Bounding the Flow Time in Online Scheduling with Structured Processing Sets

L.-C. Canon, A. Dugois and L. Marchal

Anthony Dugois
Inria, LIP, ENS Lyon, France

Journée SCALE

December 3, 2021

Table of Contents

1. Introduction

2. Processing Set Structures and Related Bounds

3. Maximum Load under Biased Popularity

4. Conclusion

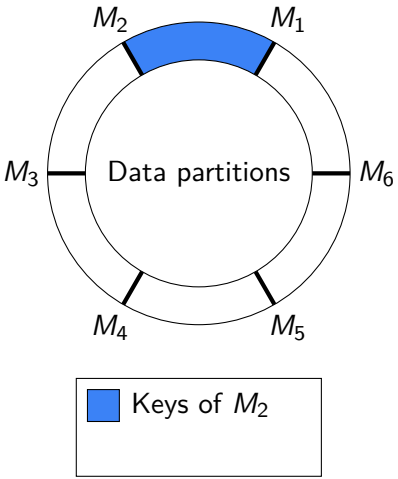
Applicative context

Schedule a stream of requests $T_1, T_2, \ldots$ in key-value stores in order to optimize the response time of each request.

Applicative context

Schedule a stream of requests $T_1, T_2, \ldots$ in key-value stores in order to optimize the response time of each request.

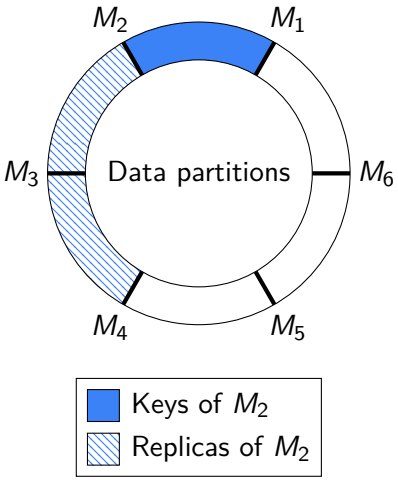
- Each server $M_1, M_2, \ldots$ holds a data partition.



Applicative context

Schedule a stream of requests $T_1, T_2, \ldots$ in key-value stores in order to optimize the response time of each request.

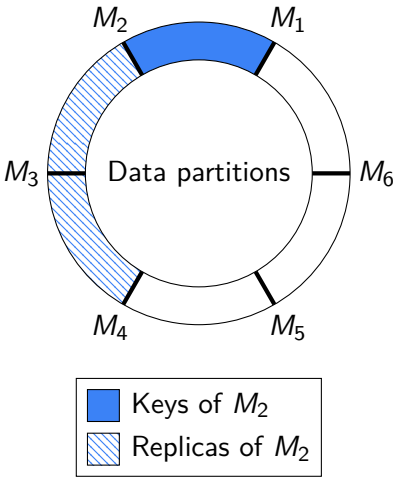
- Each server $M_1, M_2, \ldots$ holds a data partition.
- Each value is replicated on different locations.



Applicative context

Schedule a stream of requests $T_1, T_2, \ldots$ in key-value stores in order to optimize the response time of each request.

- Each server $M_1, M_2, \ldots$ holds a data partition.
- Each value is replicated on different locations.
- Each request holds a key, looking for the associated value.



Constraints

- Only a subset of servers can execute a given request (processing set restriction).

Constraints

- Only a subset of servers can execute a given request (processing set restriction).
- Requests arrive over time (online model).

Constraints

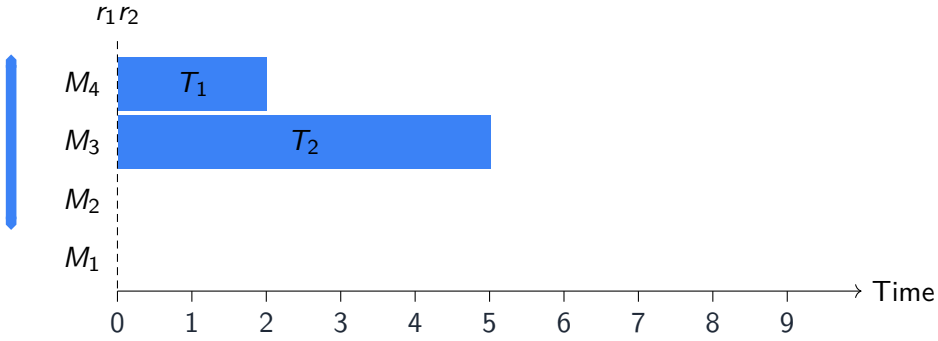
- Only a subset of servers can execute a given request (processing set restriction).
- Requests arrive over time (online model).
- Neither preemption nor simultaneous execution are allowed.

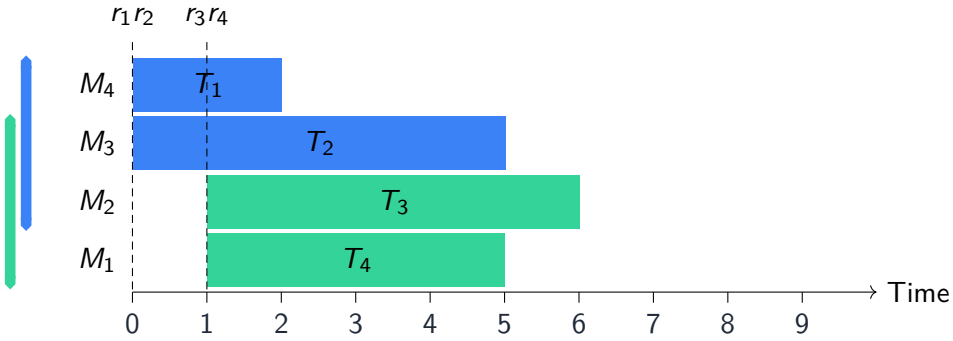
Constraints

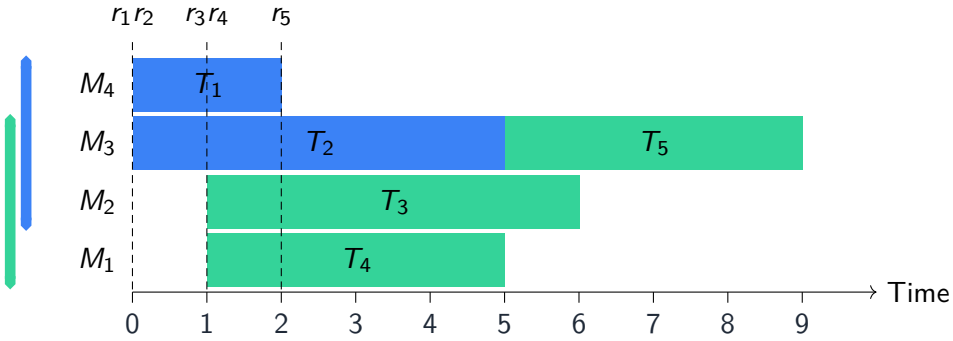
- Only a subset of servers can execute a given request (processing set restriction).
- Requests arrive over time (online model).
- Neither preemption nor simultaneous execution are allowed.

Objective

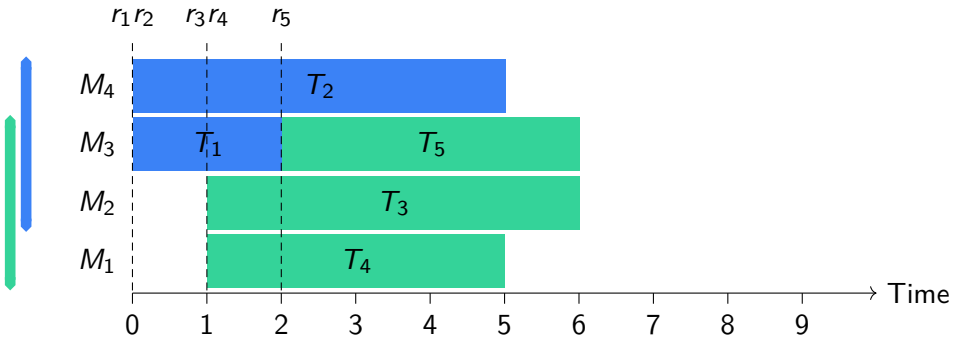
Minimize the maximum flow time $F_{\max} = \max F_i$ (F_i is the difference between the *release time* r_i and the *completion time* C_i of request i).


r_1 r_2
M_4
M_3
M_2
M_1
T_1
T_2
Time
0
1
2
3
4
5
6
7
8
9


r_1 r_2
r_3 r_4
M_4
M_3
M_2
M_1
T_1
T_2
T_3
T_4
Time
0
1
2
3
4
5
6
7
8
9



$$F_{\max} = C_5 - r_5 = 7$$



$$F_{\max} = C_3 - r_3 = 5$$

Table of Contents

1. Introduction

2. Processing Set Structures and Related Bounds

3. Maximum Load under Biased Popularity

4. Conclusion

General

The processing sets exhibit no particular structure.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	0	1	0	1	0	1
T_3	0	0	1	0	1	1
T_4	1	1	0	1	1	0
T_5	0	0	0	0	0	1

General

The processing sets exhibit no particular structure.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	0	1	0	1	0	1
T_3	0	0	1	0	1	1
T_4	1	1	0	1	1	0
T_5	0	0	0	0	0	1

Lower bound: $\Omega(m)$ [Anand et al., 2017]

(m is the number of machines)

General

The processing sets exhibit no particular structure.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	0	1	0	1	0	1
T_3	0	0	1	0	1	1
T_4	1	1	0	1	1	0
T_5	0	0	0	0	0	1

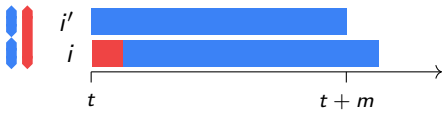
Lower bound: $\Omega(m)$ [Anand et al., 2017]

(m is the number of machines)

Can we do better if processing sets are structured?

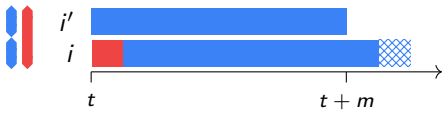
Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.



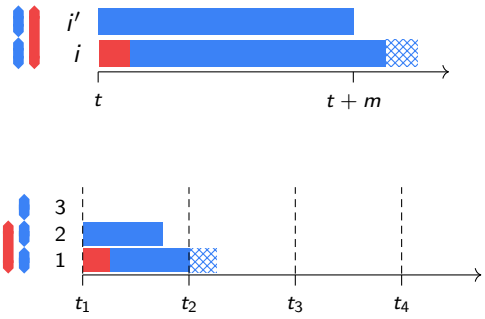
Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t + m - 1$, and feasible on only one machine.



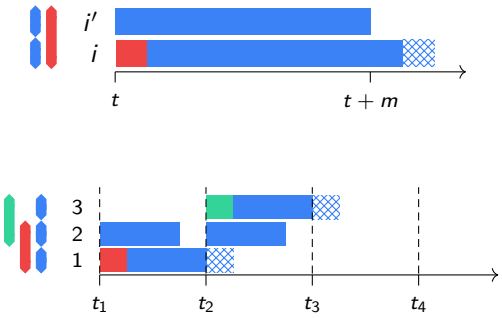
Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+m-1$, and feasible on only one machine.
   - 1 final task released at $t+m$, and feasible only on the machine that executed the critical task.



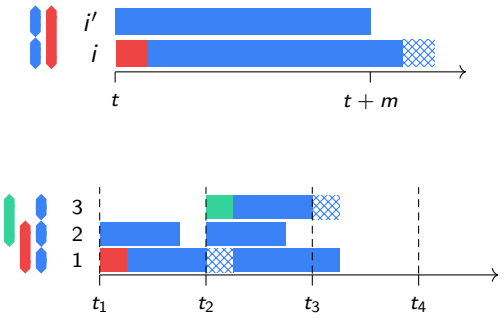
Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+m-1$, and feasible on only one machine.
   - 1 final task released at $t+m$, and feasible only on the machine that executed the critical task.
2. Compose blocks to accumulate delay.



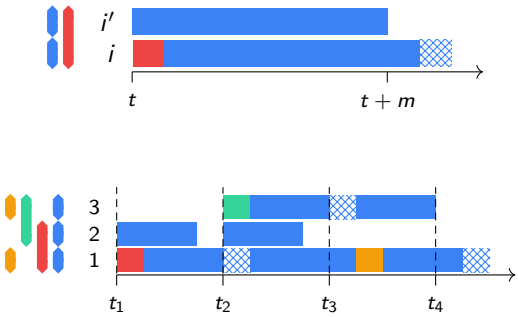
Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+m-1$, and feasible on only one machine.
   - 1 final task released at $t+m$, and feasible only on the machine that executed the critical task.
2. Compose blocks to accumulate delay.



Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+m-1$, and feasible on only one machine.
   - 1 final task released at $t+m$, and feasible only on the machine that executed the critical task.
2. Compose blocks to accumulate delay.



Proof sketch.

1. Define a block of unit tasks on machines i, i':
   - 1 critical task (in red) released at t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+m-1$, and feasible on only one machine.
   - 1 final task released at $t+m$, and feasible only on the machine that executed the critical task.
2. Compose blocks to accumulate delay.

Nested

Two different processing sets are either nested or disjoint.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	0	0	0
T_3	0	0	0	0	1	1
T_4	1	1	0	0	0	0
T_5	0	0	0	0	1	0

Nested

Two different processing sets are either nested or disjoint.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	0	0	0
T_3	0	0	0	0	1	1
T_4	1	1	0	0	0	0
T_5	0	0	0	0	1	0

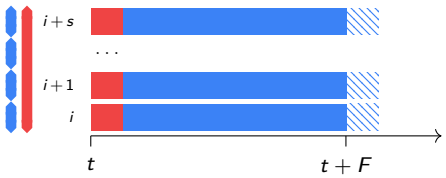
Lower bound: $\frac{1}{3}\lfloor \log_2(m) + 2 \rfloor$

Proof sketch.

1. Let $F = \log(m) + 2$.

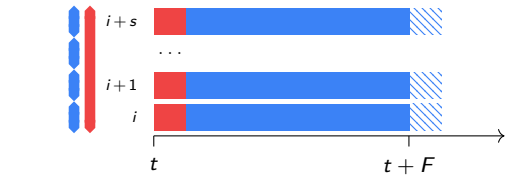
Proof sketch.

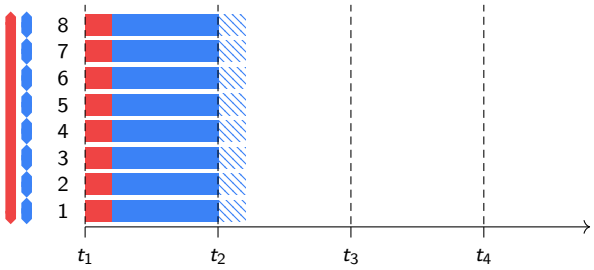
1. Let $F = \log(m) + 2$.
2. Define a block of tasks on machines $i, \ldots, i+s$:
   - s critical tasks (in red) released at time t.



Proof sketch.

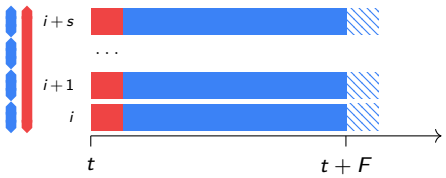
1. Let $F = \log(m) + 2$.
2. Define a block of tasks on machines $i, \ldots, i+s$:
   - s critical tasks (in red) released at time t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+F-1$, and feasible on only one machine.

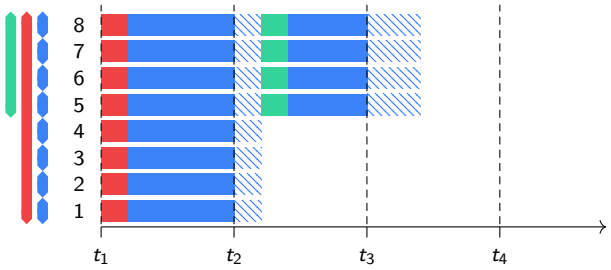




Proof sketch.

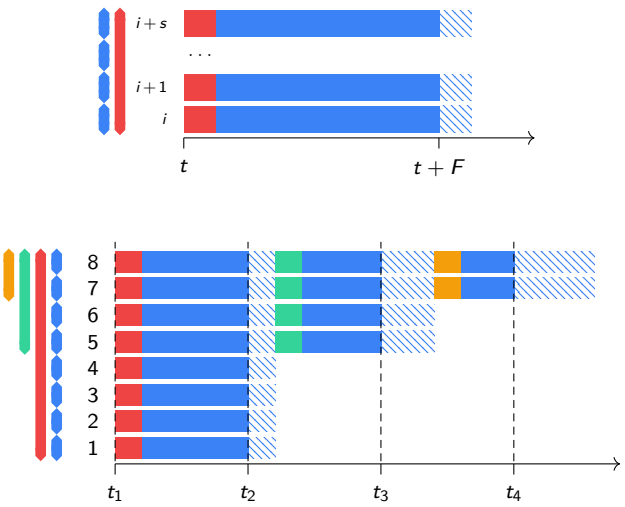
1. Let $F = \log(m) + 2$.
2. Define a block of tasks on machines $i, \ldots, i+s$:
   - s critical tasks (in red) released at time t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+F-1$, and feasible on only one machine.
3. Compose blocks to accumulate delay, by choosing the next subinterval that contains the most uncompleted blue tasks.





Proof sketch.

1. Let $F = \log(m) + 2$.
2. Define a block of tasks on machines $i, \ldots, i+s$:
   - s critical tasks (in red) released at time t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+F-1$, and feasible on only one machine.
3. Compose blocks to accumulate delay, by choosing the next subinterval that contains the most uncompleted blue tasks.



Proof sketch.

1. Let $F = \log(m) + 2$.
2. Define a block of tasks on machines $i, \ldots, i+s$:
   - s critical tasks (in red) released at time t.
   - A set of filling tasks (in blue) released at each time unit $t, \ldots, t+F-1$, and feasible on only one machine.
3. Compose blocks to accumulate delay, by choosing the next subinterval that contains the most uncompleted blue tasks.

Inclusive

Two different processing sets must be nested.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	1	1	0
T_3	0	1	1	1	0	0
T_4	0	1	1	0	0	0
T_5	0	0	1	0	0	0

Inclusive

Two different processing sets must be nested.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	1	1	0
T_3	0	1	1	1	0	0
T_4	0	1	1	0	0	0
T_5	0	0	1	0	0	0

Immediate Dispatch (i.d.)

Each task is definitely scheduled as soon as it enters in the system.

Inclusive

Two different processing sets must be nested.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	1	1	0
T_3	0	1	1	1	0	0
T_4	0	1	1	0	0	0
T_5	0	0	1	0	0	0

Immediate Dispatch (i.d.)

Each task is definitely scheduled as soon as it enters in the system.

Lower bound (i.d.): $\lfloor \log_2(m) + 1 \rfloor$

Constant size

Each processing set has the same size k.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	0	1	1	0	0
T_2	1	1	1	0	0	0
T_3	0	1	0	1	0	1
T_4	0	1	1	0	1	0
T_5	1	0	1	0	0	1

Constant size

Each processing set has the same size k.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	0	1	1	0	0
T_2	1	1	1	0	0	0
T_3	0	1	0	1	0	1
T_4	0	1	1	0	1	0
T_5	1	0	1	0	0	1

Immediate Dispatch (i.d.)

Each task is definitely scheduled as soon as it enters in the system.

Lower bound (i.d.): $\lfloor \log_k(m) \rfloor$

Full replication

No processing set restriction.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	1	1	1
T_3	1	1	1	1	1	1
T_4	1	1	1	1	1	1
T_5	1	1	1	1	1	1

Full replication

No processing set restriction.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	1	1	1
T_3	1	1	1	1	1	1
T_4	1	1	1	1	1	1
T_5	1	1	1	1	1	1

Earliest Finish Time (EFT)

When a task arrives in the system, schedule it on an eligible machine such that its completion time is minimized.

Full replication

No processing set restriction.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	1	1	1
T_2	1	1	1	1	1	1
T_3	1	1	1	1	1	1
T_4	1	1	1	1	1	1
T_5	1	1	1	1	1	1

Earliest Finish Time (EFT)

When a task arrives in the system, schedule it on an eligible machine such that its completion time is minimized.

$(3 - 2/m)$-competitive [Bender et al., 1998]

Disjoint

Two different processing sets are either equal or disjoint.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	0	0	0	0
T_2	1	1	0	0	0	0
T_3	0	0	1	1	1	0
T_4	0	0	1	1	1	0
T_5	0	0	0	0	0	1

Disjoint

Two different processing sets are either equal or disjoint.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	0	0	0	0
T_2	1	1	0	0	0	0
T_3	0	0	1	1	1	0
T_4	0	0	1	1	1	0
T_5	0	0	0	0	0	1

EFT is $(3 - 2/\max|\mathcal{M}_i|)$-competitive

($|\mathcal{M}_i|$ is the size of processing set of task i)

Disjoint

Two different processing sets are either equal or disjoint.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	0	0	0	0
T_2	1	1	0	0	0	0
T_3	0	0	1	1	1	0
T_4	0	0	1	1	1	0
T_5	0	0	0	0	0	1

EFT is $(3 - 2/\max|\mathcal{M}_i|)$-competitive

($|\mathcal{M}_i|$ is the size of processing set of task i)

Proof sketch.

1. Use the disjoint processing sets to define subinstances of the problem.
2. Apply EFT in parallel on each subinstance.

Interval of constant size

Each processing set is a contiguous interval of size k.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	0	0	0
T_2	0	1	1	1	0	0
T_3	0	0	0	1	1	1
T_4	0	0	1	1	1	0
T_5	0	1	1	1	0	0

Interval of constant size

Each processing set is a contiguous interval of size k.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	0	0	0
T_2	0	1	1	1	0	0
T_3	0	0	0	1	1	1
T_4	0	0	1	1	1	0
T_5	0	1	1	1	0	0

Very common in key-value stores.

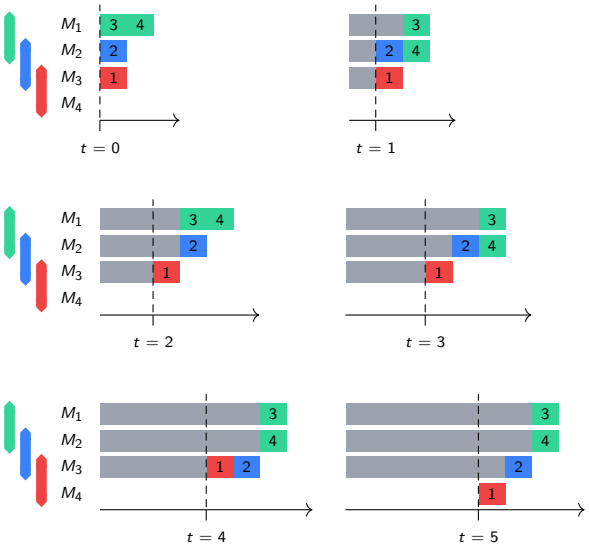
Interval of constant size

Each processing set is a contiguous interval of size k.

	M_1	M_2	M_3	M_4	M_5	M_6
T_1	1	1	1	0	0	0
T_2	0	1	1	1	0	0
T_3	0	0	0	1	1	1
T_4	0	0	1	1	1	0
T_5	0	1	1	1	0	0

Very common in key-value stores.

Lower bound (EFT): $m - k + 1$



Proof sketch.

1. m tasks are released in-order at each time unit:
   - $m - k$ tasks, feasible only on their own interval;
   - k tasks, feasible only on the first interval.
2. Show that the delay profile converges to a stable profile:
   - 2.1 While the stable profile is not reached, the total delay is strictly increasing with time.
   - 2.2 At each time, either the stable profile is not reached, or there is a delay larger than $m - k$.

Table of Contents

1. Introduction

2. Processing Set Structures and Related Bounds

3. Maximum Load under Biased Popularity

4. Conclusion

Maximum Load under Biased Popularity

- Unit tasks arrive in the system according to a Poisson process of parameter λ.

Maximum Load under Biased Popularity

- Unit tasks arrive in the system according to a Poisson process of parameter λ.
- No replication at the beginning.

Maximum Load under Biased Popularity

- Unit tasks arrive in the system according to a Poisson process of parameter λ.
- No replication at the beginning.
- Each machine has a non-uniform probability to hold the data needed by a task.

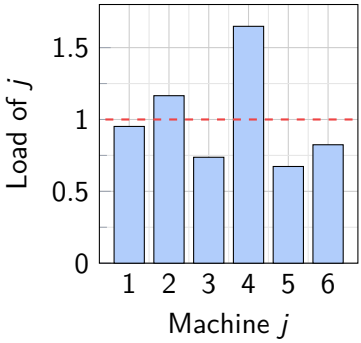
Maximum Load under Biased Popularity

- Unit tasks arrive in the system according to a Poisson process of parameter λ.
- No replication at the beginning.
- Each machine has a non-uniform probability to hold the data needed by a task.

Load of a machine j

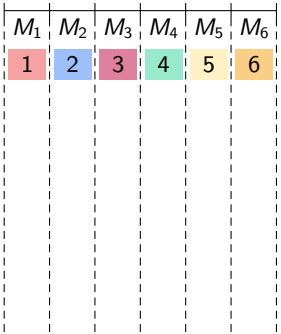
Average number of tasks sent on j at each time unit:

$$\text{Load of } j = \lambda \cdot \mathit{Prob}(\text{choosing } j)$$



Maximum Load under Biased Popularity

What is the theoretical maximum load λ one can achieve when data are replicated?

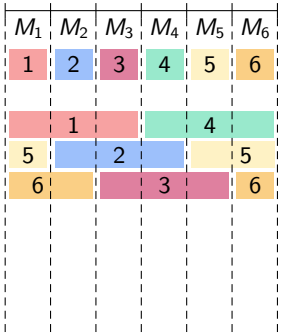


No replication

Maximum Load under Biased Popularity

What is the theoretical maximum load λ one can achieve when data are replicated?

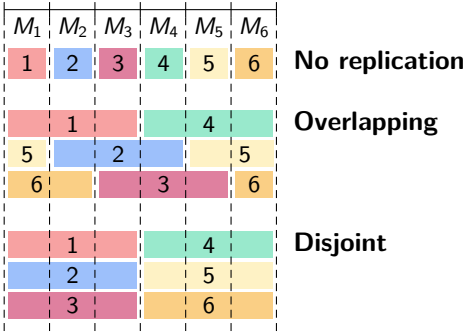
- Overlapping intervals: key-value stores.



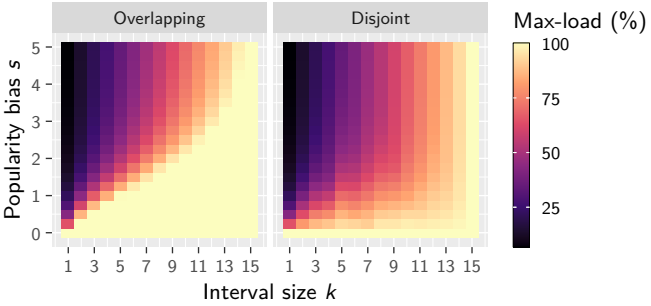
Maximum Load under Biased Popularity

What is the theoretical maximum load λ one can achieve when data are replicated?

- Overlapping intervals: key-value stores.
- Disjoint intervals: good competitive ratio.

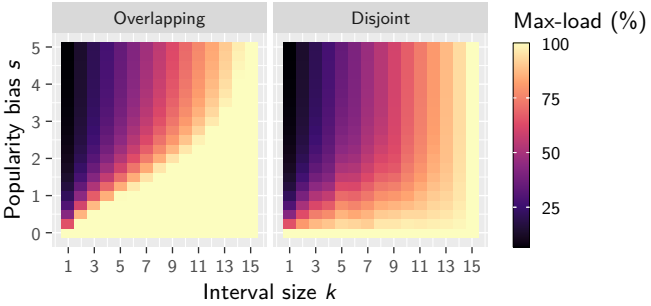


Maximum Load under Biased Popularity



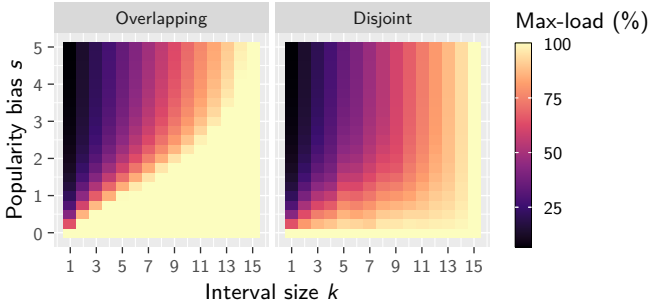
- Popularity is modeled by Zipf distribution with parameter s.

Maximum Load under Biased Popularity



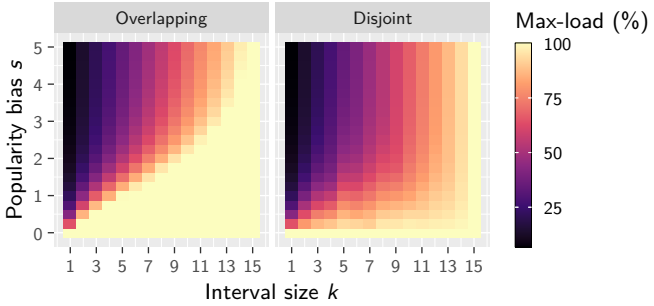
- Popularity is modeled by Zipf distribution with parameter s.
- 15 machines.

Maximum Load under Biased Popularity



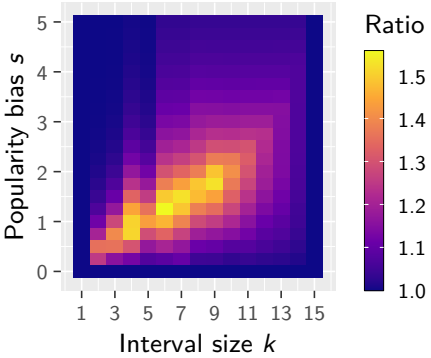
- Popularity is modeled by Zipf distribution with parameter s.
- 15 machines.
- Interval size k ranges from 1 to 15.

Maximum Load under Biased Popularity



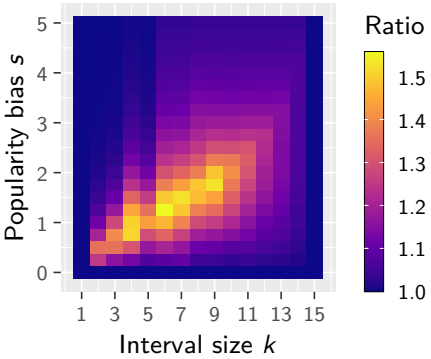
- Popularity is modeled by Zipf distribution with parameter s.
- 15 machines.
- Interval size k ranges from 1 to 15.
- Maximization of load λ is solved with a Linear Program.

Maximum Load under Biased Popularity



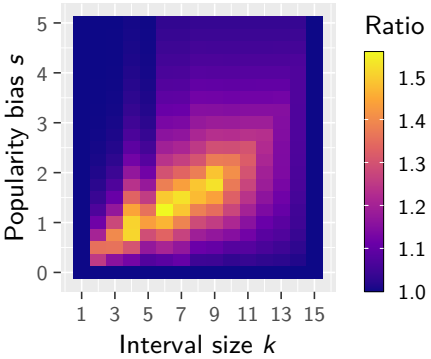
- Ratio between overlapping and disjoint cases.

Maximum Load under Biased Popularity



- Ratio between overlapping and disjoint cases.
- Load up to 50% higher with overlapping intervals.

Maximum Load under Biased Popularity



- Ratio between overlapping and disjoint cases.
- Load up to 50% higher with overlapping intervals.
- Load up to 35% higher with overlapping intervals for common parameters in key-value stores (3 replicas and moderate bias).

Table of Contents

1. Introduction

2. Processing Set Structures and Related Bounds

3. Maximum Load under Biased Popularity

4. Conclusion

Conclusion

- Competitive ratios often increase with the number of machines m, even with regular structures in processing sets.

Conclusion

- Competitive ratios often increase with the number of machines m, even with regular structures in processing sets.
- EFT has a strong guarantee for disjoint processing sets $(3 - 2/k)$ but suffers from overlapping intervals (lower bound: $m - k + 1$).

Conclusion

- Competitive ratios often increase with the number of machines m, even with regular structures in processing sets.
- EFT has a strong guarantee for disjoint processing sets $(3 - 2/k)$ but suffers from overlapping intervals (lower bound: $m - k + 1$).
- However, overlapping intervals enable a theoretical load up to 50% higher than disjoint intervals when popularity biases are introduced.

Anand, S., Bringmann, K., Friedrich, T., Garg, N., and Kumar, A. (2017).
Minimizing maximum (weighted) flow-time on related and unrelated machines.
Algorithmica, 77(2):515–536.

Bender, M. A., Chakrabarti, S., and Muthukrishnan, S. (1998).
Flow and stretch metrics for scheduling continuous job streams.
In *SODA*, volume 98, pages 270–279.