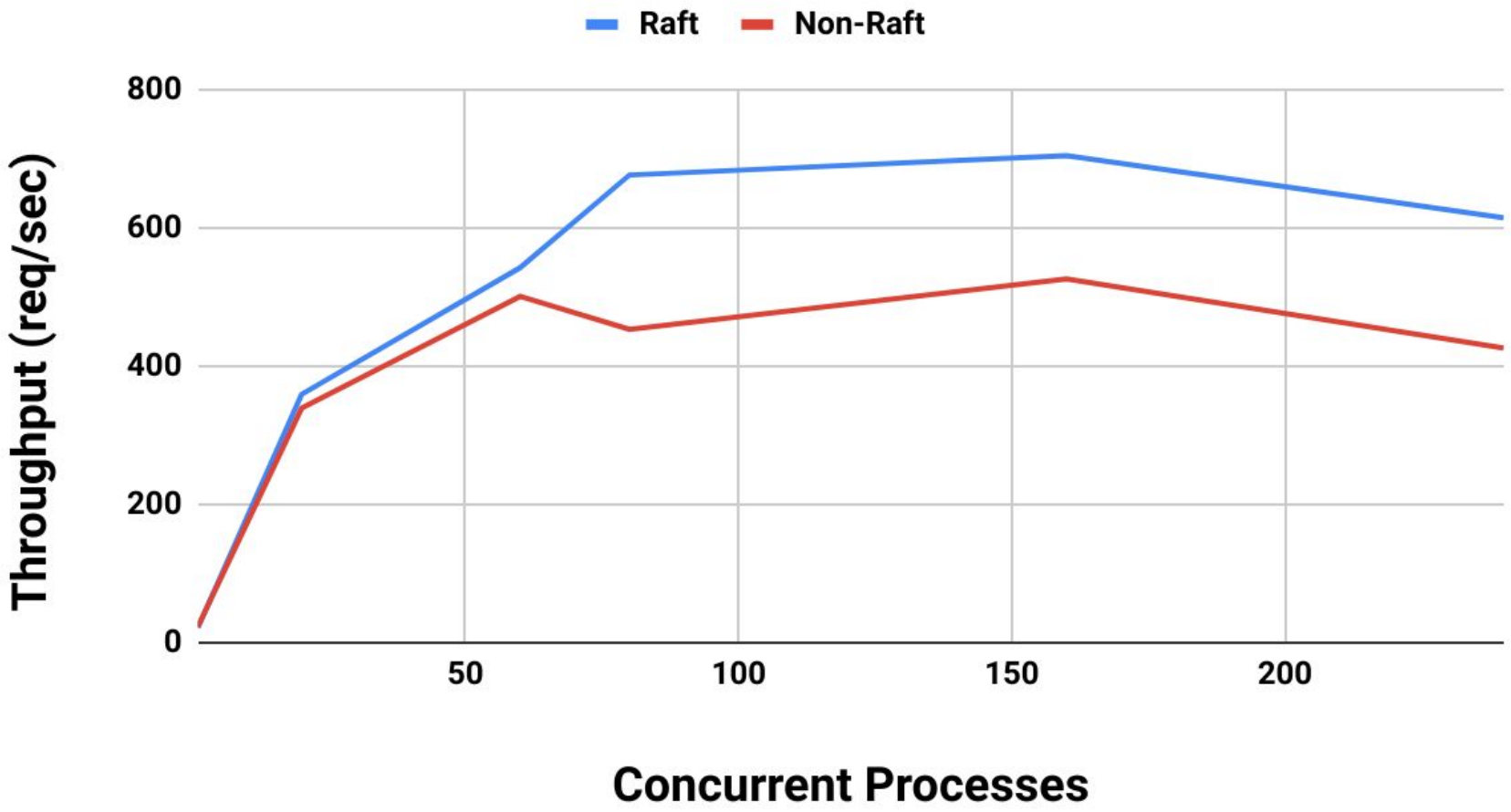
Read Throughput vs Number of Processes

Number of Concurrent Processes	Throughput (Requests / sec)	Average Latency (ms)
1	21.63	46.21
20	359	51.56
60	542	121.065
80	676	190.36
160	704	224.56
240	614	406.46

Read Throughput vs Number of Processes (no Raft Cluster)

Number of Concurrent Processes	Throughput (Requests / sec)	Average Latency (ms)
1	22.86	45.29
20	338.8	56.05
60	500.52	118.32
80	452.8	176.63
160	526	303.36
240	426	503.98

Reads: Raft and Non-Raft



Read Throughput vs Number of Mixed Processes

Number of Concurrent Processes	Throughput (Requests / sec)	Average Latency (ms)
30 Reads / 30 Writes	697.5	43.02
40 Reads / 40 Writes	714.28	56.05
80 Reads / 80 Writes	526.9	241.63
120 Reads / 120 Writes	276.38	457.07

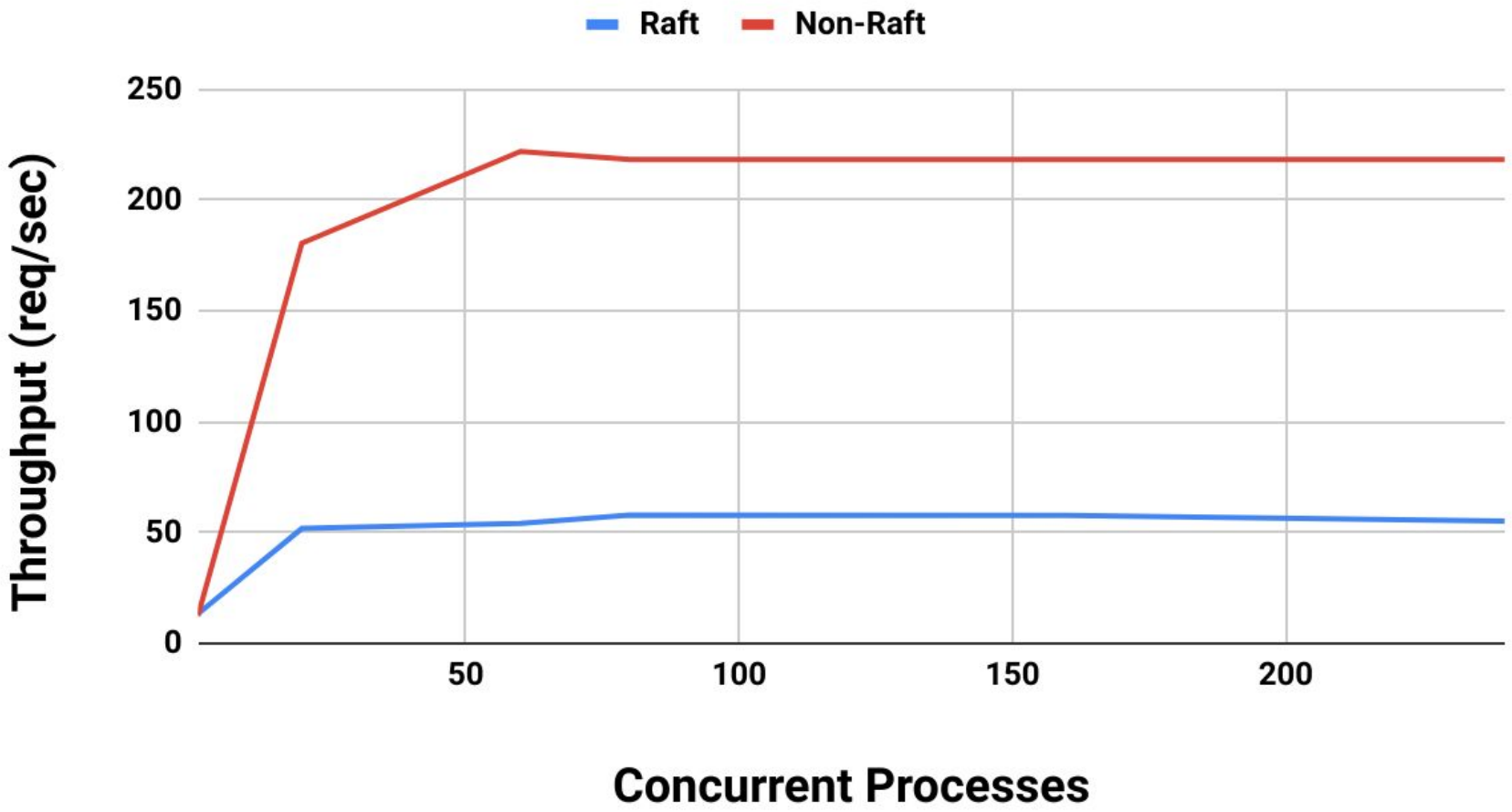
Write Throughput vs Number of Processes

Number of Concurrent Processes	Throughput (Requests / sec)	Average Latency (ms)
1	13.3	75.19
20	51.83	389
60	55.2	1087.36
80	57.92	1382.42
160	57.78	2768.75
240	55.2	4231

Write Throughput vs Number of Processes (no Raft Cluster)

Number of Concurrent Processes	Throughput (Requests / sec)	Average Latency (ms)
1	12.37	80.83
20	180.62	110.72
60	222	269.69
80	218.4	365.74
160	56% dropped requests	N/A
240	46% dropped requests	N/A

Writes: Raft and Non-Raft



Write Throughput vs Number of Mixed Processes

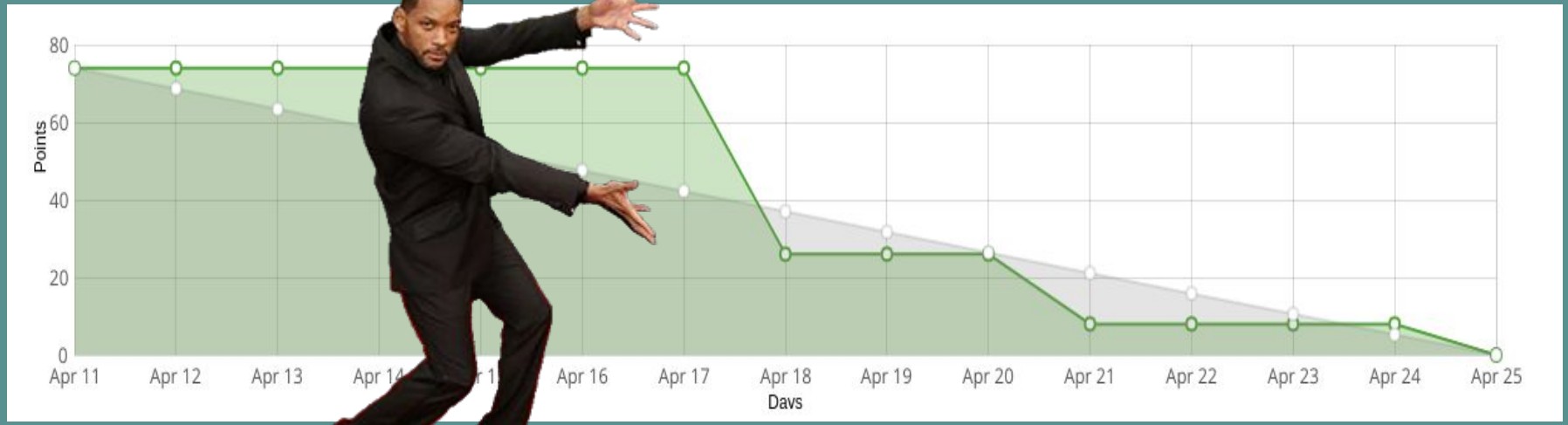
Number of Concurrent Processes	Throughput (Requests / sec)	Average Latency (ms)
30 Reads / 30 Writes	57.6	537.28
40 Reads / 40 Writes	54	739.86
80 Reads / 80 Writes	59.2	1397.14
120 Reads / 120 Writes	56.29	2318.2

Analysis

- Reads benefit from being in a load-balanced cluster
    - Requests can be shared across multiple-machines
    - Latency of the cluster was worse until ~160 concurrent connections
- Writes within a public cloud domain (Google -> Google) are faster than cross cloud due to locality
- Consensus and cluster-level locking reduce peak write throughput by 3.5x
    - Raft log is unbounded
    - The actual writes to the DB are small
    - Sensitive to network

DEMO TIME!

Sprint 5 Burndown



Challenges

- Needed to re-create entire infrastructure
- Throughput tests are highly network dependent
- Cached DNS-IP mapping

Future Optimizations

- Writes are slow because the leader has to lock the raft log for performing consensus and then db writes
  - Help?
- Bound the Raft log
  - Do we need to retain a log if both DBs are up?
  - Delete Raft entries after a certain size
- Use a more performant consensus library or algorithm
  - Ectd Raft (used by Kubernetes)
  - Leaderless Paxos

Thank you!
Questions?