Taming Tail Latency in Key-Value Stores: a Scheduling Perspective

S. Ben Mokhtar, L.-C. Canon, A. Dugois, L. Marchal and E. Rivière

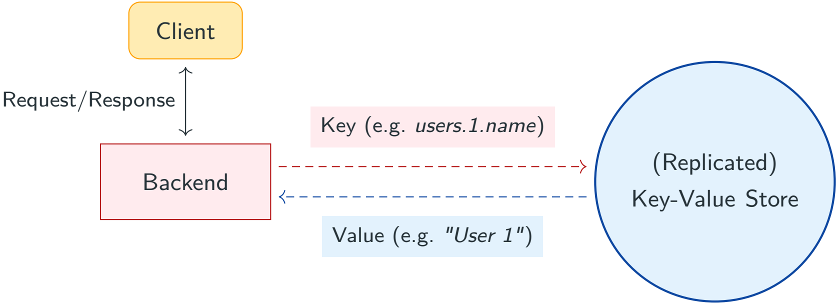
Anthony Dugois
Inria, LIP, ENS Lyon, France

27th International European Conference on Parallel and Distributed Computing

September 2, 2021

Table of Contents

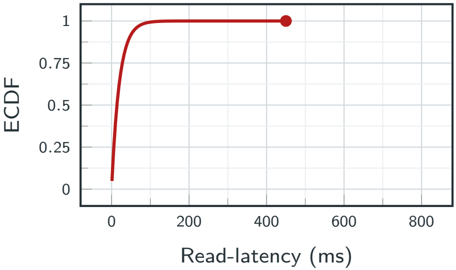
Introduction



- Strongly fault-tolerant
- Highly scalable
- Highly available
- Examples: Dynamo, Cassandra, Redis

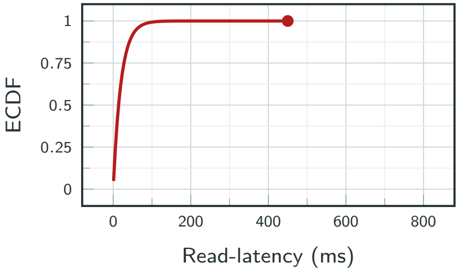
Introduction

- One user request requires multiple data items.
- Overall latency is that of the slowest request.
- A small fraction of request may result in overall degradation.



Introduction

- One user request requires multiple data items.
- Overall latency is that of the slowest request.
- A small fraction of request may result in overall degradation.



Tail Latency

Slowing a small fraction of requests (< 5 %) may degrade the QoS for most users.

Introduction

How to mitigate tail latency

Common approach: avoid that a request be sent to a busy server when a more available one would have answered faster $\rightarrow$ **this is scheduling!**

Introduction

How to mitigate tail latency

Common approach: avoid that a request be sent to a busy server when a more available one would have answered faster $\rightarrow$ **this is scheduling!**

Questions

- Is there an optimal scheduling strategy?

Introduction

How to mitigate tail latency

Common approach: avoid that a request be sent to a busy server when a more available one would have answered faster $\rightarrow$ **this is scheduling!**

Questions

- Is there an optimal scheduling strategy?
- Can we bound the performance of an offline strategy?

Introduction

How to mitigate tail latency

Common approach: avoid that a request be sent to a busy server when a more available one would have answered faster → **this is scheduling!**

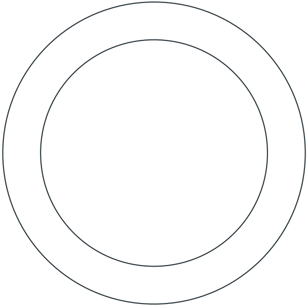
Questions

- Is there an optimal scheduling strategy?
- Can we bound the performance of an offline strategy?
- Which information do we need to build an *efficient* strategy?

Table of Contents

Model

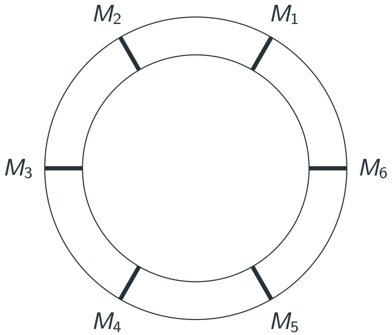
What happens in a KVS



Model

What happens in a KVS

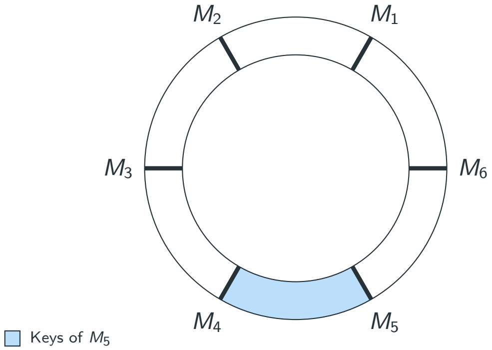
Servers are named $M_1, M_2, \ldots$



Model

What happens in a KVS

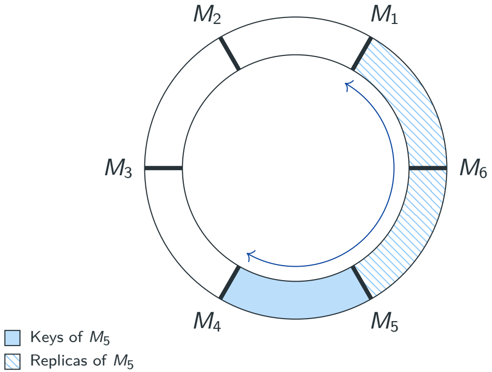
Servers are named $M_1, M_2, \ldots$



Model

What happens in a KVS

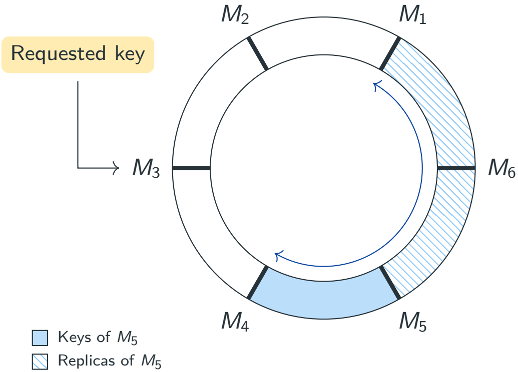
Servers are named $M_1, M_2, \ldots$



Model

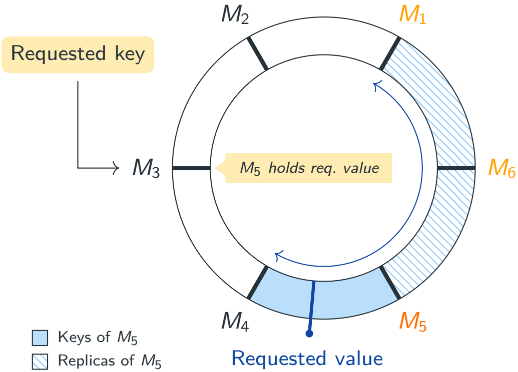
What happens in a KVS

Servers are named $M_1, M_2, \ldots$



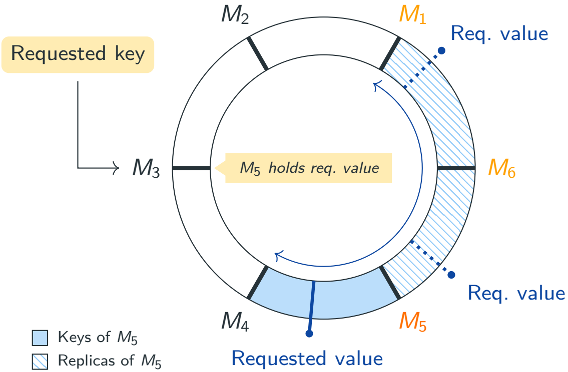
1. When a request reaches the cluster...

Servers are named $M_1, M_2, \dots$



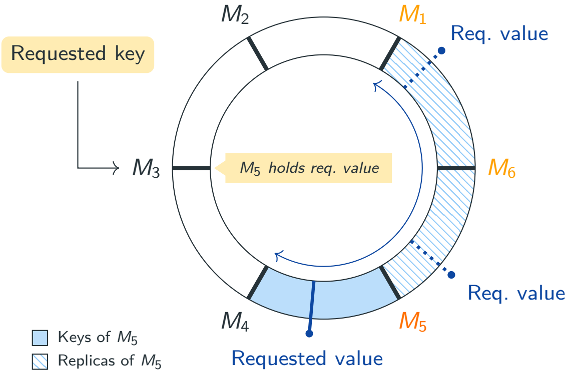
1. When a request reaches the cluster...
2. ...locate the key

Servers are named $M_1, M_2, \ldots$



1. When a request reaches the cluster. . .
2. . . . locate the key
3. . . . choose to which server direct the request.

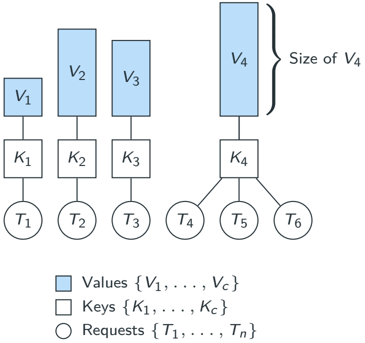
Servers are named $M_1, M_2, \ldots$



1. When a request reaches the cluster...
2. ...locate the key
3. ...choose to which server direct the request.
4. The chosen server processes the request.

Model

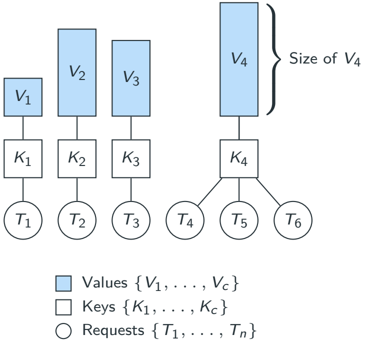
Application and platform



- Each request carries a key.

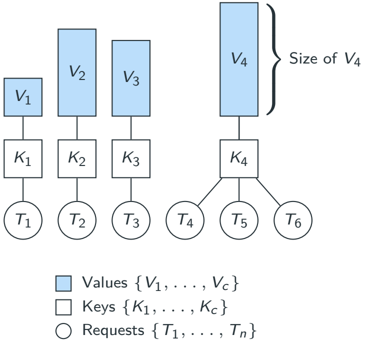
Model

Application and platform



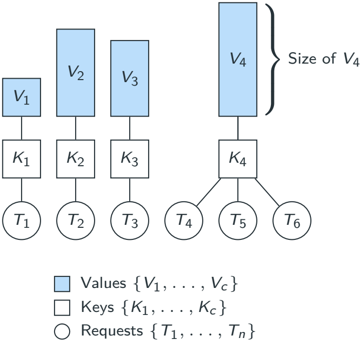
- Each request carries a key.
- Each key is associated to a value.

Model
Application and platform



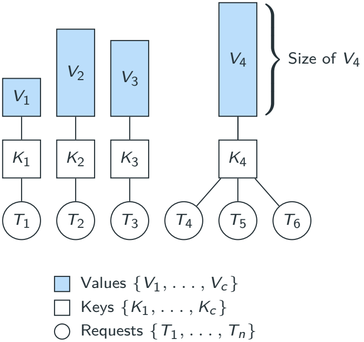
- Each request carries a key.
- Each key is associated to a value.
- Requests have different processing times.

Model
Application and platform



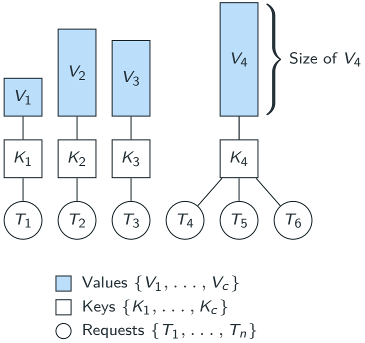
- Each request carries a key.
- Each key is associated to a value.
- Requests have different processing times.
- Processing time of a request is linear in requested value size.

Model
Application and platform



- Each request carries a key.
- Each key is associated to a value.
- Requests have different processing times.
- Processing time of a request is linear in requested value size.
- A request is released at time r_i.

Model
Application and platform



- Each request carries a key.
- Each key is associated to a value.
- Requests have different processing times.
- Processing time of a request is linear in requested value size.
- A request is released at time r_i.
- Preemption is not allowed.

Read-latency $\longrightarrow$ Flow time $F_i = C_i - r_i$ of each request (where C_i is the completion time)

Read-latency $\longrightarrow$ Flow time $F_i = C_i - r_i$ of each request (where C_i is the completion time)

99th quantile $\longrightarrow$ $\sim$ Max-flow $F_{\max} = \max F_i$

Read-latency	$\longrightarrow$	Flow time $F_i = C_i - r_i$ of each request (where C_i is the completion time)
99th quantile	$\longrightarrow$	$\sim$ Max-flow $F_{\max} = \max F_i$
Fairness/equity	$\longrightarrow$	Weighted max-flow $\max w_i F_i$ *(e.g., minimize the stretch instead of pure latency)*

Read-latency $\longrightarrow$ Flow time $\boxed{F_i = C_i - r_i}$ of each request (where C_i is the completion time)

99th quantile $\longrightarrow$ $\sim$ Max-flow $\boxed{F_{\max} = \max F_i}$

Fairness/equity $\longrightarrow$ Weighted max-flow $\boxed{\max w_i F_i}$ *(e.g., minimize the stretch instead of pure latency)*

Scheduling Problem

Considering our constraints and objective, we are interested in the scheduling problem $P|\mathcal{M}_i, \text{online-}r_i| \max w_i F_i$ (Graham's notation).

Table of Contents

Our problem is clearly NP-hard (by reduction to the parallel makespan problem). First approach: try to relax the problem and find which variants are easier.

Our problem is clearly NP-hard (by reduction to the parallel makespan problem). First approach: try to relax the problem and find which variants are easier.

- Start with the single-server problem; we give an optimal algorithm.

Algorithm 2 Single-Simple

Require: w_i

1: schedule requests by non-increasing order of w_i

Theorem

Single-Simple solves $1||\max w_i C_i$ in polynomial-time.

- By a simple exchange argument, SINGLE-SIMPLE is optimal on parallel platforms (with full replication) when costs are homogeneous.

Theorem

SINGLE-SIMPLE solves $P|p_i = p| \max w_i C_i$ in polynomial-time.

Scheduling

Without release times ($F_i = C_i$)

- By a simple exchange argument, SINGLE-SIMPLE is optimal on parallel platforms (with full replication) when costs are homogeneous.

Theorem

SINGLE-SIMPLE solves $P|p_i = p|\max w_i C_i$ in polynomial-time.

- For heterogeneous costs, we find that it approximates the problem by a factor $2 - 1/m$, with m the number of machines.

Theorem

SINGLE-SIMPLE is a tight $(2 - 1/m)$-approximation for $P||\max w_i C_i$.

- The preemptive version of our problem is solved by Legrand et al.[1]

Theorem

$R|r_i, pmtn|\max w_i F_i$ can be solved in polynomial-time.

[1] Legrand, A., Su, A., Vivien, F.: Minimizing the stretch when scheduling flows of divisible requests. Journal of Scheduling (2008).

- The preemptive version of our problem is solved by Legrand et al.[1]

Theorem

$R|r_i, pmtn|\max w_i F_i$ can be solved in polynomial-time.

Model restricted replication in unrelated setting

If a request i cannot be executed by a server j, set the corresponding processing time $p_{ij} = \infty$. As a consequence, solving $R|r_i, pmtn|\max w_i F_i$ also solves $P|\mathcal{M}_i, r_i, pmtn|\max w_i F_i$.

[1] Legrand, A., Su, A., Vivien, F.: Minimizing the stretch when scheduling flows of divisible requests. Journal of Scheduling (2008).

Scheduling
Offline problem

- Recall we do not allow preemption in our model, mainly because of the overhead of migration. By a reduction to the parallel makespan problem, we show that the problem becomes NP-complete when migration is not allowed...

Theorem

The non-migratory version of $R|r_i, pmtn| \max w_i F_i$ is NP-complete.

- Recall we do not allow preemption in our model, mainly because of the overhead of migration. By a reduction to the parallel makespan problem, we show that the problem becomes NP-complete when migration is not allowed...

Theorem

The non-migratory version of $R|r_i, pmtn|\max w_i F_i$ is NP-complete.

- ...but the preemptive version still provides a **lower bound** for our problem!

Table of Contents

Scheduling happens at two levels.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
   - EFT: send request to the least-loaded server.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
    - EFT: send request to the least-loaded server.
    - EFT-S: same as EFT, but specializes some servers for short requests.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
   - EFT: send request to the least-loaded server.
   - EFT-S: same as EFT, but specializes some servers for short requests.
   - State-of-the-art heuristics: Héron[2], LOR, Random.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Simulations
Online heuristics

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
   - EFT: send request to the least-loaded server.
   - EFT-S: same as EFT, but specializes some servers for short requests.
   - State-of-the-art heuristics: Héron[2], LOR, Random.
2. **Local Execution**: choose in which order the server executes the local queue.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
   - EFT: send request to the least-loaded server.
   - EFT-S: same as EFT, but specializes some servers for short requests.
   - State-of-the-art heuristics: HÉRON[2], LOR, RANDOM.
2. **Local Execution**: choose in which order the server executes the local queue.
   - FIFO: classic first-in first-out queue.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
   - EFT: send request to the least-loaded server.
   - EFT-S: same as EFT, but specializes some servers for short requests.
   - State-of-the-art heuristics: Héron[2], LOR, Random.
2. **Local Execution**: choose in which order the server executes the local queue.
   - FIFO: classic first-in first-out queue.
   - MWF: sort queue by non-increasing weighted flow time.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Simulations
Online heuristics

Scheduling happens at two levels.

1. **Replica Selection**: choose which resource will execute the request.
   - EFT: send request to the least-loaded server.
   - EFT-S: same as EFT, but specializes some servers for short requests.
   - State-of-the-art heuristics: HÉRON[2], LOR, RANDOM.
2. **Local Execution**: choose in which order the server executes the local queue.
   - FIFO: classic first-in first-out queue.
   - MWF: sort queue by non-increasing weighted flow time.

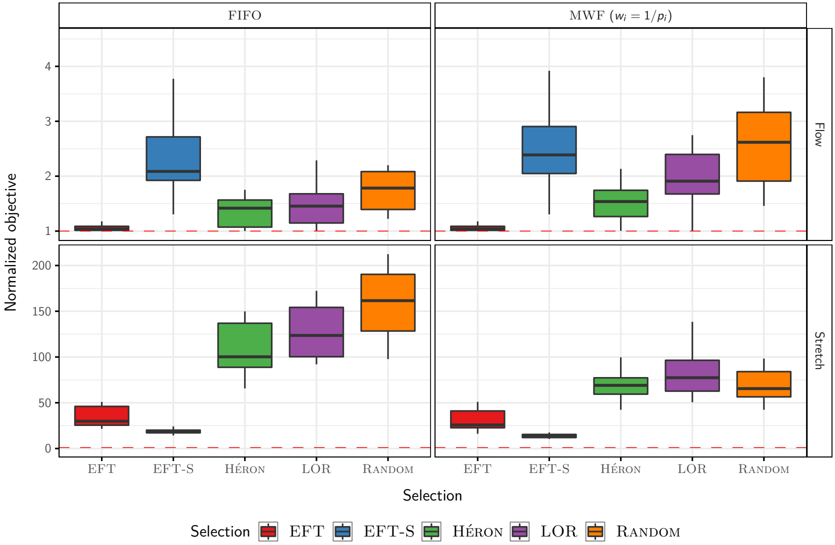
Note on EFT

EFT is very difficult to achieve in a real system. One would consider a degraded version in practice; our goal here is to evaluate the best possible situation.

[2] Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).

Simulations

Distribution of normalized read-latency and stretch maximums (1000 requests)



Takeaways

- EFT allows to get very close to the lower bound for read-latency.
- EFT is the most stable heuristic for read-latency between scenarios.
- $\mathrm{EFT\text{-}S}$ is not good for read-latency, but it is the best for the stretch.
- MWF improves the stretch objective of all selection heuristics without worsening read-latency too much.

Table of Contents

Conclusion

- Formal model of key-value store.
- Intractability of the related scheduling problem, even for some restricted variants.
- Comparison of online heuristics with a lower bound.

Conclusion

- Formal model of key-value store.
- Intractability of the related scheduling problem, even for some restricted variants.
- Comparison of online heuristics with a lower bound.

Answers to initial questions

- Is there an optimal scheduling strategy?
  $\rightarrow$ **The problem is NP-hard.**

Conclusion

- Formal model of key-value store.
- Intractability of the related scheduling problem, even for some restricted variants.
- Comparison of online heuristics with a lower bound.

Answers to initial questions

- Is there an optimal scheduling strategy?
  $\rightarrow$ **The problem is NP-hard.**
- Can we bound the performance of an offline strategy?
  $\rightarrow$ **Yes! The preemptive version provides a lower bound.**

Conclusion

- Formal model of key-value store.
- Intractability of the related scheduling problem, even for some restricted variants.
- Comparison of online heuristics with a lower bound.

Answers to initial questions

- Is there an optimal scheduling strategy?
  $\rightarrow$ **The problem is NP-hard.**
- Can we bound the performance of an offline strategy?
  $\rightarrow$ **Yes! The preemptive version provides a lower bound.**
- What information do we need to build an *efficient* strategy?
  $\rightarrow$ **Simulations show that a good knowledge of current load is critical.**