Advanced Distributed Load Balancing

Milestone 5
Praktikum: Cloud Databases
Technical University of Munich

Carl-Leander Henneking, Levin Tschürtz, Siradanai Treitel **(Group 1)**

2023-08-07

https://miro.medium.com/max/3240/1*uhEr0YiK6Bv95uM9jOy6OQ.jpeg, accessed 2023-08-01.

Agenda

1.0 Project
- 1.1 Issue
- 1.2 What’s new?
- 1.2 What happens under load?
- 1.3 Recalculation of Buckets

2.0 Demo

3.0 Benchmarking
- 3.1 Approach
- 3.2 Results & Interpretation

4.0 Q&A

1.1 Issue

- Allocating the key-ranges of KV-Servers based on a hash of the IP:Port string
  → leads to an imbalanced distribution of key-ranges
- Even with equally distributed key-ranges, the inserted data can be distributed unevenly among all servers
  → we waste computational power by not being able to adjust the key-ranges

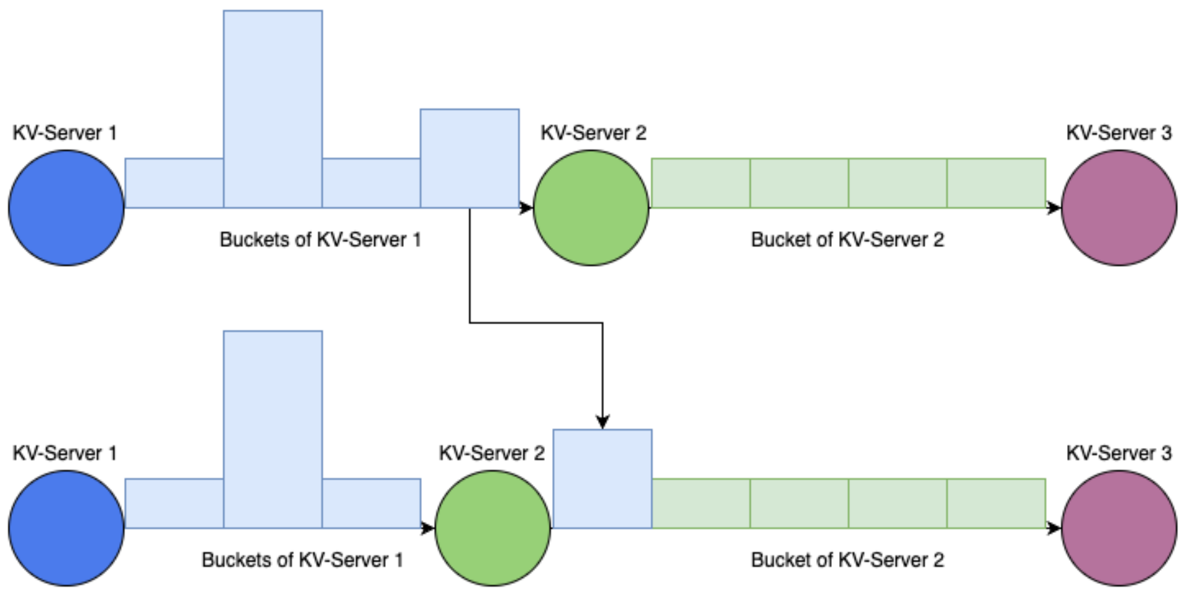
1.2 What’s new?

- Customizable start & end key-range for each KV-Server
- KV-Server key-range partitioning into **N** buckets
- Usage metrics measuring load on a KV-Server
- Offloading of buckets (based on a threshold **T**) to neighbouring nodes when KV-Server is experiencing higher load (GET/PUT/DELETE) than neighbours

Result
Eventual keyrange distribution among servers
that matches the key distribution of the underlying data &
divides computational requirements among all available servers more equally

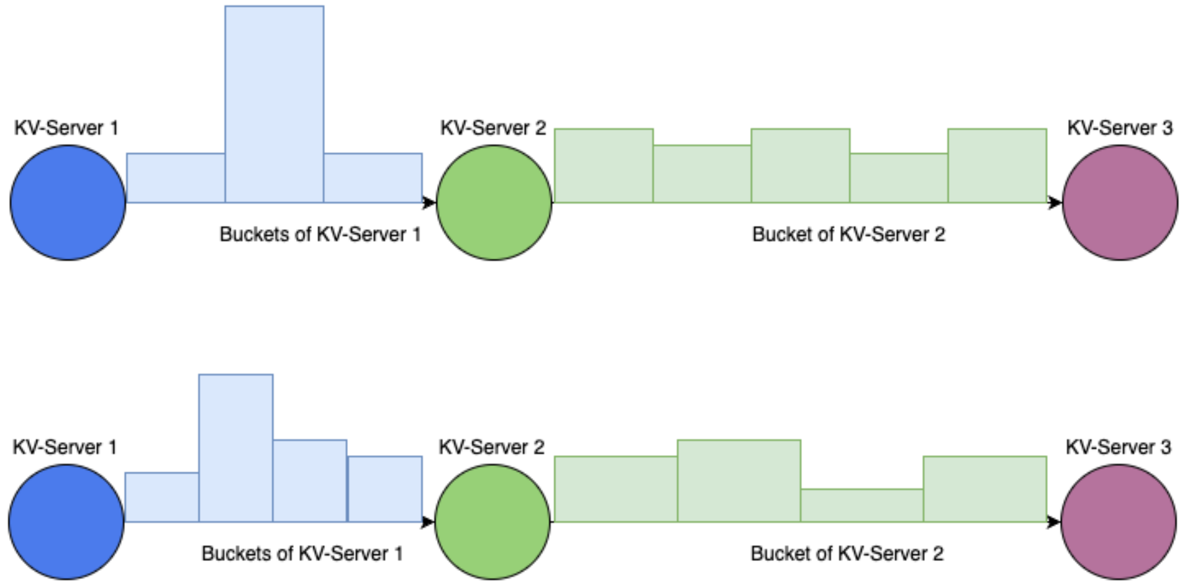
1.3 What happens under load?

- KV-Server experiences high load → requests usage metrics form its neighbouring KV-Servers
- Neighbour with lowest load takes over responsibility of at least **T**% of the keys that the high-load server holds



1.4 Recalculation of Buckets

- Both KV-Servers undertake a recalculation of their buckets → split the key-range up
- They resize their bucket key-ranges to accommodate a total of **N** buckets once again



⌥⌘1 java -jar /Users/carl/IdeaProjects/ms5/kv-client/target/kv-client.jar

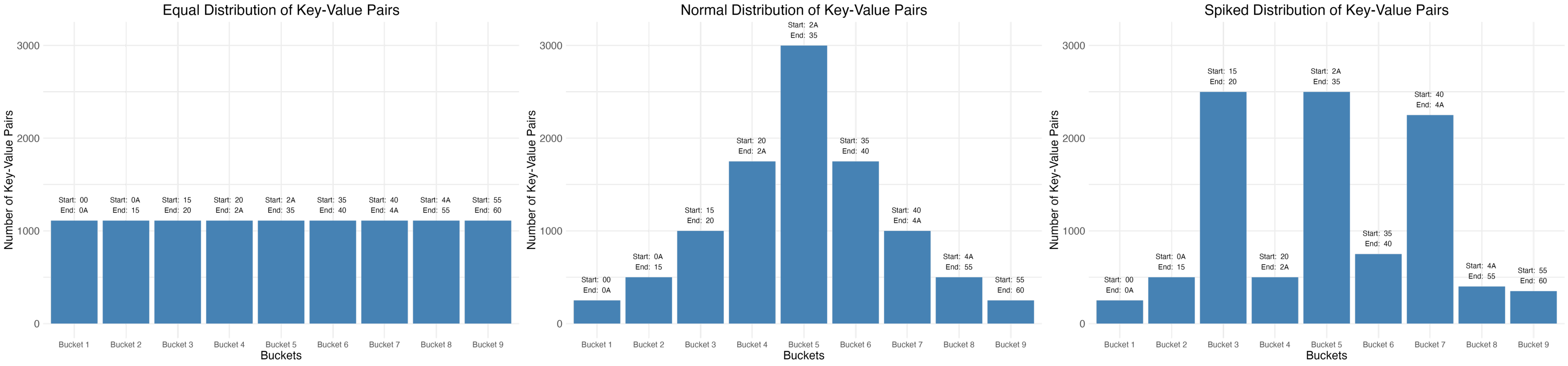
```
EchoClient>
```

⌥⌘2 java -jar /Users/carl/IdeaProjects/ms5/kv-server/target/kv-server.jar -p 8001

⌥⌘3 java -jar /Users/carl/IdeaProjects/ms5/kv-server/target/kv-server.jar -p 8002

3.1 Benchmarking - Approach

- We pre-defined three different key-value pair distributions to mimic real world datasets and their intrinsic key distributions
- 10k keys in all different combinations



3.1 Benchmarking - Approach

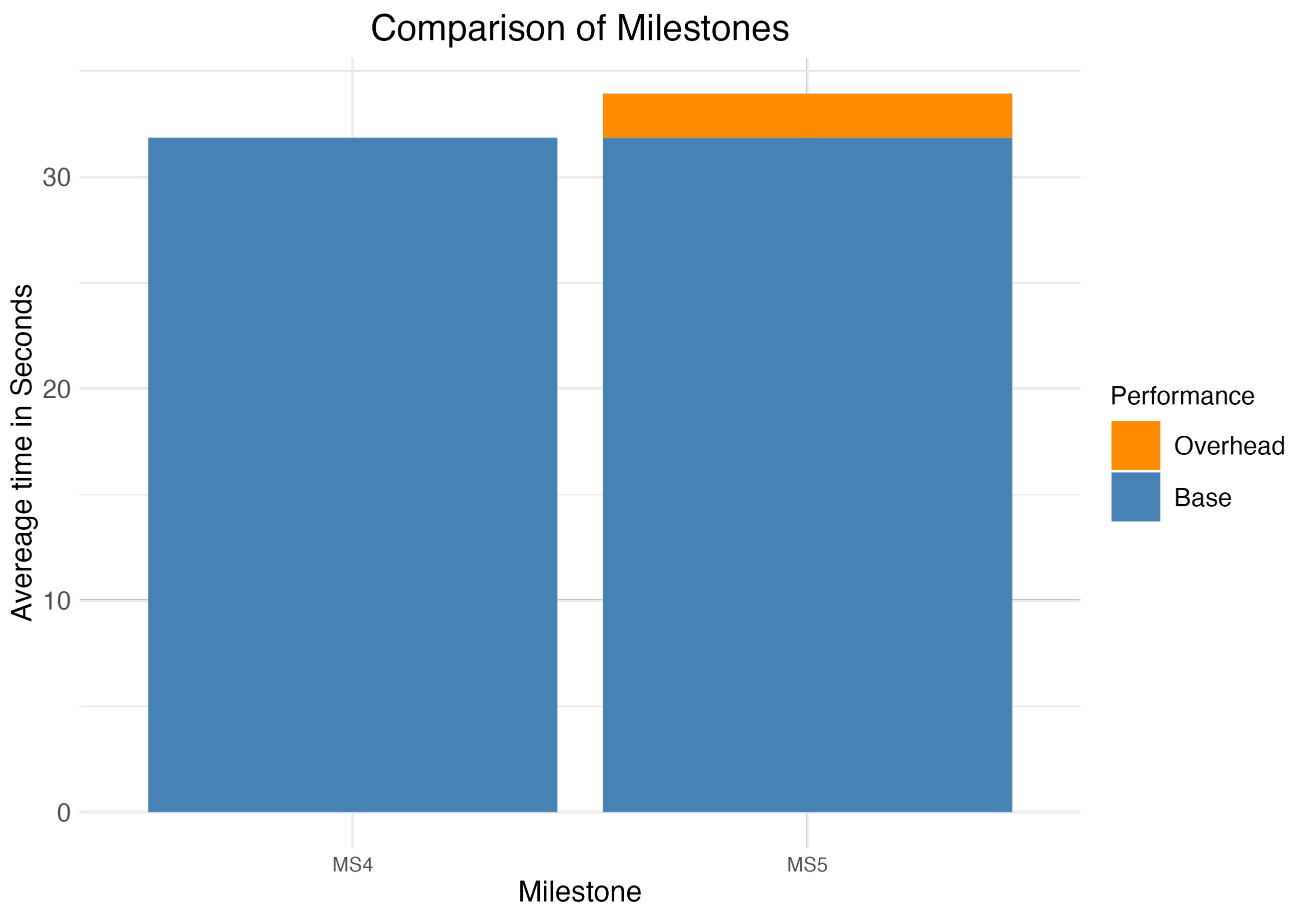
- Based on these key distributions we conducted a sensitivity analysis to find the optimal values for the number of buckets **N** and offload threshold value **T**
- Using these optimal value for bucket count **N** and threshold **T** we compared the performance of GET/PUT/DELETE operations and a mixed case of GETs and PUTs

Test Environment (Local):

- Apple M1
- 8GB Memory

3.2 Benchmarking - Results

When forcing MS5 to act under MS4 conditions (no load-averse offloading of keys)
→ overhead of managing the buckets and frequency table of one KV server to be **~2s**

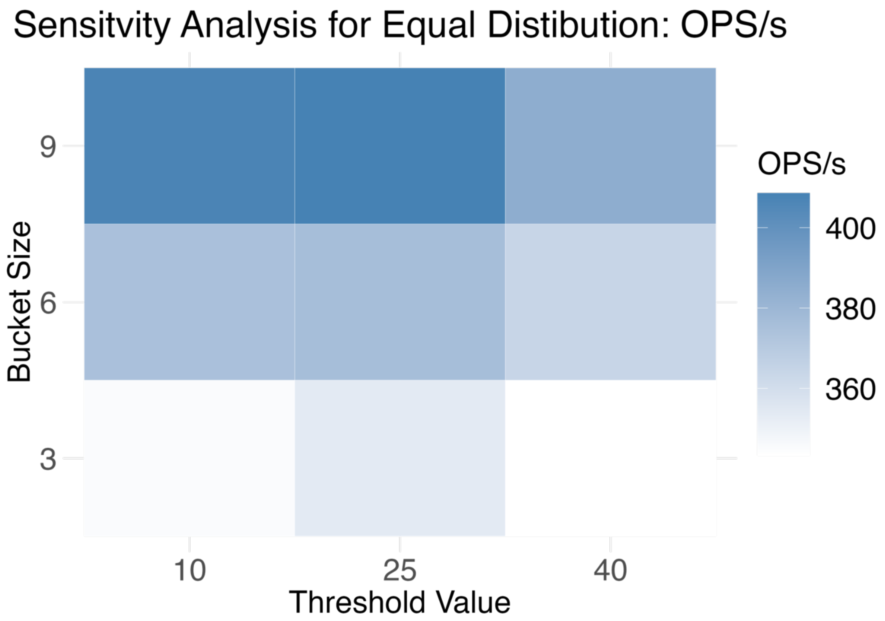


3.2 Benchmarking - Results

Equal Distribution (F1)

Threshold / Bucket Count	10%	25%	45%
9	24.56	24.48	25.97
6	26.67	26.57	27.44
3	28.95	28.27	29.12

OPS/s Heatmap

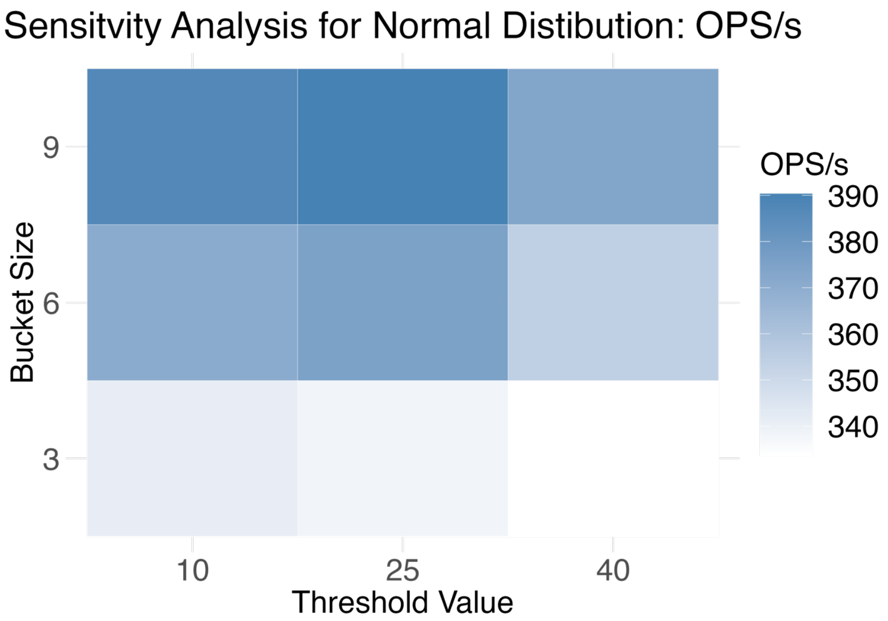


3.2 Benchmarking - Results

Normal Distribution (F2)

Threshold / Bucket Count	10%	25%	45%
9	25.83	25.62	26.76
6	26.96	26.66	28.25
3	29.31	29.58	29.97

OPS/s Heatmap

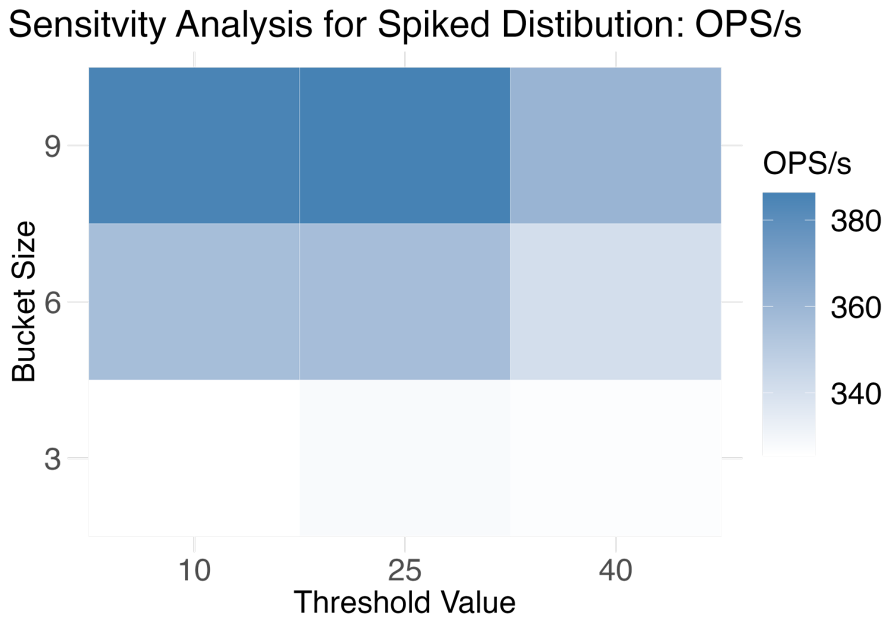


3.2 Benchmarking - Results

Spiked Distribution (F3)

Threshold / Bucket Count	10%	25%	45%
9	25.96	25.89	27.71
6	28.06	28.05	29.35
3	30.71	30.48	30.62

OPS/s Heatmap



3.2 Benchmarking - Results

- By accessing the server(s) with 3 concurrent clients we allow for concurrent GET/PUT/DELETE operations
  → performance improvement
- Due to the distribution and offloading of keys, the computational load is split among more than one single KV-Server
  → better performance by distribution the load