



Our System in a Nutshell

Distributed KV Store for recipes

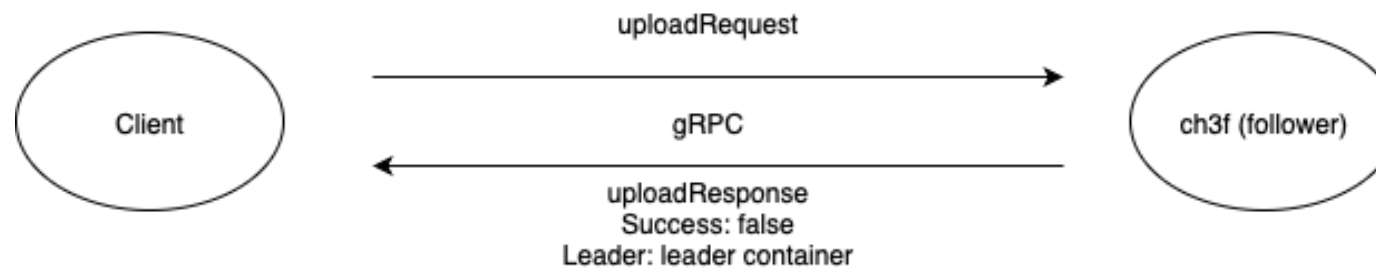
Clients can upload or download recipes

Using leader election, all uploads are managed by the leader

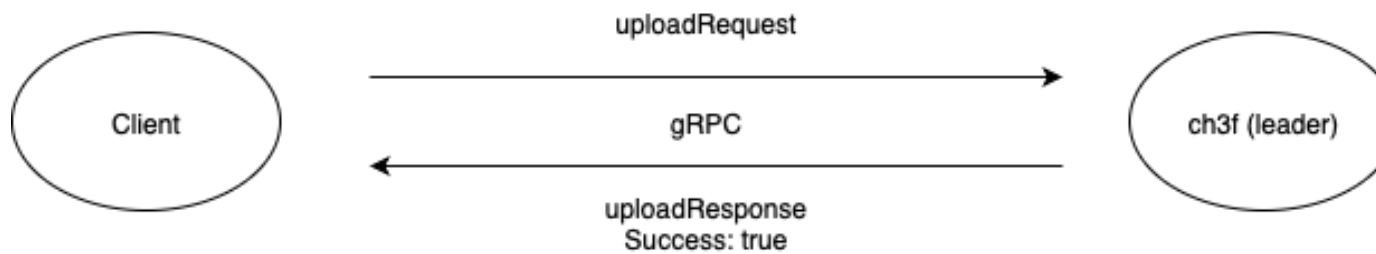
Downloading is possible on all nodes, including followers

Uploading a Recipe

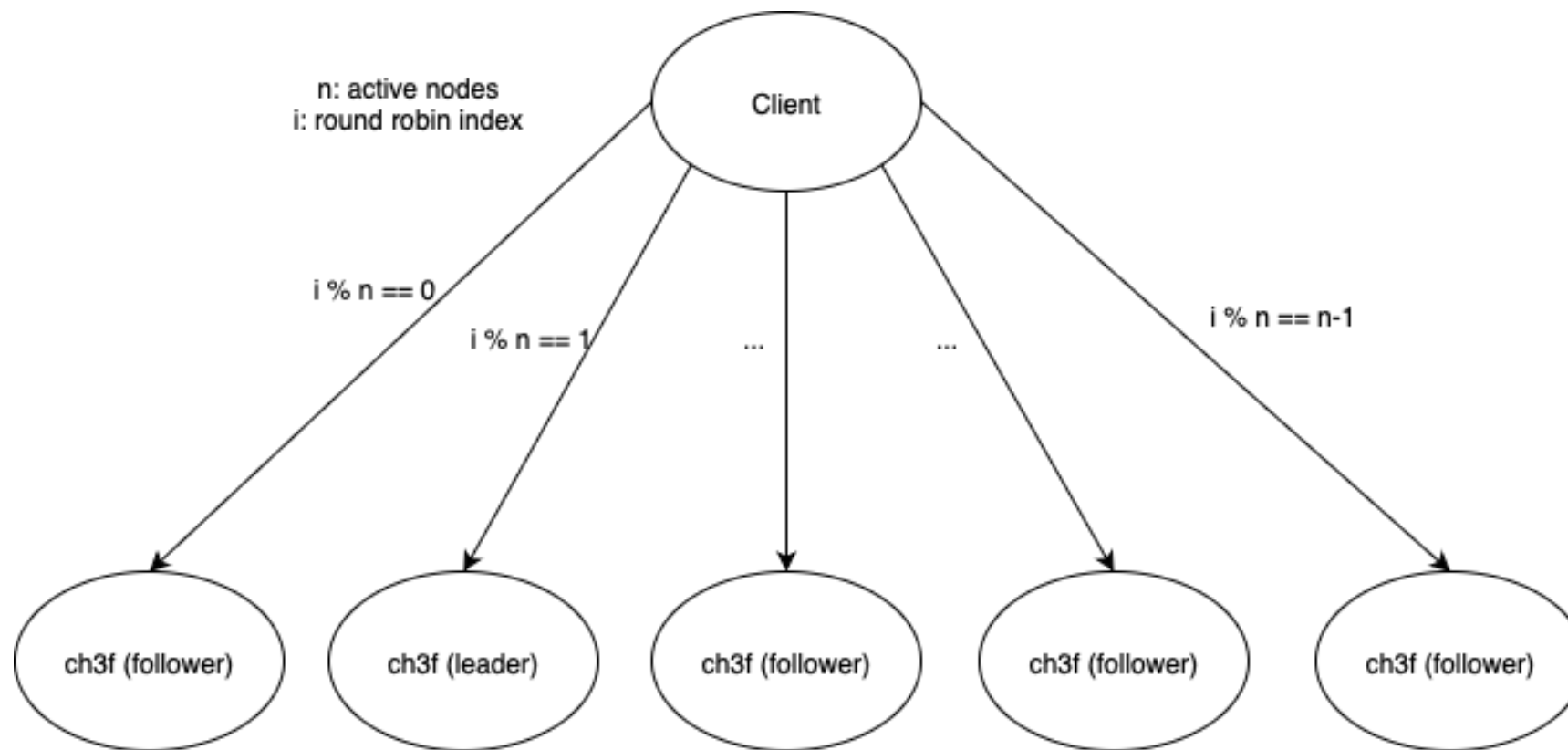
Request a random node (round robin), not the leader in this case:



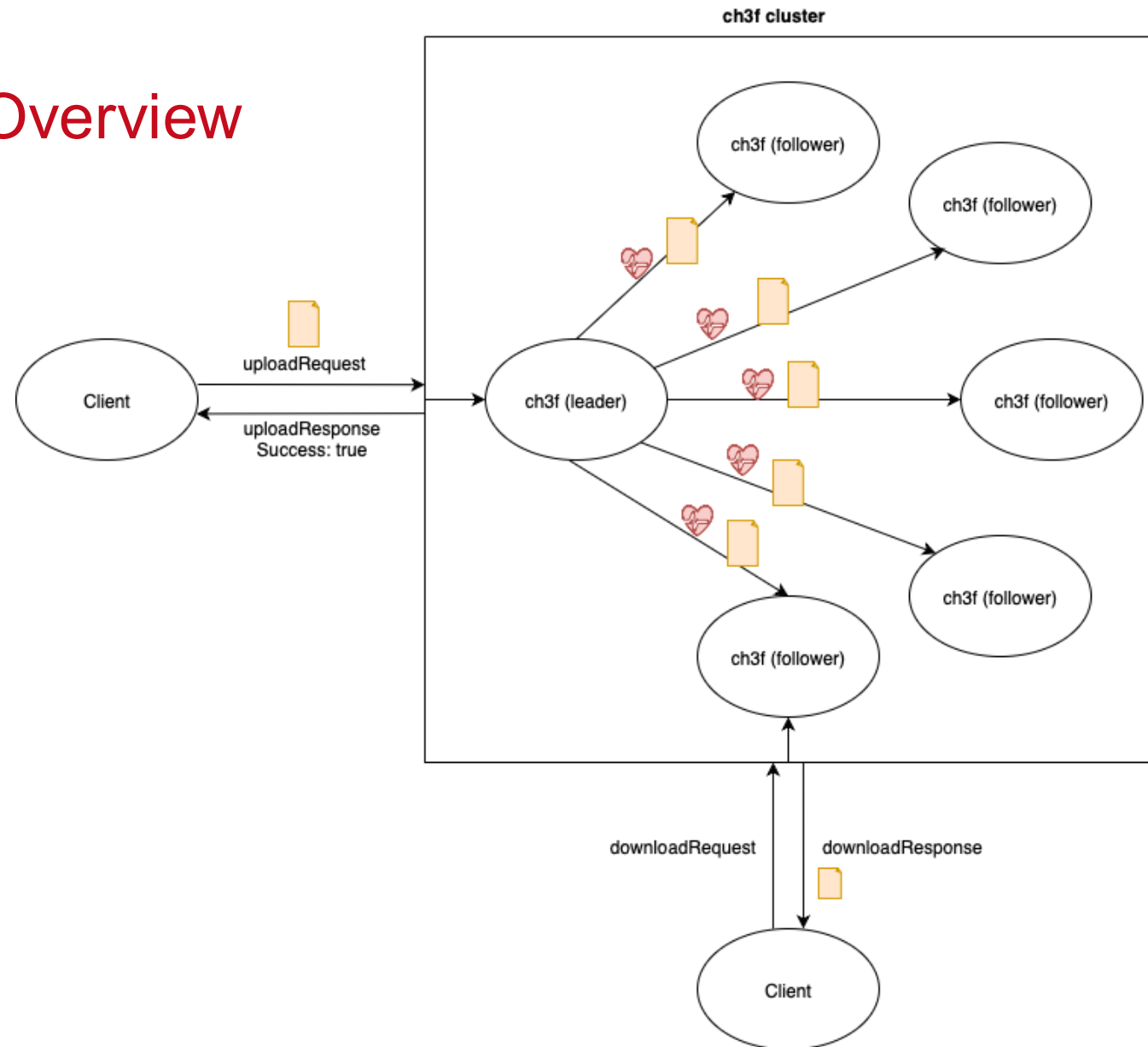
Retry to the leader:



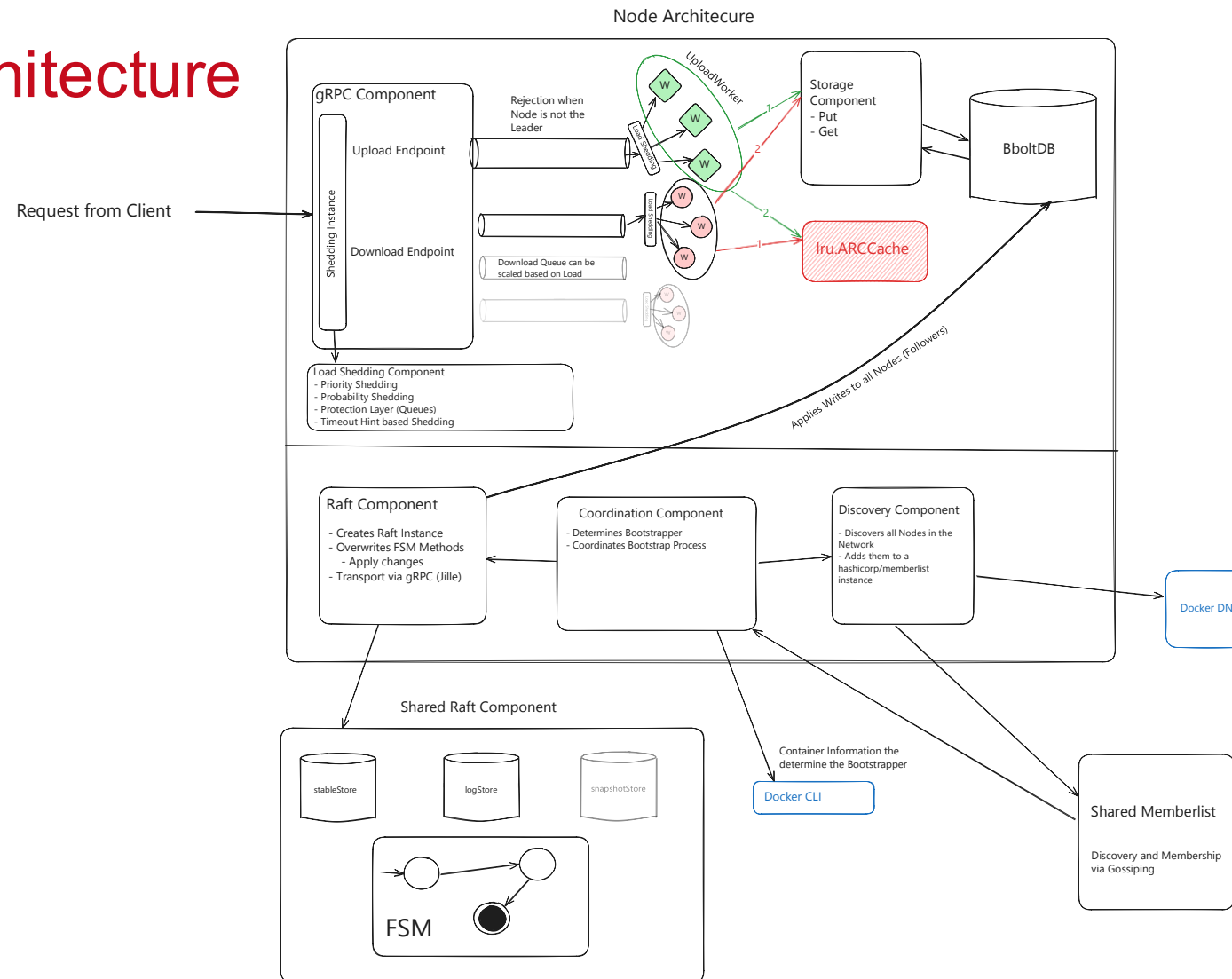
Downloading a Recipe – Round Robin Algorithm



Cluster Overview



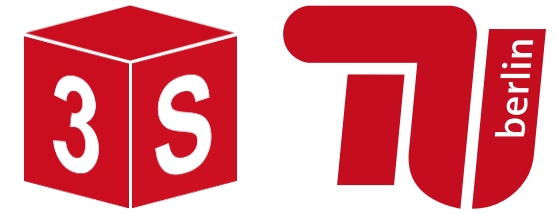
Node Architecture





Requirements Overview

1. The shared state consists of **persistently stored recipes**, raft logs and a member list.
2. Scaling out is possible, since the number of nodes can be specified in the .env file.
3. We combined several **load shedding** strategies and retries with backoff.
4. Two other strategies implemented from the Amazon Builder's Library or lecture are **leader election** and **caching**.



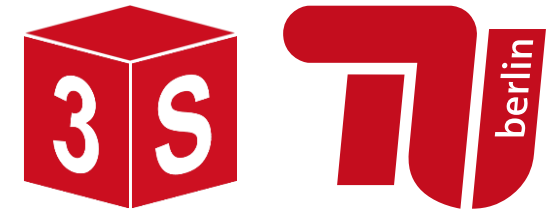
1st/2nd Requirement: Shared State and Scalability

Shared State, Horizontal Scalability

Single Client and 3 Nodes:

1. Client uploads to a random node
2. Not successful – Retry on Leader
3. Downloading from the 3 nodes

```
2025-07-12T20:56:59.006Z INFO client/main.go:14 Client 9bd4860838ce
started
2025-07-12T20:57:29.038Z INFO client/main.go:21 Benchmark-Client:
9bd4860838ce, starts to send requests
2025-07-12T20:57:29.053Z INFO client/benchmark.go:211 Retrying with
new leader: 23aa8da0e150:8080
2025-07-12T20:57:32.165Z INFO client/benchmark.go:220 Successfully
uploaded file after retrv: rice eggburger
2025-07-12T20:57:34.169Z INFO client/benchmark.go:226 Now reading
recipe from every other Node
2025-07-12T20:57:34.171Z INFO client/grpc.go:43 Sending Download
Request to target: 172.19.0.2:8080
2025-07-12T20:57:36.181Z INFO client/benchmark.go:236 Download Recipe:
rice_eggburger with Content: spinach rice salmon cheese until golden brown
bake quinoa beef beef salmon cook chicken
2025-07-12T20:57:36.181Z INFO client/grpc.go:43 Sending Download
Request to target: 172.19.0.3:8080
2025-07-12T20:57:38.192Z INFO client/benchmark.go:236 Download Recipe:
rice_eggburger with Content: spinach rice salmon cheese until golden brown
bake quinoa beef beef salmon cook chicken
2025-07-12T20:57:38.192Z INFO client/grpc.go:43 Sending Download
Request to target: 172.19.0.4:8080
2025-07-12T20:57:40.205Z INFO client/benchmark.go:236 Download Recipe:
rice_eggburger with Content: spinach rice salmon cheese until golden brown
```

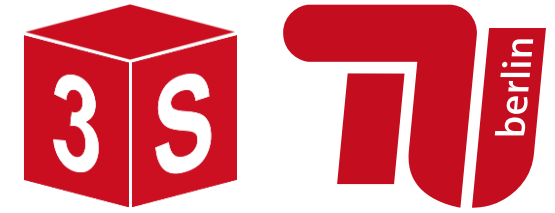
Shared State, Horizontal Scalability

Single Client and 5 Nodes:

All 5 Nodes return the same recipe

```
2025-07-12T21:03:38.644Z INFO client/main.go:14 Client 9b17373c6fba started
2025-07-12T21:04:08.649Z INFO client/main.go:21 Benchmark-Client: 9b17373c6fba, starts to send requests
2025-07-12T21:04:08.655Z INFO client/benchmark.go:211 Retrying with new leader: 498cc5274be9:8080
2025-07-12T21:04:11.723Z INFO client/benchmark.go:220 Successfully uploaded file after retry: rice_beefpasta
2025-07-12T21:04:13.723Z INFO client/benchmark.go:226 Now reading recipe from every other Node
2025-07-12T21:04:13.726Z INFO client/grpc.go:43 Sending Download Request to target: 172.19.0.4:8080
2025-07-12T21:04:15.737Z INFO client/benchmark.go:236 Download Recipe: rice_beefpasta with Content: mushroom chicken tofu bake tofu stir stir beans tofu egg spinach cook
2025-07-12T21:04:15.737Z INFO client/grpc.go:43 Sending Download Request to target: 172.19.0.5:8080
2025-07-12T21:04:17.741Z INFO client/benchmark.go:236 Download Recipe: rice_beefpasta with Content: mushroom chicken tofu bake tofu stir stir beans tofu egg spinach cook
2025-07-12T21:04:17.742Z INFO client/grpc.go:43 Sending Download Request to target: 172.19.0.6:8080
2025-07-12T21:04:19.753Z INFO client/benchmark.go:236 Download Recipe: rice_beefpasta with Content: mushroom chicken tofu bake tofu stir stir beans tofu egg spinach cook
2025-07-12T21:04:19.753Z INFO client/grpc.go:43 Sending Download Request to target: 172.19.0.2:8080
2025-07-12T21:04:21.760Z INFO client/benchmark.go:236 Download Recipe: rice_beefpasta with Content: mushroom chicken tofu bake tofu stir stir beans tofu egg spinach cook
2025-07-12T21:04:21.760Z INFO client/grpc.go:43 Sending Download Request to target: 172.19.0.3:8080
2025-07-12T21:04:23.769Z INFO client/benchmark.go:236 Download Recipe: rice_beefpasta with Content: mushroom chicken tofu bake tofu stir stir beans tofu egg spinach cook
```

3rd Requirement: Mitigation Strategies



Mitigation Strategies: Load Shedding

The load shedding is based on the cpu usage of the container and the fact that we **prioritize downloads over uploads**. (pkg/cluster/loadshed):

1. Priority Load shedding: shed only **uploads** if the cpu usage is >90%
2. Probabilistic Load shedding: shed only a percent of requests (see next slide)
3. Timeout shedding: shed all requests, if the timeout is expected to exceed before finished

Probabilistic Load Shedding - Policies

Downloads:

CPU Usage in %	Shedded requests
[95, 100]	25%
[90, 95)	10%
[85, 90)	5%
[0, 85)	0%

Uploads:

CPU Usage in %	Shedded requests
[80, 100]	25%
[75, 80)	10%
(70, 75)	5%
[0, 70]	0%

Mitigation Strategies: Load Shedding

It was hard to show load shedding – the machine must be under heavy pressure.

The other strategies are found in: `pkg/cluster/loadshed`

This case: Load Shedding based on Timeout Hints

```
2025/07/12 19:57:17 Received Download Request from client:
172.19.0.13:48296, Recipe-Filename: rice_tofusandwich
2025/07/12 19:57:17 Remaining time for execution is to low, request:
mushroom_salmonpizza, been shed
2025/07/12 19:57:17 Remaining time for execution is to low, request:
cheese_beansburger, been shed
2025/07/12 19:57:18 Creating a job now for: cheese_salmonsoup
2025/07/12 19:57:18 Now Enqueueing Job: cheese_salmonsoup
2025/07/12 19:57:18 Received Download Request from client:
172.19.0.12:55818, Recipe-Filename: rice_beefpizza
```



Mitigation Strategies: Queuing

Upload and download requests are queued separately (`pkg/upload/`, `pkg/download/`)

Enqueueing is bounded by length and available virtual memory

Shedding happens before and after the Queue

Download Queues scale dynamically based on system load



Mitigation Strategies: Retries, Backoff with Jitter

We use Retries, with exponential Backoff and Jitter for Cluster formation

The Raft-GRPC library handles Retries for critical operations (e.g. log replication)

We also use Retries with Backoff and Jitter within the Test-Client logic (`pkg/client/grpc.go`)



4th Requirement: Two Lecture or Builders Library Strategies

Leader Election – Raft Overview

1. builds a cluster of nodes: nodes can be followers, candidates or the leader.
2. no conflicting concurrent writes: leader serializes all writes to a log.
3. replication: broadcasts log entries to all available follower nodes.
4. keeps data consistent: if a node fails, all missing log entries will be sent to it, to catch up.
5. In our implementation, initially, the first node that starts is designated to perform the bootstrap process and start the cluster as the leader.

Leader Election

If the leader fails – detected by its peers due to missing heartbeats – a new election starts:

1. Nodes that have not received a heartbeat put themselves into candidate state.
2. To prevent that every node puts itself into candidate state and votes itself simultaneously, each node has a randomized but bounded heartbeat timeout.
3. Remaining followers will vote for a candidate whose log is at least as up-to-date as their own, ensuring that the leader has all necessary log entries.
4. The candidate that receives a majority of votes becomes the new leader.



Leader Election

```
2025-07-13T18:39:45.629Z [DEBUG] raft: asking for vote: term=2
from=7bea5516f2c0 address=172.19.0.3:50051
2025-07-13T18:39:45.629Z [DEBUG] raft: asking for vote: term=2
from=eb68f24bfe03 address=172.19.0.4:50051
2025-07-13T18:39:45.629Z [DEBUG] raft: vote granted: from=a77f62e0cb61
term=2 tally=1
2025-07-13T18:39:45.634Z [DEBUG] raft: vote granted: from=7bea5516f2c0
term=2 tally=2
2025-07-13T18:39:45.634Z [INFO] raft: election won: term=2 tally=2
2025-07-13T18:39:45.634Z [INFO] raft: entering leader state: leader="Node
at :50051 [Leader]"
2025-07-13T18:39:45.634Z [INFO] raft: added peer, starting replication:
peer=7bea5516f2c0
2025-07-13T18:39:45.634Z [INFO] raft: added peer, starting replication:
peer=eb68f24bfe03
2025-07-13T18:39:45.635Z [WARN] raft: appendEntries rejected, sending older
logs: peer="{Voter eb68f24bfe03 172.19.0.4:50051}" next=1
```



Further Strategies: Caching

Each Node uses an ARC (Adaptive Replacement Cache) for fast access

Download requests check the cache before hitting storage

Recipes are cached on both upload and download

When full, the cache evicts least-used entries, balancing recency and frequency



Client Interaction, Testing the Requirements



The Workflow of the Simulation

1. Nodes of our “ch3f” service are discovered via Docker DNS (`cluster/discovery.go`, `cluster/coordinator.go`)
2. A leader is elected using Hashicorp’s Raft library (`cluster/raft.go`)
3. Clients initially upload 10 recipes each by calling the gRPC endpoints of the nodes (`client/benchmark.go`, `client/grpc.go`)
4. Subsequent requests are ~ 99.5% downloads and, ~0.5% uploads. Correlated request spikes around ~15s, 25s, 45s and 50s cause sharp RPS increases.



Dependencies

BboltDB: (go.etcd.io/bbolt)

Member list: (github.com/hashicorp/memberlist)

Raft: (github.com/hashicorp/raft)

CPU, disk and memory metrics: (github.com/shirou/gopsutil/v3/)

ARC Cache: (github.com/hashicorp/golang-lru)

Logging: (go.uber.org/zap)

Env-Utils (Linus Gustafsson): (github.com/linusgith/goutils/pkg/env_utils)

Raft Transport Management: (github.com/Jille/raft-grpc-transport)