A log-linear $(2 + 5/6)$-approximation algorithm for parallel machine scheduling with a single orthogonal resource

Adrian Naruszko Bartłomiej Przybylski Krzysztof Rządca

University of Warsaw, Faculty of Mathematics, Informatics and Mechanics

Aussois, May 14th, 2024

Introduction - Burst buffers

- Burst buffers are an intermediate layer between CPU and filesystem shared by all supercomputer nodes (in some architectures at least, see the Euro-Par paper [KR21])
- They offer low latency and high bandwidth, providing much better performance than the filesystem itself (e.g. 4x faster write on supercomputer Summit, average time spent on I/O reduced from 8% to 3% of application runtime [20])
- Due to the high price of the solution, they are of limited capacity

Introduction - Burst buffers

- Burst buffers are an intermediate layer between CPU and filesystem shared by all supercomputer nodes (in some architectures at least, see the Euro-Par paper [KR21])
- They offer low latency and high bandwidth, providing much better performance than the filesystem itself (e.g. 4x faster write on supercomputer Summit, average time spent on I/O reduced from 8% to 3% of application runtime [20])
- Due to the high price of the solution, they are of limited capacity

Introduction - Burst buffers

- Burst buffers are an intermediate layer between CPU and filesystem shared by all supercomputer nodes (in some architectures at least, see the Euro-Par paper [KR21])
- They offer low latency and high bandwidth, providing much better performance than the filesystem itself (e.g. 4x faster write on supercomputer Summit, average time spent on I/O reduced from 8% to 3% of application runtime [20])
- Due to the high price of the solution, they are of limited capacity

Introduction - Why do we need burst buffers?

- On supercomputers, rate of CPU speed growth outperforms the development done in the storage layer
- Comparing the Oak Ridge Laboratory machines, each new generation gets FLOPS ratio about 10 times greater, but the IO throughput grows only 2-4 times compared to the previous one [24]

Introduction - Burst buffers as an orthogonal resource

- Every job may require some amount of burst buffers
- Scheduler must ensure that in every moment sum of those requirements does not exceed burst buffer capacity
- It means burst buffers introduce orthogonal resource constraint on the schedule
- The scheduling problem with orthogonal resource is NP-hard in general, but we may approximate

Introduction - Burst buffers as an orthogonal resource

- Every job may require some amount of burst buffers
- Scheduler must ensure that in every moment sum of those requirements does not exceed burst buffer capacity
- It means burst buffers introduce orthogonal resource constraint on the schedule
- The scheduling problem with orthogonal resource is NP-hard in general, but we may approximate

Introduction - Burst buffers as an orthogonal resource

- Every job may require some amount of burst buffers
- Scheduler must ensure that in every moment sum of those requirements does not exceed burst buffer capacity
- It means burst buffers introduce orthogonal resource constraint on the schedule
- The scheduling problem with orthogonal resource is NP-hard in general, but we may approximate

Introduction - Burst buffers as an orthogonal resource

- Every job may require some amount of burst buffers
- Scheduler must ensure that in every moment sum of those requirements does not exceed burst buffer capacity
- It means burst buffers introduce orthogonal resource constraint on the schedule
- The scheduling problem with orthogonal resource is NP-hard in general, but we may approximate

Model description

- Every job i has its processing time $p_i \in \mathbb{R}_+$ and resource requirement $r_i \in [0, 1]$
- There is 1 unit of continuous resource available, shared among all machines
- Goal is to minimize C_{max}, that is the completion time of the last running job

Model description

- Every job i has its processing time $p_i \in \mathbb{R}_+$ and resource requirement $r_i \in [0, 1]$
- There is 1 unit of continuous resource available, shared among all machines
- Goal is to minimize C_{max}, that is the completion time of the last running job

Model description

- Every job i has its processing time $p_i \in \mathbb{R}_+$ and resource requirement $r_i \in [0, 1]$
- There is 1 unit of continuous resource available, shared among all machines
- Goal is to minimize C_{max}, that is the completion time of the last running job

Model description

- Jobs:
  - run on a single machine only (*sequential*)
  - processing times and and resource requirements are known in advance (*clairvoyant*)
  - require fixed amount of resource that cannot change during job execution (*rigid*)
  - are known from the beginning (*off-line*)
  - once scheduled cannot be paused or stopped (*non-preemptable*)
  - run on m identical machines with equal access to the resource

Literature

There are several studies based on the proposed model:

- Garey and Graham gave general approximation guarantee of $3 - 3/m$ for *list schedulers* [GG75]
- Niemeyer and Wiese presented a PTAS with $(2 + \epsilon)$-approximation factor (with additional restrictions) [Nie13]
- Jansen, Raack and Mau presented an asymptotic FPTAS, which produces a schedule of length $(1 + \epsilon)$ OPT $+O(\frac{1}{\epsilon^2})p_{\max}$ [JMR19]

First one is very general, other two are complex!

Is there something in between?

Our goal was to design a scheduling algorithm:

- for a classic scheduling problem (defined by the model proposed earlier)
- with makespan approximation factor of $c < 3 - 3/m$ - there is already a guarantee of $3 - 3/m$ for every *list scheduler* [GG75]
- with reasonable time complexity
- that is considerably easier to implement than current state-of-the-art algorithms [Nie13], [JMR19]
- that is build on top of some basic, well known facts, which interact with each other in a complex way

Is there something in between?

Our goal was to design a scheduling algorithm:

- for a classic scheduling problem (defined by the model proposed earlier)
- with makespan approximation factor of $c < 3 - 3/m$ - there is already a guarantee of $3 - 3/m$ for every *list scheduler* [GG75]
- with reasonable time complexity
- that is considerably easier to implement than current state-of-the-art algorithms [Nie13], [JMR19]
- that is build on top of some basic, well known facts, which interact with each other in a complex way

Is there something in between?

Our goal was to design a scheduling algorithm:

- for a classic scheduling problem (defined by the model proposed earlier)
- with makespan approximation factor of $c < 3 - 3/m$ - there is already a guarantee of $3 - 3/m$ for every *list scheduler* [GG75]
- with reasonable time complexity
- that is considerably easier to implement than current state-of-the-art algorithms [Nie13], [JMR19]
- that is build on top of some basic, well known facts, which interact with each other in a complex way

Is there something in between?

Our goal was to design a scheduling algorithm:

- for a classic scheduling problem (defined by the model proposed earlier)
- with makespan approximation factor of $c < 3 - 3/m$ - there is already a guarantee of $3 - 3/m$ for every *list scheduler* [GG75]
- with reasonable time complexity
- that is considerably easier to implement than current state-of-the-art algorithms [Nie13], [JMR19]
- that is build on top of some basic, well known facts, which interact with each other in a complex way

Is there something in between?

Our goal was to design a scheduling algorithm:

- for a classic scheduling problem (defined by the model proposed earlier)
- with makespan approximation factor of $c < 3 - 3/m$ - there is already a guarantee of $3 - 3/m$ for every *list scheduler* [GG75]
- with reasonable time complexity
- that is considerably easier to implement than current state-of-the-art algorithms [Nie13], [JMR19]
- that is build on top of some basic, well known facts, which interact with each other in a complex way

Tools

The algorithm is based on a few simple observations:

- Optimal schedule length may be bounded with
  - jobs processing time: $OPT \geq \sum p_i/m$
  - jobs resource requirement: $OPT \geq \sum p_i \cdot r_i$
- Jobs without resource requirement can be scheduled in an $4/3 - \frac{1}{3m}$ approximation using LPT routine [Gra66]

Tools

The algorithm is based on a few simple observations:

- Optimal schedule length may be bounded with
  - jobs processing time: $OPT \geq \sum p_i/m$
  - jobs resource requirement: $OPT \geq \sum p_i \cdot r_i$
- Jobs without resource requirement can be scheduled in an $4/3 - \frac{1}{3m}$ approximation using LPT routine [Gra66]

Tools

The algorithm is based on a few simple observations:

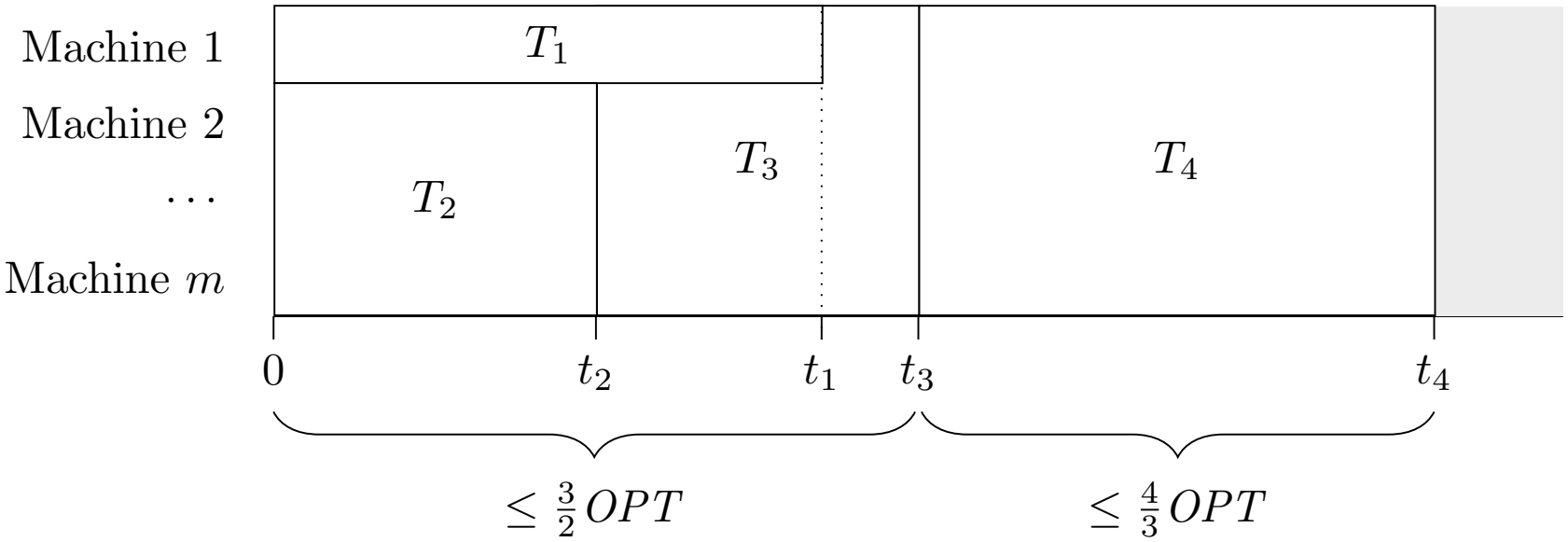
- Optimal schedule length may be bounded with
  - jobs processing time: $OPT \geq \sum p_i/m$
  - jobs resource requirement: $OPT \geq \sum p_i \cdot r_i$
- Jobs without resource requirement can be scheduled in an $4/3 - \frac{1}{3m}$ approximation using LPT routine [Gra66]

Tools

The algorithm is based on a few simple observations:

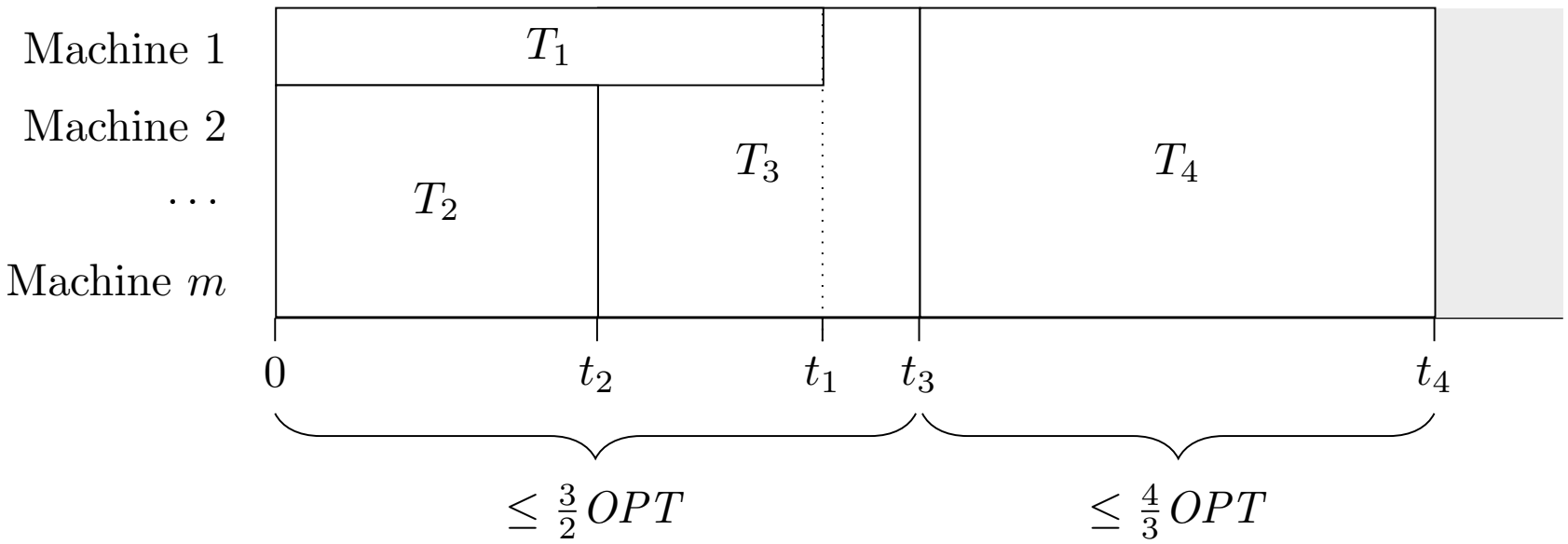
- Optimal schedule length may be bounded with
  - jobs processing time: $OPT \geq \sum p_i/m$
  - jobs resource requirement: $OPT \geq \sum p_i \cdot r_i$
- Jobs without resource requirement can be scheduled in an $4/3 - \frac{1}{3m}$ approximation using LPT routine [Gra66]

A sketch of our schedule



In the first phase (steps 1, 2, 3) we fill T_1, T_2 and T_3 blocks in such a way that $t_3 \leq 3/2$ OPT. In the second phase (step 4) we fill T_4 in such a way that $t_4 \leq (2\frac{5}{6} - \frac{1}{3m})$ OPT. We make sure that the resource is no longer a constraint before the second phase.

A sketch of our schedule

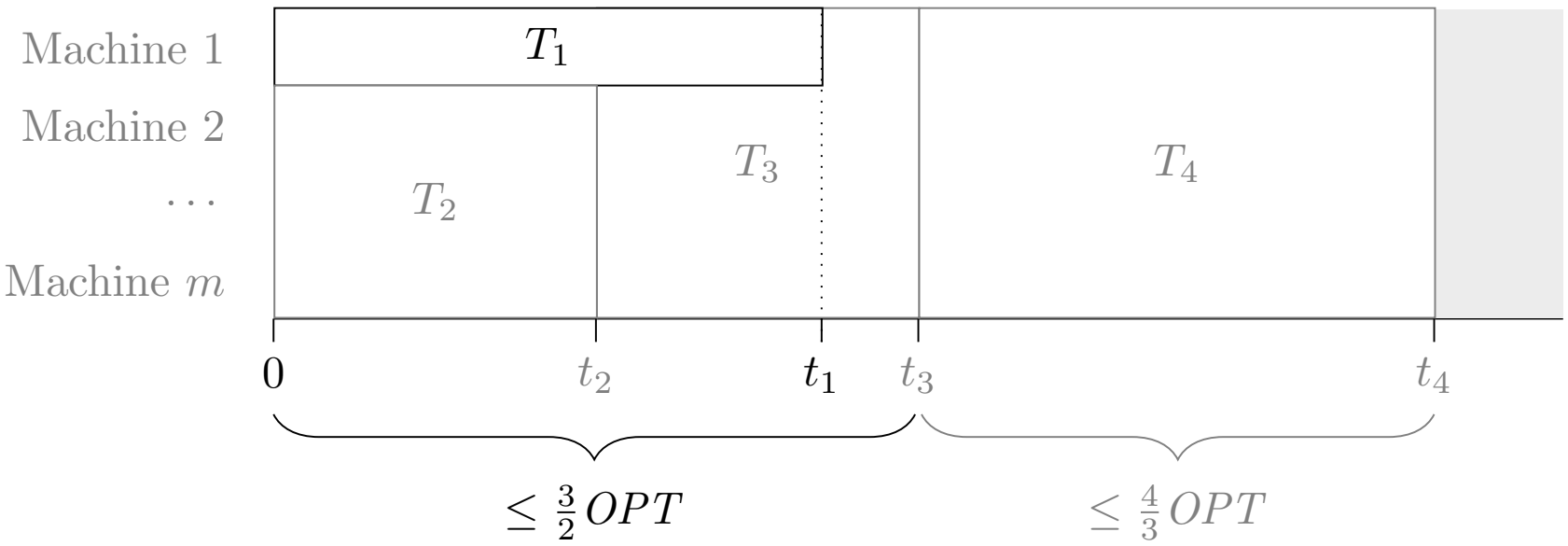


We achieve $\leq 3/2 OPT$ when:

- All machines are occupied OR
- At least 2/3 of the resource is used in every moment

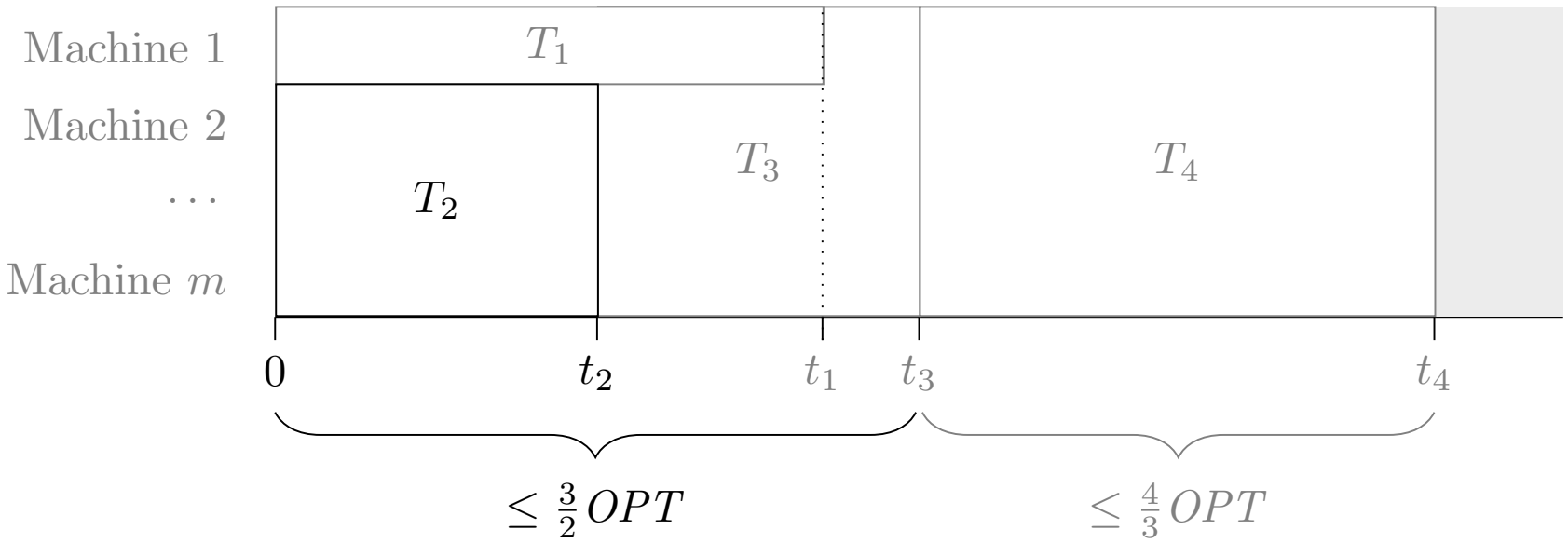
Step 1: Schedule *heavy* jobs

Heavy jobs ($r_i > 1/2$) are scheduled in weakly decreasing order of their resource requirements



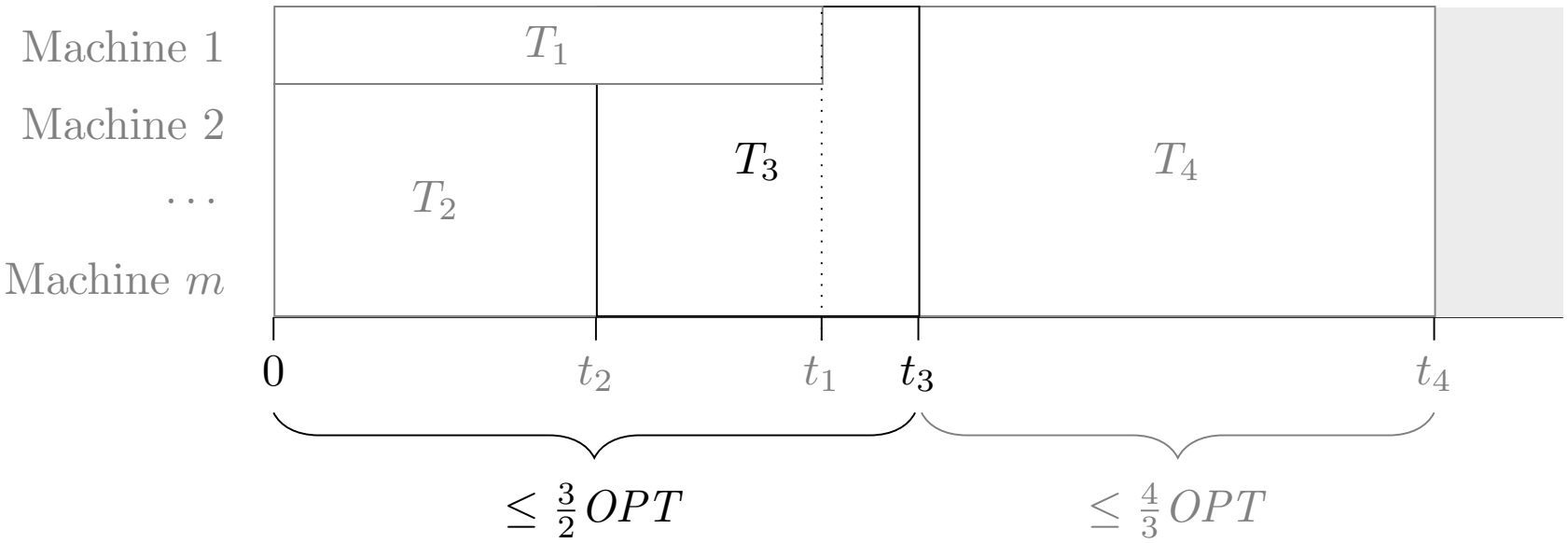
Step 2: Schedule *light* jobs

Light jobs ($r_i \leq 1/3$) are scheduled along *heavy* ones - in strict order given by their resource requirements - until at least 2/3 of the resource is used at a time by all jobs combined



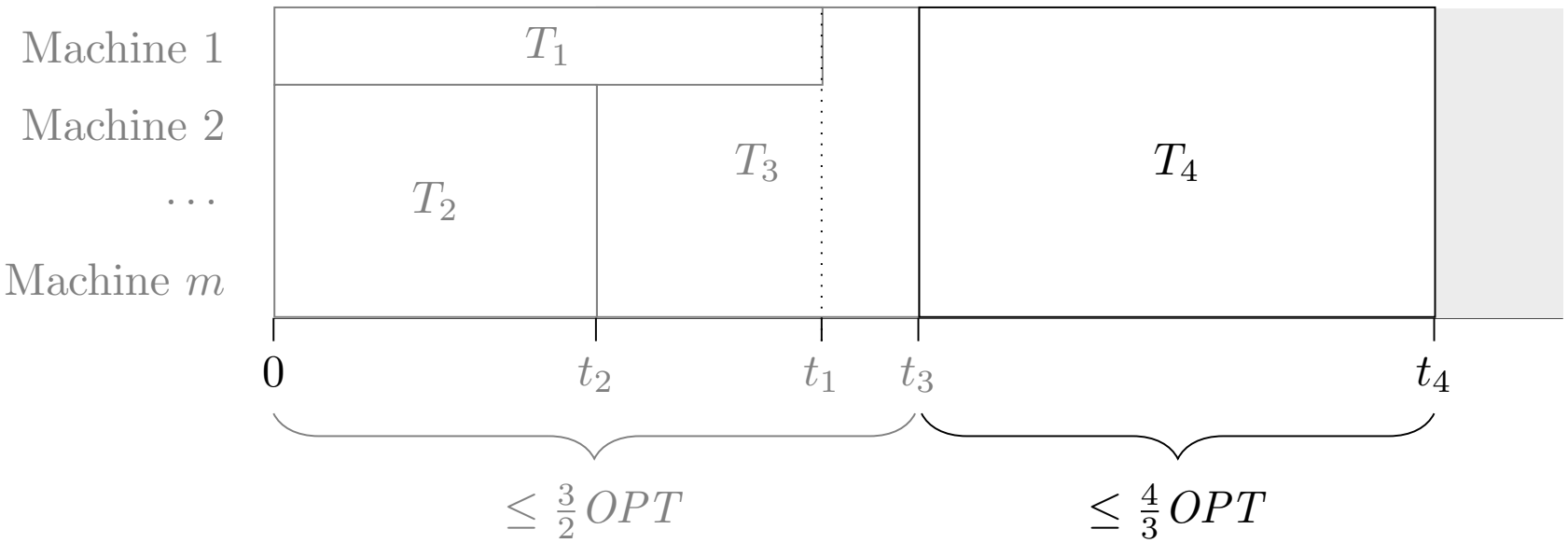
Step 3: Schedule *medium* jobs and remaining *light* jobs

- *Medium* jobs ($1/3 < r_i \leq 1/2$) are scheduled by a *list scheduler* along and after *heavy* jobs on at most two machines, in order of weakly increasing resource requirement
- *Light* jobs are scheduled along *medium* ones the same way as previously



Step 4: Schedule all remaining jobs using *LPT* routine

- After steps 1-3 we still have jobs to schedule.
- However, by careful design we make sure that every m jobs require not more than the whole resource.
- Thus we schedule them using *LPT* routine as for jobs without resource requirements.



Experiments

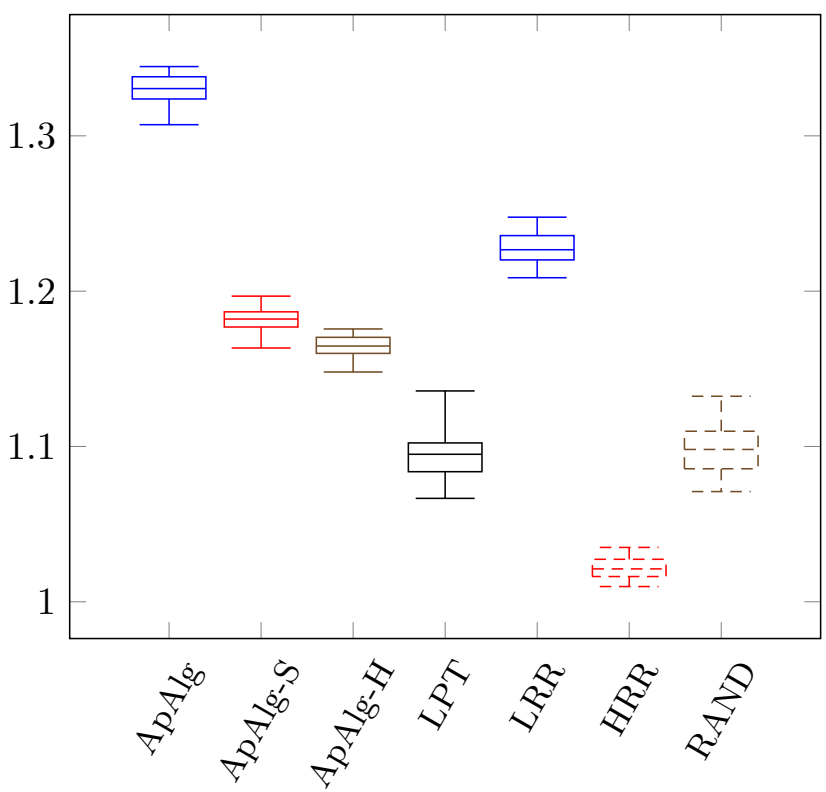
In order to compare the algorithm with existing solutions we have tested 3 versions of the algorithm and some *list scheduler* variations:

- *ApAlg* - pure implementation of our algorithm
- *ApAlgS* & *ApAlgH* - two heuristics based on *ApAlg*, that improve its behavior in average case
- *LPT - Longest Processing Time* prioritized by p_i
- *HRR - Highest Resource Requirement* prioritized by r_i
- *LRR - Lowest Resource Requirement* prioritized by $-r_i$
- *RAND - Random List Scheduler* prioritized by random order

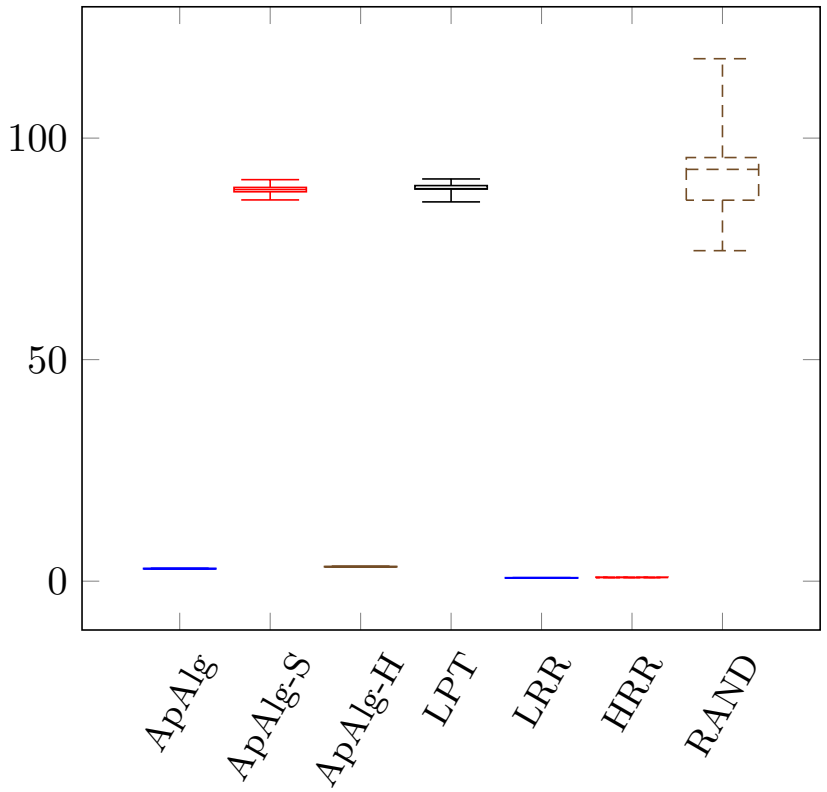
For every pair of parameters (various numbers up to 10000 jobs and 100 machines) we tested them on 30 instances derived from real data from Czech MetaCentrum Supercomputer Grid [15].

Experiment results

Tests were performed on various params, below we show the largest test with 10000 jobs and 100 machines. The $C_{\max}$ values are normalized by the lower-bounds on the optimal schedule lengths.



(a) Makespan to lower-bound ratio



(b) Runtime [s]

Conclusions

- Presented algorithm has approximation factor limited by $2\frac{5}{6} - \frac{1}{3m}$, which, for large m, is better than $3 - \frac{3}{m}$ guarantee for *list schedulers*
- It is base on list scheduling, so it is easy to implement
- Average-case performance factor seems to be far better than the theoretical bound for all tested algorithms
- Majority of tested *list schedulers* are performing better than *ApAlg*
- It is not clear whether presented *list schedulers* may be forced to produce schedule, which length is close to the mentioned guarantees
- Due to its log-linear time complexity presented algorithm may be used in combination with *list schedulers*

Bibliography

[Gra66] R. L. Graham. "Bounds for certain multiprocessing anomalies.". In: *Bell System Technical Journal* 45 (1966), pp. 1563–1581 (cit. on pp. 20–23).

[GG75] M. R. Garey and R. L. Graham. "Bounds for multiprocessor scheduling with resource constraints". In: *SIAM Journal on Computing* 4 (1975), pp. 187–200 (cit. on pp. 14–19).

[Nie13] Wiese A. Niemeier M. "Scheduling with an Orthogonal Resource Constraint.". In: *Erlebach T., Persiano G. (eds) Approximation and Online Algorithms.* 7846 (2013) (cit. on pp. 14–19).

[15] *MetaCentrum workload logs*. https://www.cs.huji.ac.il/labs/parallel/workload/l_metacentrum2/index.html. Feb. 2015 (cit. on p. 30).

[JMR19] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation Schemes for Machine Scheduling with Resource (In-)Dependent Processing Times". In: 15.3 (2019). ISSN: 1549-6325. DOI: 10.1145/3302250 (cit. on pp. 14–19).

[20] *Burst buffers parameters on summit*. https://www.olcf.ornl.gov/wp-content/uploads/2020/02/Burst_Buffer_Training_June2020.pdf. 2020 (cit. on pp. 2–4).

[KR21] Jan Kopański and Krzysztof Rządca. "Plan-based Job Scheduling for Supercomputers with Shared Burst Buffers". In: *Euro-Par* (2021) (cit. on pp. 2–4).

[24] *Burst buffers parameters on oak ridge supercomputers*. https://www.olcf.ornl.gov/frontier/. 2024 (cit. on p. 5).