Chair of Computer Architecture and Parallel Systems
TUM School of Computation, Information and Technology
Technical University of Munich

Deadline Miss Minimization Scheduling for License-Constrained CAE Jobs in Hybrid Cloud Infrastructure

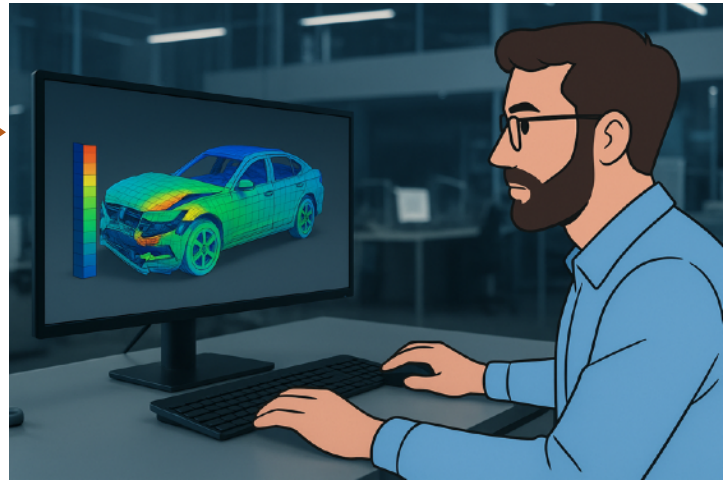
- Mohamed Noaman, **Srishti Dasgupta**, Michael Gerndt -

28th Workshop on Job Scheduling Strategies for Parallel Processing
39th IEEE International Parallel and Distributed Processing System
Milan, Italy
3 - 4 June, 2025

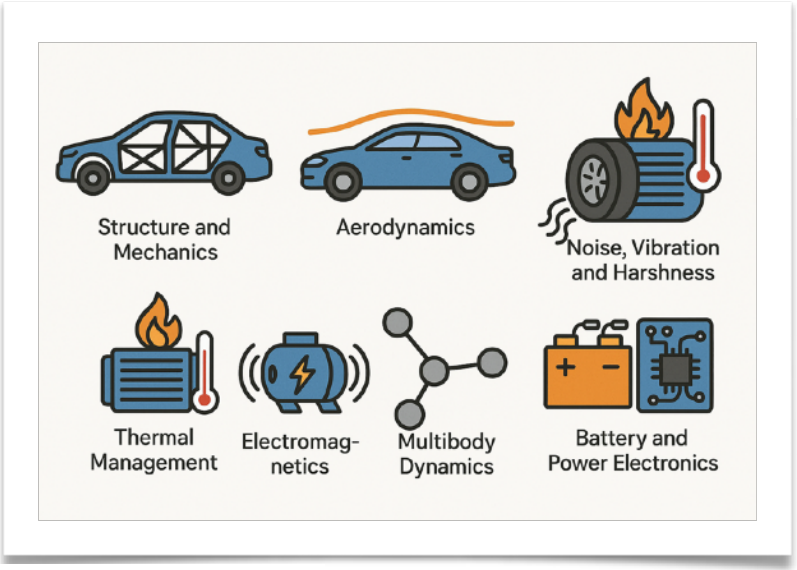
Motivation & Problem Statement

What is a CAE Simulation

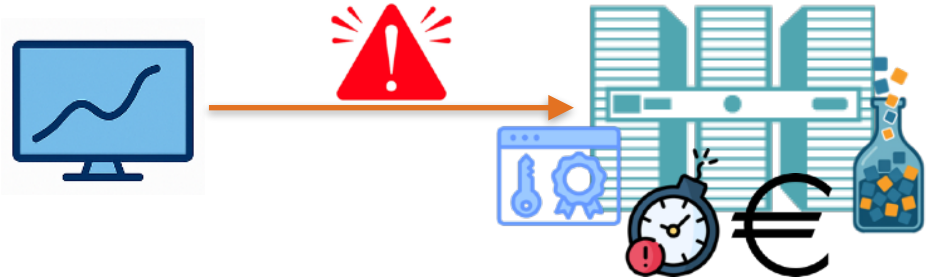
"Computer-Aided Engineering(CAE) software has revolutionized car modeling by enabling virtual testing and optimization of structural, thermal, aerodynamic, and dynamic performance — drastically reducing time, cost, and physical prototypes."



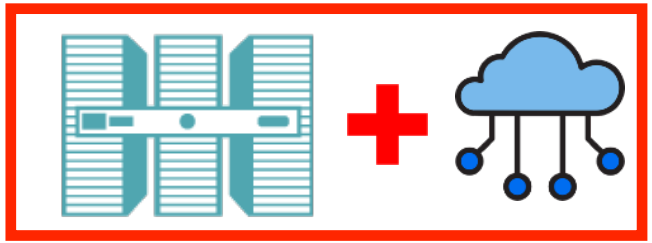
Bottlenecks in the CAE Simulation Field



Often computationally heavy requiring hundreds of cores

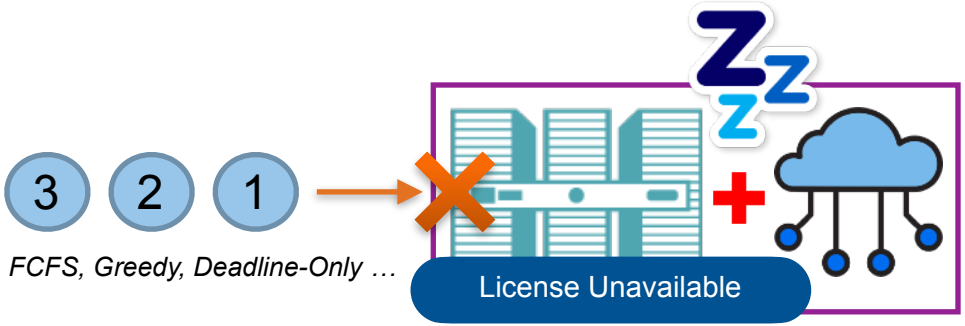


CAE tools drive safety & innovation—but come with tight deadlines, expensive licenses, and compute bottlenecks.



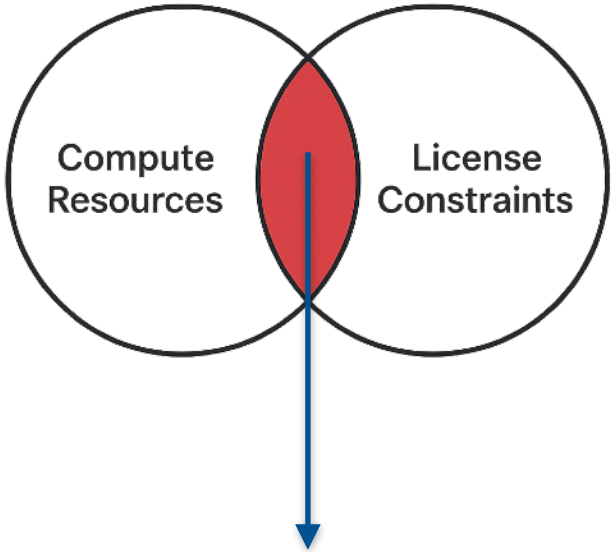
Hybrid Infrastructure

Why Advanced Scheduling is Needed ?



Current schedulers overlook the interdependency of licenses and compute — leading to missed deadlines and wasted cloud costs.

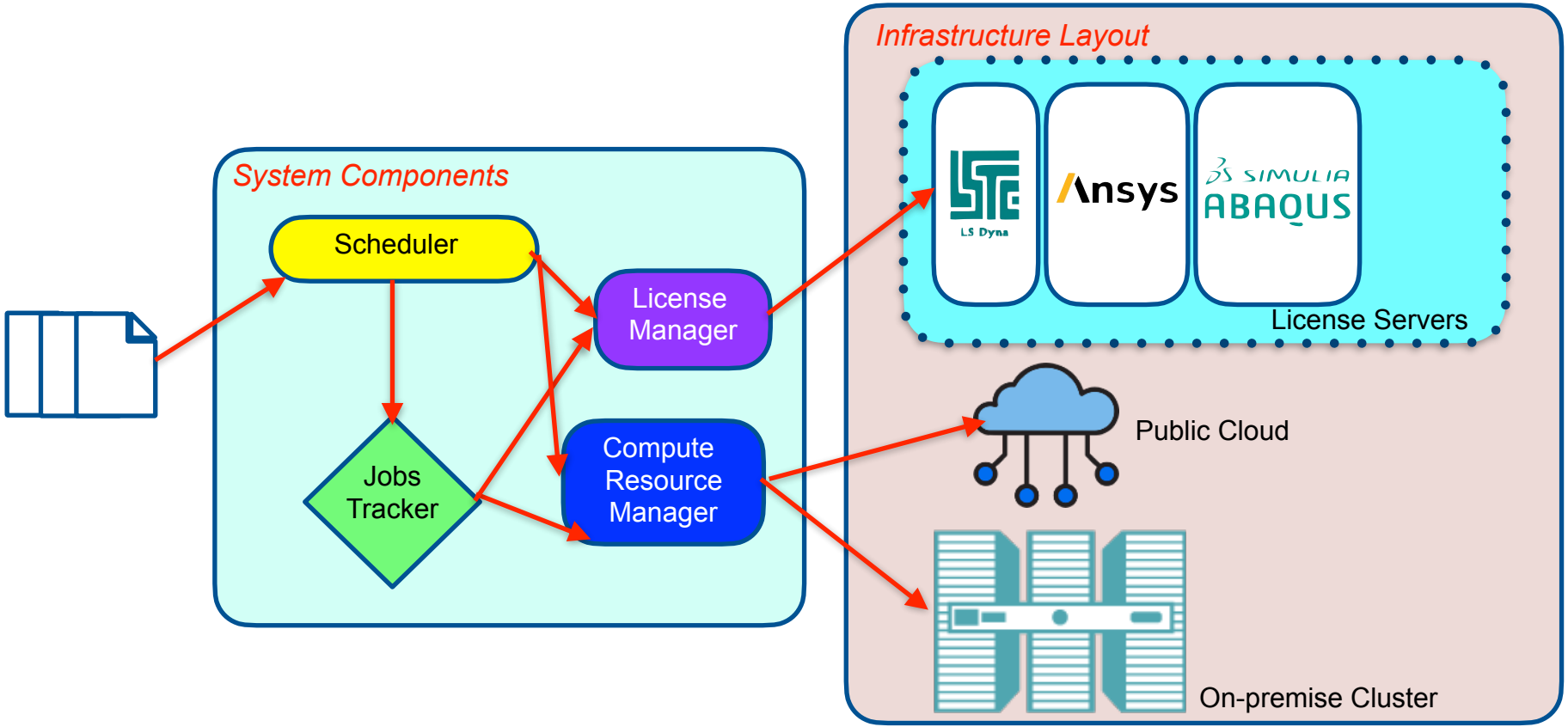
License should also be treated as Resource



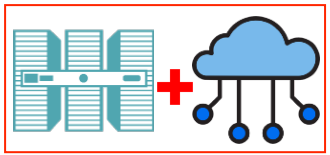
License-Aware Deadline Miss Minimization Algorithm

Proposed Solution: LADMM_backfill Algorithm

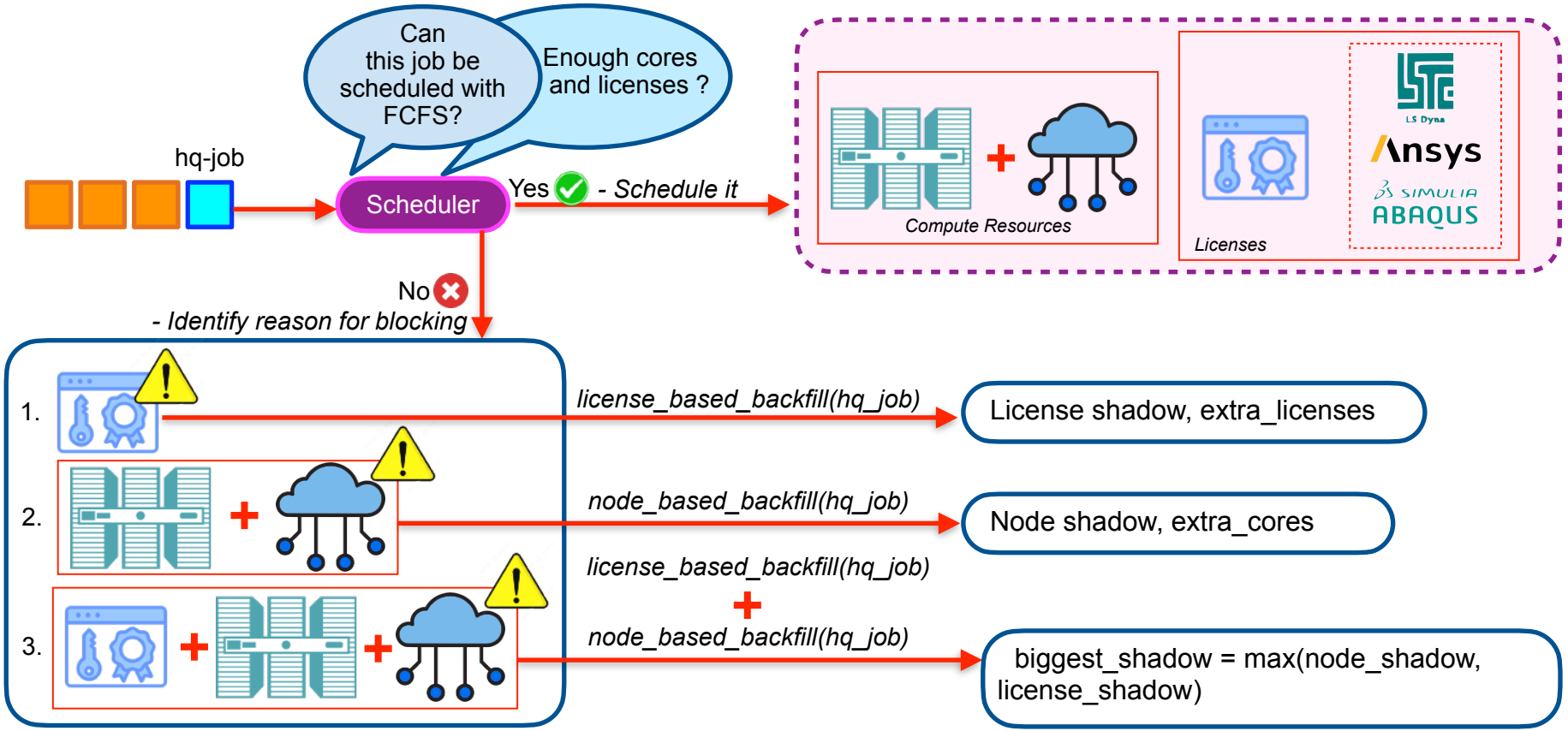
System Design



LADMM_backfill Algorithm: Core Objectives

- *Built in principle of EASY backfilling strategy*
-  Aimed at minimizing deadline misses
-  Integrates real-time availability of floating licenses
-  Favors jobs requiring critical resources
-  Handles both On-Prem HPC clusters and public cloud nodes

LADMM_backfill Algorithm: Initial Resource Allocation



LADMM_backfill Algorithm: Scoring & Backfilling

- **Total Score for Job j**:

$$S_j = S_{critical} + S_{deadline}$$

- **Critical Resources Heuristics**($S_{critical}$)

$$S_{critical} = C_n \cdot t_j + C_T \cdot (R_j / R_{min})$$

C_n : critical resources licensing factor
t_j : number of licenses required by jobj
C_T : critical resources runtime factor
R_{min} : runtime of the shortest job in the queue
R_j : runtime of jobj

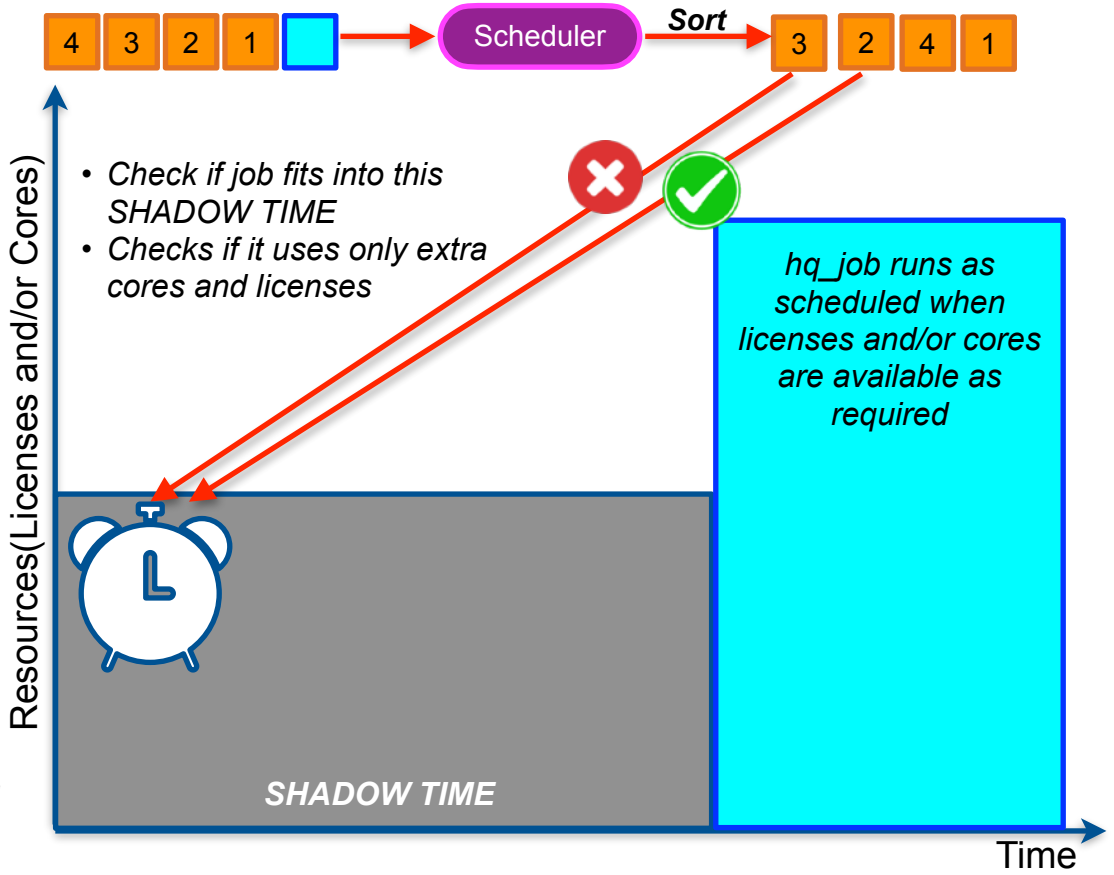
- **Deadline Heuristics**($S_{deadline}$)

$$S_{deadline} = \begin{cases} (D_B \cdot D_{min}) / (D_j - T_{current} + R_j), \\ \quad If\ D_j - T_{current} + R_j > 0 \\ 0, \ otherwise \end{cases}$$

D_B : deadline urgency weight
D_{min} : smallest deadline gap in the queue
D_j : deadline of job j
$T_{current}$: current simulation time
R_j : runtime of jobj

How scoring was tuned ?

- A Pareto front was used
  125000 jobs were simulated with varying D_B, C_T & C_n
- Selected weights that ***maximized License Minimization*** & ***minimized Deadline Misses***



Experimental Setup and Results

Experimental Setup

- Licensing Model
  - Floating License Servers
  - 3 CAE Licenses chosen: LS-Dyna, Abaqus & Ansys with their **cost models** [1]
  - Fixed number of licenses available: Ranging from 50 to 250
- Job Traces
  - From Parallel Workfloads Archive [2]
    - Submission Time, Core Count, runtime
  - Deadline = Runtime * Factor(2 to 5)
- Each Job assigned a CAE software randomly, with Ansys jobs running specifically on On-premise
- Job Scale:
  - Total jobs: 45000 with core requirements 1 to 50
- Infrastructure :
  - On-premise: considered 528 total cluster cores (48 and 96 cores machines)
  - Cloud: based on AWS m7 instances with 1:1 vcpu to physical core mapping, reflecting license constraints
    - Pay-As-You go model with per-minute billing
  - Used simulus [3] to build the infrastructure

Baselines Used

1. First Come First Served: ***FCFS***
2. Greedy Technique sorting jobs w.r.t deadlines : ***GREEDY_deadline***
3. Heuristics-based Backfilling: ***HEURISTICS_backfill***
   - Attempts backfilling via scoring

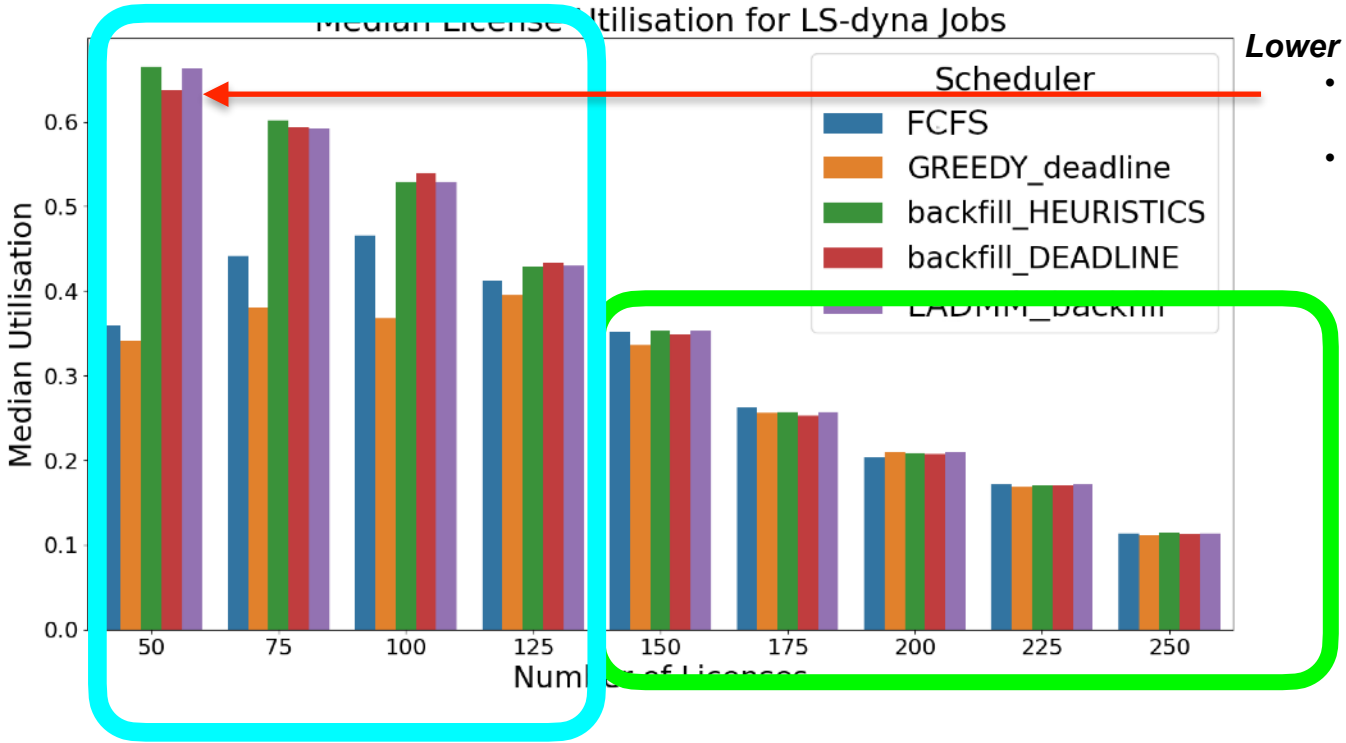
$$S_j = S_{critical} + S_{deadline''} + S_{starvation}$$

$S_{critical}$	Prioritises jobs requiring more licenses and shorter runtimes, deemed “critical”
$S_{deadline''}$	Prioritizes jobs closer to deadline, with respect to the average job queue
$S_{starvation}$	Boosts long-waiting jobs to avoid starvation

** Also uses a metric “*Ansys Ratio*” which monitors and prioritized Ansys jobs on on-premise if they are being starved.

4. Deadline-based Backfilling : ***DEADLINE_backfill***
   - Checks if a job can meet its deadline, and attempts safe backfilling using shadow time calculations if not
   - If above is not possible, moves job to an “impossible queue”

Median License Utilization of LS-Dyna Jobs

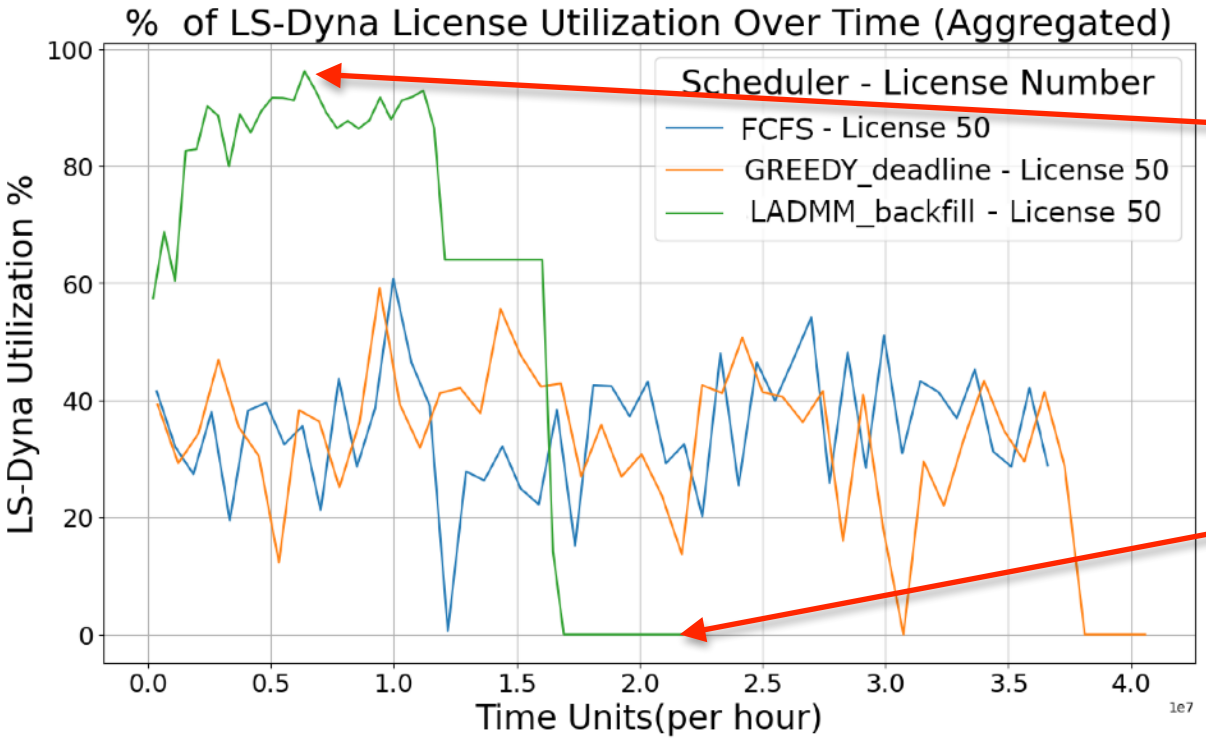


Lower License Count:

- ~ 84-94% reduction vs FCFS and GREEDY_deadline
- Comprable to backfilling baselines

Higher License Count: performance converges due to reduced resource contention

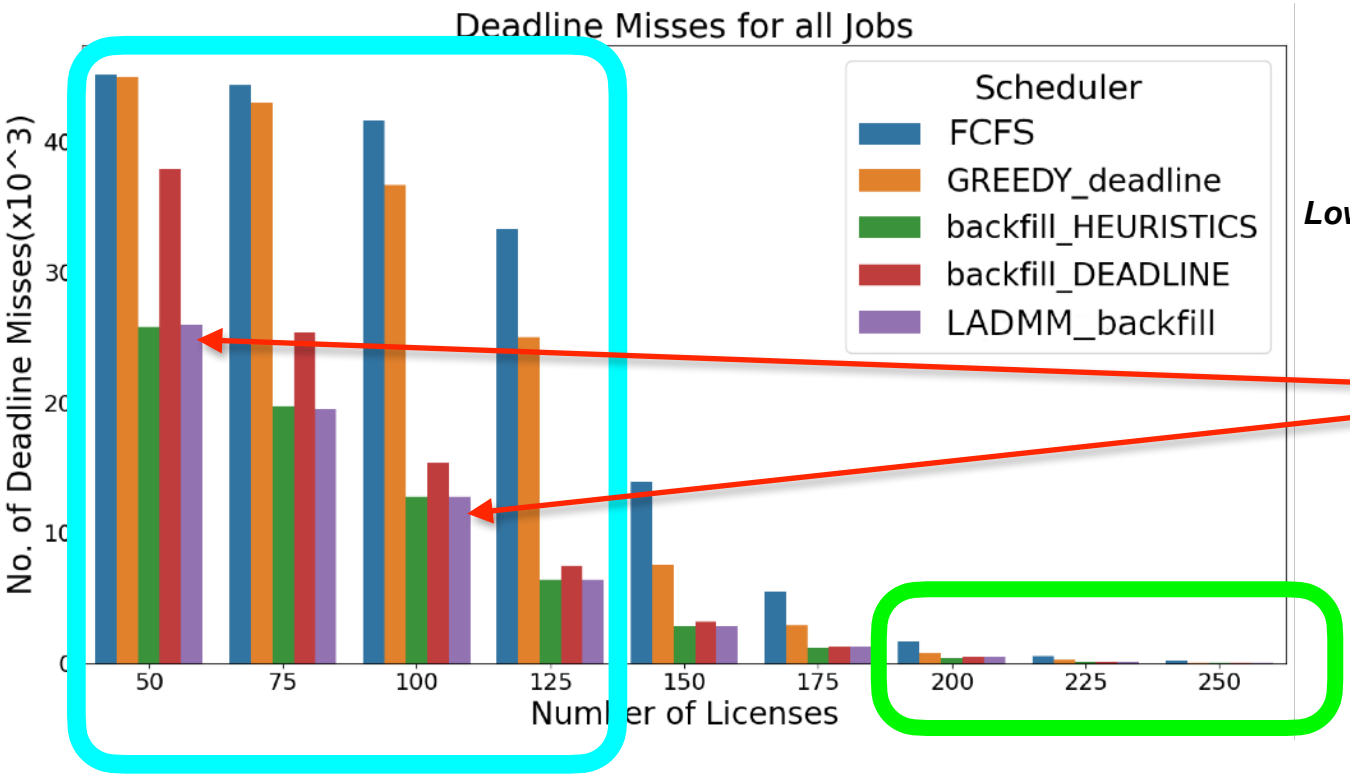
% of License Utilization of LS-Dyna Jobs



High License Utilisation: almost 90% vs ~40% of FCFS and GREEDY_deadline

Shorter Makespan: batch of jobs finish much faster

Deadline Miss Ratio

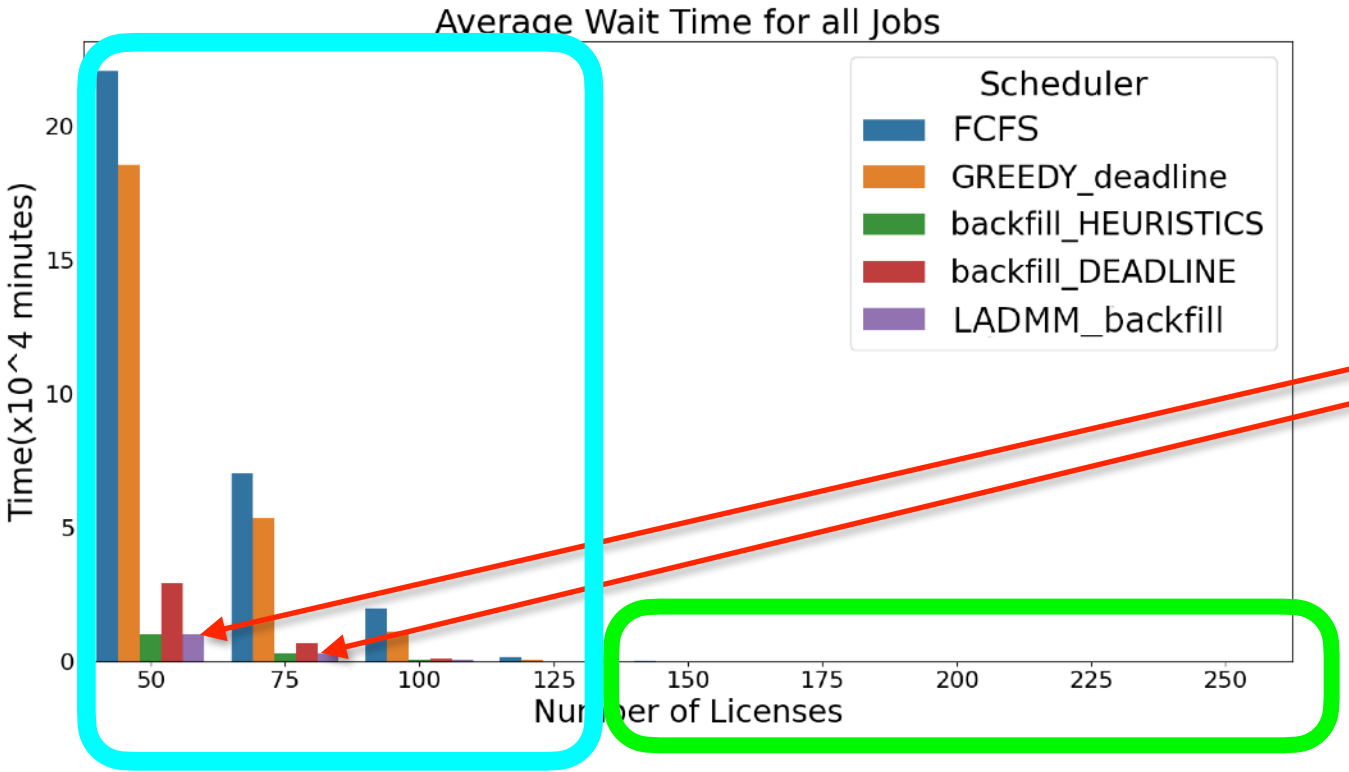


Lower License Count:

- ~ 43-80% reduction vs FCFS
- ~ 43-74% reduction vs GREEDY_deadline
- ~ 14 -32 % reduction vs backfill_DEADLINE & backfill_HEURISTICS

Higher License Count: performance converges due to reduced resource contention

Average Wait Time



Lower License Count:

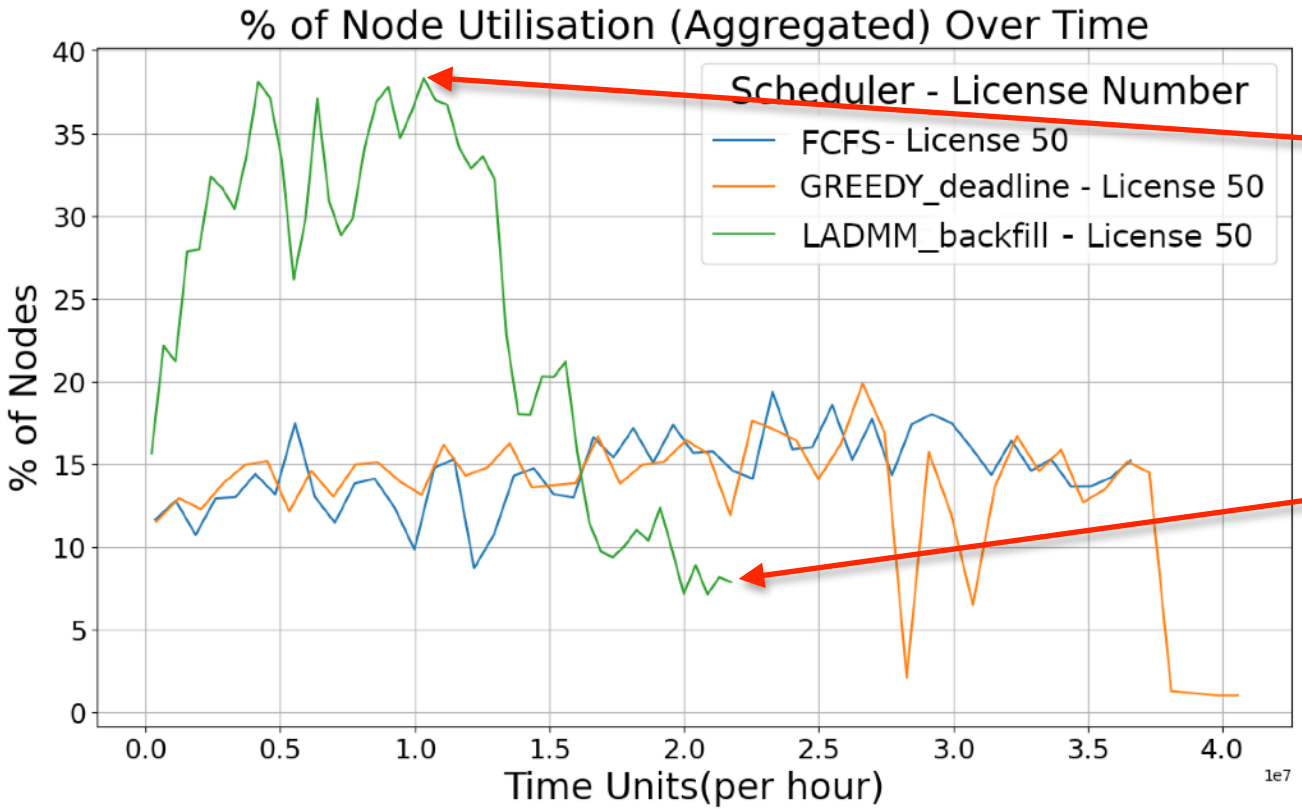
- ~ 97% reduction vs FCFS
- ~ 93% reduction vs GREEDY_deadline
- ~ 50% reduction vs backfill_DEADLINE

Higher License Count: performance converges due to reduced resource contention

Public Cloud Costs(x1000 $)

License Number	FCFS	GREEDY deadline	BACKFILLING heuristics	BACKFILLING deadline	LADMM backfill
50	0	0	3353.485	0	0
75	0	0	229.009	0	0
100	0	0	934.808	0	0
125	1.062	0	907.626	.965	0
150	1.106	324	313.283	4.069	4.130
175	7.261	2.649	125.788	4.788	5.392
200	6.292	6.099	45.241	5.882	7.470
225	7.705	7682	13.616	5.033	8.880
250	10.062	8.186	14.365	5.943	9.899

Global Node Utilisation



High Node Utilization: *avg* ***15-20%*** *higher as compared to baselines*

Shorter Makespan: batch of jobs finish much faster

Conclusion & Future Work

- Proposed **LADMM_backfill**: a license-aware, deadline-driven scheduler for hybrid HPC-cloud environments.
- Integrates **license and core availability** using optimized scoring heuristics.
- Outperforms FCFS, Greedy, and other backfilling baselines in:
  - ⏱️ Deadline misses: ↓ up to **80%**
  - 🔑 License utilization: ↑ up to **94%**
  - 🕒 Wait time: ↓ up to **97%**
  - 💰 Cost: Maintained or reduced, especially under tight license constraints.

- Extend to **workflow-based scheduling** with inter-task dependencies.
- Support **pay-as-you-go licensing models** for greater flexibility.
- Integrate **dynamic pricing** and **ML-based demand prediction**.
- Validate at **larger scales** and across **diverse CAE workloads** for robustness.

Thank You !
Questions ?

[1]Henkel, N., Treiber, S.: Cost-effective sizing of your HPC cluster for CAE sim ulations. In: IEEE Int. Conf. High Perform. Comput. (2015). https://api.semanticscholar.orgCorpusID:35686422
[2]https://www.cs.huji.ac.il/labs/parallel/workload/
[3]https://pypi.org/project/simulus/