Solving the Restricted Assignment Problem to Schedule Multi-Get Requests in Key-Value Stores

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

Anthony Dugois

FEMTO-ST, University of Franche-Comté

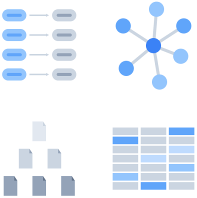
Besançon, France

Euro-Par Conference 2024

Madrid, Spain

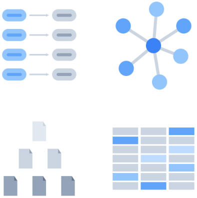
August 29, 2024

Applicative Context: Persistent Key-Value Stores



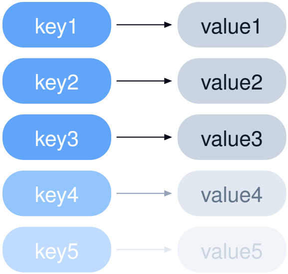
Graph Databases
Document Stores
Wide-Column Databases
In-Memory Key-Value Store
Persistent Key-Value Stores
...

Applicative Context: Persistent Key-Value Stores



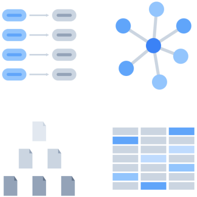
Graph Databases
Document Stores
Wide-Column Databases
In-Memory Key-Value Store
Persistent Key-Value Stores
...

(Persistent) Key-Value Stores



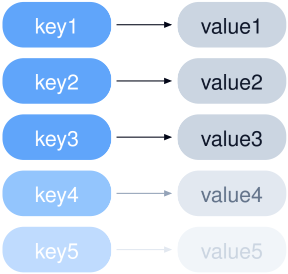
`get(key1)`   `put(key2,value2)`

Applicative Context: Persistent Key-Value Stores



Graph Databases
Document Stores
Wide-Column Databases
In-Memory Key-Value Store
Persistent Key-Value Stores
...

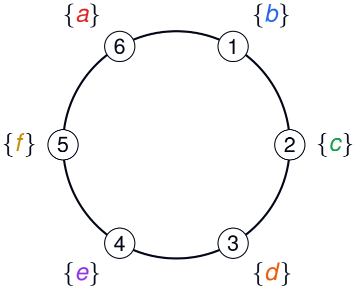
(Persistent) Key-Value Stores



`get(key1)` `put(key2,value2)`

`mget(key1, key2, key4)`

Multi-Get Requests in Distributed Key-Value Stores

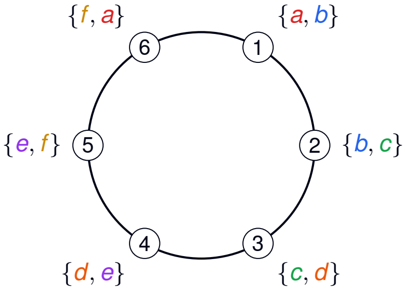


- Each server holds a data partition

◯ Servers of the key-value stores

$\{x, y\}$ Data partition of the server

Multi-Get Requests in Distributed Key-Value Stores

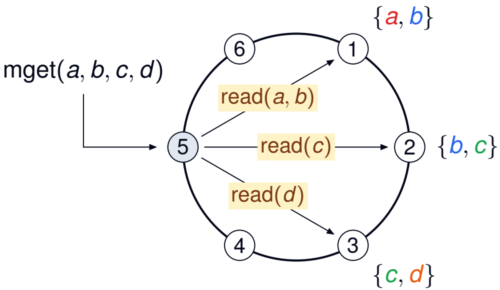


- Each server holds a data partition
- Data replicated on k successor servers

◯ Servers of the key-value stores

$\{x, y\}$ Data partition of the server

Multi-Get Requests in Distributed Key-Value Stores

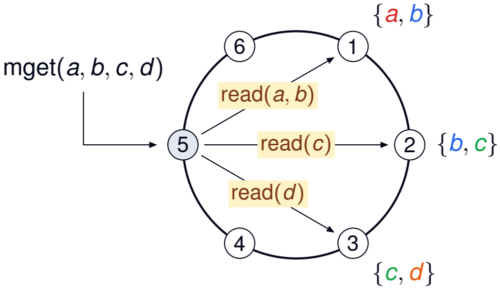


- Each server holds a data partition
- Data replicated on k successor servers
- Scheduling a **multi-get** request implies splitting into sub-requests to read data

◯ Servers of the key-value stores

$\{x, y\}$ Data partition of the server

Multi-Get Requests in Distributed Key-Value Stores



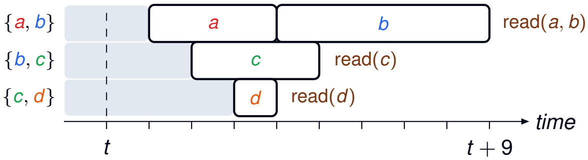
◯ Servers of the key-value stores

$\{x, y\}$ Data partition of the server

- Each server holds a data partition
- Data replicated on k successor servers
- Scheduling a **multi-get** request implies splitting into sub-requests to read data

→ Splitting is not trivial: reading different items takes different times, and server loads are different

Multi-Get Request Splitting

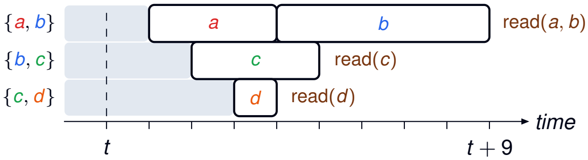


$C_{\max} = t + 9$, non-optimal

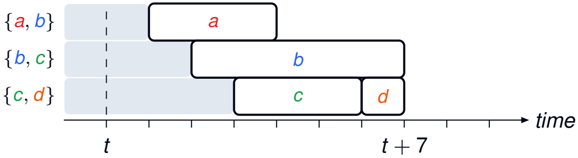
$C_{\max}$ = maximum completion time of read operations (makespan)

t = arriving time of the multi-get request

Multi-Get Request Splitting



$C_{\max} = t + 9$, non-optimal



$C_{\max} = t + 7$, optimal

$C_{\max}$ = maximum completion time of read operations (makespan)

t = arriving time of the multi-get request

(Almost) Equivalent Scheduling Problem

Restricted Assignment on Intervals (RAI)

- **Input**:
  - n jobs (= read operations)
  - m identical machines (= servers)
  - processing times p_j (= times to read items)
  - intervals of compatible machines $\langle a_j, b_j \rangle = \{a_j, a_j + 1, \cdots, b_j\}$
- **Output**: sets of jobs J_i to put on each machine i
- **Objective**: minimize makespan

Results

- **Existing work**:
  - 2-approximation for $R \,||\, C_{\max}$ [Lenstra et al., 1990]
  - $(2 - 1/2^k)$-approximation for $P \,|\, \mathcal{M}_j \,|\, C_{\max}$ when $p_j \in \{1, 2, \cdots, 2^k\}$ [Biró et al., 2014]
  - No PTAS for RAI unless $P = NP$ [Maack et al., 2020]
  - Optimal algorithm for RAI when $p_j = 1$ and m is fixed [Lin et al., 2004]

$\rightarrow$ Real-time scheduling: cannot afford too complex algorithms

Results

- **Existing work**:
  - 2-approximation for $R \,||\, C_{\max}$ [Lenstra et al., 1990]
  - $(2 - 1/2^k)$-approximation for $P \,|\, \mathcal{M}_j \,|\, C_{\max}$ when $p_j \in \{1, 2, \cdots, 2^k\}$ [Biró et al., 2014]
  - No PTAS for RAI unless $P = NP$ [Maack et al., 2020]
  - Optimal algorithm for RAI when $p_j = 1$ and m is fixed [Lin et al., 2004]

  $\rightarrow$ Real-time scheduling: cannot afford too complex algorithms

- **In this talk**:

  $\rightarrow$ Efficient $(2 - 1/m)$-approximation for RAI

  $\rightarrow$ Efficient $(4 - 2/m)$-approximation for a generalized version

  $\rightarrow$ Efficient and qualitative heuristics

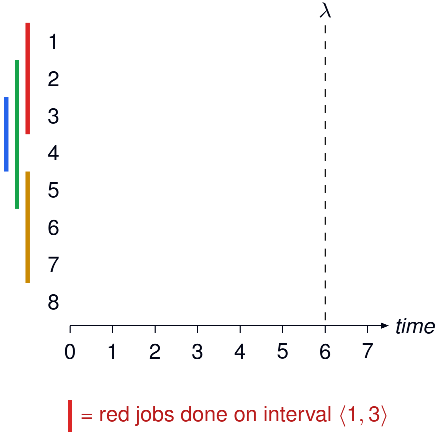
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



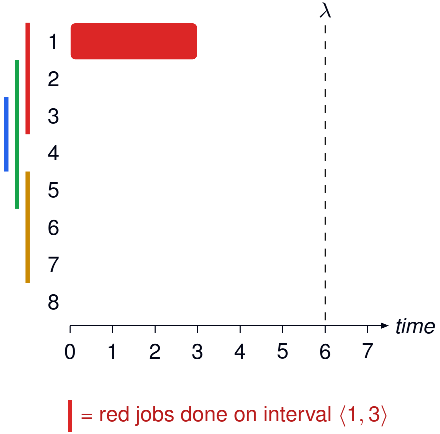
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



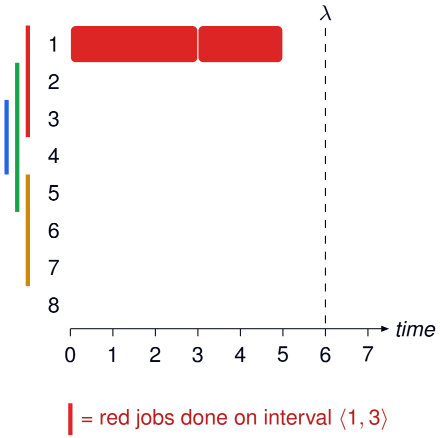
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



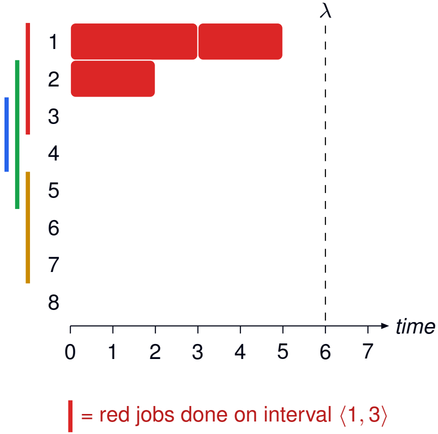
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



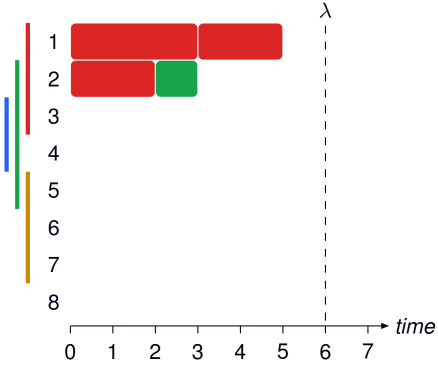
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



= red jobs done on interval $\langle 1, 3 \rangle$

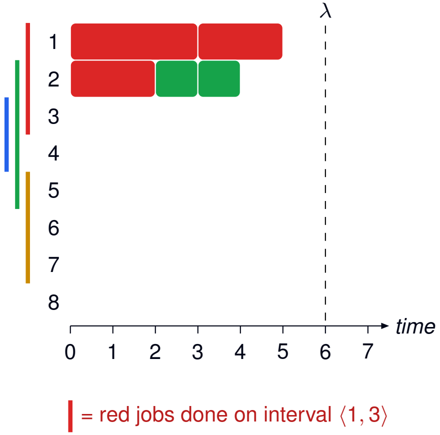
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



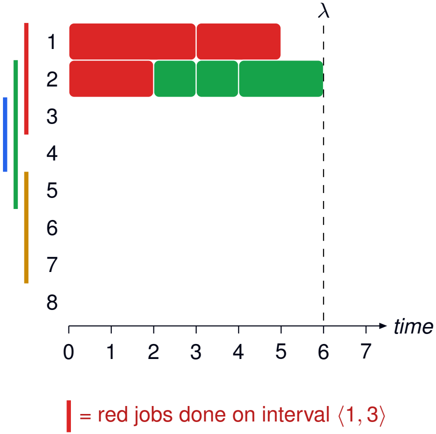
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



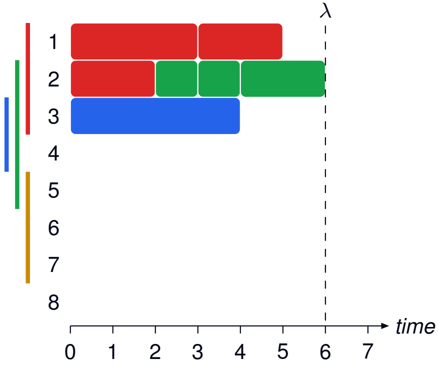
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



= red jobs done on interval $\langle 1, 3 \rangle$

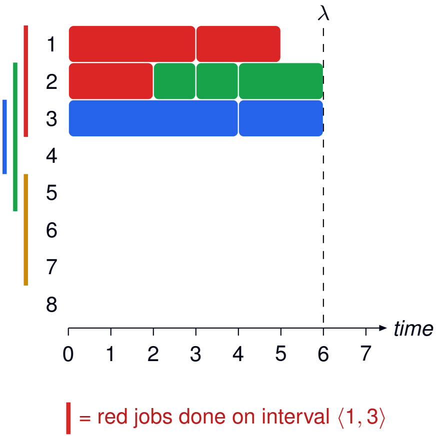
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



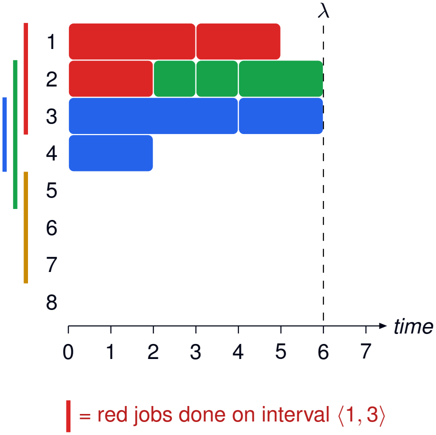
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



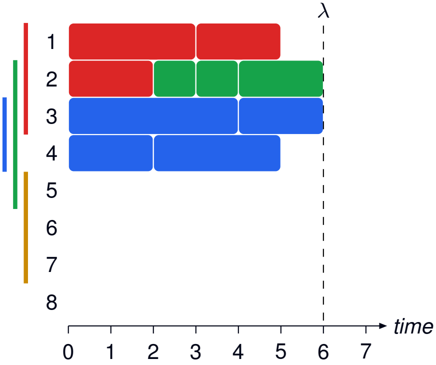
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



= red jobs done on interval $\langle 1, 3 \rangle$

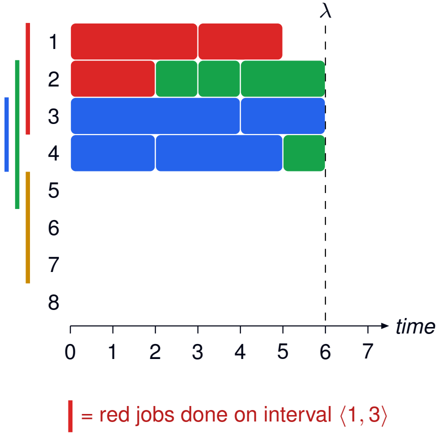
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



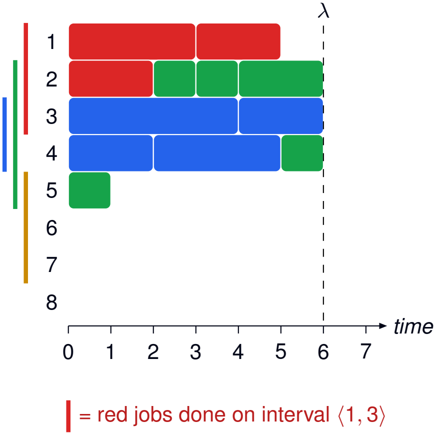
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



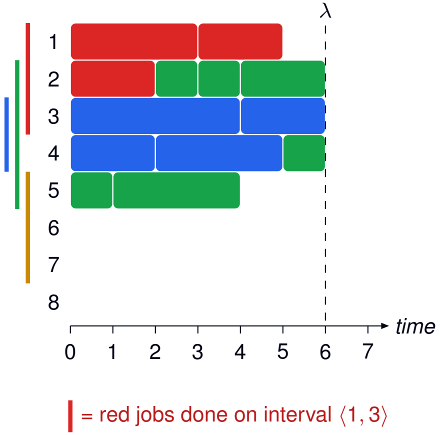
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



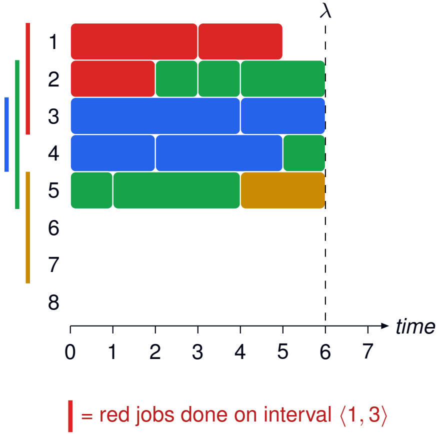
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



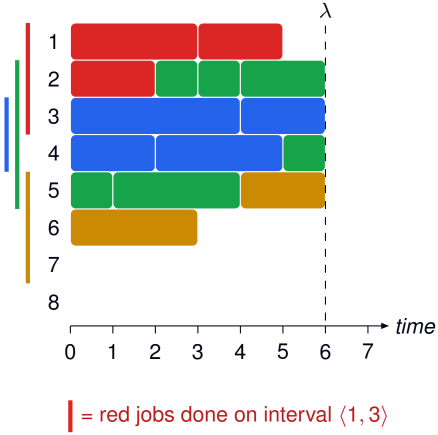
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



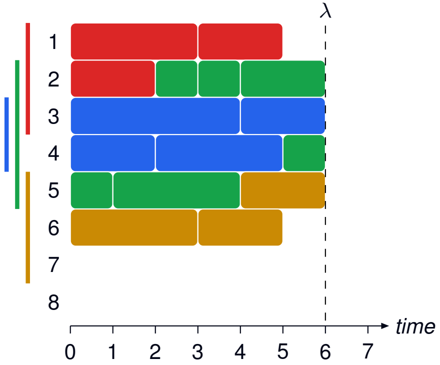
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



= red jobs done on interval $\langle 1, 3 \rangle$

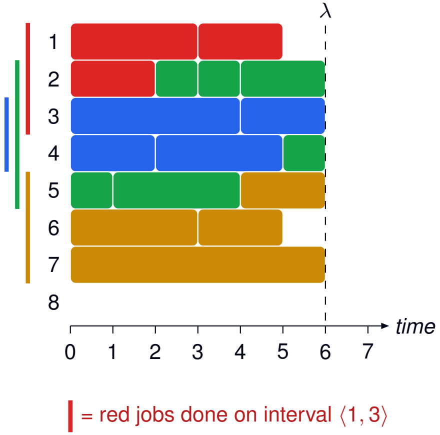
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to compute λ



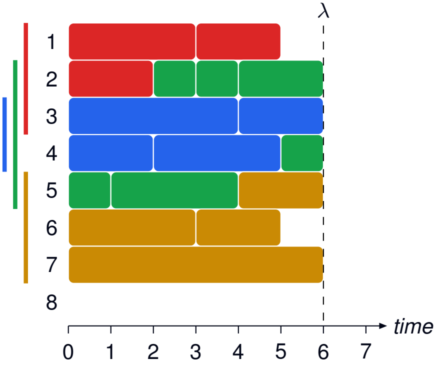
Estimated Least Flexible Job

ELFJ Algorithm

Parameter: estimated makespan λ.

Starting from the first machine, build a schedule that finishes no later than time λ by processing in priority the most constrained job.

Time complexity: $O(n \log n + mn)$ + time to **compute λ**



= red jobs done on interval $\langle 1, 3 \rangle$

Result

Computing Makespan λ

- $\lambda = \tilde{w}_{\text{max}} + (1 - 1/m)\, p_{\text{max}}$
- p_{max} = maximum processing time among jobs
- $\tilde{w}_{\text{max}} = \max_{\alpha \leq \beta} \tilde{w}_{\langle \alpha, \beta \rangle}$ where $\tilde{w}_{\langle \alpha, \beta \rangle}$ is the average amount of work that must be done on interval $\langle \alpha, \beta \rangle$

Result

Computing Makespan λ

- $\lambda = \tilde{w}_{\text{max}} + (1 - 1/m)\, p_{\text{max}}$
- p_{max} = maximum processing time among jobs
- $\tilde{w}_{\text{max}} = \max_{\alpha \leq \beta} \tilde{w}_{\langle \alpha, \beta \rangle}$ where $\tilde{w}_{\langle \alpha, \beta \rangle}$ is the average amount of work that must be done on interval $\langle \alpha, \beta \rangle$

Theorem

ELFJ is a $(2 - 1/m)$-approximation algorithm for RAI with arbitrary jobs.

Result

Computing Makespan λ

- $\lambda = \tilde{w}_{\text{max}} + (1 - 1/m)\, p_{\text{max}}$
- p_{max} = maximum processing time among jobs
- $\tilde{w}_{\text{max}} = \max_{\alpha \leq \beta} \tilde{w}_{\langle \alpha, \beta \rangle}$ where $\tilde{w}_{\langle \alpha, \beta \rangle}$ is the average amount of work that must be done on interval $\langle \alpha, \beta \rangle$

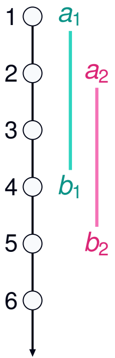
  $\rightarrow$ Can be computed in time $O(m^2 + n)$ with dynamic programming

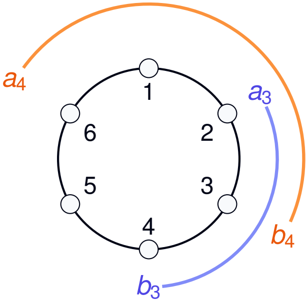
Theorem

ELFJ is a $(2 - 1/m)$-approximation algorithm for RAI with arbitrary jobs.

Runs in time $O(m^2 + n \log n + mn)$.

Generalizing the Problem with Circular Intervals





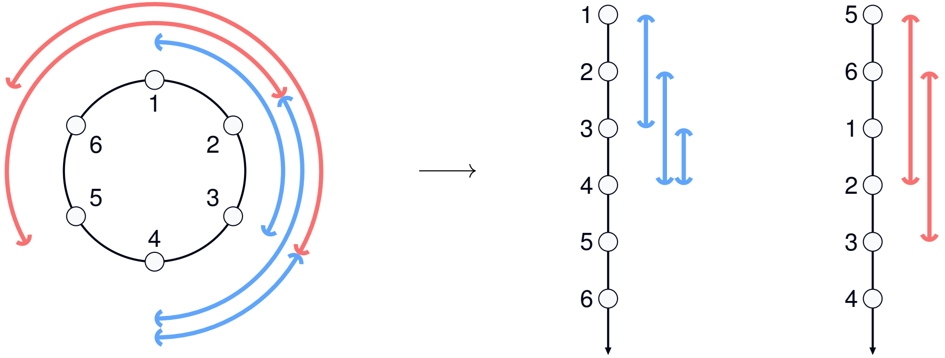
Regular RAI problem

$$\langle \alpha, \beta \rangle = \{\alpha, \alpha + 1, \cdots, \beta\}$$

Circular RAI problem

$$\langle \alpha, \beta \rangle = \begin{cases} \{\alpha, \alpha + 1, \cdots, \beta\} & \text{if } \alpha \leq \beta \\ \{\alpha, \cdots, m\} \cup \{1, \cdots, \beta\} & \text{otherwise.} \end{cases}$$

Splitting the Circular Problem into Regular Sub-Problems



Circular instance $\mathcal{I}$
with all jobs

Regular sub-instance $\mathcal{I}_1$
with *regular* jobs

Regular sub-instance $\mathcal{I}_2$
with *circular* jobs

Result

Double ELFJ Algorithm (DELFJ)

Apply ELFJ for regular jobs, then for the circular ones, and merge schedules.

Result

Double ELFJ Algorithm (DELFJ)

Apply ELFJ for regular jobs, then for the circular ones, and merge schedules.

Theorem

DELFJ is a $(4 - 2/m)$-approximation algorithm for circular RAI.

Proof sketch:

1. By previous theorem:
   - $\text{ELFJ}(\mathcal{I}_1) \leq (2 - 1/m)\,\text{OPT}(\mathcal{I}_1)$
   - $\text{ELFJ}(\mathcal{I}_2) \leq (2 - 1/m)\,\text{OPT}(\mathcal{I}_2)$
2. Moreover, $\text{OPT}(\mathcal{I}_1) \leq \text{OPT}(\mathcal{I})$ and $\text{OPT}(\mathcal{I}_2) \leq \text{OPT}(\mathcal{I})$
3. Finally, $\text{DELFJ}(\mathcal{I}) \leq \text{ELFJ}(\mathcal{I}_1) + \text{ELFJ}(\mathcal{I}_2) \leq (4 - 2/m)\,\text{OPT}(\mathcal{I})$

Searched Least Flexible Job

SLFJ Algorithm

Progressively searches for a feasible makespan λ by successively applying ELFJ.

```
compute w̃_max as if jobs were unitary
δ ← 0
repeat
    apply ELFJ with λ = ⌈w̃_max⌉ + δ
    δ ← INCREASE(δ)
until all jobs are assigned
```

Note 1: this terminates because it always finds a solution when $\delta \geq p_{\max}$.

Note 2: quality and speed both depend on the INCREASE function.

Heuristics

Two Variants

- Arithmetic SLFJ
  - INCREASE : $\delta \to \delta + 1$
  - Better quality, slower convergence
- Geometric SLFJ
  - INCREASE : $\delta \to \max(1, 2\delta)$
  - OK quality, faster convergence

Simulation Setup

Baseline

- RANDOM: randomly assign each job to a compatible machine
- EFT-MIN: assign each job to the *first* compatible machine that completes it the earliest
- EFT-RAND: same as EFT-MIN, but with a randomized tie-break

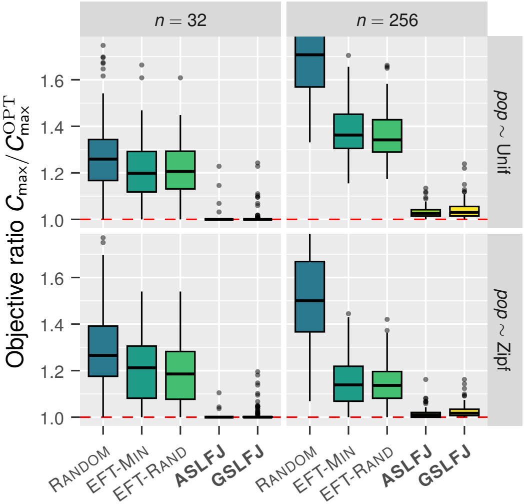
Instance generation

- 48 servers
- Replication factor of 3
- 100 000 keys
- Reading times drawn from exponential distribution
- Two parameters:
  - n = number of read operations in a multi-get request
  - pop = popularity distribution of the keys

Makespan of Each Multi-Get Request

- Ratio between makespan of **each multi-get request** and optimal solution (lower is better)
- Fixed sizes (32 and 256)
- Popularity distributions:
  - Unif: keys have the same probability to be requested
  - Zipf: the probability of a key to be requested is correlated to its rank

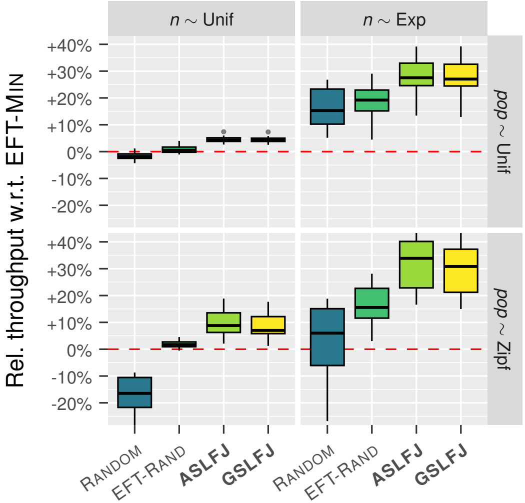
→ GSLFJ converges faster, with a quality almost equal to ASLFJ



Throughput of a Stream of Requests

- Ratio between throughput of **a stream of requests** and solution given by baseline heuristic EFT-MIN (higher is better)
- Size distributions:
  - Unif (between 1 and 256)
  - Exp: “small” requests more probable

→ Optimizing multi-get requests individually leads to global improvements



Conclusion

- $(2 - 1/m)$-approx. for RAI
- Circular (generalized) version of RAI + $(4 - 2/m)$-approx.
- Heuristics give close-to-optimal solutions in practice
- Optimizing individual requests leads to global improvements

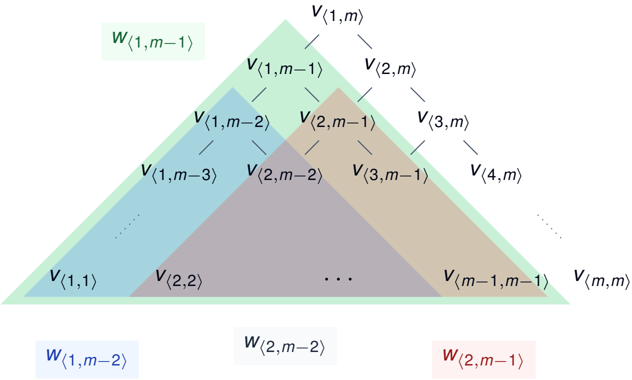
- **Perspectives**:
  - Is there a better approx. for circular RAI?
  - What guarantees can we have if only estimations are available for p_j?
  - Experiment heuristics in actual systems

Thank you!

Read the paper

Reach me at **anthony.dugois@univ-fcomte.fr**

Computing Makespan λ



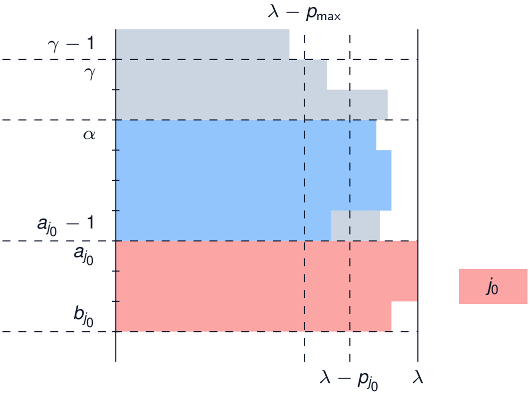
- $v_{\langle \alpha,\beta \rangle}$: total work of jobs j such that $a_j = \alpha$ and $b_j = \beta$.
- $w_{\langle \alpha,\beta \rangle}$: total work of jobs j such that $\alpha \leq a_j \leq b_j \leq \beta$.

Dynamic programming: $w_{\langle \alpha,\beta \rangle} = v_{\langle \alpha,\beta \rangle} + w_{\langle \alpha,\beta-1 \rangle} + w_{\langle \alpha+1,\beta \rangle} - w_{\langle \alpha+1,\beta-1 \rangle}$

Time complexity: $O(m^2 + n)$

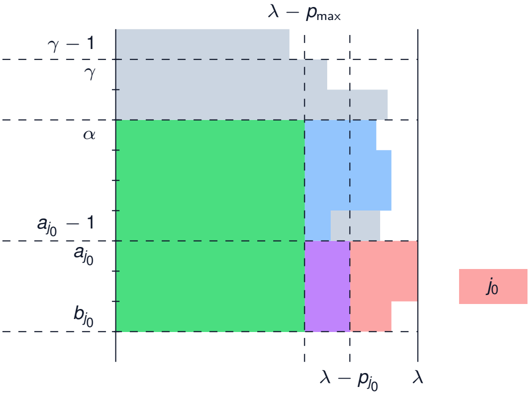
Proof Sketch

1. $\lambda = \tilde{w}_{\max} + (1 - 1/m)\, p_{\max}$
2. Suppose by contradiction that a job j_0 cannot be scheduled before λ
3. Progressively search for a machine α such that almost all the work done on $\alpha, \alpha + 1, \cdots, b_{j_0}$ is included in $w_{\langle \alpha, b_{j_0} \rangle}$
   - either all jobs done on $\langle a_{j_0}, b_{j_0} \rangle$ are included in $w_{\langle a_{j_0}, b_{j_0} \rangle}$, or
   - there is a job j_1 done on $\langle a_{j_0}, b_{j_0} \rangle$ such that $a_{j_1} < a_{j_0}$

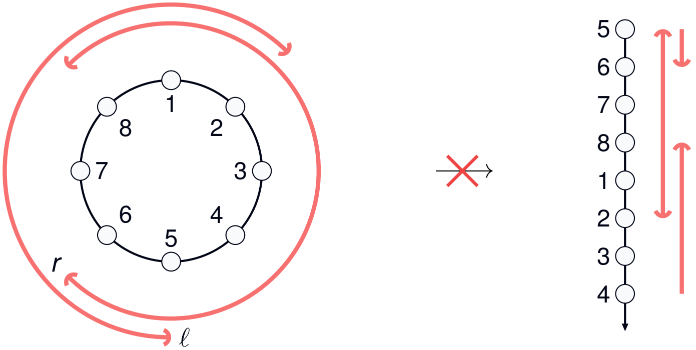


Proof Sketch

1. $\lambda = \tilde{w}_{\max} + (1 - 1/m)\, p_{\max}$
2. Suppose by contradiction that a job j_0 cannot be scheduled before λ
3. Progressively search for a machine α such that almost all the work done on $\alpha, \alpha+1, \cdots, b_{j_0}$ is included in $w_{\langle \alpha, b_{j_0} \rangle}$
   - either all jobs done on $\langle a_{j_0}, b_{j_0} \rangle$ are included in $w_{\langle a_{j_0}, b_{j_0} \rangle}$, or
   - there is a job j_1 done on $\langle a_{j_0}, b_{j_0} \rangle$ such that $a_{j_1} < a_{j_0}$
4. $w_{\langle \alpha, b_{j_0} \rangle} > (b_{j_0} - \alpha + 1)(\lambda - p_{\max}) + (b_{j_0} - a_{j_0} + 1)(\lambda - p_{j_0} - (\lambda - p_{\max})) + p_{j_0}$,
5. Leads to $\lambda < \tilde{w}_{\langle \alpha, b_{j_0} \rangle} + (1 - 1/m)\, p_{\max}$
6. $\rightarrow$ Contradicts 1.



Necessary Condition for Splitting



$\mathcal{I}_2$ cannot be regularized if $\ell \leq r$