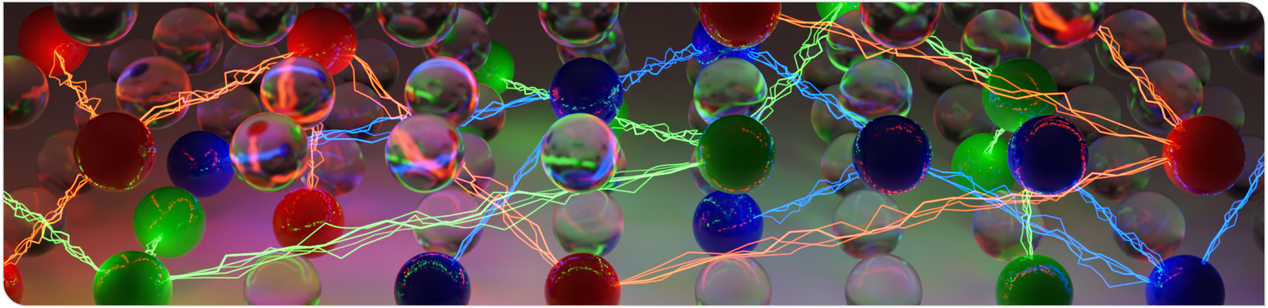
Scalable SAT Solving on Demand

Highlights of Parallel Computing | ALGO 2024, Nantes

Dominik Schreiber, Peter Sanders | June 17, 2024



Motivation: SAT Solving

The NP-complete problem SAT [Cook 1971]

Given a propositional formula $F := \bigwedge_{c \in C} \left(\bigvee_{\ell \in c} \ell \right)$, find a satisfying variable assignment for F or report unsatisfiability.

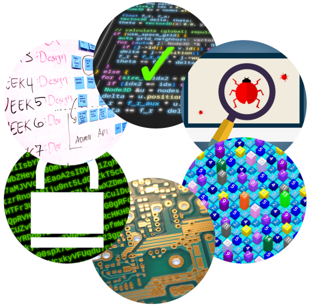
Motivation: SAT Solving

The NP-complete problem SAT [Cook 1971]

Given a propositional formula $F := \bigwedge_{c \in C} \left(\bigvee_{\ell \in c} \ell \right)$, find a satisfying variable assignment for F or report unsatisfiability.

SAT Solving: Fundamental building block for plethora of applications

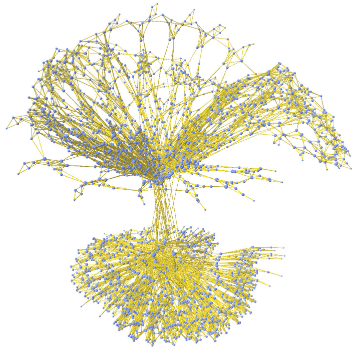
- Planning and scheduling
- Formal verification
- Testing and debugging
- Cryptanalysis
- Theorem proving
- Electronic circuit design



SAT: Limits of Feasibility

Observation

We often face SAT instances of practical relevance which are infeasible to solve with current methods.


Formula encoding two multiplier circuits and their logical equivalence; 4k variables, 13k clauses

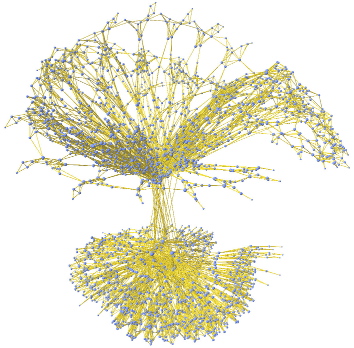
SAT: Limits of Feasibility

Observation

We often face SAT instances of practical relevance which are infeasible to solve with current methods.

Objective

Push the frontier of feasible problems using modern distributed environments (HPC, clouds).



Formula encoding two multiplier circuits and their logical equivalence; 4k variables, 13k clauses

SAT: Limits of Feasibility

Observation

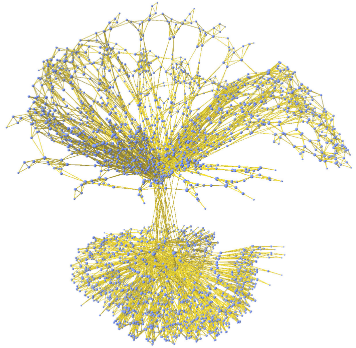
We often face SAT instances of practical relevance which are infeasible to solve with current methods.

Objective

Push the frontier of feasible problems using modern distributed environments (HPC, clouds).

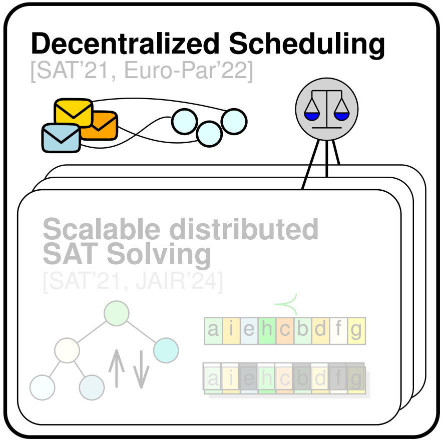
Challenges

- Strongly sublinear scaling of parallel SAT solvers
- Execution times unknown in advance



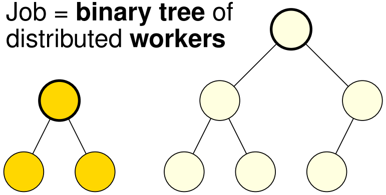
Formula encoding two multiplier circuits and their logical equivalence; 4k variables, 13k clauses

Overview


Decentralized Scheduling
[SAT'21, Euro-Par'22]
Scalable distributed SAT Solving
[SAT'21, JAIR'24]
a i e h c b d f g
a i e h c b d f g

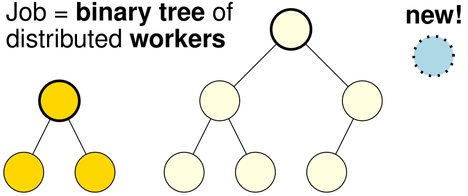
Decentralized Malleable Scheduling [SAT'21, Euro-Par'22]

Job = **binary tree** of distributed **workers**



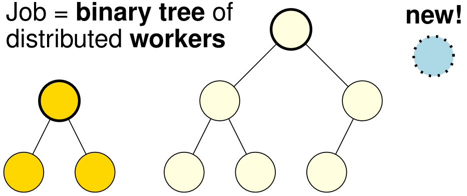
Decentralized Malleable Scheduling [SAT'21, Euro-Par'22]

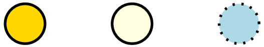
Job = **binary tree** of distributed **workers**



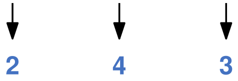
Decentralized Malleable Scheduling [SAT'21, Euro-Par'22]

Job = **binary tree** of distributed **workers**



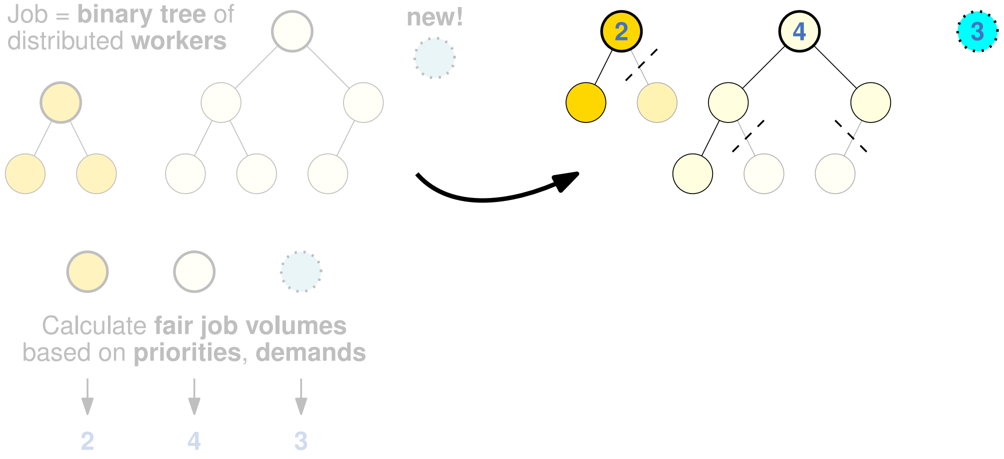


Calculate **fair job volumes** based on **priorities**, **demands**



Decentralized Malleable Scheduling [SAT'21, Euro-Par'22]

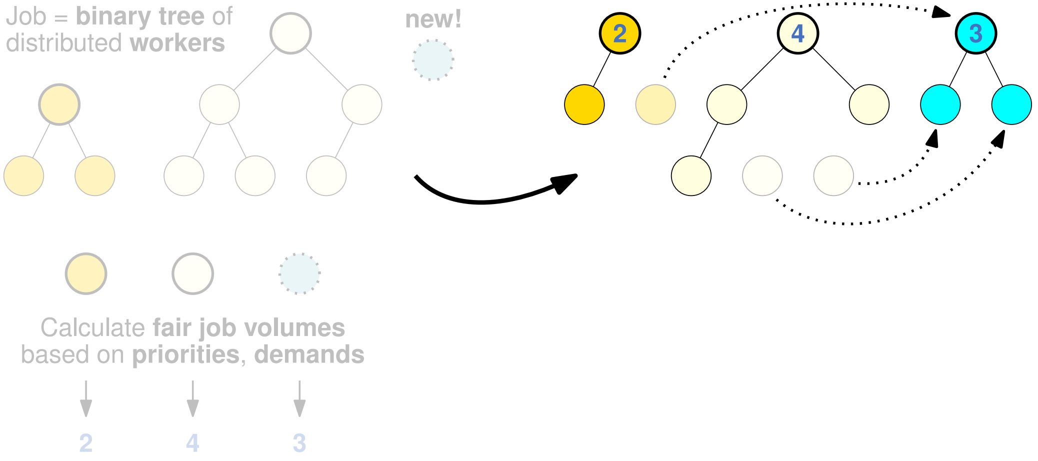
Job = **binary tree** of distributed **workers**



Calculate **fair job volumes** based on **priorities**, **demands**

Decentralized Malleable Scheduling [SAT'21, Euro-Par'22]

Job = **binary tree** of distributed **workers**



Calculate **fair job volumes** based on **priorities**, **demands**

Scheduling: Experiments [Euro-Par'22]

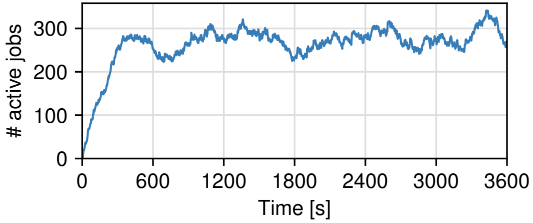
- 128 machines of **SuperMUC-NG**
  – 1536 processes $\times$ 4 cores
- Random arrival of random tasks

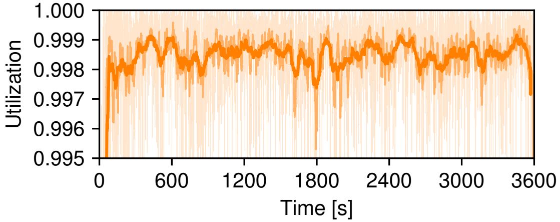
400 problems from Int'l SAT Competition 2020

Scheduling: Experiments [Euro-Par'22]

- 128 machines of **SuperMUC-NG**
  – 1536 processes $\times$ 4 cores
- Random arrival of random tasks

400 problems from Int'l SAT Competition 2020

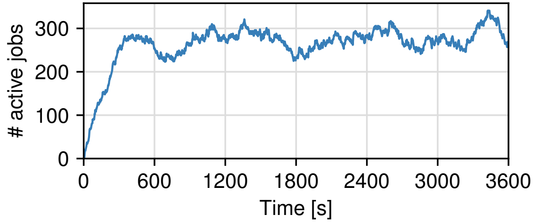




Scheduling: Experiments [Euro-Par'22]

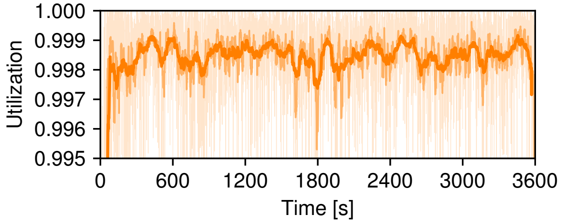
- 128 machines of **SuperMUC-NG**
  – 1536 processes × 4 cores
- Random arrival of random tasks

400 problems from Int'l SAT Competition 2020

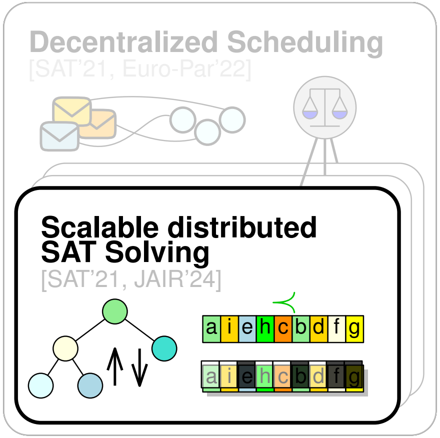


Mean latencies

≈ 10 ms for scheduling a 1st worker
≈ 1 ms for calculating fair volumes
≈ 6 ms for finding+adding further workers



Overview

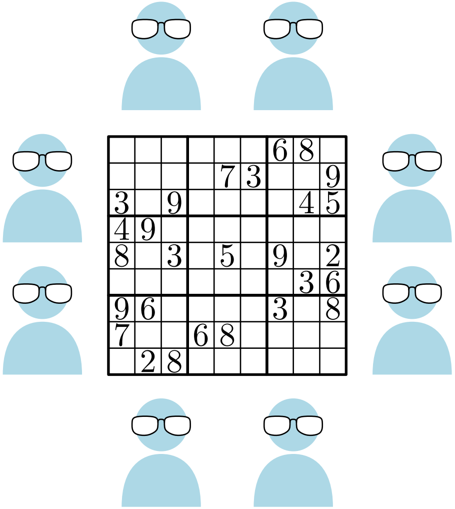

Decentralized Scheduling
[SAT'21, Euro-Par'22]
Scalable distributed SAT Solving
[SAT'21, JAIR'24]
a i e h c b d f g
a i e h c b d f g

Parallel Logical Reasoning

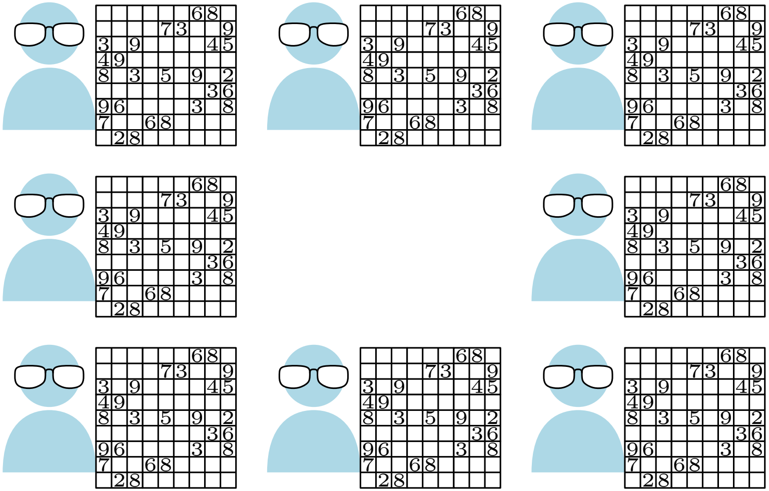
The assembly of logicians

- Complex logic puzzle
- *n* logic experts want to solve the puzzle
- Experts tend to work the best undisturbed

How to coordinate our experts?



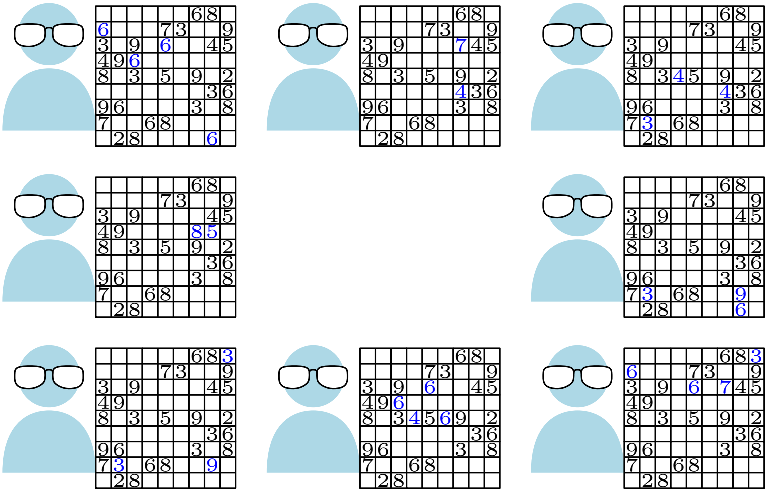
Parallel Logical Reasoning



Parallel portfolio

- All experts work on original problem independently

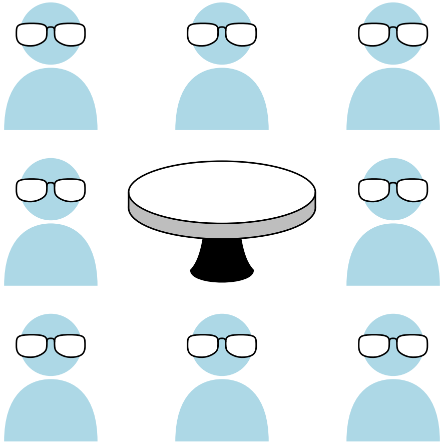
Parallel Logical Reasoning



Parallel portfolio

- All experts work on original problem independently

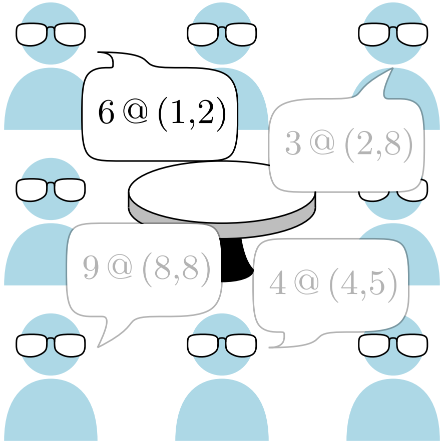
Parallel Logical Reasoning



Parallel portfolio

- All experts work on original problem independently
- Brief meetings to exchange crucial insights

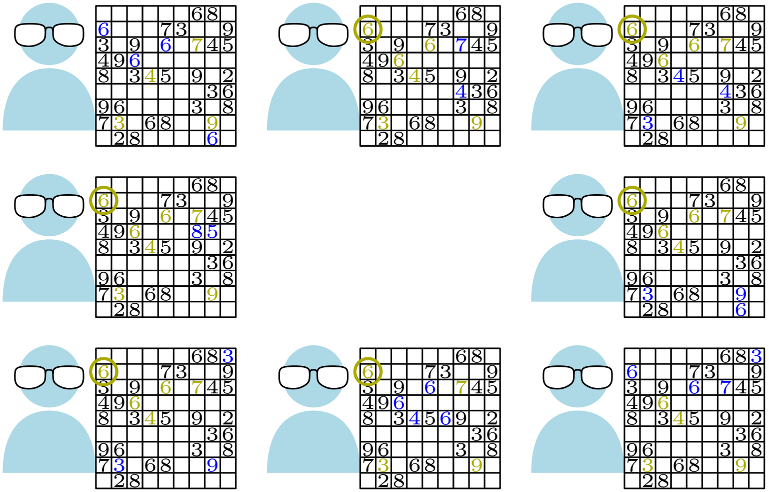
Parallel Logical Reasoning



Parallel portfolio

- All experts work on original problem independently
- Brief meetings to exchange crucial insights

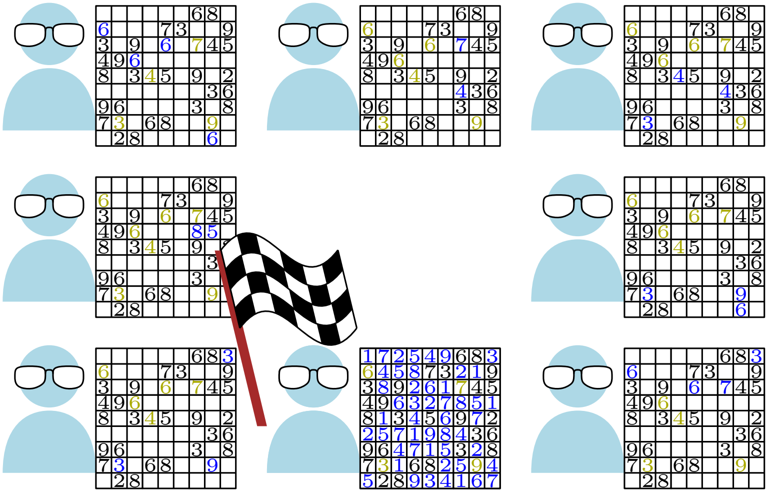
Parallel Logical Reasoning



Parallel portfolio

- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving

Parallel Logical Reasoning



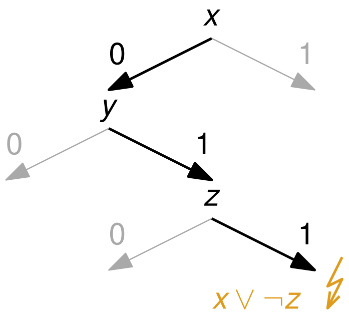
Parallel portfolio

- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving
- Only one expert needs to find a solution!

Distributed SAT solving: State of the art

Parallel SAT solving [Hamadi et al. 2010]

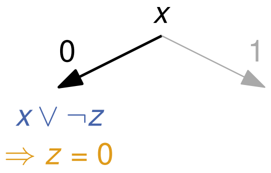
- Experts $\equiv$ sequential search algorithms



Distributed SAT solving: State of the art

Parallel SAT solving [Hamadi et al. 2010]

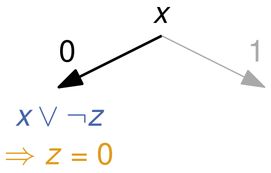
- Experts $\equiv$ sequential search algorithms



Distributed SAT solving: State of the art

Parallel SAT solving [Hamadi et al. 2010]

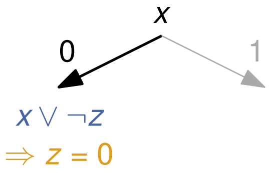
- Experts $\equiv$ sequential search algorithms
- Shared information $\equiv$ learned conflict clauses



Distributed SAT solving: State of the art

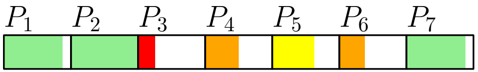
Parallel SAT solving [Hamadi et al. 2010]

- Experts $\equiv$ sequential search algorithms
- Shared information $\equiv$ learned conflict clauses



Prior state of the art: **HordeSat** [Balyo et al. 2015]

- Periodic clause exchange
  - Concatenation of fixed-size clause buffers
  - Duplicates, unused space in buffers



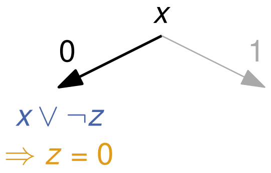
Distributed SAT solving: State of the art

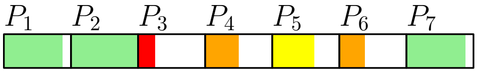
Parallel SAT solving [Hamadi et al. 2010]

- Experts $\equiv$ sequential search algorithms
- Shared information $\equiv$ learned conflict clauses

Prior state of the art: **HordeSat** [Balyo et al. 2015]

- Periodic clause exchange
  - Concatenation of fixed-size clause buffers
  - Duplicates, unused space in buffers
- Experiments with $\leq$ 2048 cores
  - Individual super-linear speedups ($>$ 2048)
  - Median speedup at 2048 cores: 13 (efficiency 0.6%)

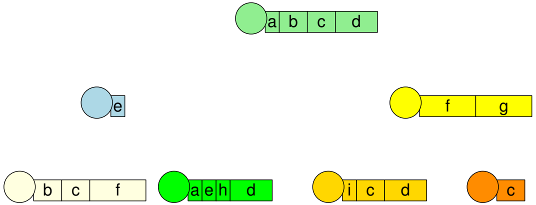




Clause sharing: Our approach [SAT'21, JAIR'24]

Exchange of useful clauses

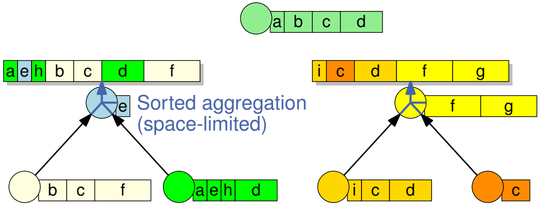
1.



Clause sharing: Our approach [SAT'21, JAIR'24]

Exchange of useful clauses

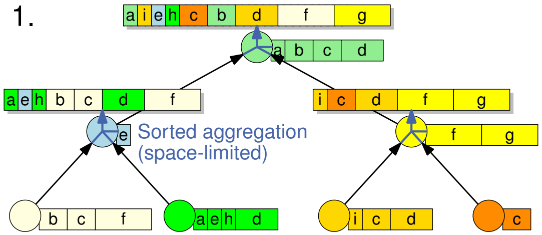
1.



Clause sharing: Our approach [SAT'21, JAIR'24]

Exchange of useful clauses

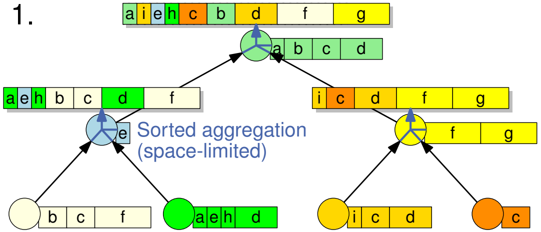
1.



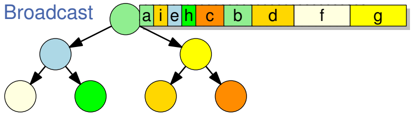
Clause sharing: Our approach [SAT'21, JAIR'24]

Exchange of useful clauses

1.

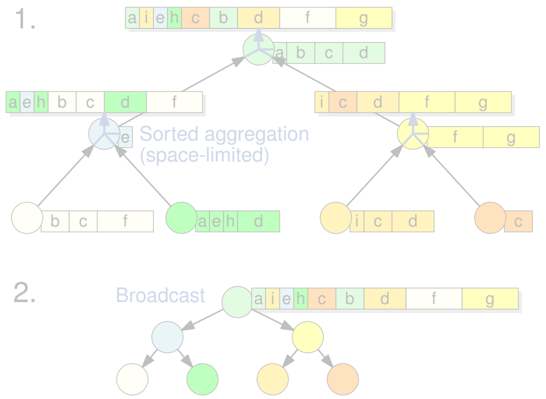


2.

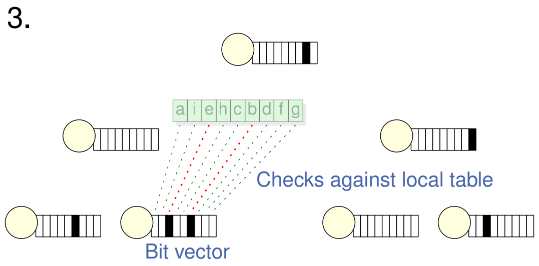


Clause sharing: Our approach [SAT'21, JAIR'24]

Exchange of useful clauses

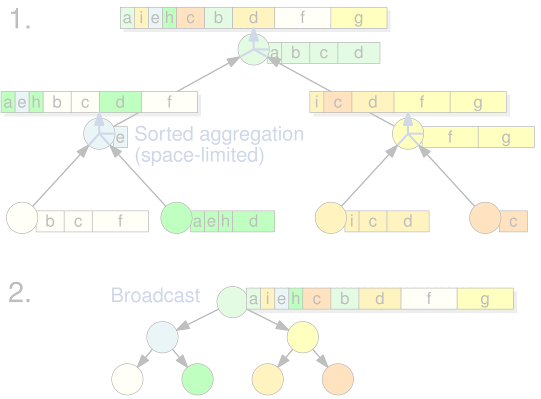


Filtering of recently shared clauses

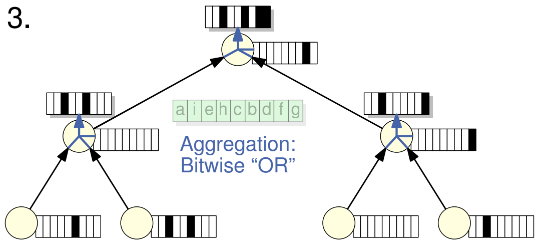


Clause sharing: Our approach [SAT'21, JAIR'24]

Exchange of useful clauses



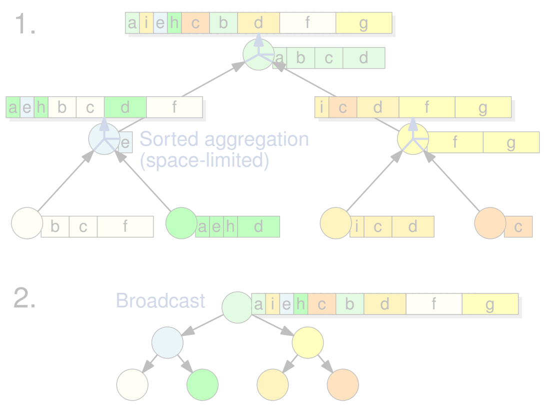
Filtering of recently shared clauses

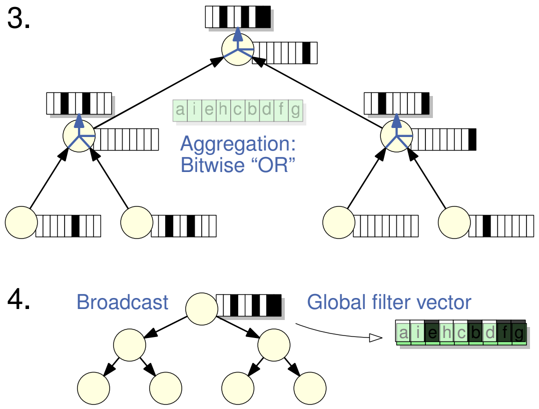


Clause sharing: Our approach [SAT'21, JAIR'24]

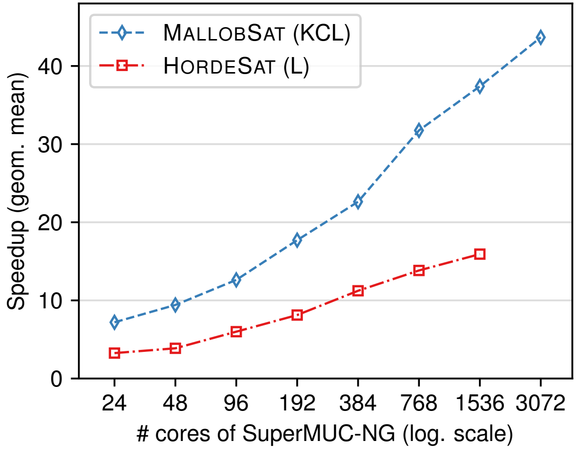
Exchange of useful clauses

Filtering of recently shared clauses



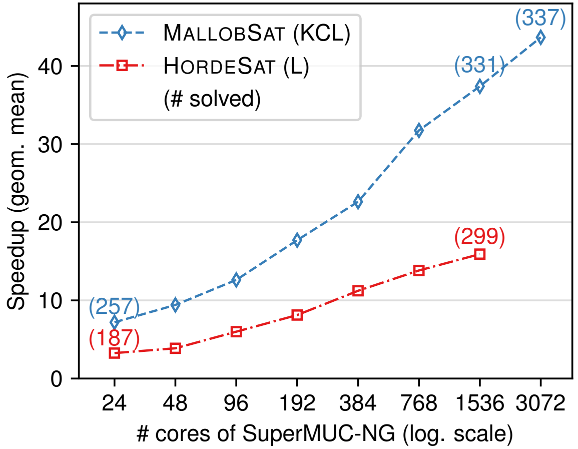


MallobSat: Results [JAIR'24]



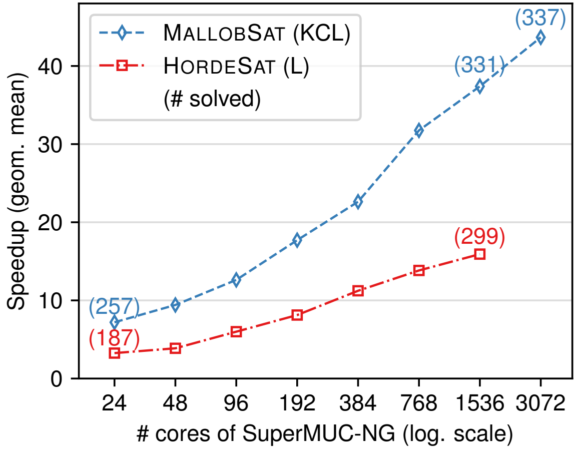
400 problems from SAT Comp. 2021 · Seq. baseline Kissat_MAB-HyWalk · Seq. limit 32 h (331 solved) · Par. limit 300 s

MallobSat: Results [JAIR'24]



400 problems from SAT Comp. 2021 · Seq. baseline Kissat_MAB-HyWalk · Seq. limit 32 h (331 solved) · Par. limit 300 s

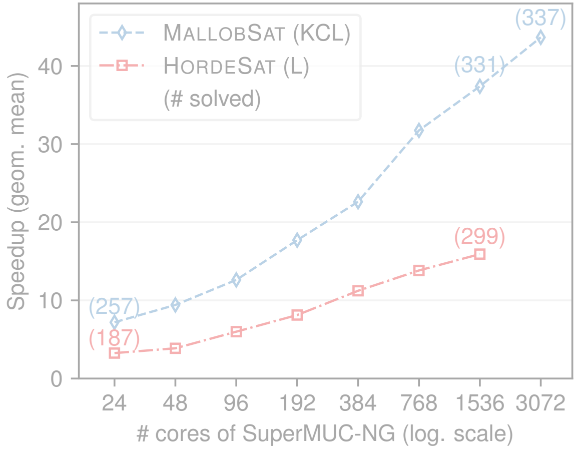
MALLOBSAT: Results [JAIR'24]



Seq. time $\geq$ 1 h $\Rightarrow$ Speedup **419** at 3072 cores

400 problems from SAT Comp. 2021 · Seq. baseline KISSAT_MAB-HYWALK · Seq. limit 32 h (331 solved) · Par. limit 300 s

MallobSat: Results [JAIR'24]



Seq. time $\geq$ 1 h $\Rightarrow$ Speedup **419** at 3072 cores

Malleable scheduling

- 6400 cores, 2 h wallclock time, 400 formulas
- Rigid: Each task gets $\frac{6400}{400} = 16$ cores
  $\Rightarrow \varnothing$ Response time: 26.7 min
- Malleable: Resources of done jobs are redistributed to remaining jobs
  $\Rightarrow \varnothing$ Response time: 21.1 min (-21%)

400 problems from SAT Comp. 2021 · Seq. baseline Kissat_MAB-HyWalk · Seq. limit 32 h (331 solved) · Par. limit 300 s

Conclusion

Testimonials

"*Mallob-mono is now, by a **wide** margin, the most powerful SAT solver on the planet.*" —Byron Cook, Amazon Science, 2021

`https://www.amazon.science/blog/automated-reasonings-scientific-frontiers`

Best cloud solver @ International SAT Competition 2020–2023

References

`github.com/domschrei/mallob`

[SAT'21] Schreiber, Sanders: *Scalable SAT Solving in the Cloud*

[Euro-Par'22] Sanders, Schreiber: *Decentralized Job Scheduling of Malleable NP-hard Jobs*

[JOSS'22] Sanders, Schreiber: *Mallob: Scalable SAT Solving on Demand with Decentralized Job Scheduling*

[JAIR'24] Schreiber, Sanders: *MallobSat: Scalable SAT Solving by Clause Sharing* (to appear)

References

Behnke, Gregor, Daniel Höller, and Susanne Biundo. “totSAT – Totally-ordered hierarchical planning through SAT.” AAAI 2018.

Bercher, Pascal, Ron Alford, and Daniel Höller. “A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations.” IJCAI 2019.

Audemard, Gilles and Laurent Simon. “Predicting learnt clauses quality in modern SAT solvers.” IJCAI 2009.

Balyo, Tomáš, Peter Sanders, and Carsten Sinz. “Hordesat: A massively parallel portfolio SAT solver.” SAT 2015.

Biere, Armin. “CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018.” SAT Competition 2018.

Biere, Armin, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020.” SAT Competition 2020.

Cook, Stephen. “The complexity of theorem proving procedure.” 3rd Symp. on Theory of Computing (1971).

Feitelson, Dror G. “Job scheduling in multiprogrammed parallel systems.” IBM Research Report (1997).

Gao, Yu. “Kissat MAB prop in SAT Competition 2023.” SAT Competition 2023.

Hamadi, Youssef, Said Jabbour, and Lakhdar Sais. “ManySAT: a parallel SAT solver”. JSAT (2010).

Heule, Marijn J. H., Norbert Manthey, and Tobias Philipp. “Validating Unsatisfiability Results of Clause Sharing Parallel SAT Solvers.” POS 2014.

Zheng, Jiongzhi, Kun He, Zhuo Chen, et al. “Combining Hybrid Walking Strategy with Kissat MAB, CaDiCaL, and LStech-Maple.” SAT Competition 2022.

Image Sources

2 · Circuit: `https://www.rawpixel.com/image/5907876/photo-image-background-public-domain-technology`

2 · Planning/scheduling: `https://www.pexels.com/photo/blue-printer-paper-7376/`

2 · Cryptography: `https://pixabay.com/vectors/computer-encrypt-encryption-1294045/`

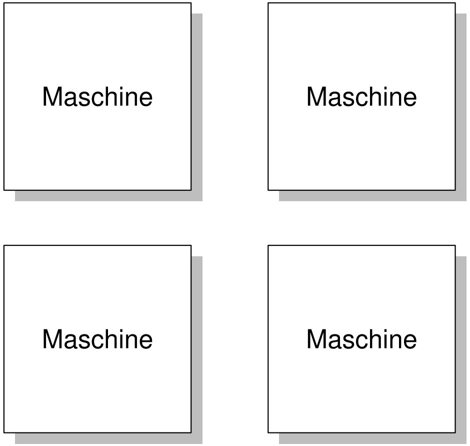
2 · Colored grid:
`https://www.quantamagazine.org/the-number-15-describes-the-secret-limit-of-an-infinite-grid-20230420/`

2 · Debugging: `https://technofaq.org/posts/2017/12/heres-everything-you-need-to-know-about-software-testing/`

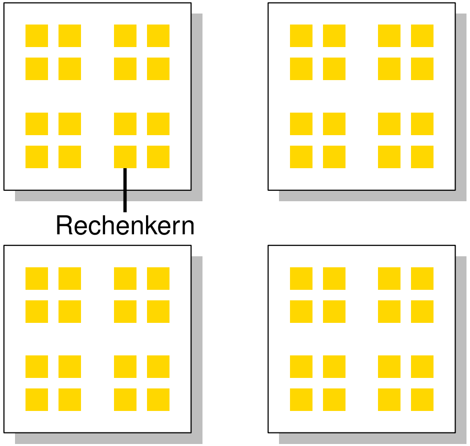
Appendix

(German / English)

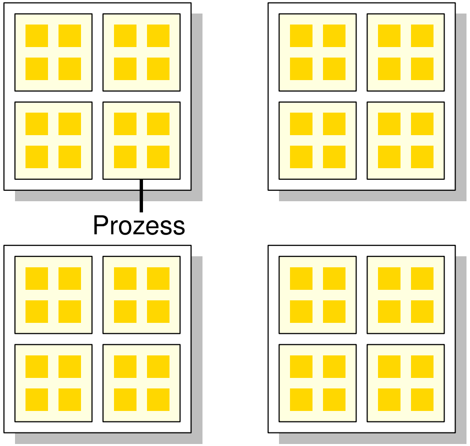
Ausführungsumgebung



Ausführungsumgebung



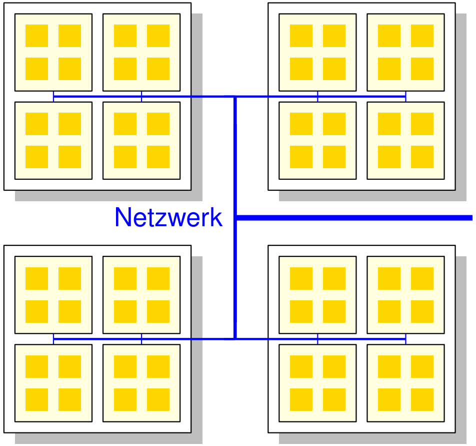
Ausführungsumgebung



Verteilte Rechenumgebung des Schedulers

- *m* verteilte Prozesse

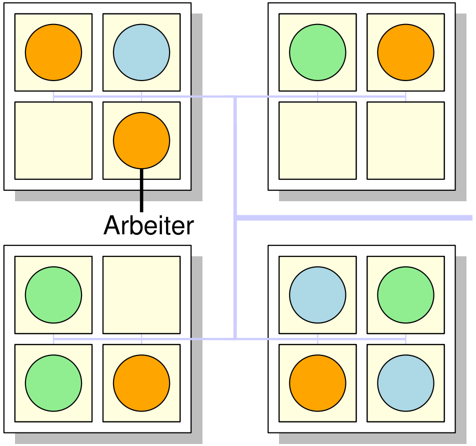
Ausführungsumgebung



Verteilte Rechenumgebung des Schedulers

- m verteilte Prozesse

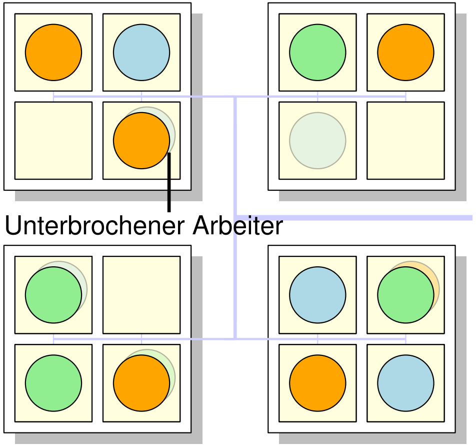
Ausführungsumgebung



Verteilte Rechenumgebung des Schedulers

- m verteilte Prozesse
- Arbeiter: Ausführungskontext einer bestimmten Aufgabe auf einem bestimmten Prozess
- Je Prozess:
  $\leq$ 1 aktive Arbeiter
  $\leq c$ unterbrochene Arbeiter

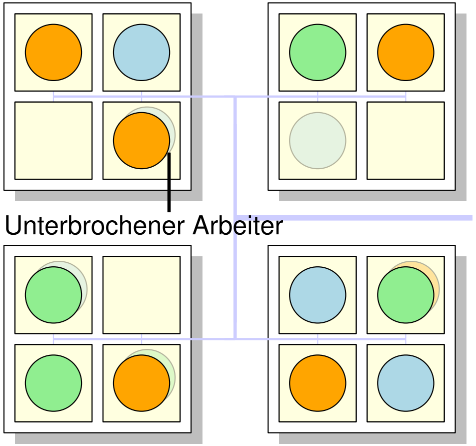
Ausführungsumgebung



Verteilte Rechenumgebung des Schedulers

- m verteilte Prozesse
- Arbeiter: Ausführungskontext einer bestimmten Aufgabe auf einem bestimmten Prozess
- Je Prozess:
  $\leq$ 1 aktive Arbeiter
  $\leq c$ unterbrochene Arbeiter

Ausführungsumgebung



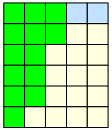
Verteilte Rechenumgebung des Schedulers

- *m* verteilte Prozesse
- Arbeiter: Ausführungskontext einer bestimmten Aufgabe auf einem bestimmten Prozess
- Je Prozess:
  $\leq$ 1 aktive Arbeiter
  $\leq$ *c* unterbrochene Arbeiter
- Eigenschaften jeder Aufgabe $j \in J$:
  - Priorität $p_j \in \mathbb{R}^+$
  - Max. Ressourcen-Bedarf $d_j \in \mathbb{N}^+$

Scheduling: Motivation

Definition: Malleability [Feitelson 1997]

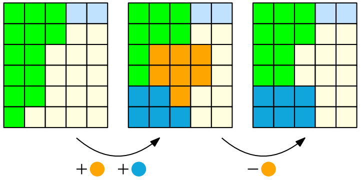
A parallel computation is **malleable** if it supports a fluctuating number of processing elements throughout its execution.



Scheduling: Motivation

Definition: Malleability [Feitelson 1997]

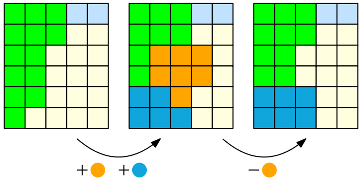
A parallel computation is **malleable** if it supports a fluctuating number of processing elements throughout its execution.



Scheduling: Motivation

Definition: Malleability [Feitelson 1997]

A parallel computation is **malleable** if it supports a fluctuating number of processing elements throughout its execution.



Why malleable scheduling for SAT solving?

- Execution times unknown ⇒ Flexible reactions beneficial
- Sublinear scaling ⇒ Parallel processing of multiple formulas increases efficiency
- Malleability easy to achieve → 2nd part of the talk

Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j

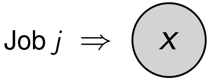
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



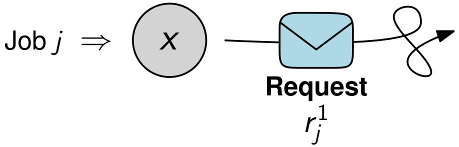
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



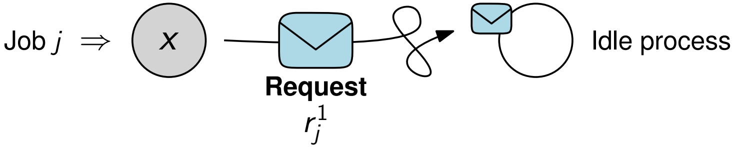
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



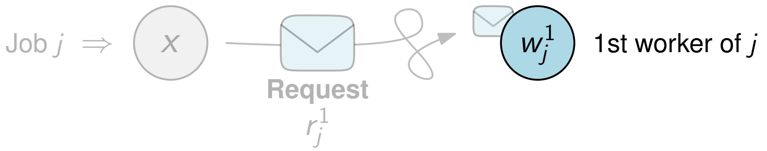
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



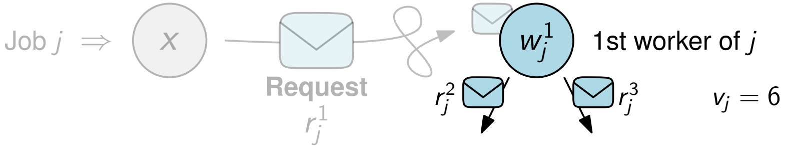
Our scheduling approach [SAT’21, Euro-Par’22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



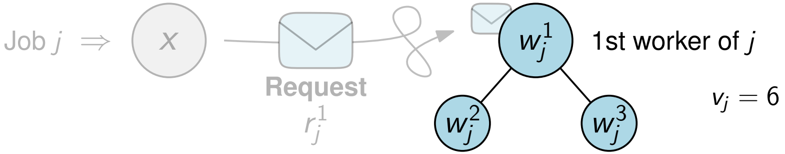
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



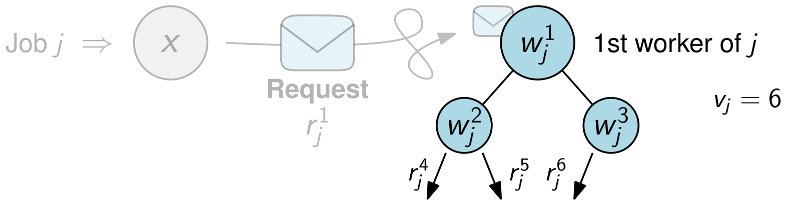
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



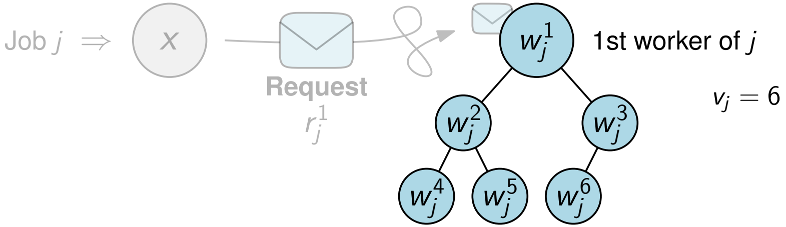
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



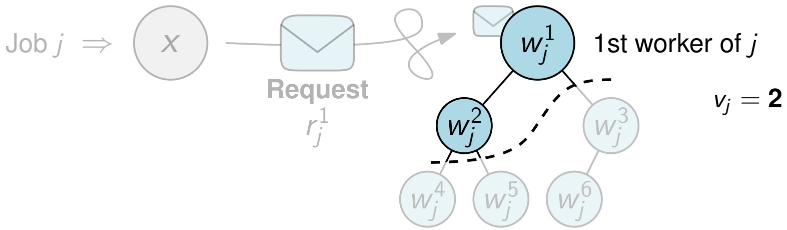
Our scheduling approach [SAT'21, Euro-Par'22]

Job model: priority p_j; max. resource demand d_j; set of exclusive associated resources (workers)

Problem 1: For each active job j, determine a fair number $1 \leq v_j \leq d_j$ of workers in such a way that $v_j \propto p_j$

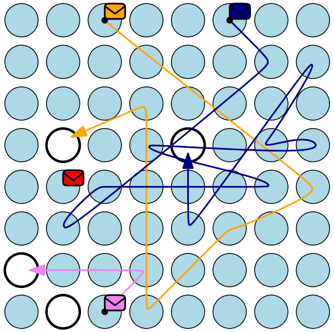
- Theory: Fully scalable algorithm with span $\mathcal{O}(\log m)$ via collective operations
- Practice: Aggregate events which alter system state $\rightarrow$ locally compute new assignments

Problem 2: Assign v_j actual processes to each job j



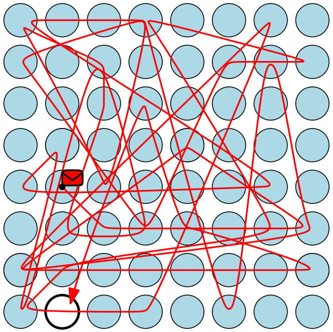
Zuordnung von Anfragen und Prozessen [Euro-Par’22]

Random-Walk-Methode



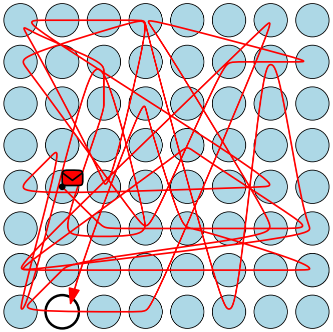
Zuordnung von Anfragen und Prozessen [Euro-Par’22]

Random-Walk-Methode

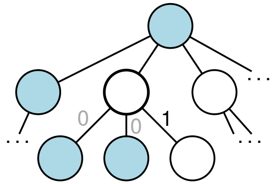


Zuordnung von Anfragen und Prozessen [Euro-Par’22]

Random-Walk-Methode

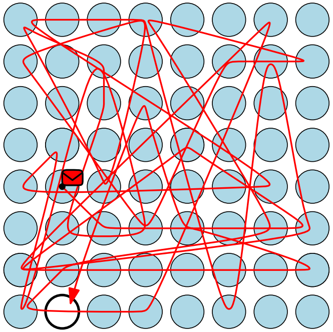


Prozess-Baum

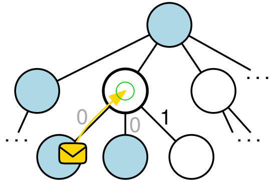


Zuordnung von Anfragen und Prozessen [Euro-Par’22]

Random-Walk-Methode

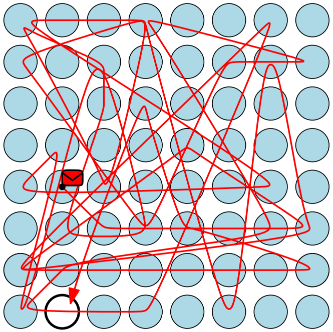


Prozess-Baum

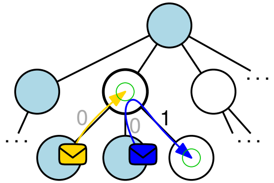


Zuordnung von Anfragen und Prozessen [Euro-Par’22]

Random-Walk-Methode

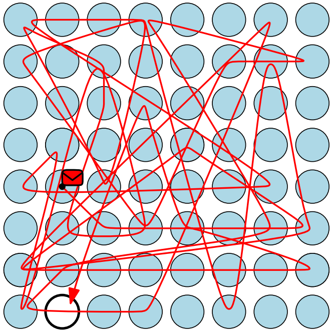


Prozess-Baum

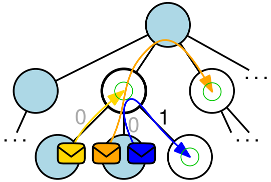


Zuordnung von Anfragen und Prozessen [Euro-Par'22]

Random-Walk-Methode

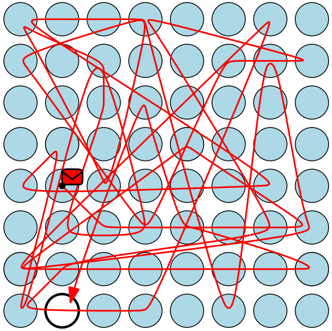


Prozess-Baum

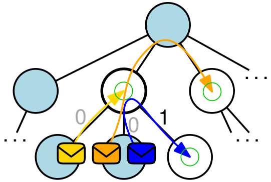


Zuordnung von Anfragen und Prozessen [Euro-Par’22]

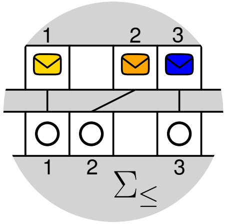
Random-Walk-Methode



Prozess-Baum



Async. Präfixsummen



Our system MallobSat [SAT'21, JAIR'24]

Solver configuration

- Interfaces for well-performing solvers (Kissat, CaDiCaL, Lingeling, Glucose) [Biere et al. 2018, 2020; Audemard & Simon 2009]
- Little randomization $\Rightarrow$ effective work subdivision via clause sharing

Our system MallobSat [SAT'21, JAIR'24]

Solver configuration

- Interfaces for well-performing solvers (Kissat, CaDiCaL, Lingeling, Glucose) [Biere et al. 2018, 2020; Audemard & Simon 2009]
- Little randomization ⇒ effective work subdivision via clause sharing

Malleability

- Process tree for all communication
- Arbitrary addition, removal of solver processes

Our system MALLOBSAT [SAT'21, JAIR'24]

Solver configuration

- Interfaces for well-performing solvers (KISSAT, CADICAL, LINGELING, GLUCOSE) [Biere et al. 2018, 2020; Audemard & Simon 2009]
- Little randomization ⇒ effective work subdivision via clause sharing

Malleability

- Process tree for all communication
- Arbitrary addition, removal of solver processes

"*Mallob-mono is now, by a* wide *margin, the most powerful SAT solver on the planet.*"

—Byron Cook, Amazon Distinguished Scientist, 2021

`https://www.amazon.science/blog/automated-reasonings-scientific-frontiers`

Int. SAT Competition 2020–2023