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
1. Introduction

2. Model
   - What happens in a KVS
   - Application and platform
   - Objective

3. Scheduling
   - Without release times ($F_i = C_i$)
   - Offline problem

4. Simulations
   - Online heuristics
   - Results

5. Conclusion
Introduction

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

Key-Value Store Backend

Client

Request/Response

Key (e.g. `users.1.name`)

Value (e.g. "User 1")

(Replicated) Key-Value Store
• 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.
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!**
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?
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?
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

1. Introduction

2. Model
   - What happens in a KVS
   - Application and platform
   - Objective

3. Scheduling
   - Without release times \((F_i = C_i)\)
   - Offline problem

4. Simulations
   - Online heuristics
   - Results

5. Conclusion
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 $M_1, M_2, \ldots$
Servers are named $M_1, M_2, \ldots$

1. When a request reaches the cluster...

- Requested key
- Keys of $M_5$
- Replicas of $M_5$
Model
What happens in a KVS

Servers are named $M_1, M_2, \ldots$

1. When a request reaches the cluster…
2. …locate the key
Model
What happens in a KVS

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.
Model
What happens in a KVS

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.

\[
\begin{align*}
&\text{Values } \{V_1, \ldots, V_c\} \\
&\text{Keys } \{K_1, \ldots, K_c\} \\
&\text{Requests } \{T_1, \ldots, T_n\}
\end{align*}
\]
Model
Application and platform

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

Diagram:
- Values \{V_1, \ldots, V_c\}
- Keys \{K_1, \ldots, K_c\}
- Requests \{T_1, \ldots, T_n\}

Size of V_4
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 $r_i$. 

Values $\{V_1, \ldots, V_c\}$
Keys $\{K_1, \ldots, K_c\}$
Requests $\{T_1, \ldots, T_n\}$
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.
Model

Objective

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

99th quantile $\rightarrow$ $\sim$ Max-flow $F_{\text{max}} = \max F_i$
Model

Objective

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

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

Fairness/equity $\rightarrow$ Weighted max-flow $\max w_i F_i$ (e.g., minimize the stretch instead of pure latency)
Model

Objective

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

**99th quantile**  →  $\sim$ Max-flow $F_{\text{max}} = \max F_i$

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

Scheduling Problem

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

2. Model
   - What happens in a KVS
   - Application and platform
   - Objective

3. Scheduling
   - Without release times \((F_i = C_i)\)
   - Offline problem

4. Simulations
   - Online heuristics
   - Results

5. Conclusion
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.
Scheduling
Without release times \((F_i = C_i)\)

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

Theorem
\texttt{SINGLE-SIMPLE} solves \(P|p_i = p|\max w_i C_i\) 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.

• 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 \( P|| \max w_i C_i \).
Scheduling
Offline problem

- The preemptive version of our problem is solved by Legrand et al.\textsuperscript{1}

\textbf{Theorem}

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

The preemptive version of our problem is solved by Legrand et al. \(^1\)

---

**Theorem**

\[ R|\text{r}_i, \text{pmtn}| \max w_i F_i \] 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 \( R|\text{r}_i, \text{pmtn}| \max w_i F_i \) also solves \( P|\text{M}_i, \text{r}_i, \text{pmtn}| \max w_i F_i \).

---

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 $R|r_i, pmtn|\max w_iF_i$ is NP-complete.

...but the preemptive version still provides a lower bound for our problem!
1. Introduction

2. Model
   - What happens in a KVS
   - Application and platform
   - Objective

3. Scheduling
   - Without release times ($F_i = C_i$)
   - Offline problem

4. Simulations
   - Online heuristics
   - Results

5. Conclusion
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\textsuperscript{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.

Simulations
Online heuristics

Scheduling happens at two levels.

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

---

Scheduling happens at two levels.

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

---

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.

---

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.

---

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.

---

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.

---

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.

\(^2\text{Jaiman, V., Ben Mokhtar, S., Quéma, V., Chen, L. Y., Rivière, E.: Héron: Taming tail latencies in key-value stores under heterogeneous workloads. 37th Symposium on Reliable Distributed Systems (2018).}\)
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: Héron², 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.

---

Simulations

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

- FIFO
- MWF \( (w_i = 1/p_i) \)

Selection: EFT, EFT-S, HÉRON, LOR, RANDOM

The box plots show the distribution of normalized objectives for different selection methods under FIFO and MWF scheduling policies.
Simulations
Results

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.
- EFT-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.
1. Introduction

2. Model
   - What happens in a KVS
   - Application and platform
   - Objective

3. Scheduling
   - Without release times \((F_i = C_i)\)
   - Offline problem

4. Simulations
   - Online heuristics
   - Results

5. Conclusion
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?
  → **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?
  → **The problem is NP-hard.**

• Can we bound the performance of an offline strategy?
  → **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?
  → **The problem is NP-hard.**

• Can we bound the performance of an offline strategy?
  → **Yes! The preemptive version provides a lower bound.**

• What information do we need to build an efficient strategy?
  → **Simulations show that a good knowledge of current load is critical.**