On Optimal Resource Allocation Algorithms for Parallelisable Jobs

Aadya Mudgil [1] Supervisor: Arpan Mukhopadhyay [1]

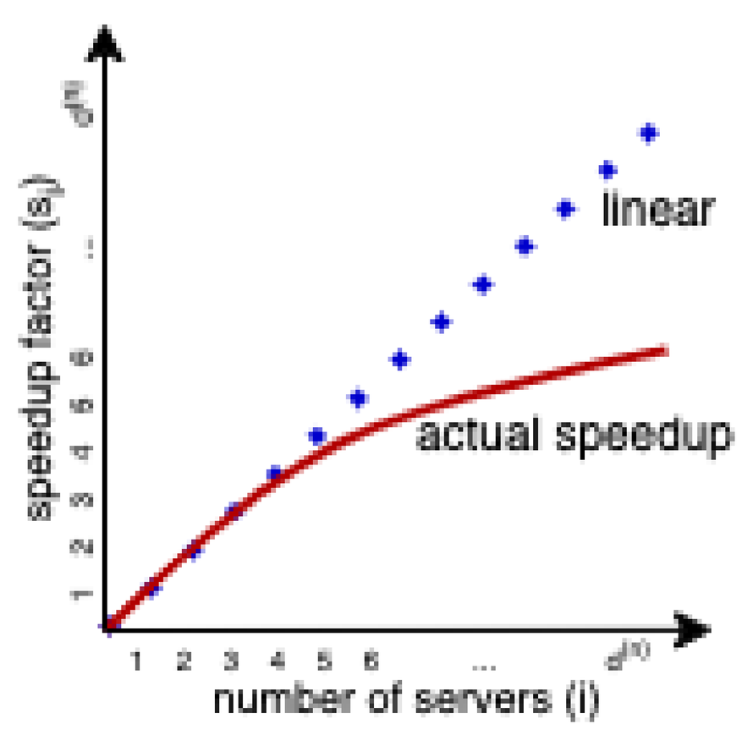
[1]Department of Computer Science

Introduction

Unprecedented demands are being placed on compute resources due to HPC jobs such as next-gen machine learning. Distributed systems rely on resource provisioning algorithms to parallelise these jobs across many nodes; however, traditional algorithms are unlikely to perform- an **inefficient and costly** waste of power and time.

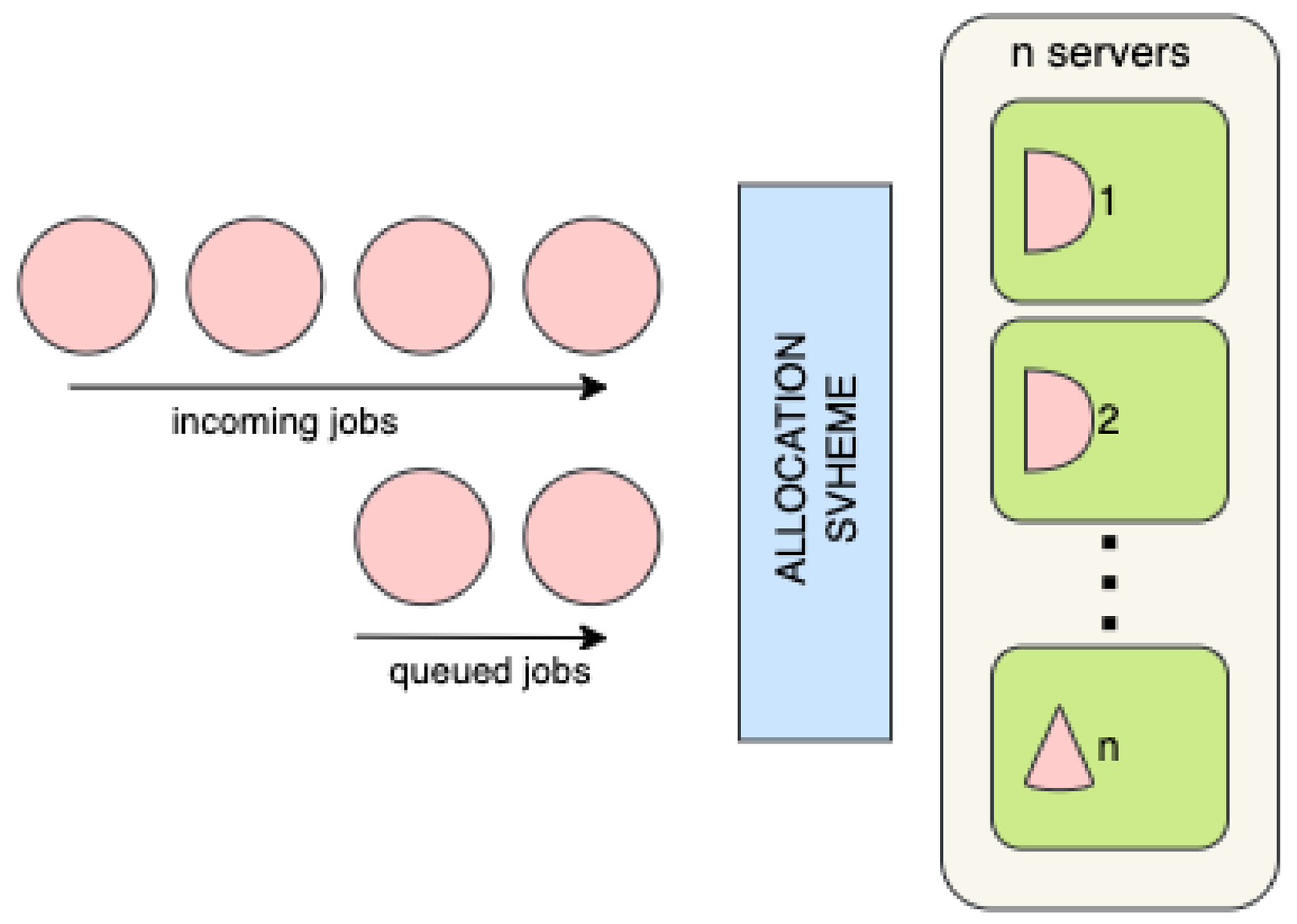
This poster aims to **design the optimal algorithm for parallelisable tasks**, under a real-world workload, and validate theoretical expectations with the results from system simulations.

Formulating the Optimisation Problem



Consider a system with n servers, where jobs are parallelisable up to $d^{(n)}$ servers. Jobs are all of the same class: same arrival rate (λ), mean size (μ), concave speedup function following Amdahl's law (vector S)- where s_i is how much quicker a job runs compared to a single server.

While allocating a job to more servers lowers its execution time, it reduces the number of servers available to future jobs. If a job arrives at a point where all servers are occupied, it must queue— this wait time is unacceptable.



We aim to find an asymptotically optimal allocation algorithm that minimizes the execution time across all submitted jobs and eliminates wait time as system size increases ($n \to \infty$).

Ghanbarian et al.[2] formalise this tradeoff in the following convex optimisation problem:

Objective Function: minimise execution time across all jobs

$$\underset{y=(y_i, i\in[d^{(n)}])}{\text{minimize}} \quad \frac{1}{\lambda^{(n)}} \sum_{i\in[d^{(n)}]} y_i$$

Constraints

$$\text{subject to} \quad r(y) = \sum_{i\in[d^{(n)}]} s_i y_i = \frac{\lambda^{(n)}}{\mu^{(n)}}, \qquad \text{(P)}$$

← **1) Rate conservation constraint:** for a stable Markovian system arrival rate = departure rate

$$q_1(y) = \sum_{i\in[d^{(n)}]} i y_i \le 1,$$

← **2) Busy servers ≤ total servers**

$$y_i \ge 0, \quad \forall i \in [d^{(n)}],$$

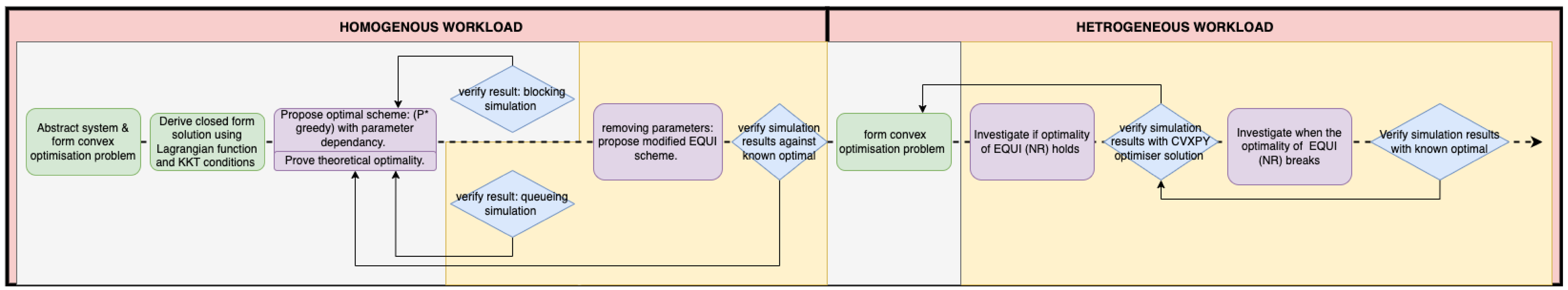
← **3) Non-negative results constraint**

Solving (P) gives: $y_i^{*,(n)}$, $p_i^{*,(n)} = \frac{s_i y_i^{*,(n)}}{\lambda^n}$

From the closed form solution of (P) they derive an optimal allocation scheme, greedy($p^{*,(n)}$) that allocates each job to as many servers as possible up to $i \in [d^{(n)}]$ servers with probability $p_i^{*,(n)}$. They prove this result using Stein's method and have verified this through system simulation.

A limitation of using greedy($p^{*,(n)}$) in industry is the parameter dependency: it requires λ, s_i to be known in advance. To eliminate this, we explore an alternative scheme: EQUI with NO REPACKING: EQUI-NR

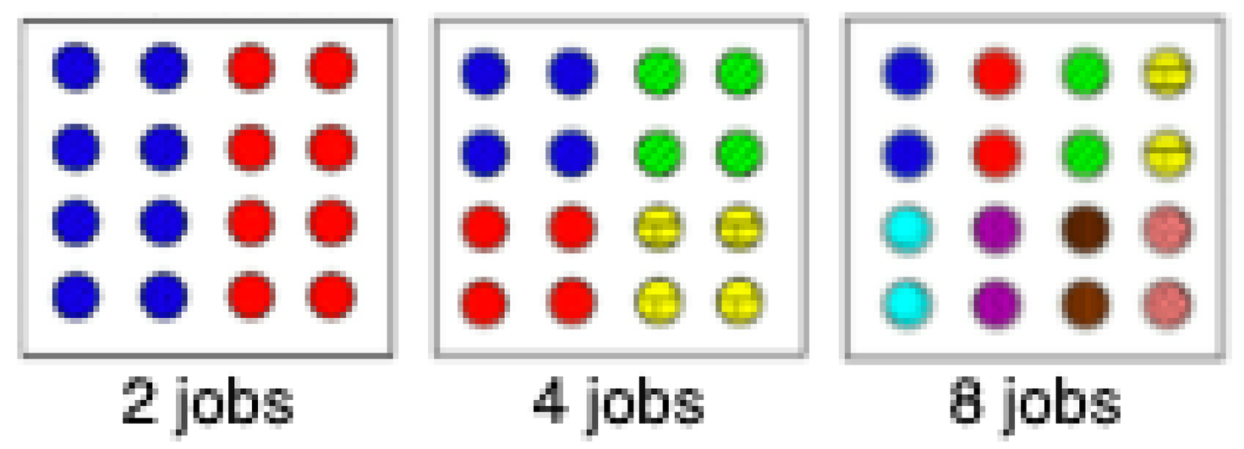
Methodology



Flowchart 1. Grey area is existing work, yellow area is the contribution of this project

Overcoming Parameter Dependency with EQUI-NR

The EQUI policy is an optimal[1] , simple and preemptive scheme where all servers are evenly distributed among the existing jobs as seen in the figure[3] . EQUI involves **repacking**: jobs currently running on a server can be removed and reallocated, leading to inefficiencies.



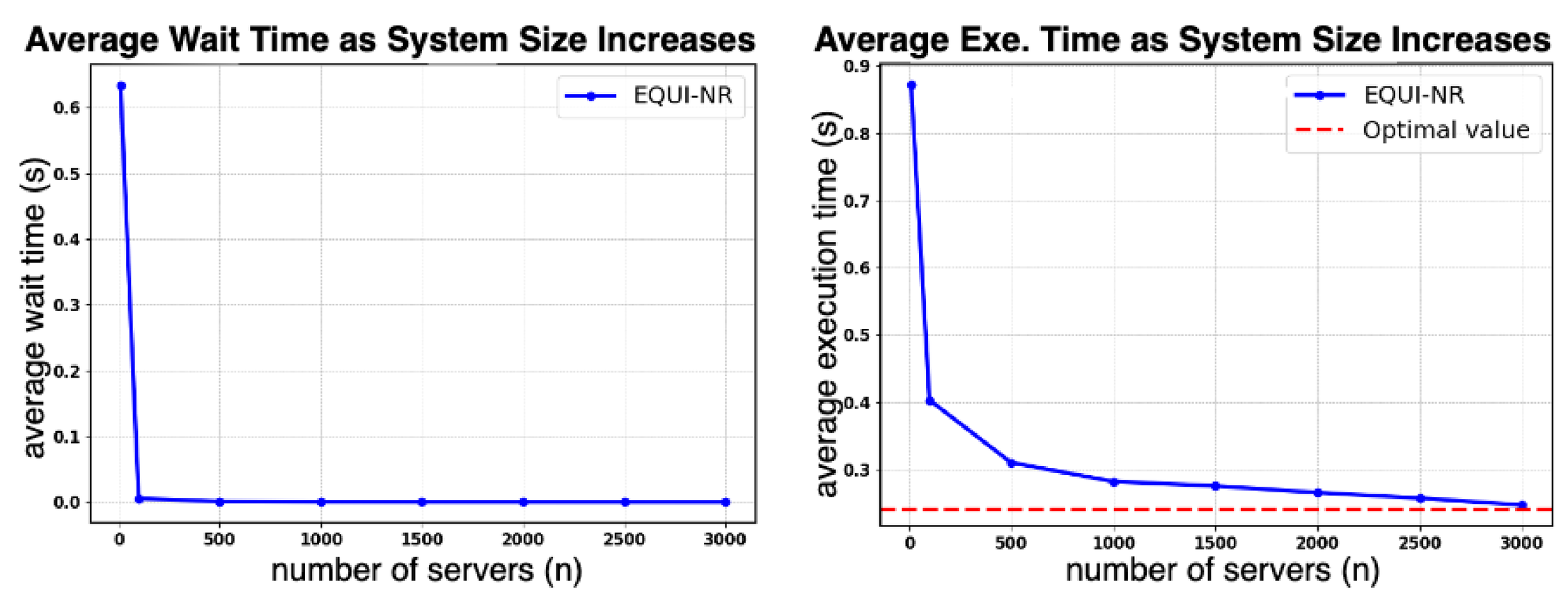
To address this, we propose the modified EQUI-NR that eliminates this repacking by distributing $min(z,$ available servers, $d^{(n)})$ servers to **only the new or queued jobs**, with z calculated by:

ALGORITHM: EQUI-NR

1: **Input:** n (number of servers in the system), $Y(t)$ (total number of jobs in the system)
2: Compute $x = \frac{n}{Y(t)}$ **on every job entry or exit**
3: Compute $p = \frac{x-\lfloor x\rfloor}{\lceil x\rceil-\lfloor x\rfloor}$
4: $z = \lceil x\rceil$ servers with probability p, $\lfloor x\rfloor$ servers with probability $1-p$
5: **Output:** z (the computed allocation)

Simulation results verify the optimality of EQUI-NR for jobs of the same class

The simulation is run for speedup curve $S(i) = i^{0.5}$, tracking results for n = 10 to 3000.



Graphs 1 and 2: Graph 1 shows that the wait (queueing) time converges to zero as system size increases. In Graph 2, the processing time converges to the same limit as the greedy($p^{*,(n)}$) scheme, which is the optimal value.

Above results match the expectation for an optimal algorithm, which is derived from solving the closed-form solution for problem (P).

Heterogeneous Workload

Thus far, we have considered an abstracted workload in which all submitted jobs are the same type. We now extend this analysis to a more realistic, heterogeneous workload, where there are j distinct classes of jobs, each with varying parameters.

Heterogeneous workload convex optimisation problem[2] :

$$\underset{y=(y_{ij})}{\text{minimize}} \quad \frac{1}{\lambda} \sum_{i,j} y_{i,j}$$

$$\text{subject to} \quad \sum_{i\in[d_j]} s_{i,j} y_{i,j} = \frac{\lambda_j}{\mu_j}, \forall j \in [l], \qquad \text{(Q)}$$

$$\sum_{i,j} i y_{i,j} \le 1,$$

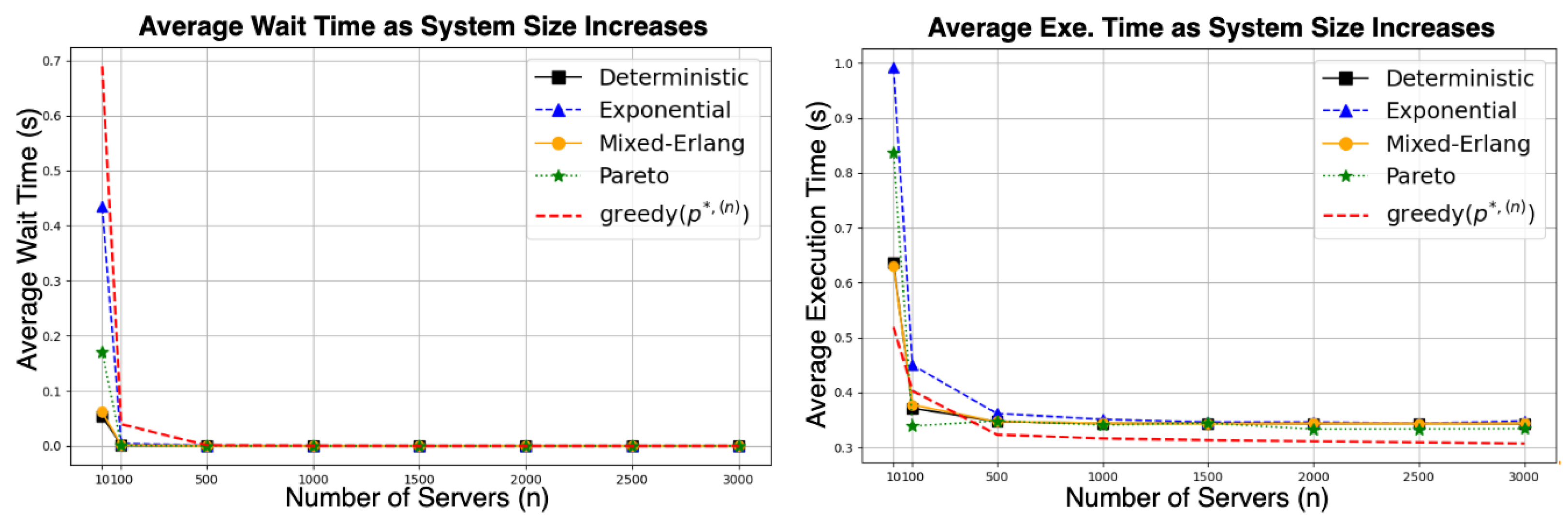
$$y_{i,j} \ge 0, \quad \forall j \in [l], i \in [d_j]$$

Investigating EQUI-NR under Heterogeneous Workload

Unlike optimization problem (P), problem (Q) lacks a closed-form solution, complicating how we can verify if an algorithm converges to the optimal average execution time. We use a CVXPY optimizer to solve (Q), obtaining the optimal greedy($p^{*,(n)}$) value for the heterogeneous case, and compare it with the performance of EQUI-NR.

Simulation results for EQUI-NR under heterogeneous workload

The simulation is run with a workload of 5 distinct job classes that have different parameters. Multiple job size distributions are used to investigate insensitivity.



EQUI-NR performs efficiently, but not optimally. We observe that the average wait (queuing) time converges to 0 as the system size increases, which is the desired result. However, the greedy($p^{*,(n)}$) scheme, which shows the optimal behavior, converges to an average execution time that is lower than EQUI-NR.

We also discover that EQUI-NR is asymptotically insensitive, this is highlighted in the table below. Across all 4 job size distributions investigated, both the average wait time and execution time follow a very similar path of convergence. This is a desirable characteristic as real-world workloads will have varying job sizes.

Table 1. Performance metrics for EQUI-NR across various job size distributions

	Mean Execution Time				Mean Wait Time			
Servers	Exp	Det	Erlang	Pareto	Exp	Det	Erlang	Pareto
10	0.9924	0.6358	0.6306	0.8379	0.4350	0.0548	0.0617	0.1714
100	0.4498	0.3711	0.3774	0.3388	0.0048	0.0012	0.0014	0.0009
500	0.3615	0.3470	0.3469	0.3484	0.0003	0	0	0
1000	0.3512	0.3435	0.3436	0.3407	0	0	0	0
1500	0.3462	0.3434	0.3430	0.3442	0	0	0	0
2000	0.3462	0.3433	0.3439	0.3301	0	0	0	0

Conclusions & Further Research

1. The asymptotic optimality of the EQUI-NR scheme for the single class job case is confirmed with simulation results.
2. EQUI-NR is effective but not optimal for heterogeneous workloads.
3. We observe the asymptotic insensitivity of EQUI-NR for job classes with inherent size defined by different probability distributions.
4. **The simulation platform is released publicly (link in QR code) to profile own algorithms.**

---

1. Mathematically prove and issue performance guarantees for EQUI-NR with heterogeneous workloads
2. Use reinforcement learning to train an ML model that dynamically determines useful parameters we can use to improve EQUI-NR to get optimal processing time.

References

[1] Benjamin Berg, Thomas Bonald, and Isi Mitrani. Towards optimality in parallel job scheduling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):40, 2017.

[2] Samira Ghanbarian, Arpan Mukhopadhyay, Ravi R. Mazumdar, and Fabrice M. Guillemin. On optimal server allocation for moldable jobs with concave speed-up. *arXiv preprint arXiv:2406.09427*, 2024.

[3] Mor Harchol-Balter. Open problems in queueing theory inspired by datacenter computing. *Queueing Systems*, 97:3–37, 2021.

aadyamudgil1@gmail.com