國立臺灣科技大學
資訊工程系

碩士學位論文

雲端 GPU 渲染排程問題 (CGRSP)：異質 GPU 叢集之數學建模與實驗評估

Cloud GPU Rendering Scheduling Problem (CGRSP): A Mathematical Formulation and Empirical Evaluation on Heterogeneous GPU Fleets

研究生　　：Michael Liem (林墾森)
學號　　：M11215818
指導教授　　：Prof. Yuan-Shin Hwang (黃元欣)
共同指導教授：Prof. Vincent F. Yu (喻奉天)

中華民國一一五年十月一日

摘　要

本論文提出「雲端 GPU 渲染排程問題」(CGRSP)，針對異質 GPU 叢集上的三維渲染工作負載建立嚴謹的數學模型與評估框架。

基於 6,627 筆 RunPod Secure Cloud 真實渲染任務，建構 Python 離散事件模擬器，比較十種排程演算法（八種基準策略，含一種自適應 EDF-SPT 混合策略，以及本研究提出之新型預期式滾動視界（RH）排程器與成本感知截止期限風險（CADR）排程器），採用異質截止期限分布（20% 緊急 1 小時、80% 寬鬆 8 小時）。

壅塞日條件下（∼950 任務/天，$C_{\max} = 5$，$\rho > 1$），RH 達成最低截止期限違反率 (9.50%) 且等待時間與 SPT 並列最低（140.24 分鐘，配對檢定不顯著，$p = 0.671$），於違反率上顯著優於 SPT+Rescue（其等待時間優勢經統計校正後未達顯著）。RH 之預期機制受負載閾值控制：飽和以下時爲即將到來的緊急（急件）任務保留閒置容量，將湧浪日違反率自 6.05% 降至 1.59%；飽和時關閉保留，故飽和負載之結果不變。CADR 達成接近 RH 之違反率（10.09%），同時大幅降低 RH 之平均延遲時間（6.43 分鐘，相較 RH 之 30.86 分鐘），在壅塞飽和下違反率亦大幅低於 SPT+Rescue，等待時間則與 SPT+Rescue 無統計差異（$p = 0.927$）。SPT 達成最低等待時間（141.03 分鐘，與 RH 並列），但違反率顯著高於 EDF（18.60% vs 15.81%）；SPT+Rescue 提供明顯更低之違反率（12.50%）並具保守截止期限保護。FIFO、LCF 與 Balanced 因忽略截止期限而產生 24.85% 至 26.39% 違反率。湧浪日負載下，於截止期限掃描之基準策略中，緊急截止期限比例 $\leq 30\%$ 時 EDF 之違反率顯著低於 SPT，比例爲 50% 時兩者無法區分（差異不顯著，$p = 0.69$），比例 $\geq 70\%$ 時 SPT 之違反率顯著低於 EDF。

本論文貢獻具統計支撐的排程指引：當僅需最小化截止期限違反數量時，以 RH 作爲預設策略（最低違反率且等待時間與 SPT 並列最低）；當違反的嚴重程度（延遲時間）亦重要時，採用 CADR（接近 RH 之違反率且延遲時間大幅降低）；SPT 仍爲低等待之強健基準；提升 $C_{\max}$ 超出 RunPod 預設值 5 以降低殘餘違反率。

關鍵詞： 雲端 GPU 排程、異質 GPU 叢集、離散事件模擬、截止期限感知排程、RunPod Secure Cloud、三維渲染工作負載

Abstract

This thesis formalises the Cloud GPU Rendering Scheduling Problem (CGRSP), the challenge of scheduling batch 3D rendering workloads on heterogeneous on-demand GPU fleets under stochastic node availability and mixed deadline constraints. Three elements distinguish CGRSP from prior GPU scheduling work: empirically calibrated GPU heterogeneity (RTX 3090, A5000, A4000, A4500 on RunPod Secure Cloud), a stochastic, stock-status-driven model of qualitative node availability, and a bi-objective function that treats quality of service (waiting time and soft-deadline misses) as the primary criterion and rental cost as a secondary, tiebreaking criterion.

A Python discrete-event simulator calibrated against 6,627 real rendering jobs evaluates ten scheduling policies (eight benchmarks, including an adaptive EDF-SPT hybrid, plus the novel anticipative Rolling-Horizon (RH) scheduler and the Cost-Aware Deadline-Risk (CADR) scheduler) across four experimental phases using a heterogeneous deadline distribution (20% tight 1-hour, 80% loose 8-hour). Under hectic capacity saturation ($\sim$950 jobs/day, $C_{\max} = 5$, $\rho > 1$), RH achieves the lowest deadline miss rate (9.50%) at the lowest wait tier (140.24 min, statistically tied with SPT, $p = 0.671$), significantly outperforming SPT+Rescue on miss (its wait advantage over SPT+Rescue is not statistically significant after correction). RH's anticipation is load-gated: below saturation it reserves idle capacity for imminent tight (rush-order) arrivals, cutting surge-day miss from 6.05% to 1.59%, while the offered-load gate disables reservation under saturation so the saturated-load results are unchanged. CADR achieves near-RH miss (10.09%) while cutting RH's mean tardiness substantially (6.43 min vs 30.86 min), also achieving substantially lower miss than SPT+Rescue at hectic saturation, at a wait the paired test cannot distinguish from SPT+Rescue's ($p = 0.927$). SPT achieves the lowest wait (141.03 min, tied with RH) but is significantly outperformed by EDF on miss rate (18.60% vs 15.81%), and SPT+Rescue achieves notably lower miss than SPT with conservative deadline protection. FIFO, LCF, and Balanced incur 24.85% to 26.39% miss from deadline ignorance. At surge-day load, among the baseline policies in the deadline sweep, EDF achieves significantly lower miss

than SPT at tight-deadline fractions $\leq$30%, the two are statistically indistinguishable at 50% ($p = 0.69$), and SPT achieves significantly lower miss at $\geq$70%.

These findings yield deployment rules for cloud rendering operators: deploy RH when minimising the count of missed deadlines is the sole priority (lowest miss, wait tied with SPT); deploy CADR when the severity (lateness) of misses also matters (near-RH miss with substantially lower tardiness); SPT remains a strong low-wait baseline; increase $C_{\max}$ beyond the RunPod default of 5 to reduce residual miss at saturation.

Keywords: Cloud GPU scheduling, heterogeneous GPU fleet, discrete-event simulation, deadline-aware scheduling, RunPod Secure Cloud, 3D rendering workloads

Acknowledgements

誌謝

I would like to express my sincere gratitude to my advisor, Prof. Yuan-Shin Hwang, for his invaluable guidance, encouragement, and patience throughout the course of this research. His expertise in computer systems and scheduling algorithms provided the intellectual foundation upon which this thesis was built. I am equally grateful to my co-advisor, Prof. Vincent F. Yu, for his insightful suggestions on the optimisation formulation and his support throughout the research process.

I am grateful to the members of my oral examination committee for their constructive feedback and insightful questions, which significantly improved the quality of this work.

Special thanks go to my labmates and fellow graduate students in the Department of Computer Science and Information Engineering at NTUST, whose discussions and companionship made the research journey both productive and enjoyable.

I acknowledge RunPod Inc. for making GPU rental pricing data publicly available, which enabled the empirical calibration of the simulator used in this study.

Finally, I owe my deepest gratitude to my family for their unwavering support, understanding, and encouragement throughout my graduate studies. This work is dedicated to them.

Michael Liem (林墾森)

Taipei, Taiwan

July 2026

Contents

摘要 (Chinese Abstract) i

Abstract ii

Acknowledgements (誌謝) iv

符號索引 (List of Symbols) ix

List of Figures (圖目錄) xii

List of Tables (表目錄) xiv

List of Algorithms (演算法目錄) xvi

1 Introduction 1

1.1 Research Background 1

1.2 Computational Complexity 3

1.3 Research Objective 4

1.4 Research Contribution 5

1.5 Scope and Limitations 6

1.6 Organisation of Thesis 7

2 Background and Related Work 9

2.1 Cloud GPU Computing 9

2.2 Classical Job Scheduling 10

2.2.1 Shortest Processing Time (SPT) 11

2.2.2 Earliest Deadline First (EDF) 11

2.2.3 First-In-First-Out (FIFO) 12
2.2.4 Other Heuristics 12
2.3 Heterogeneous Machine Scheduling 13
2.4 Reinforcement Learning for Resource Management 14
2.5 Discrete-Event Simulation 15
2.6 3D Rendering Workloads 16
2.7 Summary 16
3 Problem Formulation and Mathematical Model 19
3.1 Mathematical Notation 20
3.2 Problem Description 20
3.3 Mathematical Model 24
3.4 Modelling Simplifications and Assumptions 29
3.5 Numerical Example 31
4 System Design and Scheduling Algorithms 33
4.1 System Architecture Overview 33
4.2 Discrete-Event Simulator Core 34
4.3 GPU Fleet Model 35
4.4 Job Generator 38
4.5 Scheduling Algorithms 38
4.5.1 FIFO: First-In-First-Out 39
4.5.2 SPT: Shortest Processing Time 39

4.5.3 EDF: Earliest Deadline First 40
4.5.4 LCF: Lowest Cost First 40
4.5.5 Balanced: Weighted Speed-Cost Composite 41
4.5.6 Random: Random Baseline 41
4.5.7 Design Progression 41
4.5.8 SPT+Rescue: Hybrid Throughput-Deadline Policy 43
4.5.9 Illustrative Scheduling Example 45
4.5.10 Rolling-Horizon: Anticipative Receding-Horizon Scheduler . . . 45
4.5.11 CADR: Cost-Aware Deadline-Risk 51
4.5.12 Adaptive: Queue-Pressure EDF-SPT Hybrid 54
4.6 Metrics Collection 56
5 Experimental Results and Analysis 58
5.1 Experimental Setup 58
5.1.1 Platform and Data 58
5.1.2 Evaluation Phases 64
5.1.3 Statistical Methodology 65
5.2 Phase 2: Seven-Scheduler Benchmark 66
5.3 Phase 3 Statistical Validation 68
5.4 Ablation Analysis 74
5.4.1 Capacity Lower Bound 77
5.5 Sensitivity Analysis 78

5.5.1 Day-Type Sensitivity ($C_{\max} = 5$) 78
5.5.2 Deadline Tightness Sensitivity 81
5.5.3 Concurrency Limit Sensitivity 82
5.5.4 Rescue Threshold Sensitivity 86
5.5.5 Critical-Ratio Threshold Sensitivity 87
5.5.6 Queue-Pressure Threshold Sensitivity 88
5.5.7 Anticipation Reservation Sensitivity 89
5.5.8 Availability-Model Sensitivity 91
5.5.9 Objective-Weight Sensitivity 93
5.6 Phase 4: Monthly Operational Simulation 97
5.7 Summary of Findings 102
6 Discussion and Future Work 104
6.1 Theoretical Implications 104
6.1.1 Choosing Between RH and CADR 109
6.2 Limitations and Future Research 111
6.2.1 Limitations 111
6.2.2 Future Research 113
6.3 Concluding Remarks 115
References 117

符號索引
List of Symbols

Symbol	Description
$\mathcal{J}$	Set of all rendering jobs
$\mathcal{N}$	Set of nodes
$\mathcal{G}$	Set of resource (GPU) types
$\mathcal{K}$	Set of discrete complexity levels
$\mathcal{J}_t$	Set of jobs waiting in the queue at time t
j	Job index
m	Node index
g	Resource type index
$g(m)$	Resource type of node m
g_j	Resource type of the node assigned to job j: $g(m)$ for the m with $x_j^m = 1$
κ_j	Complexity level of job j
d_j	Deadline of job j
$T_{\text{tight}}, T_{\text{loose}}$	Tight / loose deadline windows (3600 s / 28800 s)
T_j	Deadline window assigned to job j: $T_j \in \{T_{\text{tight}}, T_{\text{loose}}\}$
t_j^{submit}	Submission time of job j
t_j^{dispatch}	Dispatch (slot-acquisition) time of job j
t_j^{start}	Start time of job j
t_j^{complete}	Completion time of job j
$W(j, g)$	Realised execution time of job j on GPU type g (log-normal)
$\overline{W}(\kappa, g)$	Mean (expected) execution time for complexity κ on GPU type g
$\sigma_{\kappa,g}$	Log-space dispersion of execution time within stratum (κ, g)
n_g	Number of provisioned nodes of resource type g in the fleet

Symbol	Description
p_g	Per-second rental price of GPU type g
$s_g(t)$	Stock status of GPU type g at time t (High / Medium / Low)
β_g	Baseline high-stock probability of GPU type g
$\phi(t)$	Time-of-day stock multiplier at time t
τ_j^{prov}	Provisioning delay incurred when job j acquires an instance (stock-status dependent; unbilled, counts toward w_j)
L_s, U_s	Lower / upper provisioning-delay bounds for stock status s
$p_g^H(t), p_g^M(t)$	High- / Medium-stock probabilities of resource type g at time t
C_{max}	Maximum simultaneous GPU instances (concurrency limit)
λ	Job arrival rate (jobs/second)
ρ	Offered load: $\lambda \overline{W} / C_{\text{max}}$
$x_j^m \in \{0,1\}$	Assignment variable: 1 if job j assigned to node m
$a_m^t \in \{0,1\}$	Idle indicator for slot m at time t (1 if free, 0 if running)
w_j	Waiting time of job j
r_j	Response time of job j
$\delta_j \in \{0,1\}$	Deadline adherence: 1 if $t_j^{\text{complete}} \leq d_j$
τ_j	Tardiness of job j: $\max(0, t_j^{\text{complete}} - d_j)$
c_j	Execution cost of job j
Δ_{miss}	Deadline miss rate: $\frac{1}{\lvert\mathcal{J}\rvert} \sum_j (1 - \delta_j)$
C_{QoS}	QoS cost component, the primary lexicographic criterion: $\sum_j [\alpha_1 w_j + \alpha_2 (1 - \delta_j)]$
C_{Cost}	Operational cost component, the secondary (tiebreaking) criterion: $\sum_j c_j$
α_1, α_2	QoS penalty weights inside C_{QoS} (per second of waiting, per missed deadline)
$w_{\text{qos}}, w_{\text{cost}}$	Scalarisation weights (weighted-sum robustness variant; RH placement surrogate)

Symbol	Description
θ	SPT+Rescue laxity threshold (600 s)
θ_{eff}	Adaptive's queue-pressure-widened threshold ($= \theta$ normally, $= T_{\text{loose}}$ when $\lvert Q\rvert > P$)
w_c	SPT-family node-selection cost weight (0.3)
w_b	Balanced speed-cost weight (0.8)
$\psi(s_m)$	LCF multiplicative stock-status penalty ($\{1.0, 1.05, 1.15\}$)
$\text{sp}(s_m)$	SPT/Balanced-family additive stock-status penalty ($\{0, 0.2, 1.0\}$)
$L_j(t)$	Laxity of job j at time t: $d_j - t - \min_{m \in I} \overline{W}(\kappa_j, g(m))$ over the idle-node set I
e_j	Planning estimate of job j's service time: $\min_m \overline{W}(\kappa_j, g(m))$
t_{free}	Soonest time a slot frees for a job not dispatched now (Rolling-Horizon)
ρ_{res}	Rolling-Horizon reservation count: idle slots held for anticipated tight arrivals (1)
ρ_{gate}	Offered-load gate above which anticipation is disabled (0.95)
$\text{CR}_j(t)$	Critical ratio of job j at time t: $(d_j - t)/e_j$ (CADR)
CR^*	CADR critical-ratio threshold separating at-risk from safe jobs (3)
P	Adaptive queue-pressure threshold: backlog size above which the urgent tier widens (10)

List of Figures

1.1 The CGRSP problem at a glance 3

2.1 GPU fleet execution-time heterogeneity and pricing 17

3.1 CGRSP scheduling decision at a single event 23

3.2 Single-job lifecycle and metric intervals 28

3.3 Numerical example: three-job schedule on two GPUs 32

4.1 CGRSP system architecture 33

4.2 Scheduler design lineage 42

4.3 SPT+Rescue laxity rescue decision flow 43

4.4 Illustrative Gantt chart: FIFO vs SPT+Rescue 46

4.5 Rolling-Horizon slot timelines and job classification 49

4.6 CADR critical-ratio triage and cost-aware placement 53

5.1 Empirical dataset: arrival pattern and execution times 60

5.2 Diurnal availability model and real arrival pattern 63

5.3 Observed RunPod availability record 64

5.4 Phase 2 benchmark: normal vs hectic day 67

5.5 Phase 3: 95% confidence intervals 69

5.6 Phase 3: per-seed distributions 70

5.7 Paired t-test effect sizes 70

5.8 Ablation waterfall across the design chain 75

5.9 Day-type sensitivity results 80

5.10 Deadline-tightness sensitivity 82

5.11 Concurrency limit sensitivity 85

5.12 SPT+Rescue threshold sensitivity 87

5.13 CADR critical-ratio threshold sensitivity 88

5.14 Anticipation reservation sensitivity 90

5.15 Objective-weight sensitivity 96

5.16 Operational cost by scheduler 98

5.17 Phase 4 monthly results 99

5.18 Daily miss rate over the simulated month 99

5.19 Miss rate vs tardiness trade-off 101

List of Tables

2.1 Positioning of CGRSP against related work 17

3.1 Sets and Indices 20

3.2 System Parameters 21

3.3 Decision Variables and State Variables 22

4.1 GPU fleet configuration 36

5.1 Phase 2 benchmark results 67

5.2 Phase 3 statistical validation 69

5.3 Pairwise paired t-test results 71

5.4 Ablation analysis: evolutionary chain 75

5.5 Day-type sensitivity summary 79

5.6 Deadline-tightness sweep 81

5.7 Concurrency limit sensitivity 84

5.8 SPT+Rescue threshold sensitivity 86

5.9 CADR critical-ratio threshold sensitivity 88

5.10 Anticipation reservation sensitivity 90

5.11 Availability-model sensitivity 91

5.12 Objective-weight sensitivity 94

5.13 Tardiness-objective re-scoring 96

5.14 Phase 4 monthly summary 98

5.15 Summary of key experimental findings 103

6.1 RH vs CADR decision guidance 110

List of Algorithms

1 FIFO Scheduler 39

2 SPT Scheduler 40

3 SPT+Rescue Scheduler 44

4 Rolling-Horizon Scheduler 51

5 CADR Scheduler 54

6 Adaptive EDF-SPT Hybrid Scheduler 56

Chapter 1 Introduction

1.1 Research Background

Three-dimensional (3D) rendering is the computationally intensive process of generating photorealistic images from geometric scene descriptions, a cornerstone technology in film production, video game development, architectural visualisation, and interactive media. A single high-fidelity frame may require hours of computation on a dedicated workstation; rendering a full animation sequence at production quality demands thousands of such frames [1]. The consequent demand for computational power has driven the 3D rendering industry towards cloud GPU platforms, where rendering studios rent time on GPU nodes rather than maintaining on-premise hardware infrastructure.

Cloud GPU markets have matured rapidly. Platforms such as RunPod, Lambda Labs, and CoreWeave now offer heterogeneous GPU fleets (comprising nodes with different architectures, memory capacities, and performance profiles) at a range of price points, with billing granularity as fine as one second [2–4]. A rendering studio submitting jobs to such a platform faces a multi-dimensional optimisation challenge: jobs arrive stochastically, GPU node availability fluctuates due to competing customers, maintenance windows, and hardware failures, and jobs carry soft or hard deadlines tied to production schedules. Selecting the wrong GPU for a job, or failing to schedule deadline-critical frames promptly, can cascade into missed delivery milestones and financial penalties.

Scheduling in heterogeneous cloud computing environments has been an active research area since the early 2000s [5, 6]. However, the specific combination of features present in cloud GPU rendering (per-second billing, qualitative rather than exact node availability information, stochastic provisioning delays driven by competing customer demand, and real-world GPU performance heterogeneity at the rendering-task level) has not been studied as a unified problem. Existing surveys on GPU scheduling in data centres [7] focus primarily on deep learning training workloads, where the access model (reserved

clusters or dedicated hardware) differs fundamentally from the marketplace rental model used in cloud rendering.

This study addresses this gap by formulating the **Cloud GPU Rendering Scheduling Problem (CGRSP)**: a mathematical model and empirical evaluation framework for scheduling 3D rendering workloads on heterogeneous, stochastically available GPU fleets. At its core, CGRSP asks: *given a stream of rendering jobs arriving over time, each with a deadline, a complexity level, and an estimated execution duration, and given a heterogeneous fleet of GPU nodes with stochastic availability, which GPU should each job be assigned to, and in what order should jobs be processed, to jointly minimise waiting time, deadline miss rate, and operational cost?*

Three challenges make CGRSP hard in practice:

1. **GPU Heterogeneity.** The fleet in this study consists of four GPU types (RTX 3090, RTX A5000, RTX A4000, and RTX A4500) with differing CUDA core counts, VRAM capacities, per-hour rental prices, and empirical rendering speeds. A simple rule such as "assign the fastest GPU" maximises throughput but ignores cost; "assign the cheapest GPU" minimises cost but may increase waiting time and deadline violations. The optimal assignment depends on the job's complexity, deadline urgency, and the current fleet state.

2. **Stochastic Node Availability.** In a public cloud GPU market, a studio cannot guarantee that any specific node will remain available for the duration of a job. Customer demand fluctuates diurnally, maintenance windows are scheduled by the platform, and hardware failures occur with non-zero probability. The simulator models this through a stochastic, stock-status-driven availability model, and the scheduler observes only a coarsened state abstraction (idle / running, plus per-type stock status), reflecting what is actually visible to a cloud consumer through a platform API.

3. **Bi-Objective Optimisation.** Rendering studios care about both *turnaround time* (minimising the time a job waits before execution begins) and *cost* (minimising the total GPU rental fees). These objectives are in partial tension: the fastest GPU delivers the best turnaround at the highest per-hour cost, whereas the cheapest GPU types

reduce cost while increasing execution time (fleet pricing in Table 4.1). A scheduler must navigate this trade-off explicitly.

Figure 1.1 sketches the end-to-end problem: rendering jobs arrive stochastically and queue for capacity; at each event a scheduler decides which job runs on which GPU; the heterogeneous fleet executes the work; and each job's outcome is recorded as its waiting time, deadline adherence, and rental cost. The three challenges above map directly onto this flow: heterogeneity lives in the fleet, stochastic availability sits between the scheduler and the fleet, and the bi-objective tension appears in the outcomes.

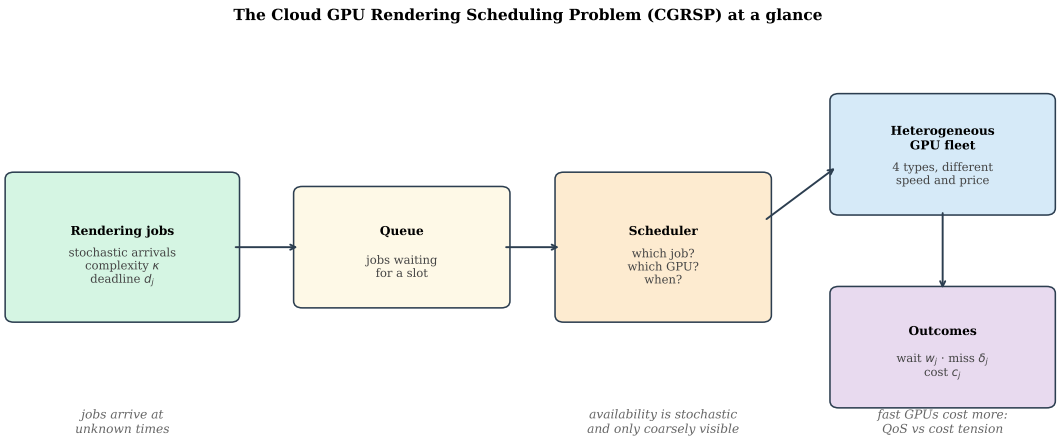

Figure 1.1: The Cloud GPU Rendering Scheduling Problem at a glance. Rendering jobs with a complexity level and a deadline arrive stochastically and wait in a queue; the scheduler decides which job to place on which GPU and when; the heterogeneous fleet (four GPU types with different speeds and prices, and stochastic availability) executes the jobs; and per-job outcomes (waiting time, deadline adherence, cost) are the quantities the scheduler is judged on.

The formulation is grounded in 6,627 real rendering jobs collected from RunPod Secure Cloud and is evaluated through a discrete-event simulator calibrated to March 2026 RunPod pricing.

1.2 Computational Complexity

The offline version of CGRSP (where all n jobs are known in advance) is a generalisation of the weighted job scheduling problem on parallel unrelated machines with

deadlines, which is NP-hard in the strong sense [8]. For the online version (jobs revealed at their arrival times), no online algorithm can achieve a bounded competitive ratio against the offline optimum in the worst case for deadline-constrained scheduling without preemption; even in the preemptive uniprocessor setting, optimal online scheduling requires full future knowledge unless the system is underloaded, a limitation established in the classical real-time scheduling literature building on [9]. This intractability motivates the use of practical heuristic policies rather than exact optimisation. The heuristic policies are benchmarked in Chapter 5 against a FIFO baseline rather than against an intractable optimum; reinforcement learning is a natural future extension (Section 6.2) but is not evaluated in this study.

1.3 Research Objective

This study pursues four research objectives:

1. **Formal Problem Formulation.** Define CGRSP as a rigorous mathematical optimisation problem with explicit notation, decision variables, constraints, and a bi-objective function, enabling reproducible comparison with future work.

2. **Simulator Construction and Calibration.** Build a discrete-event simulator that faithfully reproduces the RunPod Secure Cloud environment, including empirically measured GPU execution times, a stochastic availability model, Poisson job arrivals, and per-second cost tracking, then validate it against 6,627 real rendering jobs.

3. **Comparative Evaluation of Scheduling Heuristics.** Implement and statistically compare ten scheduling algorithms: FIFO (baseline), Shortest Processing Time (SPT), Earliest Deadline First (EDF), Lowest Cost First (LCF), Balanced (weighted speed-cost composite), Random, SPT+Rescue (SPT with laxity-based deadline rescue), an adaptive queue-pressure EDF-SPT hybrid, the novel anticipative Rolling-Horizon (RH) scheduler, and the Cost-Aware Deadline-Risk (CADR) scheduler. Assess them across a four-phase evaluation programme: single-seed benchmark, 30-

seed statistical validation, day-type and deadline-tightness sensitivity, and a 30-day monthly simulation calibrated to real RunPod billing data.

4. **Deployment Guidance.** Translate experimental findings into actionable, statistically validated recommendations for cloud rendering operators: establish when each of the two proposed schedulers should be deployed (RH when the count of missed deadlines is the priority, CADR when the severity of misses also matters), quantify when SPT's wait-time advantage outweighs its deadline risk, identify the tight-deadline fraction threshold at which EDF becomes preferable, and characterise the role of the concurrency limit $C_{\max}$ as the primary performance lever.

1.4 Research Contribution

The primary contributions of this study are:

- A complete mathematical formulation of CGRSP, including a three-state availability model, a bi-objective QoS-cost function, and all assignment and temporal constraints (Chapter 3).
- A validated discrete-event simulator implemented in Python, calibrated against the full empirical rendering-job dataset (Chapter 4).
- A statistically rigorous comparison of ten scheduling algorithms (eight benchmarks, including an adaptive EDF-SPT hybrid, plus the two proposed schedulers) on the saturated hectic-day configuration of Section 5.1.1 ($n = 30$ seeds, $\rho > 1$), using paired t-tests, Cohen's d effect sizes, and a Holm-Bonferroni correction. The comparison establishes SPT as the lowest-wait baseline (statistically tied with RH), but significantly outperformed by EDF on miss, and quantifies the deadline penalty of the deadline-oblivious policies (Chapter 5).
- A day-type sensitivity study (quiet/normal/hectic/surge, 30 seeds each) showing that the scheduler ranking is regime-dependent: RH achieves the lowest miss on hectic days (above saturation, $\rho > 1$), while the Adaptive hybrid, EDF, and RH are tied for the lowest miss on below-saturation surge days (Chapter 5).

- A deadline-tightness sensitivity study under the surge-day configuration of Section 5.1.1 (30 seeds per sweep point) locating the crossover between deadline-driven and shortest-job-first ordering: among the seven baseline policies in the sweep, EDF attains significantly lower miss than SPT when tight fraction $\leq 30\%$, the two are statistically indistinguishable at the midpoint of the sweep (50%), and SPT attains significantly lower miss from $\geq 70\%$; no scheduler achieves 0% misses across all fractions at surge-day load (Chapter 5).

- A 30-day monthly operational simulation (Phase 4, 10 replications) calibrated from RunPod billing data (March 2026), confirming that the hectic-day ranking persists at monthly scale, with RH and CADR achieving near-identical lowest monthly miss, and identifying the RunPod serverless concurrency limit $C_{\max}$ as the binding throughput constraint (Chapter 5).

- Two complementary deadline-aware schedulers: (i) the Rolling-Horizon (RH) scheduler, which at each event plans over the slot-availability timeline and anticipated arrivals, keeping shortest-processing-time ordering for jobs with slack, promoting only jobs the timeline shows would otherwise miss, and demoting already-doomed jobs so they do not displace savable ones; RH achieves the lowest deadline miss rate of the ten policies with wait statistically tied with SPT, significantly outperforming SPT+Rescue on miss; and (ii) the Cost-Aware Deadline-Risk (CADR) scheduler, which classifies jobs by a scale-free critical ratio and selects the cheapest deadline-feasible node; CADR matches RH's miss rate at substantially lower tardiness and also achieves substantially lower miss than SPT+Rescue, at essentially the same wait. The study establishes deployment criteria distinguishing the two schedulers on miss-count versus miss-severity priority (Chapter 5; see also Research Objective 4 above).

1.5 Scope and Limitations

This study focuses on the problem of scheduling 3D rendering workloads on cloud GPU platforms, specifically evaluated on the RunPod Secure Cloud platform using pricing data from March 2026. The evaluation is based on the empirical dataset of real rendering

jobs collected between July 2025 and June 2026, spanning four GPU types (RTX 3090, RTX A5000, RTX A4000, RTX A4500) and three scene complexity levels.

Several limitations bound the scope of this work. The concurrency limit is fixed at the default RunPod serverless account limit ($C_{\max}$, Section 5.1.1), which serves as the primary throughput constraint throughout all experiments. The availability model is calibrated to qualitative empirical patterns rather than exact platform measurements, which are not publicly available; a sensitivity sweep over the assumed baseline availability probabilities (Chapter 5) shows the scheduler comparison is stable across the operable range of that sweep rather than at every point of it, since the scarcest settings leave the fleet inoperable for every policy alike and the most eased setting lets several policies reach zero misses (Chapter 6). The study also assumes non-preemptive scheduling, as CGRSP models each rendering frame as a single indivisible job; tile-based rendering across multiple GPUs per frame is not considered. These limitations are discussed in detail, together with directions for future research, in Chapter 6.

1.6 Organisation of Thesis

The remainder of this thesis is organised as follows:

Chapter 2 reviews related work on cloud GPU scheduling, classical scheduling heuristics, and discrete-event simulation methodology, motivating the design choices made in this study.

Chapter 3 presents the formal mathematical formulation of CGRSP: notation tables, an informal problem description of the system model and state representation, a unified mathematical model (the bi-objective lexicographic objective, assignment constraints, the stochastic availability model, and temporal/QoS constraints), and an illustrative numerical example.

Chapter 4 describes the system architecture: the discrete-event simulator core, the RunPod fleet model, the job generator, and the implementations of all ten scheduling algorithms.

Chapter 5 presents the full experimental evaluation: the seven-scheduler Phase 2 hectic-day benchmark (the seven non-hybrid baseline policies; RH, CADR, and the Adaptive hybrid are evaluated from Phase 3 onward), Phase 3 statistical analysis over 30 seeds with ablation tracing the design chain (FIFO $\rightarrow$ SPT $\rightarrow$ SPT+Rescue $\rightarrow$ RH, plus CADR and the CADR-order-only ablation), a critical-ratio threshold sensitivity analysis for CADR, an anticipation reservation sensitivity analysis, a concurrency limit sensitivity analysis ($C_{\max} \in \{3, 5, 7, 10\}$), a rescue threshold sensitivity analysis, day-type and deadline-tightness sensitivity, a tardiness-objective robustness study, and Phase 4 monthly operational simulation.

Chapter 6 concludes the study with a discussion of theoretical implications, practical deployment guidance, limitations, and directions for future research.

Chapter 2 Background and Related Work

2.1 Cloud GPU Computing

The commercialisation of GPU computing as a cloud service began with NVIDIA's CUDA platform [10] enabling general-purpose parallel computation on graphics hardware. Early cloud providers (AWS, Azure, GCP) offered GPU instances as part of virtual machine (VM) rental services, billed hourly with a minimum session length of a full hour on AWS [11]. The rise of deep learning as a primary GPU workload in the 2010s drove rapid expansion of GPU instance types and the emergence of spot-pricing markets where unused capacity is offered at discounted rates with potential preemption [12–14].

By the early 2020s, a second tier of GPU cloud providers, such as RunPod, Lambda Labs, CoreWeave, and Vast.ai, emerged with billing granularity as fine as one second, without minimum session lengths. These platforms aggregate GPU hardware from multiple data-centre operators and offer a heterogeneous catalogue of GPU types (consumer-grade RTX series, workstation-grade A-series, data-centre A100/H100) at widely varying price points. For rendering workloads, which tend to be short (tens of seconds to a few minutes per frame) and embarrassingly parallel at the frame level, per-second billing and the ability to spin up many low-cost nodes simultaneously make these platforms attractive alternatives to dedicated render farms.

The key distinguishing feature of on-demand cloud GPU rental relevant to this thesis is *qualitative availability*. Unlike reserved VM instances with guaranteed exclusive access and pre-provisioned capacity, an on-demand instance must be acquired afresh from a pool whose stock fluctuates with competing demand: when stock is scarce, acquiring an instance of a given type incurs a provisioning delay. From the perspective of the scheduling algorithm, this manifests as a coarsened observable state: the platform API exposes a qualitative per-type stock status, but not the underlying cause (customer demand, mainte-

nance, hardware fault). This is the abstraction encoded in the CGRSP stochastic availability model, which targets the RunPod Secure Cloud on-demand model (where a running instance is not reclaimed) rather than spot preemption.

A critical distinction separating marketplace GPU rental from the managed-cluster context of most scheduling literature is rarely made explicit. **Reserved-cluster schedulers**, including HEFT [6], Gavel [15], and Optimus [16], operate on GPU pools under the operator's exclusive control; node availability is guaranteed and the scheduling problem reduces to job-to-node assignment. **Marketplace scheduling**, by contrast, must treat GPU availability as externally determined: competing customer demand, platform maintenance windows, and hardware failures collectively determine which GPU types can be provisioned at any given moment, outside the operator's control. This distinction is structural, not incidental: the scheduler must treat availability as a stochastic first-class input rather than a deterministic constraint. This is precisely the gap the CGRSP formulation closes. A third body of work, **spot and transient-instance scheduling**, does treat cloud availability as stochastic, but models it as mid-job *revocation* risk: checkpointing against instance loss [12], batch services that absorb revocations [13], and bidding strategies over spot price processes [14]. The on-demand Secure Cloud setting of this thesis is structurally different: a held instance is never reclaimed, so no checkpoint or migration machinery is relevant, and scarcity manifests instead as a stochastic provisioning delay at acquisition time, observed only as a coarse per-type stock status inside every dispatch decision. That acquisition-side availability problem is the one the surveyed spot literature does not address.

2.2 Classical Job Scheduling

Job scheduling on a set of machines to minimise one or more performance criteria is one of the most well-studied topics in operations research and computer science [8]. Graham et al. [17] established the three-field notation $\alpha|\beta|\gamma$ for scheduling problems. For CGRSP, the problem is closest to $P_m|\text{online}, r_j, d_j|\sum \omega_j C_j$ (parallel identical machines with online arrivals, release times, deadlines, and weighted completion time; the weight

is written ω_j here to avoid a clash with the waiting time w_j), except that the machines in CGRSP are not identical: they differ in speed and cost, and their speed ranking depends on the job, making it a generalisation to unrelated parallel machines (R_m) rather than uniform parallel machines (Q_m). A uniform-machine model would require one speed factor per machine that applies to every job, and the fleet violates this, because the ranking inverts across complexity levels: at $\kappa = 2$ the RTX A4000 is the fastest of the four types (68.7 s) and the RTX A4500 the slowest, whereas at $\kappa = 3$ that pair swaps ends and the A4500 becomes the fastest (88.7 s) with the A4000 at 98.2 s (Table 4.1).

2.2.1 Shortest Processing Time (SPT)

The Shortest Processing Time rule assigns the available processor to the ready job with the smallest processing time. Smith [18] proved that SPT minimises average weighted completion time on a single machine. On parallel machines, SPT applied independently on each machine remains optimal for minimising average flow time in offline settings [8]. In online settings with Poisson arrivals, SPT minimises mean response time (a classical result from M/G/1 queueing theory [8]). SPT's weakness in deadline-constrained settings is its tendency to starve large, long-running jobs: a steady stream of short jobs keeps a machine busy and can delay deadline-critical long jobs past their deadlines.

2.2.2 Earliest Deadline First (EDF)

Earliest Deadline First, attributed to Jackson [19] and later formalised by Lawler and Moore [20], prioritises the ready job with the nearest deadline. EDF is optimal on a single preemptive machine in the feasibility sense: if any policy can meet every deadline, EDF does [9]. On non-preemptive parallel machines, the setting of CGRSP, EDF does not provide a worst-case guarantee for deadline violation minimisation. The Phase 3 deadline sensitivity experiments in Chapter 5 instead separate its comparison against shortest-job-first ordering (SPT) into three regimes: EDF attains significantly lower miss when the tight-deadline fraction is low ($\leq$ 30%; at 30%, $t = 10.705$, $p < 10^{-10}$), the two are statistically indistinguishable at the midpoint of the sweep (50%, $t = 0.397$, $p = 0.69$), and

SPT attains significantly lower miss at high tight fractions ($\geq 70\%$; at 70%, $t = -11.549$, $p < 10^{-11}$).

2.2.3 First-In-First-Out (FIFO)

FIFO processes jobs in the order they arrive. It provides fairness guarantees (no starvation), requires no knowledge of processing time or deadline, and has long served as the default scheduling policy in batch computing systems, dating to the earliest resident-monitor mainframe operating systems [21]. FIFO is used as the primary baseline in this study because it represents the "do nothing special" approach. In the M/M/c queueing model, FIFO achieves the same throughput as any work-conserving policy (by work-conservation arguments [8]), but it does not minimise mean response time or deadline adherence.

2.2.4 Other Heuristics

Lowest Cost First (LCF) prioritises the GPU type with the lowest effective rental cost per job, defined as the product of the rental rate and the expected execution time: $\text{cost}_{\text{eff}}(j, m) = p_{g(m)} \cdot \mathbb{E}[W(\kappa_j, g(m))]$. LCF is motivated by cost-minimisation objectives, relevant for rendering studios operating under tight production budgets where deadline pressure is low. A complicating factor is GPU availability reliability: a "cheap" GPU type with consistently low stock may increase effective turnaround through long provisioning delays. This is addressed in the CGRSP implementation through a proportional stock-status penalty added to the effective cost.

Balanced scheduling combines processing speed (SPT-like) and cost (LCF-like) criteria through a weighted composite score. A weight $w_b \in [0, 1]$ controls the trade-off: $w_b \to 1$ approaches SPT behaviour (speed priority), $w_b \to 0$ approaches LCF (cost priority). In the CGRSP implementation, $w_b = 0.8$ (speed-biased) is used, with global-max normalisation to prevent one criterion from dominating due to differing numerical scales. (The weight is written w_b to avoid a clash with the objective weights α_1, α_2 of Chapter 3.) Unlike EDF and SPT, Balanced does not reorder jobs; it sorts only on node selection,

maintaining FIFO job ordering. This design choice ensures that Balanced is statistically distinguishable from both EDF (which reorders by deadline) and SPT (which reorders by processing time) in the comparative evaluation.

2.3 Heterogeneous Machine Scheduling

When machines have different speeds, the scheduling problem becomes significantly harder. Braun et al. [5] provided the first systematic comparison of scheduling heuristics on heterogeneous machines, evaluating 11 algorithms including min-min, max-min, genetic algorithms, and simulated annealing in terms of makespan. Their study on synthetic workloads found that min-min (which assigns each task to the machine that finishes it the earliest) outperforms more complex methods for moderate-heterogeneity settings. CGRSP's SPT policy is analogous to min-min when execution times are known.

The Heterogeneous Earliest Finish Time (HEFT) algorithm of Topcuoglu et al. [6] is the most widely cited heuristic for scheduling task-dependency DAGs on heterogeneous parallel machines. HEFT ranks tasks by upward rank (estimated time to completion including successors) and assigns each task to the machine that minimises its finish time. For CGRSP, jobs are independent (no DAG dependencies), so HEFT reduces to SPT-like behaviour with heterogeneous execution time estimation, making it conceptually close to the SPT implementation described in Chapter 4.

More recent work addresses GPU heterogeneity specifically. Ye et al. [7] surveyed deep learning workload scheduling in GPU data centres, cataloguing approaches ranging from static profiling-based assignment to online reinforcement learning-based policies. Peng et al. [16] proposed Optimus, a performance model-driven scheduler for deep learning jobs that reduces training time by dynamically adjusting resource allocation. These approaches differ from CGRSP in that they focus on *training* jobs (which can be checkpointed and migrated) rather than batch rendering jobs (which are typically non-preemptive and short).

Narayanan et al. [15] proposed Gavel, a heterogeneity-aware cluster scheduler for deep learning workloads that formulates GPU-type assignment as a multi-objective optimisation over time-sharing policies (round-robin, fairness, and priority scheduling). Gavel explicitly models the performance differential across heterogeneous GPU types and demonstrates that heterogeneity-aware scheduling substantially reduces average job completion time compared to homogeneous-assumption baselines, a result conceptually aligned with CGRSP's finding that SPT's wait-time advantage depends on exploiting per-complexity GPU performance differences. Gavel, however, targets reserved GPU clusters with guaranteed node availability, is evaluated on long-running training jobs (hours, checkpointable), and does not enforce per-job deadlines. CGRSP differs on all three dimensions: stochastic on-demand availability with provisioning delay, short non-preemptive rendering frames, and mixed tight/loose deadline constraints.

2.4 Reinforcement Learning for Resource Management

The application of reinforcement learning (RL) to scheduling and resource management has received substantial attention since the seminal work of Mao et al. [22], who trained a neural network policy using REINFORCE to schedule jobs on parallel machines, achieving lower average completion time than heuristic baselines in the regime where job sizes are unknown at arrival. Their formulation (a discrete action space with one action per job in the queue, and a state derived from the job-size histogram and resource occupancy) is conceptually related to meta-scheduler approaches for cloud GPU scheduling.

Subsequent work applied deep Q-networks (DQN) [23] to various scheduling domains. Chen and Chen [24] applied DQN specifically to GPU job scheduling, training a policy to select job-GPU pairings based on job type, GPU utilisation, and historical completion time. Their results show DQN outperforming FIFO and SPT in average job completion time, but the approach requires tens of thousands of training episodes on simulated workloads, a substantially larger training budget than the heuristic-only approach pursued in this thesis.

Zhou et al. [25] surveyed deep RL methods for cloud resource scheduling, identifying three main challenges: (1) large state-action spaces requiring function approximation, (2) sparse and delayed rewards, and (3) non-stationarity from competing workloads. These challenges motivate the heuristic-based approach in CGRSP, where domain knowledge is encoded directly into the scheduling rules rather than learned from scratch.

2.5 Discrete-Event Simulation

Discrete-event simulation (DES) models the evolution of a system through a chronologically ordered sequence of events, each of which can modify system state and schedule future events [26]. DES is the standard methodology for evaluating scheduling algorithms when analytical queueing models are intractable, as is the case for heterogeneous machines with general service time distributions.

Sargent [27] established the validation and verification (V&V) framework for simulation models that this thesis follows: (1) conceptual model validity (does the model represent the real system?), (2) operational validity (do simulation outputs match real system outputs?), and (3) data validity (are model inputs accurate?). The CGRSP simulator is validated against the 6,627-job empirical dataset at the operational validity level, with log-normal service times whose per-stratum means and log-space dispersions are fitted to the measured execution times.

Law and Kelton [28] identified statistical output analysis, including confidence intervals, variance reduction techniques, and sensitivity analysis, as essential for credible simulation results. This thesis follows these recommendations: all reported metrics are averages over multiple independent replications (seeds), and 95% confidence intervals are reported for Phase 3 statistical results.

2.6 3D Rendering Workloads

Three-dimensional (3D) rendering is the process of computing the colour of each pixel in an image from a geometric scene description, camera model, and light sources. Path tracing [29], the algorithm used by most modern renderers (Cycles in Blender, V-Ray, Arnold), is embarrassingly parallel at the pixel level: each pixel's colour is estimated independently by tracing thousands of random light paths. This means that rendering a scene is naturally decomposed into per-frame jobs, and each frame can be rendered on a single GPU without inter-node communication.

Rendering job duration depends primarily on scene complexity (polygon count, light count, material complexity), image resolution, and the number of samples per pixel [29]. In this thesis, job complexity is abstracted to three levels ($\kappa \in \{1, 2, 3\}$), with per-stratum log-normal execution times whose means are derived from the 6,627-job dataset. The dataset spans a wide range of complexities: high-complexity jobs take roughly 40% to 70% longer than low-complexity jobs, depending on GPU type (per-stratum means in Table 4.1). This heterogeneity in job duration is precisely what makes SPT effective: by front-loading short jobs, it reduces the average number of jobs waiting in the queue.

The specific GPU types included in this study (RTX 3090, A5000, A4000, A4500) are representative of the consumer and prosumer GPU tiers available on marketplace GPU platforms as of early 2026. They cover a factor of $\sim$1.8 in rental price and a factor of $\sim$1.3 in average rendering speed (Table 4.1), providing meaningful heterogeneity for the scheduling decision. Figure 2.1 visualises this heterogeneity: the speed ranking of the four GPU types is non-monotone across complexity levels, and price does not track speed, so neither dimension alone determines the best assignment.

2.7 Summary

Table 2.1 summarises the positioning of CGRSP relative to closely related work. Among the studies surveyed, CGRSP is unique in combining all four distinguishing fea-

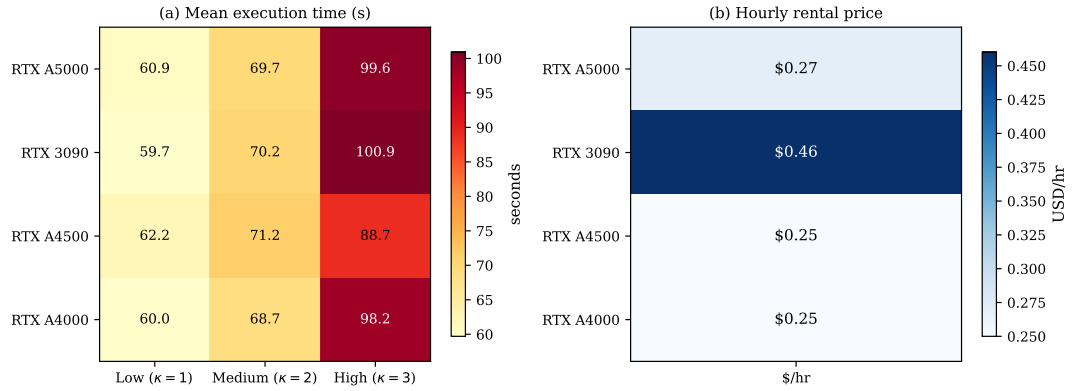

Figure 2.1: GPU fleet heterogeneity. Panel (a): mean execution time (seconds) per GPU type and scene complexity level κ. The RTX A4500 is fastest at high complexity (88.7 s) despite having fewer CUDA cores than the RTX 3090 (100.9 s), illustrating that the GPU speed ranking is non-monotone with respect to hardware specifications. Panel (b): hourly rental price. The RTX 3090 is the most expensive ($0.46/hr) yet is the slowest at high complexity and only second-fastest on average; the A4500 and A4000 share the same price ($0.25/hr) while differing in speed, making node selection a genuine multi-criteria optimisation.

tures: (1) real-world calibrated GPU heterogeneity, (2) stochastic qualitative availability (marketplace-driven, observed only as coarse stock status), (3) per-second billing with deadline constraints, and (4) rigorous statistical evaluation across multiple seeds, day types, and concurrency limits.

Table 2.1: Positioning of CGRSP against related scheduling studies

Work	**GPU Hetero.**	**Stoch. Avail.**	**Deadlines**	**Stat. Valid.**
Braun et al. [5]	Synthetic	×	×	×
HEFT [6]	Yes	×	×	×
Mao et al. [22]	×	×	×	✓
Chen & Chen [24]	Yes	×	×	Limited
Ye et al. [7]	Yes (survey)	×	Partial	×
Narayanan et al. [15] (Gavel)	Yes	×	×	✓
CGRSP (this thesis)	✓	✓	✓	✓

Beyond the four dimensions in Table 2.1, CGRSP is, to the best of the author's knowledge, the first scheduling study of marketplace cloud GPU rental applied specifically to *batch 3D rendering workloads*. Compared to deep learning training, the dominant focus of GPU scheduling literature [15, 22, 24], rendering frames have three properties that

shape the scheduling design. First, execution is *short and non-preemptive*: individual frames complete in tens of seconds to a few minutes, making checkpoint-and-resume impractical and favouring simple online dispatch policies over the long-horizon optimisation that DL training schedulers can exploit. Second, *complexity is observable at submission*: scene complexity level (κ) is treated as known when a job enters the queue (in deployment it would be classified from scene statistics such as polygon and light counts that are available before rendering; in the calibration dataset it is proxied by render-time terciles, Section 5.1.1), providing the execution-time estimate that SPT and SPT+Rescue rely on. Third, *GPU performance is non-monotone across complexity levels*: for high-complexity frames ($\kappa = 3$), the RTX A4000 (6,144 CUDA cores, 98.2 s) renders faster than the RTX 3090 (10,496 cores, 100.9 s; see Table 4.1), a reversal that requires per-complexity node selection rather than a fixed GPU speed ranking. These three rendering properties shape the CGRSP formulation and calibration methodology.

Chapter 3 Problem Formulation and Mathematical Model

This chapter presents the mathematical formulation of the Cloud GPU Rendering Scheduling Problem (CGRSP). The problem models an on-demand GPU cloud where rendering jobs arrive continuously and must be scheduled to balance quality of service (QoS) and per-second operational costs. The formulation addresses three key challenges: (1) heterogeneous GPU types with varying performance and per-second rental rates, (2) stochastic provisioning delays driven by fluctuating market scarcity for each GPU type, and (3) temporal deadline constraints ensuring timely job completion.

The system operates as an on-demand GPU pool with $|\mathcal{N}|$ nodes across $|\mathcal{G}|$ resource types, serving a stream of rendering jobs with heterogeneous complexity levels. Each job is assigned to at most one node; a GPU instance of the node's type is provisioned for the duration of the job and billed per second of execution. The scheduler pursues a bi-objective criterion: minimising QoS penalties (waiting time and deadline violations) as the primary criterion while treating per-second operational cost as a secondary, tiebreaking criterion, subject to node availability and assignment constraints.

Occupancy and billing are deliberately modelled as two independent axes. On the occupancy axis, a node behaves as a persistent, single-tenant slot: once acquired it is held exclusively by one job for that job's full duration and is not shared, pre-empted, or reclaimed mid-execution. On the billing axis, the provider charges strictly for realised execution, not for the act of acquiring a slot: the stochastic provisioning delay τ_j^{prov} incurred while a slot is being acquired (Eq. 3.8) is unbilled, and per-second charges accrue only once execution actually begins (Eq. 3.15). This combination, a held, exclusive slot for the occupancy model paired with pay-only-for-execution billing, is what the fleet and cost model in this chapter formalise.

3.1 Mathematical Notation

Tables 3.1, 3.2 and 3.3 define all symbols (sets and indices, system parameters, and decision and state variables, respectively) used throughout the formulation.

Table 3.1: Sets and Indices

Symbol	Description
$\mathcal{J}$	Set of all rendering jobs
$\mathcal{N}$	Set of nodes
$\mathcal{G}$	Set of resource types
$\mathcal{K}$	Set of discrete complexity levels
$\mathcal{J}_t$	Set of jobs waiting in the queue at time t
$j \in \mathcal{J}$	Job index
$m \in \mathcal{N}$	Node index
$g \in \mathcal{G}$	Resource type index
$\kappa \in \mathcal{K}$	Complexity level index
$t \in \mathbb{R}^+$	Continuous time variable

3.2 Problem Description

Figure 3.1 illustrates the decision structure at a single scheduling event: the scheduler observes the current job queue (each job carrying a complexity level and deadline class) and the fleet state (each slot's GPU type, stock status, and occupancy), then assigns jobs to idle slots under the lexicographic objective. This snapshot is the atomic unit repeated throughout the discrete-event simulation; the provisioning delay τ_j^{prov} incurred between dispatch and execution start (formalised in Section 3.3) is shown explicitly on the dispatch step.

The system operates as a discrete-event simulator with continuous time representation. At each scheduling event, the scheduler observes the current state and makes as-

Table 3.2: System Parameters

Symbol	Description
n_g	Number of nodes of resource type g
$s_g(t)$	Stock status of resource type g at time t (High / Medium / Low)
p_g	Per-second rental price of resource type g
λ	Mean job arrival rate (jobs per second)
ρ	Offered load, $\rho = \lambda \overline{W}/C_{\text{max}}$, where $\overline{W}$ is the fleet-mean expected execution time
κ_j	Complexity level of job j
d_j	Deadline for job j
t_j^{submit}	Submission time of job j
$W(j,g)$	Realised execution time of job j on resource type g; it depends on j only through the complexity level κ_j, so the class-level form $W(\kappa_j, g)$ is used where convenient
$\overline{W}(\kappa,g)$	Mean (expected) execution time of complexity class κ on resource type g
β_g	Baseline high-stock probability of resource type g
$\phi(t)$	Time-of-day stock multiplier at time t
τ_j^{prov}	Provisioning delay incurred when job j acquires an instance (stock-status dependent)
$g(m)$	Resource type of node m
C_{max}	Concurrency limit: number of on-demand slots ($\lvert\mathcal{N}\rvert = C_{\text{max}}$)
L_s, U_s	Lower and upper provisioning-delay bounds for stock status s
$p_g^H(t), p_g^M(t)$	High- and Medium-stock probabilities of resource type g at time t
$T_{\text{tight}}, T_{\text{loose}}$	Per-job deadline window: tight class (rush orders) or loose class (standard renders); platform values and class proportions in Section 5.1.1
α_1, α_2	QoS penalty weights inside C_{QoS}: penalty per second of waiting (α_1) and penalty per missed deadline (α_2)
$w_{\text{qos}}, w_{\text{cost}}$	Scalarisation weights balancing QoS and cost, used in the weighted-sum robustness variant (Section 5.5.9) and in the Rolling-Horizon placement surrogate (Chapter 4)

Table 3.3: Decision Variables and State Variables

Symbol	Description
$x_j^m \in \{0,1\}$	Assignment: 1 if job j executes on node m (realised placement over the horizon), 0 otherwise
$a_m^t \in \{0,1\}$	Idle indicator for slot m at time t (1 if idle and able to accept a job, 0 if currently running)
t_j^{dispatch}	Dispatch (slot-acquisition) time of job j: state variable determined by the scheduling policy
t_j^{start}	Execution start time of job j (Eq. 3.9)
t_j^{complete}	Completion time of job j
w_j	Waiting time of job j
r_j	Response time of job j
$\delta_j \in \{0,1\}$	Deadline adherence indicator: 1 if job j completes on time ($t_j^{\text{complete}} \leq d_j$), 0 otherwise
$\tau_j = \max(0, t_j^{\text{complete}} - d_j)$	Tardiness of job j: auxiliary quantity (time past deadline, 0 if on time); defined for completeness, not part of the objective
c_j	Execution cost of job j (Eq. 3.15)

signment decisions. The state space includes both observable information and internal provisioning dynamics.

Observable State (Scheduler View): The scheduler receives a simplified three-state representation for each node:

- **Unavailable**: Slot cannot be rented. This state is retained for generality (it arises on a spot market); under the on-demand Secure Cloud model studied here a held slot is never reclaimed, so this state does not occur in the experiments.
- **Idle**: Slot is free and can be assigned to jobs
- **Running**: Slot is currently executing an assigned job

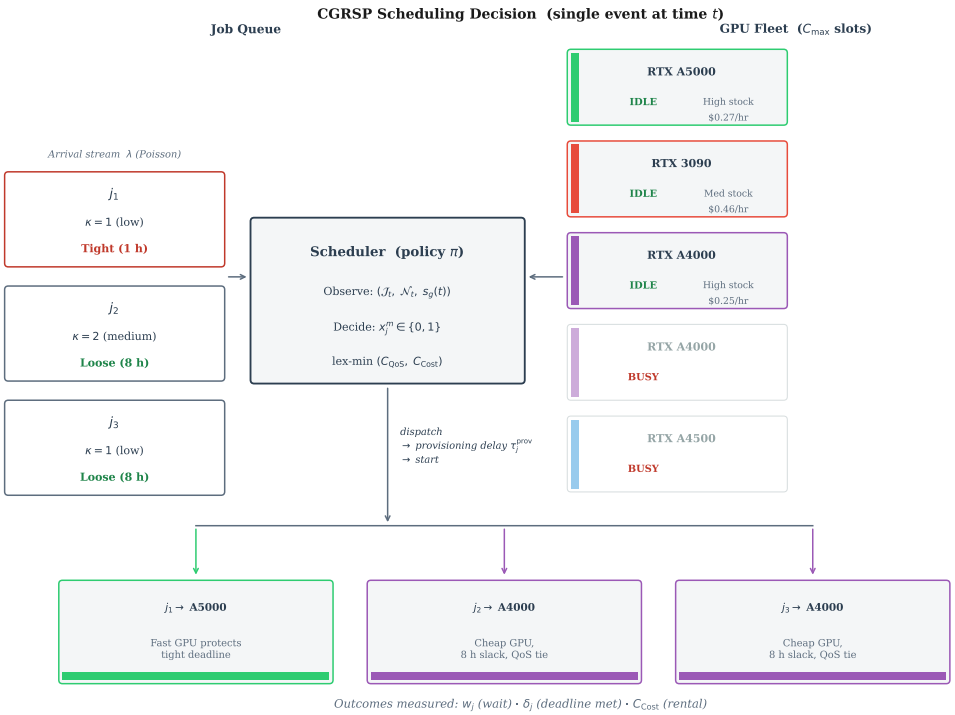

Figure 3.1: CGRSP scheduling decision at a single event. The scheduler observes the job queue (jobs j_1–j_3 with complexity κ and deadline class) and the GPU fleet ($C_{\max}$ slots with type, stock status, and occupancy). It assigns jobs to idle slots under the lexicographic objective: QoS first (minimise wait and deadline misses), then cost as a tiebreaker. Here j_1 carries a tight deadline and is routed to a fast, high-stock GPU; j_2 and j_3 have ample slack, so cost decides and both go to the cheapest idle slot. Busy slots (greyed) are unavailable. Dispatch incurs the provisioning delay τ_j^{prov} before a job starts running. The outcomes recorded per job are waiting time w_j, deadline adherence δ_j, and rental cost.

Additional observations: current job queue $\mathcal{J}_t$ with job characteristics $(\kappa_j, d_j, t_j^{\text{submit}})$, current simulation time t, and historical performance metrics.

In addition to per-node availability, the scheduler observes a per-type **stock status** $s_g(t) \in \{\text{High}, \text{Medium}, \text{Low}\}$ summarising the provisioning reliability of resource type g: the likelihood that a fresh instance of type g can be acquired without a long provisioning delay. Stock status is distinct from the idle indicator a_m^t: a_m^t records whether a held slot is currently free, whereas $s_g(t)$ reflects market-level scarcity of type g. Stock status governs the provisioning delay incurred when a fresh instance of type g is acquired, formalised in Section 3.3, and additionally biases node selection (Chapter 4) away from scarce or unreliable types.

The fleet model enforces at most one job per node at any time: each provisioned instance is exclusively assigned to a single job until completion, and once started a job runs to completion without interruption, since checkpoint-restart mechanisms are expensive or unavailable for rendering workloads. While GPU virtualisation could enable multi-tenancy, the one-job-per-node restriction reflects common rendering workload requirements, where jobs use full GPU capacity for optimal performance.

The on-demand Secure Cloud pool studied here makes every slot rentable at any time: the idle indicator $a_m^t \in \{0, 1\}$ records only whether slot m is currently free (1) or running a job (0), and a running slot is never reclaimed by the provider. Market scarcity is therefore not realised as denial of service but as the provisioning delay τ_j^{prov} formalised in Section 3.3 (Eq. 3.8); the calibration of this delay against the RunPod snapshot record is given in Chapter 5.

Human Decision Elements: The formulation reflects human operational decisions through several design parameters:

- **Deadline policy**: Per-job deadline window from submission time ($d_j = t_j^{\text{submit}} + T_j$, where $T_j \in \{T_{\text{tight}}, T_{\text{loose}}\}$; values and class proportions are given in Section 5.1.1)
- **Fleet configuration** (n_g for each $g \in \mathcal{G}$): Capacity planning decision
- **Objective weights** ($w_{\text{qos}}, w_{\text{cost}}$): Business priority between QoS and cost efficiency

3.3 Mathematical Model

The scheduling objective is bi-objective, with quality of service treated as the *primary* criterion and operational cost as a *secondary*, tiebreaking criterion. QoS is optimised first; operational cost is optimised only among assignments that are equivalent on QoS. This lexicographic priority reflects the operational reality of deadline-driven rendering: a missed client deadline carries a discrete contractual cost that dwarfs the per-second rental difference between GPU types (the fleet's hourly rates span a narrow band, Table 4.1), so the fast-GPU premium is worth paying whenever it protects a deadline, while the cheap-

GPU saving is banked only when QoS is unaffected. Formally, the scheduler solves

$$\text{lex min } \big(C_{\text{QoS}},\ C_{\text{Cost}}\big) \tag{3.1}$$

$$C_{\text{QoS}} = \sum_{j \in \mathcal{J}} \big[\, \alpha_1\, w_j + \alpha_2\, (1 - \delta_j) \,\big] \tag{3.2}$$

$$C_{\text{Cost}} = \sum_{j \in \mathcal{J}} c_j \tag{3.3}$$

where lexicographic minimisation prefers any policy with lower C_{QoS} and breaks QoS ties on lower C_{Cost}. The QoS cost (Eq. 3.2) combines a waiting-time penalty and a deadline-violation penalty: w_j is the waiting time in seconds (Eq. 3.11), α_1 is the penalty per second of wait, and α_2 is a penalty per missed deadline applied through the binary miss indicator $(1 - \delta_j)$. The concrete values of α_1 and α_2 used in experiments are given in Section 5.1.1; their ratio α_2/α_1 is swept in Section 5.5.9 to verify robustness. The cost criterion C_{Cost} is the total rental spend across all provisioned instances, summing the per-job execution cost c_j, which is defined formally in Eq. 3.15 below, once the assignment variable x_j^m has been introduced through the constraints.

Subject to:

$$\sum_{m \in \mathcal{N}} x_j^m \le 1, \quad \forall j \in \mathcal{J} \tag{3.4}$$

$$\sum_{j \in \mathcal{J}} x_j^m\, \mathbf{1}\Big[t_j^{\text{start}} \le t < t_j^{\text{complete}}\Big] \le 1, \quad \forall m \in \mathcal{N},\ \forall t \tag{3.5}$$

$$x_j^m \le a_m^{t_j^{\text{dispatch}}}, \quad \forall j \in \mathcal{J},\ m \in \mathcal{N} \tag{3.6}$$

$$s_g(t) = \begin{cases} \text{High} & \text{if } u < p_g^H(t), \\ \text{Medium} & \text{if } p_g^H(t) \le u < p_g^H(t) + p_g^M(t), \\ \text{Low} & \text{otherwise,} \end{cases} \qquad \forall g \in \mathcal{G},\ u \sim \text{Uniform}(0,1) \tag{3.7}$$

$$\tau_j^{\text{prov}} \sim \text{Uniform}\Big(L_{s_{g_j}(t_j^{\text{dispatch}})},\ U_{s_{g_j}(t_j^{\text{dispatch}})}\Big), \quad \forall j \in \mathcal{J},\ g_j := g(m),\ x_j^m = 1 \tag{3.8}$$

$$t_j^{\text{start}} = t_j^{\text{dispatch}} + \tau_j^{\text{prov}}, \quad \forall j \in \mathcal{J} \tag{3.9}$$

$$d_j = t_j^{\text{submit}} + T_j, \quad \forall j \in \mathcal{J},\ T_j \in \{T_{\text{tight}}, T_{\text{loose}}\} \tag{3.10}$$

$$w_j = t_j^{\text{start}} - t_j^{\text{submit}}, \quad \forall j \in \mathcal{J} \tag{3.11}$$

$$r_j = t_j^{\text{complete}} - t_j^{\text{submit}}, \quad \forall j \in \mathcal{J} \tag{3.12}$$

$$\tau_j = \max(0,\, t_j^{\text{complete}} - d_j), \quad \forall j \in \mathcal{J} \tag{3.13}$$

$$\delta_j = \mathbf{1}[t_j^{\text{complete}} \le d_j], \quad \forall j \in \mathcal{J} \tag{3.14}$$

$$c_j = \sum_{m \in \mathcal{N}} x_j^m \, W(j, g(m))\, p_{g(m)}, \quad \forall j \in \mathcal{J} \tag{3.15}$$

$$x_j^m,\ a_m^t,\ \delta_j \in \{0,1\}, \quad w_j,\ r_j,\ \tau_j,\ c_j \ge 0, \quad \forall j \in \mathcal{J},\ m \in \mathcal{N},\ t \tag{3.16}$$

Constraints (3.4)–(3.6) ensure feasible assignments respecting the concurrency limit and slot occupancy. The assignment variable x_j^m records the realised placement over the whole horizon, so Constraint (3.5) is stated per time instant using the indicator function $\mathbf{1}[\cdot]$, a standard piece of notation that evaluates to 1 when the bracketed condition is true and 0 otherwise. Concretely, $\mathbf{1}\left[t_j^{\text{start}} \le t < t_j^{\text{complete}}\right]$ equals 1 exactly while job j is actively running on node m, i.e. from its start time up to (but not including) its completion time, and 0 at every other instant. Multiplying by x_j^m and summing over $j \in \mathcal{J}$ therefore counts how many jobs are simultaneously running on node m at instant t; bounding that count by 1 states, in plain terms, that node m can be actively executing at most one job at any given time. The half-open execution interval $[t_j^{\text{start}}, t_j^{\text{complete}})$ inside the indicator lets a slot that completes a job at t start its successor at the same instant without double counting. Constraint (3.6) requires the chosen slot to be idle at the job's dispatch instant ($a_m^{t_j^{\text{dispatch}}} = 1$). Constraint (3.4) is deliberately an inequality rather than a covering constraint ($\sum_m x_j^m = 1$) because assignment is decided per scheduling event: a job left unassigned at the current event simply remains in the queue $\mathcal{J}_t$ and is reconsidered at subsequent events, so under any work-conserving policy every job is eventually dispatched without forcing an assignment while all slots are busy. The concurrency limit $C_{\max}$ is encoded directly in $|\mathcal{N}|$: the node set $\mathcal{N}$ contains exactly $C_{\max}$ virtual on-demand slots, so at most $C_{\max}$ jobs can run simultaneously at any time t. Slots are costless when idle; rental charges apply only while a job is executing on one.

Equations (3.7)–(3.9) formalise the stochastic provisioning process that Section 3.2 introduced informally, made explicit here because the provisioning delay τ_j^{prov} it produces is a first-class component of the waiting time w_j, not an implementation detail. Each resource type g carries a stock status $s_g(t)$ drawn by comparing a single uniform variate $u \sim \text{Uniform}(0, 1)$ against cumulative thresholds set by a baseline high-stock probability β_g and a time-of-day multiplier $\phi(t)$, where $p_g^H(t) = \min(0.95,\, \beta_g\, \phi(t))$ and $p_g^M(t) = \min(0.90,\, 1.5\, \beta_g\, \phi(t))$ (Eq. 3.7). When job j is dispatched at time t_j^{dispatch} to a slot of its assigned type g_j, it incurs the provisioning delay τ_j^{prov} drawn against that type's stock status $s_{g_j}(t_j^{\text{dispatch}})$ at the dispatch instant (Eq. 3.8), whose range depends on the stock status (per-status ranges in Section 5.1.1; the Low-state delays are orders of magnitude longer than High-state ones); the realised start time then follows from the dispatch time and this delay (Eq. 3.9), so τ_j^{prov} forms a component of the waiting time w_j (Eq. 3.11). The β_g values, the $\phi(t)$ band calibration, and the resulting frequency of each stock state are calibration details given in Chapter 5.

Equation (3.10) fixes each job's absolute deadline from its per-job deadline window T_j. Unlike hard-deadline models, the CGRSP formulation treats deadlines as soft: violations are penalised in the objective rather than structurally prohibited. This reflects operational reality under the $C_{\max}$ concurrency ceiling: when the offered load $\rho = \lambda\, \overline{W} / C_{\max}$ exceeds one, misses are structurally unavoidable for some jobs regardless of scheduling policy, so a penalty formulation is the appropriate model.

Equations (3.11)–(3.14) define the QoS metrics tracked for each job. Figure 3.2 illustrates how each interval maps onto a job's timeline for both the on-time and the missed-deadline cases. Deadline violations are penalised in the objective via the binary miss indicator $(1 - \delta_j)$ rather than via continuous tardiness τ_j; tardiness (Eq. 3.13) is defined for completeness but is not used in the objective. The miss-indicator form is the appropriate QoS penalty in a rendering-studio context because a missed client deadline is a discrete, qualitatively costly event (contract penalties, pipeline delays) rather than a quantity that scales with how late the job is; once a job will miss, its exact lateness is of secondary operational concern, so the objective penalises whether a deadline is met, not by how much. A corollary is that the scheduler should not sacrifice jobs that can still be saved in order to reduce the lateness of jobs that are already doomed. Mean tardiness is nonetheless reported

as a diagnostic metric in Chapter 5, because the severity of missed deadlines matters operationally even when the count of misses is the primary objective. A robustness study (Chapter 5, Section 5.5.9) re-scores a six-policy subset (FIFO, SPT, EDF, SPT+Rescue, RH and CADR) against a tardiness-based objective, and the per-metric leaders within that subset are unchanged: RH retains the lowest miss rate and CADR attains the lowest mean tardiness. The subset boundary matters and is stated explicitly here, because that study omits the Adaptive EDF-SPT hybrid, which records a lower mean tardiness than CADR in the full ten-scheduler comparison (Section 5.3) while matching neither proposed scheduler on miss rate; CADR is therefore the lowest-tardiness policy of the re-scored six, not of the fleet of policies evaluated in this thesis. What the tardiness objective does shift is the single scalarised winner, which moves towards low-tardiness policies; the decomposed conclusions of this thesis are therefore robust to the deadline-penalty form, while any single collapsed score is not.

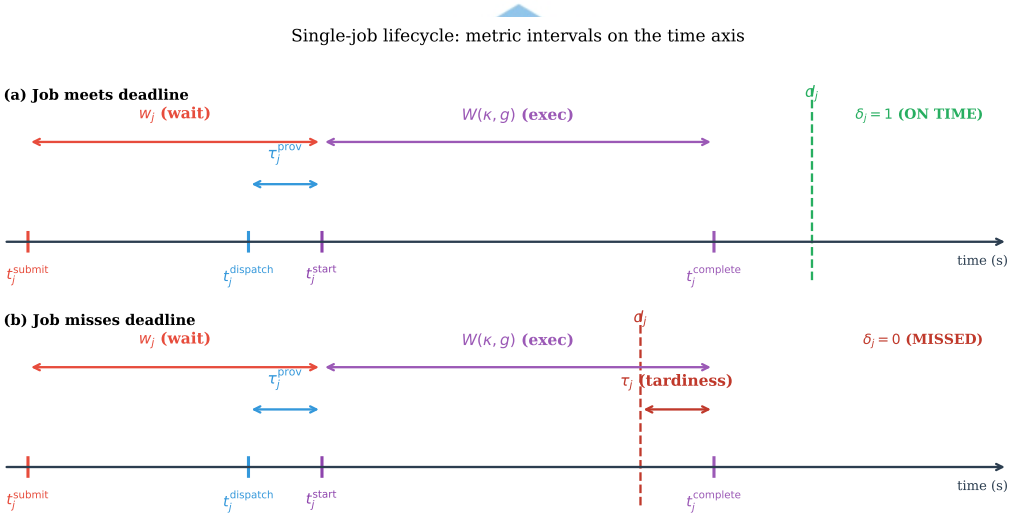

Figure 3.2: Single-job lifecycle: metric intervals mapped onto the time axis. Panel (a) shows a job that meets its deadline ($\delta_j = 1$): the waiting time w_j spans submission to execution start and includes the provisioning delay τ_j^{prov} incurred between dispatch (t_j^{dispatch}, slot acquisition) and execution start (t_j^{start}); the execution time $W(\kappa, g)$ then runs to completion before the deadline d_j. Panel (b) shows a missed deadline ($\delta_j = 0$): completion falls after d_j, and tardiness $\tau_j = t_j^{\text{complete}} - d_j > 0$.

Equation (3.15) defines the per-job execution cost c_j that Eq. 3.3 sums over $\mathcal{J}$ into the operational cost component C_{Cost}: the per-second rental fees for the provisioned GPU instances across all nodes, where $W(j, g(m))$ is the realised execution time of job j on the assigned node's type (the stochastic execution-time model is calibrated in Chapter 5). Note

the asymmetry in how the provisioning delay is accounted: τ_j^{prov} counts fully towards the waiting time w_j (Eqs. 3.9 and 3.11) and hence towards the QoS penalty, but it is *not* billed, since rental charges accrue only over the execution interval $[t_j^{\text{start}}, t_j^{\text{complete}}]$; a scarce GPU type therefore hurts the scheduler through QoS, not through cost. Finally, Constraint (3.16) states the natural domains of the decision and state variables.

Because the value of a missed deadline relative to accumulated waiting (the ratio α_2/α_1) is a business policy rather than a physical constant, no single value is canonical; the deadline-penalty ratio is swept in the objective-weight sensitivity analysis (Section 5.5.9), which additionally evaluates a fully symmetric weighted-sum scalarisation $\min\big[\, w_{\text{qos}}\, C_{\text{QoS}} + w_{\text{cost}}\, C_{\text{Cost}} \,\big]$ with $w_{\text{qos}} = w_{\text{cost}} = 0.5$ as a robustness variant. That study shows that collapsing QoS and cost into a single symmetric scalar lets a near-noise cost difference reorder policies that are clearly separated on QoS; the lexicographic objective avoids this by construction, which is why all headline comparisons are reported on the decomposed metrics (mean wait, miss rate, and cost) rather than on a scalar value.

3.4 Modelling Simplifications and Assumptions

The formulation rests on five modelling simplifications, collected here so that the assumptions behind every later result are stated once in one place rather than reconstructed from the simulator design (Chapter 4) or from the experimental chapters that follow. Each remark states the simplification, why it is defensible for the rendering workload studied, and what relaxing it would require.

1. **Remark 1, stochastic execution times.** The realised execution time $W(j, g)$ is a random variable, not a constant fixed by the pair (complexity level, resource type). It is drawn from the per-stratum log-normal service-time model specified in Chapter 5, whose expectation is the stratum mean $\overline{W}(\kappa_j, g)$. What the pair does fix is that expectation, so a job's realised duration is uncertain while its expected duration is known: schedulers plan with $\overline{W}(\kappa_j, g)$ and observe the realised value only on completion,

mirroring a dispatcher that can classify a scene's complexity before rendering but cannot know its exact walltime. Relaxing this towards heavier-tailed or correlated service times would change the variance of the QoS metrics rather than the structure of the model.

2. **Remark 2, one job per node.** Constraint (3.5) admits at most one running job per node at any instant, so a provisioned instance is held exclusively for the duration of its job. This reflects rendering practice, where a job consumes the full GPU for best performance, and it is what makes the concurrency ceiling $C_{\max}$ a meaningful capacity limit. GPU virtualisation would permit multi-tenancy and would require a per-node resource-share variable in place of the binary x_j^m.

3. **Remark 3, no pre-emption.** Once execution begins, a job runs to completion without interruption, migration, or restart, because checkpoint-restart is expensive or unavailable for rendering pipelines. A consequence used throughout the scheduler designs is that a placement decision is irrevocable for the length of the job, so anticipation must happen before dispatch rather than through later correction.

4. **Remark 4, Poisson arrivals.** Jobs arrive as a Poisson process with mean rate λ, that is, with independent, memoryless, exponentially distributed inter-arrival times, the canonical model for independent submissions from separate clients and scenes. The day-type values of λ used in the experiments are set in Chapter 5. Bursty or self-similar arrival processes would raise queueing delay at the same mean rate and are not modelled here.

5. **Remark 5, binary availability.** The idle indicator satisfies $a_m^t \in \{0, 1\}$, so the scheduler observes a definite idle or running status for every slot rather than a partial or probabilistic one. Market scarcity is not modelled as denial of service under the on-demand Secure Cloud pool studied here, since a held slot is never reclaimed; it enters instead through the stochastic provisioning delay τ_j^{prov} of Eq. (3.8), which is itself driven by the per-type stock status $s_g(t)$ of Eq. (3.7). A spot-market variant would restore the Unavailable state of Section 3.2 and make availability itself stochastic.

3.5 Numerical Example

A minimal three-job, two-GPU instance makes the interplay of constraints visible. Two resource types are in the fleet: an RTX 3090 (g_1) and an RTX A4000 (g_2). The concurrency cap is $C_{\max}$ (five slots in the experimental platform, Section 5.1.1); only three are used here, leaving two idle. Three jobs $\mathcal{J} = \{j_1, j_2, j_3\}$ arrive simultaneously at $t = 0$; nodes begin available ($a_m^0 = 1$ for all $m \in \mathcal{N}$), and every resource type starts at High stock ($s_g(0) = \text{High}$).

Job Characteristics:

Job	Complexity	Submit Time	Deadline	Priority
j_1	$\kappa = 1$ (Low)	$t = 0$	$t = 3600$ (tight)	High
j_2	$\kappa = 2$ (Medium)	$t = 0$	$t = 28800$ (loose)	Medium
j_3	$\kappa = 1$ (Low)	$t = 0$	$t = 28800$ (loose)	Low

Execution Times $W(\kappa, g)$ (seconds; see Table 4.1 for full GPU specifications):

Complexity	RTX 3090 (g_1)	RTX A4000 (g_2)
Low ($\kappa = 1$)	59.7	60.0
Medium ($\kappa = 2$)	70.2	68.7
High ($\kappa = 3$)	100.9	98.2

Figure 3.3 illustrates the assignment and timeline.

Assignment under the lexicographic objective. Job j_1 carries a tight one-hour deadline, so QoS, the primary criterion, routes it to the faster RTX 3090 ($x_{j_1}^{m_1} = 1$): paying the higher rental rate buys reduced execution time and deadline safety, and cost is not allowed to override that. Jobs j_2 and j_3 both have loose eight-hour deadlines, so they meet their deadline with zero waiting on either GPU type and are QoS-equivalent across the fleet; the secondary cost criterion therefore decides their placement, and both are routed to the cheaper RTX A4000 ($x_{j_2}^{m_2} = x_{j_3}^{m_3} = 1$), banking the lower rental rate at no QoS penalty.

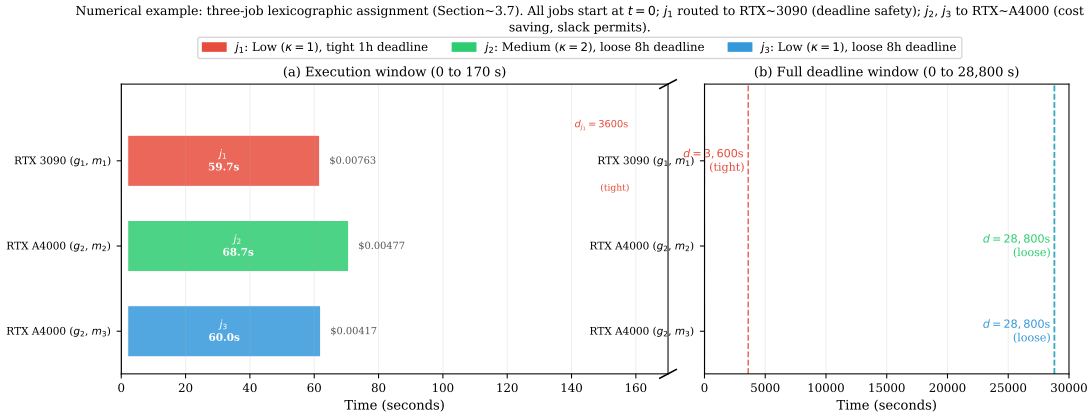

Figure 3.3: Visual aid for the three-job numerical example. (a) Execution window: all three jobs start immediately (High stock, negligible provisioning delay). j_1 (red) runs on the RTX 3090 for 59.7 s; j_2 (green) and j_3 (blue) run concurrently on separate RTX A4000 slots for 68.7 s and 60.0 s respectively. Dollar annotations show per-job cost. The tight deadline $d_{j_1} = 3{,}600$ s is indicated by a dashed red line. (b) Full deadline window: j_2 and j_3 each have an 8-hour loose deadline (28,800 s), vastly exceeding their execution times and confirming that cost is the only discriminating criterion for those two jobs.

Note that j_2's medium complexity creates no tension here: in the medium-complexity stratum the A4000's mean, a sparse-cell estimate ($n = 3$, below the reliability threshold), is in fact marginally lower (68.7 s vs 70.2 s), so the cheaper slot is also the faster one, and even where the RTX 3090 is faster (the low-complexity stratum), eight hours of slack would strip that speed advantage of any QoS value and the cost tiebreaker would still prevail. The two remaining slots are left idle ($x_j^{m_4} = x_j^{m_5} = 0$ for every j).

Outcome. All three jobs start immediately: every type begins at High stock, where the provisioning delay (Eq. 3.8) is at most a few seconds and is taken as negligible here, so every waiting-time penalty is zero ($w_{j_1} = w_{j_2} = w_{j_3} = 0$ s) and all deadlines are met (the QoS objective is at its optimum). Rental cost then works out to $59.7 \times 0.000128 + 68.7 \times 0.0000694 + 60.0 \times 0.0000694 = \$0.017$, where the per-second rates are derived from the RTX 3090 price of \$0.46/hr and the RTX A4000 price of \$0.25/hr (Table 4.1). Fleet utilisation is $3/5 = 60\%$. The key observation is that cost savings on the loose jobs j_2 and j_3 come at no QoS penalty because their deadlines are slack, whereas no such economy is available for j_1 without endangering its tight deadline: this asymmetry, QoS first and cost only among QoS-equivalent options, drives the scheduler designs described in Chapter 4.

Chapter 4 System Design and Scheduling Algorithms

4.1 System Architecture Overview

The CGRSP evaluation system comprises five tightly coupled components (Figure 4.1). The components communicate through a single shared event queue: the Job Generator places arrival events, the DES Core dispatches them and invokes the Scheduler, the Scheduler queries the Fleet Manager for idle nodes, and the Metrics Collector receives completion notifications.

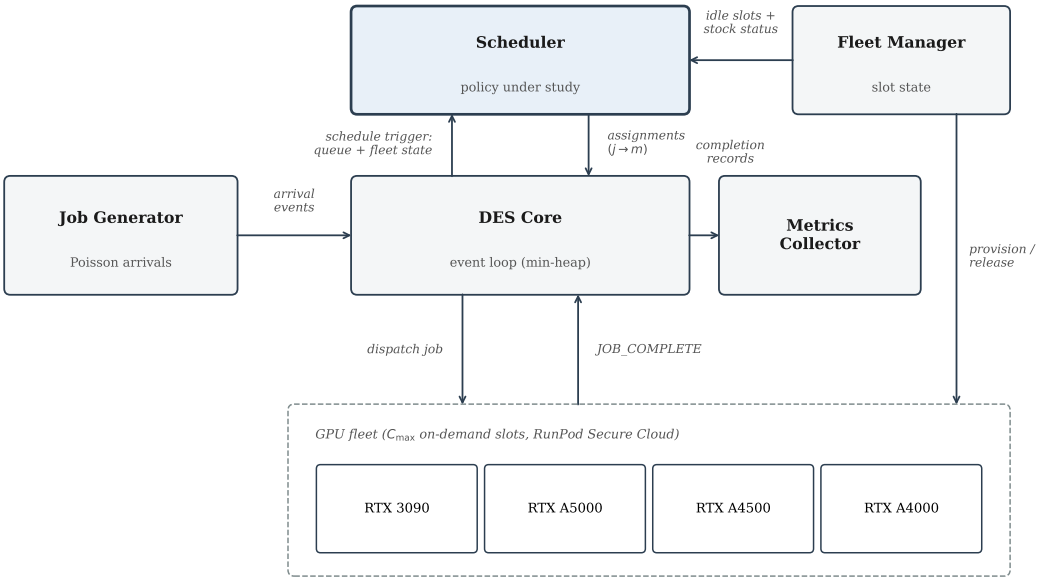

Figure 4.1: CGRSP system architecture: five components interact through a shared discrete-event queue. The Job Generator produces a Poisson job stream; the DES Core advances simulation time and invokes the Scheduler at each event; the Scheduler queries the Fleet Manager for idle slots and returns assignments; the Metrics Collector records per-job outcomes. The GPU fleet (dashed box) comprises four heterogeneous GPU types operating under stochastic stock-status provisioning-delay dynamics.

1. **Job Generator**: produces a Poisson job stream with configurable arrival rate, complexity distribution, and deadline distribution.

2. **Discrete-Event Simulator Core**: maintains a min-heap priority event queue and advances simulation time from event to event.

3. **Fleet Manager**: tracks active instance slots up to $C_{\max}$, models per-type stock availability, and reports available slots to the scheduler. An "idle" slot has no GPU instance provisioned and incurs no cost; it represents unused concurrency capacity.

4. **Scheduler**: implements the scheduling policy; receives the current queue and fleet state at each scheduling decision point and returns a list of (job, node) assignments.

5. **Metrics Collector**: receives completion notifications and records per-job outcomes (waiting time, deadline adherence, cost) for reporting.

The simulator is implemented in Python using only the standard library and NumPy, without external simulation frameworks, to ensure full control over event semantics and metric computation.

4.2 Discrete-Event Simulator Core

The simulator maintains a min-heap event queue ordered by event timestamp. Three event types drive all experiments (the implementation defines two further types, node transition and simulation end, for spot-market generality; the former is inert under the on-demand fleet):

- `JOB_ARRIVAL`(j, t): Job j arrives at time t. The job is added to the waiting queue, and a `SCHEDULE_TRIGGER` event is generated immediately.

- `JOB_COMPLETE`(j, m, t): Job j completes on node m at time t. Node m is freed, metrics are recorded, and a `SCHEDULE_TRIGGER` event is generated.

- `SCHEDULE_TRIGGER`(t): The scheduler is invoked at time t. If any idle nodes and waiting jobs exist, the scheduler returns an assignment; the simulation then generates the corresponding `JOB_COMPLETE` event at $t + \tau_j^{\text{prov}} + W(\kappa_j, g(m))$ (Eq. 3.9).

A job dispatched to a slot of type g does not start instantly: it incurs a provisioning delay drawn from the stock-status-dependent range of Eq. 3.8 before entering the **running** state. No node-failure or node-restoration events occur, because the on-demand Secure Cloud pool never reclaims a held slot (Section 3.2).

4.3 GPU Fleet Model

The fleet manager instantiates nodes according to the GPU type configuration. Each node is parameterised by:

- **GPU type** $g \in \mathcal{G}$: determines p_g and the execution time $W(\kappa, g)$.
- **Stock status**: High / Medium / Low, updated stochastically; it sets the provisioning delay (Eq. 3.8) and additionally informs node selection as a proxy for availability reliability, in scheduler-specific ways: EDF applies the stock tier as the *primary* node-selection key (before execution time); CADR applies it as a *hard filter*, excluding Low-stock types whenever an alternative exists; LCF weights it multiplicatively in its effective cost; and the SPT family, Balanced, and Adaptive add a stock penalty to their placement scores, whose Low-stock term is large enough to dominate the speed and cost terms.
- **Current state**: idle or running.

Table 4.1 shows the GPU fleet configuration used in all experiments. Pricing is taken from a RunPod Secure Cloud snapshot collected on 10 March 2026 and is treated as a **fixed deterministic constant** throughout all simulations. RunPod prices are dynamic in practice and may differ at other points in time; the cost comparisons in this thesis are therefore valid as a snapshot analysis at that date, not as a prediction of future relative costs. The execution times derive from the 6,627-job empirical dataset.

The concurrency budget is realised as $C_{\max}$ on-demand rental slots (five on the experimental platform, Section 5.1.1). Each slot is a separate cloud instance that is rented, used for one job, and then released; when the job finishes the slot becomes free and is offered

Table 4.1: GPU types available on RunPod Secure Cloud. Prices are fixed deterministic values from the 10 March 2026 snapshot (RTX A4500 and RTX A4000 share the $0.25/hr secure price at that date). Execution times from the 6,627-job empirical dataset; all experiments use $C_{\max} = 5$ (RunPod default account limit [30]) as the baseline concurrency limit.

GPU Type	**CUDA Cores**	**VRAM**	**Price ($/hr)**	**Speed Rank**[†]	**Exec. Time (avg)**	
					Low κ	High κ
RTX A5000	8,192	24 GB	0.27	1	60.9 s	99.6 s
RTX 3090	10,496	24 GB	0.46	2	59.7 s	100.9 s
RTX A4500	7,168	20 GB	0.25	3	62.2 s	88.7 s
RTX A4000	6,144	16 GB	0.25	4	60.0 s	98.2 s

[†]Speed rank by empirical average rendering time across all complexity levels (74.0, 78.1, 86.0, 97.9 s for A5000, 3090, A4500, A4000 respectively). These averages are weighted by how each type's own jobs happen to be spread across the three complexity levels, not balanced across them, and the A4000's 257 jobs sit almost entirely at high complexity (254 of them), so its pooled average is close to a high-complexity average while the A5000's is not. The Speed Rank column should therefore be read as a summary of this particular workload mix rather than as a like-for-like speed ordering: at matched high complexity the ordering is close to reversed, with the A4500 fastest, then the A4000, the A5000, and the RTX 3090 last. The simulator never uses these pooled averages; it draws service times from the per-stratum means $\overline{W}(\kappa, g)$ instead. The ranking is also non-monotone in CUDA cores: the RTX A4000, despite having the fewest cores, is the slowest on this pooled average, yet at high complexity ($\kappa = 3$) it renders faster (98.2 s) than the RTX 3090 (100.9 s), and the RTX A4500 is in fact the fastest at $\kappa = 3$ (88.7 s). Low-κ A4000/A4500 values are estimates (no low-complexity jobs for those types in the dataset).

to the scheduler again. The fleet composition (the model parameter n_g) is fixed for the whole simulation. At construction the simulator assigns a GPU type to each of the $C_{\max}$ slots by round robin over the supported-type list (RTX 3090, RTX A5000, RTX A4500, RTX A4000), which at $C_{\max} = 5$ gives two RTX 3090 slots and one each of RTX A5000, RTX A4500, and RTX A4000. A slot keeps its type for the entire run: the scheduler chooses *which* slot to dispatch to, and thereby which GPU type a job runs on, but it never re-types a slot. This composition holds in every experiment except the concurrency-limit

sensitivity study (Section 5.5.3), which varies $C_{\max}$ and so re-runs the round-robin assignment for each $C_{\max}$ value.

The fixed composition is a modelling limitation worth stating explicitly, because it bounds what any scheduling policy can achieve. Exactly one slot carries the RTX A5000, so at most one job at a time can occupy the type ranked fastest on the pooled averages of Table 4.1, however strongly a speed-seeking policy prefers it; and two of the five slots carry the most expensive RTX 3090, so under saturation, when every slot is busy, two fifths of the concurrent work runs on the costliest type whatever the policy decides. A fleet whose idle slots could be re-provisioned to a different type on demand would give the cost-aware and speed-aware policies a larger decision space than they have here. The comparisons reported in Chapter 5 are therefore conditional on this fixed heterogeneous mix, and are not claims about a fleet free to re-provision.

Execution times are modelled stochastically. When a job of complexity level $\kappa \in \{1, 2, 3\}$ is dispatched to GPU type g, its realised walltime is drawn from a log-normal distribution $W(j, g) \sim \text{LogNormal}(\mu_{\kappa,g}, \sigma^2_{\kappa,g})$ whose expected value equals the mean execution time $\overline{W}(\kappa, g)$ of the corresponding (κ, g) stratum in the empirical dataset (Eq. 5.1, Section 5.1.1). The within-stratum dispersion $\sigma_{\kappa,g}$ is fitted directly from the data as the standard deviation of ln(render time) within each stratum (pooled $\sigma_{\ln} \approx 0.11$; strata with fewer than five observations use the pooled value). The log-normal form respects the strictly positive, right-skewed shape of the empirical measurements, with a minority of jobs running substantially longer than the mean (outliers from complex geometry or large textures). Schedulers plan with the expected service time $\overline{W}(\kappa_j, g)$ as the estimate e_j, since the exact walltime of a job is not observable at dispatch; only the realised completion time is stochastic. The realised execution time is sampled from a per-simulation random stream seeded independently of the provisioning-delay draws, so that scheduler comparisons remain reproducible and differences are attributable to scheduling policy.

4.4 Job Generator

Jobs arrive according to a Poisson process with configurable mean rate λ (jobs/second). The Poisson assumption is standard for independent job-submission workloads: jobs originate from different clients and scenes, making submissions memoryless with exponentially distributed inter-arrival times, the canonical model for computer system workloads [26]. Each job is independently assigned:

- **Complexity** $\kappa_j \in \{1, 2, 3\}$: sampled from an empirical distribution (33.14% low, 33.85% medium, 33.01% high), the exact per-stratum proportions of the 6,627-job dataset (2,196 / 2,243 / 2,188 jobs respectively), which the complexity thresholds themselves define as an almost-even three-way split since they are set at the dataset's 33rd and 67th percentile of render time.
- **Deadline** $d_j = t_j^{\text{submit}} + T_j$, $T_j \in \{T_{\text{tight}}, T_{\text{loose}}\}$ (Eq. 3.10): in the baseline scenario each job draws either the tight deadline window T_{tight} (rush-order renders) or the loose window T_{loose} (standard batch renders); the values and class proportions are the platform parameters of Section 5.1.1. The tight/loose split is held constant across all day types; load intensity is varied through λ and N only.
- **Estimated duration**: set to the mean execution time $\mathbb{E}[W(\kappa_j, g_{\text{ref}})]$ for the reference GPU (RTX 3090), recorded as job metadata; schedulers compute their own per-node estimates at dispatch (Section 4.5.2).

4.5 Scheduling Algorithms

All ten scheduling algorithms implement the same interface: they receive the current waiting job queue and fleet state, and return a list of (job, node) assignments (zero or more, executed simultaneously for multiple idle nodes).

4.5.1 FIFO: First-In-First-Out

FIFO assigns waiting jobs to idle nodes in submission-time order (earliest submit time first), as summarised in Algorithm 1. Node selection among idle nodes is arbitrary (earliest-registered first in the implementation). FIFO serves as the primary baseline because it requires no job characteristic knowledge and is deadline-oblivious.

Algorithm 1 FIFO Scheduler

Require: Queue Q sorted by t_j^{submit}; set I of idle nodes

1: Sort Q by t_j^{submit} ascending
2: **for** each (j, m) in zip(Q, I) **do**
3: Assign job j to node m
4: **end for**

4.5.2 SPT: Shortest Processing Time

SPT prioritises the job with the smallest expected execution time, computed at each event as $e_j = \min_{m \in I} \overline{W}(\kappa_j, g(m))$ over the currently idle capable nodes, so the ranking always reflects the fleet actually available at dispatch. (The job metadata also carries a reference-GPU estimate $\mathbb{E}[W(\kappa_j, g_{\text{ref}})]$ set at generation time, but the scheduler does not read it.) Because all three complexity levels are calibrated to the same reference GPU, SPT naturally front-loads the shortest (low-complexity) jobs, reducing the mean number of jobs in the queue at any instant, the discrete-time analogue of the M/G/1 SPT optimality result [8, 18].

Node selection: given the sorted job queue, SPT assigns each job to the idle node minimising a normalised composite of speed, cost, and stock reliability of the same structural form as the Balanced score (Eq. 4.2),

$$\text{score}(j, m) = (1 - w_c) \frac{\overline{W}(\kappa_j, g(m))}{\overline{W}_{\min}(j)} + w_c \frac{p_{g(m)}}{p_{\min}} + \text{sp}(s_m), \tag{4.1}$$

with cost weight $w_c = 0.3$, where $\overline{W}_{\min}(j)$ is the fastest expected execution time for job j across the idle nodes, $p_{\min}$ the cheapest per-second rate among them, and $\text{sp}(s_m) \in$

$\{0, 0.2, 1.0\}$ a High/Medium/Low stock-status penalty (writing s_m as shorthand for the stock status $s_{g(m)}(t)$ of the node's type). Speed carries the larger weight (0.7), so the fastest node is selected unless a marginally slower node is sufficiently cheaper or more reliably stocked to overcome the speed term; when cost and stock are uniform across the idle nodes the rule reduces to the min-min fastest-finish heuristic of [5] applied within each scheduling decision. Algorithm 2 summarises the procedure.

Algorithm 2 SPT Scheduler

Require: Queue Q; idle node set I; current time t

Sort Q ascending by $e_j = \min_{m \in I} \overline{W}(\kappa_j, g(m))$
for each job j in Q **do**
  if $I = \emptyset$ **then**
    break
  end if
  $m^* \leftarrow \arg\min_{m \in I} \left[(1-w_c)\frac{\overline{W}(\kappa_j, g(m))}{\overline{W}_{\min}(j)} + w_c \frac{p_{g(m)}}{p_{\min}} + \mathrm{sp}(s_m)\right]$ {Eq. 4.1, w_c=0.3}
  Assign $j \to m^*$; remove m^* from I
end for

4.5.3 EDF: Earliest Deadline First

EDF prioritises the job with the nearest absolute deadline d_j. When two jobs have the same deadline, submission time is used as a tiebreaker. Node selection minimises the expected execution time $\overline{W}(\kappa_j, g(m))$ for the job's specific complexity level, with stock-availability tiers applied first (High $\prec$ Medium $\prec$ Low). Using job-specific execution times (rather than a generic speed multiplier) is necessary because GPU rank varies by complexity: for example, RTX A4000 (98.2 s) outperforms RTX 3090 (100.9 s) on high-complexity jobs despite a lower nominal speed rating.

4.5.4 LCF: Lowest Cost First

LCF minimises total operational cost by preferring the cheapest available GPU type. Node selection uses an effective cost score $c_{\text{eff}}(m) = p_{g(m)} \cdot \overline{W}(\kappa_j, g(m)) \cdot \psi(s_m)$, where

$\psi(s_m) \in \{1.0, 1.05, 1.15\}$ is a stock-status penalty (High/Medium/Low stock) reflecting availability reliability. Job ordering within LCF is by submission time (FIFO order).

The stock-status penalty ensures that when two GPUs have the same rental price, the one with higher availability reliability is preferred, avoiding types that are likely to be scarce and so incur a long provisioning delay.

4.5.5 Balanced: Weighted Speed-Cost Composite

Balanced assigns a composite score to each (job, node) pair:

$$\text{score}(j, m) = w_b \cdot \frac{\overline{W}(\kappa_j, g(m))}{\overline{W}_{\max}(j)} + (1 - w_b) \cdot \frac{p_{g(m)}}{p_{\max}} + \text{sp}(s_m) \tag{4.2}$$

where $w_b = 0.8$ (speed-biased), $\overline{W}_{\max}(j)$ is the maximum execution time for job j across all fleet nodes (per-job normalisation), $p_{\max}$ is the global maximum per-second cost across the fleet, and $\text{sp}(s_m) \in \{0.0, 0.2, 1.0\}$ is a stock-status penalty. The job ordering within Balanced is by submission time (FIFO), so Balanced's differentiation from FIFO comes entirely from its node selection score. This is deliberately distinct from EDF's deadline-based job ordering.

4.5.6 Random: Random Baseline

Random selects both job and node uniformly at random from the available sets. It serves as a lower-bound baseline to verify that the structured heuristics provide a benefit over chance.

4.5.7 Design Progression

The ten scheduling algorithms evaluated in this study represent a progressive refinement of deadline awareness and wait minimisation. FIFO provides the deadline-oblivious baseline. SPT reduces mean wait by front-loading short jobs but remains deadline-blind.

SPT+Rescue introduces the laxity rescue concept: a secondary EDF tier that promotes jobs whose deadline slack has fallen below a threshold. The Adaptive scheduler (Section 4.5.12) extends that idea with a hopeless tier and a queue-pressure rule that widens the urgent tier under congestion, interpolating between SPT and EDF. The Rolling-Horizon scheduler (Section 4.5.10) goes further: instead of fixed tiers it plans over the slot-availability timeline and anticipated arrivals, promoting a job only when the timeline shows it would otherwise miss. The CADR scheduler (Section 4.5.11) introduces a scale-free critical-ratio triage that achieves near-RH miss performance while substantially reducing the tardiness (lateness) of missed deadlines by being less aggressive in demoting already-at-risk jobs. Figure 4.2 summarises this lineage: each arrow marks the algorithmic ingredient added at that step, and the two proposed schedulers (RH and CADR) both descend from SPT+Rescue.

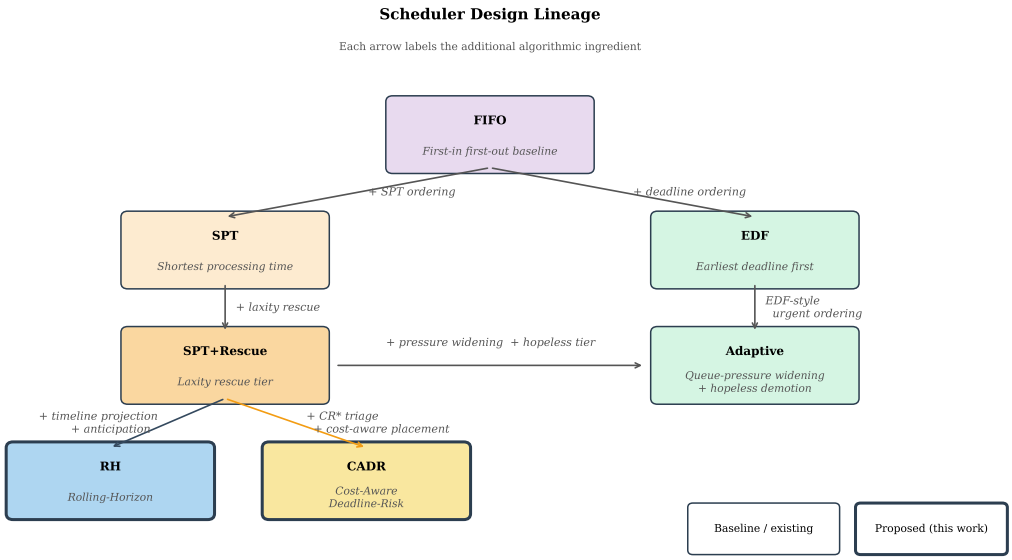

Figure 4.2: Scheduler design lineage. Each arrow labels the algorithmic ingredient added at that step. FIFO is the deadline-oblivious baseline; SPT adds shortest-job-first ordering to reduce wait; EDF adds deadline ordering to reduce misses; SPT+Rescue adds a laxity rescue tier to combine both effects; the Adaptive hybrid combines EDF-style urgent ordering with the rescue idea's queue-pressure tier widening and hopeless demotion; the proposed RH scheduler replaces fixed tiers with a forward timeline projection and adds load-gated anticipation; the proposed CADR scheduler replaces the laxity threshold with a scale-free critical-ratio triage and adds cost-aware node selection.

4.5.8 SPT+Rescue: Hybrid Throughput-Deadline Policy

SPT+Rescue introduces the laxity rescue concept as an intermediate design step. It combines SPT's wait-time advantage with an EDF-based deadline rescue mechanism: most jobs are ordered by shortest processing time, but any job whose *laxity* drops below a threshold is promoted to an urgent tier and dispatched before all normal-tier jobs.

The laxity of job j at scheduling decision time t is defined as:

$$L_j(t) = d_j - t - \min_{m \in I} \overline{W}(\kappa_j, g(m)) \tag{4.3}$$

where the minimum is taken over all currently idle nodes I. Laxity measures the slack remaining: if $L_j(t) < 0$ a miss is already inevitable; if $L_j(t)$ is small, a miss is imminent unless j is dispatched at the current event.

At each scheduling event the scheduler partitions the waiting queue into two tiers (Figure 4.3):

- **Urgent** ($L_j < \theta$): sorted by earliest absolute deadline (EDF order within tier)
- **Normal** ($L_j \geq \theta$): sorted by shortest estimated processing time (SPT order)

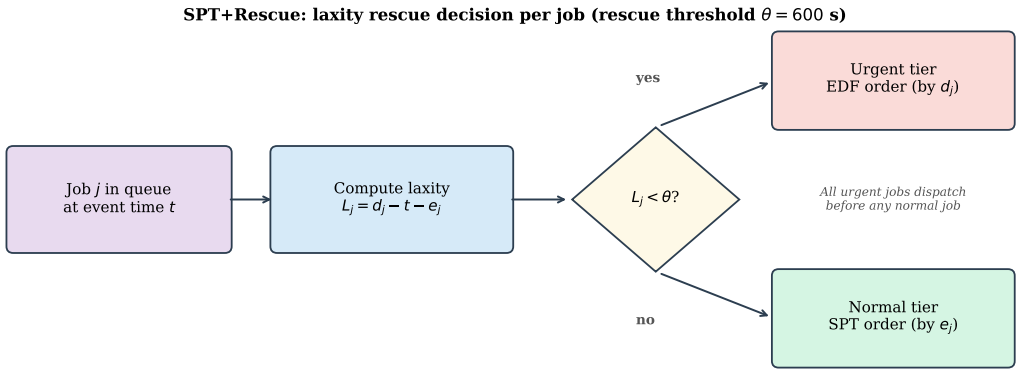

Figure 4.3: SPT+Rescue per-job decision flow. At each scheduling event, every queued job's laxity $L_j = d_j - t - e_j$ is computed against the fastest available execution time; jobs with laxity below the rescue threshold θ enter the urgent tier (EDF order), the rest stay in the normal tier (SPT order), and all urgent jobs dispatch before any normal job.

All urgent jobs are dispatched before any normal-tier job. The rescue threshold is set to $\theta = 600\,\text{s}$ (10 minutes) in all experiments. This value was chosen by the following reasoning: by the laxity definition (Eq. 4.3), a job that has just crossed the rescue threshold has laxity exactly θ, so dispatching it immediately leaves exactly $\theta = 600\,\text{s}$ to spare before the deadline, comfortably above even a high-complexity job's maximum execution time ($W \approx 150\,\text{s}$). A sensitivity study over $\theta \in \{60, 180, 300, 600, 1200\}\,\text{s}$ (Section 5.5.4) shows performance is relatively insensitive to threshold choice, with miss rates varying by only 4.73 percentage points across the whole swept range; the sweep minimum lies at $\theta = 180\,\text{s}$, but the differences within this band are comparable to sampling noise, so the default $\theta = 600\,\text{s}$ requires no tuning. Algorithm 3 summarises the two-tier procedure.

Algorithm 3 SPT+Rescue Scheduler

Require: Queue Q; idle node set I; current time t; threshold θ

for each job $j \in Q$ **do**
  $e_j \leftarrow \min_{m \in I} \overline{W}(\kappa_j, g(m))$ {fastest available expected exec time}
  $L_j \leftarrow d_j - t - e_j$
end for
$Q_{\text{urgent}} \leftarrow \{j \in Q : L_j < \theta\}$, sorted by d_j ascending
$Q_{\text{normal}} \leftarrow \{j \in Q : L_j \geq \theta\}$, sorted by e_j ascending
for each job j in $Q_{\text{urgent}} \,\|\, Q_{\text{normal}}$ **do**
  if $I = \emptyset$ **then**
    break
  end if
  $m^* \leftarrow \arg\min_{m \in I} \left[(1-w_c)\frac{\overline{W}(\kappa_j, g(m))}{\overline{W}_{\min}(j)} + w_c \frac{p_{g(m)}}{p_{\min}} + \text{sp}(s_m)\right]$ {Eq. 4.1, w_c=0.3}
  Assign $j \to m^*$; remove m^* from I
end for

SPT+Rescue combines SPT's shortest-job-first ordering with EDF deadline rescue: under capacity saturation it achieves 12.50% miss (notably lower than SPT's 18.60%) at 145.85 min wait ($d = -2.337$ vs FIFO, $p < 0.001$), providing a conservative fallback when tight-deadline compliance is a strict operational requirement.

4.5.9 Illustrative Scheduling Example

Figure 4.4 traces a concrete 7-job scenario with $C_{\max} = 3$ nodes (a minimal illustrative configuration chosen so that all 7 jobs queue up and the rescue decision is visible; all experiments use $C_{\max} = 5$) to show how SPT+Rescue differs from FIFO in a single scheduling window. Only J6 carries a tight deadline ($d_6 = 210\,\text{s}$; $\kappa_6 = 2$, medium complexity); all other jobs have loose (8-hour) deadlines. The illustrative scenario uses approximated, complexity-uniform execution times matching the figure legend (κ=1 $\approx 65\,\text{s}$, κ=2 $\approx 75\,\text{s}$, κ=3 $\approx 120\,\text{s}$, the last chosen for narrative clarity rather than as a close rounding of the empirical high-complexity mean). J0, J1, J2 arrive in the first 10 s and fill all three nodes immediately; J3 to J6 queue while all nodes are busy. When node N1 becomes free at $t = 70\,\text{s}$, FIFO dispatches J3 (next in arrival order), then at $t = 75\,\text{s}$ dispatches J4 to N2, eventually assigning J6 to N2 at $t = 140\,\text{s}$. J6 finishes at 215 s, 5 s past its deadline ($\delta_6 = 0$). SPT+Rescue instead evaluates laxity when N1 frees at $t = 70\,\text{s}$ (the sole idle node at that moment; $\overline{W}(\kappa_6{=}2, g(\text{N1})) = 75\,\text{s}$):

$$L_{J_6}(70) = d_6 - 70 - \min_{m \in I} \overline{W}(\kappa_6, g(m)) = 210 - 70 - 75 = 65\,\text{s} \tag{4.4}$$

Since $65\,\text{s}$ is below the $600\,\text{s}$ rescue threshold, J6 is classified urgent and dispatched immediately to N1. J6 starts at $t = 70\,\text{s}$ and finishes at 145 s, meeting its deadline by 65 s ($\delta_6 = 1$). Loose-deadline jobs J3 to J5 are reordered by SPT without any deadline impact.

4.5.10 Rolling-Horizon: Anticipative Receding-Horizon Scheduler

The Rolling-Horizon (RH) scheduler is the primary methodological contribution of this work. Rather than ranking the queue by a single fixed priority rule, it projects the near-future availability of the fleet and dispatches against that projection. At each scheduling event it builds the availability timeline of every usable slot (idle now, or busy until its running job's expected completion), classifies each waiting job by whether the timeline shows it would still meet its deadline if it deferred to the next slot to free, and then greedily

Figure 4.4: Illustrative CGRSP Gantt chart ($C_{\max} = 3$ nodes, 7 jobs; only J6 has a tight deadline $d_6 = 210$ s; execution times approximated from empirical data). Coloured bars show GPU execution by complexity level ($\kappa \in \{1, 2, 3\}$); the red dashed line marks J6's deadline. **FIFO** (top): J6 arrives at $t = 50$ s, waits 90 s, dispatched at $t = 140$ s, finishes at 215 s, deadline miss by 5 s ($\delta_6 = 0$). **SPT+Rescue** (bottom): laxity $L_{J_6}(70) = 65\,\text{s} < 600\,\text{s}$ threshold triggers rescue; J6 dispatched to N1 at $t = 70$ s, finishes at 145 s, meets deadline by 65 s ($\delta_6 = 1$).

places jobs, in that urgency order, each onto the slot that minimises a per-job weighted-sum surrogate of the model objective (Eq. 3.1), namely $w_{\text{qos}}(\alpha_1 w_j + \alpha_2(1-\delta_j)) + w_{\text{cost}}\, c_j$ with $w_{\text{qos}} = w_{\text{cost}} = 0.5$ (the explicit form is given in Algorithm 4). Because the wait and miss penalties dominate this sum numerically while the per-job cost term is comparatively negligible at this fleet scale (Section 5.5.9), the surrogate respects the lexicographic QoS-over-cost priority of Eq. 3.1 in practice; it is used in place of a strict lexicographic comparison only because a single differentiable score is convenient for the greedy per-slot placement. It commits only the dispatches that start at the current instant and re-plans at the next event, in the manner of receding-horizon (model-predictive) control. The placement step is a greedy, per-job assignment rather than a joint optimisation over the horizon; what is novel is that shortest-processing-time ordering, earliest-deadline urgency, and demotion of already-doomed jobs all emerge from the forward projection rather than being imposed as fixed laxity tiers, and that the urgency boundary adapts to the realised fleet state instead

of a hand-set threshold. The scheduler is additionally anticipative: below saturation it reserves a little capacity for imminent tight-deadline arrivals (described below).

The RH scheduler is anticipative but not clairvoyant: it knows the job arrival-rate profile and the deadline-window distribution (operational statistics estimable from a studio's own submission history), but it does not observe the identities or submission times of jobs that have not yet arrived. This is the standard distinction in online stochastic optimisation between distributional knowledge and realisation knowledge: the scheduler may reserve capacity for the *expected* rate of imminent tight-deadline arrivals, never for specific future jobs.

That the arrival rate is estimable in principle does not make the policy indifferent to how well it is estimated, and the sensitivity has an unusual shape that a practitioner needs to know about. The assumed λ enters the policy at exactly one place, the load gate of the anticipation mechanism described below, and the gate is a step function: it compares $\rho = \lambda \overline{W}/C_{\max}$ against ρ_{gate}, which at $C_{\max} = 5$ puts the switching point at $\lambda \approx 0.0627$ jobs/s, only about 37% below the hectic day's true rate of 0.100 jobs/s. A 30-seed sweep in which RH is given a deliberately mis-estimated arrival rate, swept as a multiple of the true λ on the hectic and surge days, confirms both halves of this shape. On the hectic day, mis-estimates that leave the assumed λ on the correct side of the switching point change nothing at all, reproducing the gated result of 9.50% miss at 140.24 min wait exactly; but an under-estimate large enough to cross the point wrongly convinces the scheduler there is spare capacity to hold back, so it reserves a slot in a saturated queue. Miss then more than doubles, and mean wait rises well above FIFO's 169.32 min ($p < 0.001$, paired against FIFO on the same seeds), which is the practically important failure: a studio that under-estimates its own peak load turns the anticipation mechanism from inert into actively harmful. Two qualifications keep this in proportion. Even in that degraded state RH's miss rate remains significantly *below* FIFO's 24.85% ($p < 0.001$), so the policy is not dominated by the baseline; the damage is concentrated in wait. And the exposure is one-sided by day type: on the surge day, whose true rate sits far below the switching point, every under-estimate and moderate over-estimate is inert, and only a fourfold over-estimate crosses the point, switching anticipation off and raising miss from 1.59% to 6.05%. The practical guidance is therefore not that λ must be known precisely, but that

it must be known well enough to place the busiest day on the correct side of ρ_{gate}; a deliberately conservative (high) estimate fails safe on saturated days, at the cost of forgoing anticipation on quiet ones.

Slot timelines. Each of the $C_{\max}$ slots is modelled as an availability timeline. An idle slot is free from the current time t; a running slot is busy until the expected completion of its job (the scheduler tracks expected completion times at dispatch, since it plans with the mean service time e_j). Every slot is either idle or running under the on-demand pool, so the timeline spans all $C_{\max}$ slots. When every slot is idle, t_{free} has no busy slot to derive from and is approximated as the completion time of a hypothetical medium-complexity job started now; this affects the urgent/normal boundary only in the fully idle fleet state. Two further implementation details qualify the picture. Placements are made only onto currently idle slots; running slots enter the plan solely through t_{free}, so the timeline classifies jobs but does not host deferred placements, which is consistent with committing only start-now dispatches. And the projection is certainty-equivalent: a running slot whose realised (log-normal) service time has already exceeded its planned mean is projected to free immediately, so at such decision events t_{free} collapses to the current instant, the urgent tier empties, and the realised ordering reduces to SPT with hopeless-job demotion. Under saturation a nontrivial fraction of decision events are in this state; the saturated-regime miss advantage therefore rests chiefly on doomed-job demotion (the RH-over-SPT+Rescue ablation step), with the EDF-style urgent tier contributing in partially loaded and below-saturation states.

Candidate ordering. For each waiting job the scheduler computes the expected service time $e_j = \min_m \overline{W}(\kappa_j, g(m))$ over capable slots and classifies it against the soonest time a slot would free for a job not dispatched now, t_{free} (Figure 4.5):

- **Urgent**: $t + e_j \leq d_j < t_{\text{free}} + e_j$. The job meets its deadline only if dispatched now. Ordered by earliest deadline (EDF).
- **Normal**: $t_{\text{free}} + e_j \leq d_j$. The job can wait for the next slot and still finish in time. Ordered by shortest e_j (SPT), which minimises mean wait.

- **Hopeless**: $t + e_j > d_j$. The deadline is unreachable even with immediate dispatch. Ordered last.

Because the objective penalises the deadline-miss indicator rather than continuous tardiness (Section 3.3), demoting hopeless jobs is correct: a job that will miss regardless should not displace a job that can still be saved. This is the key advance over SPT+Rescue, whose fixed laxity threshold promotes all low-laxity jobs, including already-doomed ones, ahead of savable normal-tier jobs.

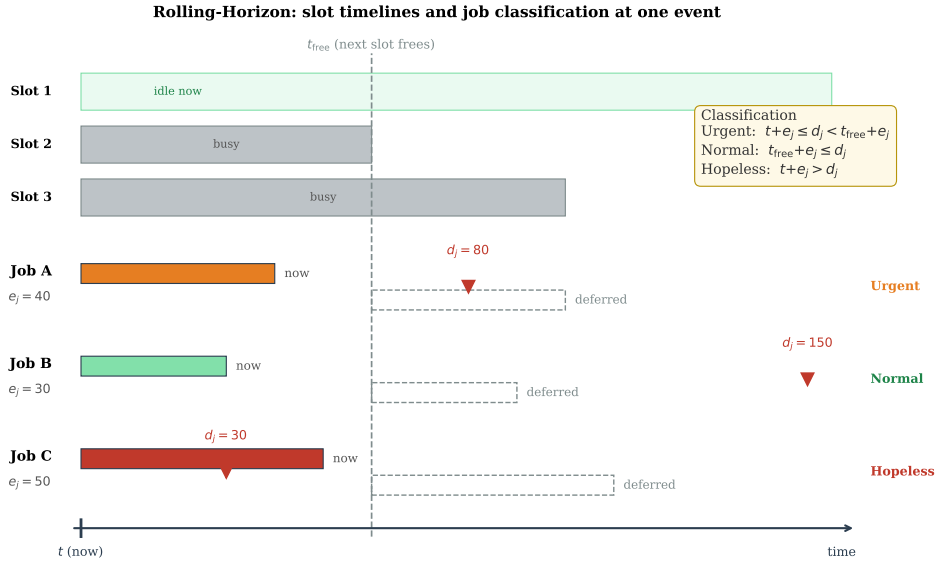

Figure 4.5: Rolling-Horizon slot timelines and job classification at a single scheduling event. Top: the fleet's availability timeline; Slot 1 is idle now, Slot 2 frees at t_{free} (the soonest time a slot becomes available for a job not dispatched now), Slot 3 frees later. Bottom: three waiting jobs, each shown with its execution window if dispatched now (solid bar) and if deferred to t_{free} (dashed outline) against its deadline (red marker). Job A meets its deadline only if dispatched now (Urgent); Job B still finishes in time after deferring (Normal); Job C misses its deadline even with immediate dispatch (Hopeless, dispatched last so it does not displace savable jobs).

Anticipation (load-gated reservation). Below saturation the scheduler reserves a small amount of idle capacity for anticipated tight-deadline (rush-order) arrivals rather than spending it on loose jobs that have ample slack, and it dispatches waiting tight jobs eagerly so that a held slot is actually taken by the rush job it was kept for. (The implementation classifies a job as tight when its deadline window is below 7,200 s, twice T_{tight}; under the baseline $\{3600, 28800\}$ s windows this selects exactly the tight class.) The reservation count ρ_{res} sets how many idle slots are held back ($\rho_{\text{res}} = 1$ by default); tight jobs

may always use any idle slot, and only loose jobs are withheld, so deadline-critical work is never starved of capacity. The mechanism is gated by the offered load $\rho = \lambda \overline{W} / C_{\max}$, computed from the known arrival-rate profile λ (distributional knowledge only, as above) and the mean service time $\overline{W}$: when $\rho \geq \rho_{\text{gate}}$ (here 0.95) the system is saturated, every slot is needed immediately, and reservation is disabled (any ρ_{res} reduces to the plain forward-simulation policy); when $\rho < \rho_{\text{gate}}$ there is genuine spare capacity to hold. This is the operational meaning of anticipation being self-limiting: it activates exactly when reserving a slot for an imminent tight arrival does not delay a job that cannot wait. Section 5.5.7 shows that load-gated anticipation substantially reduces tight-deadline miss below saturation (the surge day) while the gate correctly switches it off under saturation (the hectic day), where reserving capacity would only delay jobs that cannot be deferred. The hectic-day results are therefore produced by the plain forward-simulation policy and are unaffected by anticipation. The monthly results are not: of the four day types in the Phase 4 calendar only the hectic day exceeds the gate, so reservation is live on the surge, normal and quiet days, which together are the large majority of simulated days. Re-running the whole month with reservation disabled, on the identical schedules and per-day seeds, raises the monthly deadline-miss rate and mean tardiness significantly ($p < 0.001$ on each, paired over the monthly replications) while reducing mean wait slightly ($p < 0.01$), the same wait-for-miss trade the surge-day sweep shows in isolation. The monthly headline is thus an anticipation-enabled result, not a plain forward-simulation one; the reservation-disabled arm is retained as a reported comparison arm in the Phase 4 results file, and the shipped RH configuration is unchanged by it.

Node selection. Within a placement the GPU type is chosen by minimising the bi-objective contribution (expected wait and miss penalty, plus operational cost), with a stock-status penalty derived from $s_g(t)$ and a fixed per-type reliability prior (RTX A4000 0, A4500 0.05, A5000 0.15, 3090 0.40) whose ordering mirrors the baseline high-stock probabilities β_g. Because the wait and miss terms are identical across idle-now slots and the per-job cost term is small at this fleet scale, this prior is the deciding term among idle slots in practice: RH systematically prefers the historically better-stocked (and cheaper) types unless the deadline term forces a faster GPU. Secondary ordering tiebreakers (deadline then service time within tiers) are applied as documented in the implementation.

Algorithm 4 Rolling-Horizon Scheduler

Require: Queue Q; usable slots S (idle now, or running until expected completion); time t; arrival rate λ; reservation ρ_{res}; gate ρ_{gate}

$\rho \leftarrow \lambda \overline{W}/C_{\max}$; $R \leftarrow \rho_{\text{res}}$ if $\rho < \rho_{\text{gate}}$ else 0 {load gate: reserve only below saturation}

build slot timelines from S; $t_{\text{free}} \leftarrow$ soonest time a slot frees for a non-dispatched job

for each job $j \in Q$ **do**

  $e_j \leftarrow \min_m \overline{W}(\kappa_j, g(m))$ over capable slots

end for

order Q: hopeless ($t + e_j > d_j$) last; else if $R > 0$ and j tight, urgent (by d_j, eager); else urgent ($t_{\text{free}} + e_j > d_j$, by d_j); else normal (by e_j)

for each candidate j in order **do**

  place j at its earliest feasible start minimising $w_{\text{qos}}(\alpha_1 w_j + \alpha_2(1 - \delta_j)) + w_{\text{cost}} c_j$ plus stock-status penalty and fixed per-type reliability prior

end for

commit each job whose placement starts at t on an idle slot

if $R > 0$: withhold loose-job commits so at least R idle slots stay free for tight arrivals

discard uncommitted placements and re-plan at the next event

The RH scheduler is implemented in `CGRSP/cgrsp/schedulers/rolling_horizon_scheduler.py`. Anticipation is enabled ($\rho_{\text{res}} = 1$, load-gated) in all experiments; because the gate disables it under saturation, the hectic-day results are identical to the plain forward-simulation policy, while the surge day exhibits its benefit and the monthly results, whose calendar is mostly below the gate, carry it throughout. The reservation count and the gate are examined in Section 5.5.7.

4.5.11 CADR: Cost-Aware Deadline-Risk

The Cost-Aware Deadline-Risk (CADR) scheduler is the second scheduling contribution of this work. Like RH, CADR classifies waiting jobs into tiers based on deadline

urgency; unlike RH, it uses a scale-free critical ratio rather than a forward-simulation timeline, and it integrates cost-aware node selection as a secondary objective.

Critical ratio. The critical ratio is a classical dispatching rule from the operations-research literature [8]; the contribution here is its combination with the scale-free three-tier triage, doomed-last demotion, and stock-filtered cheapest-deadline-feasible placement. For each job j at scheduling time t with estimated service time $e_j = \min_{m \in I} \overline{W}(\kappa_j, g(m))$, the critical ratio is:

$$\mathrm{CR}_j(t) = \frac{d_j - t}{e_j} \tag{4.5}$$

A ratio $\mathrm{CR}_j \leq 1$ means the remaining time is at most the required service time, so immediate dispatch leaves no slack for any real-world delay; such jobs are treated as unreachable and dispatched last. $\mathrm{CR}_j = \mathrm{CR}^*$ is the boundary between at-risk and safe jobs, with a baseline $\mathrm{CR}^* = 3$.

Three-tier classification. At each scheduling event the waiting queue is partitioned by the critical ratio (Eq. 4.5) into three tiers (Figure 4.6a):

- **Doomed** ($\mathrm{CR}_j \leq 1$): deadline is unreachable even now; dispatched *last* so they do not displace savable jobs.
- **At-risk** ($1 < \mathrm{CR}_j \leq \mathrm{CR}^*$): deadline is reachable but slack is tight; dispatched in earliest-deadline-first (EDF) order.
- **Safe** ($\mathrm{CR}_j > \mathrm{CR}^*$): ample slack remaining; dispatched in shortest-processing-time (SPT) order to minimise mean wait.

Cost-aware node selection. Within a placement (Figure 4.6b), CADR first restricts the capable nodes to those not currently low on stock (falling back to all capable nodes only when every capable node is low-stock, since low stock signals a longer expected provisioning delay), then selects the cheapest among them whose estimated finish time satisfies the deadline (deadline-feasible cheapest node). If no deadline-feasible node exists, it falls back to the fastest node in the same stock-filtered pool. This integrates cost awareness directly into dispatch rather than as a post-hoc re-scoring.

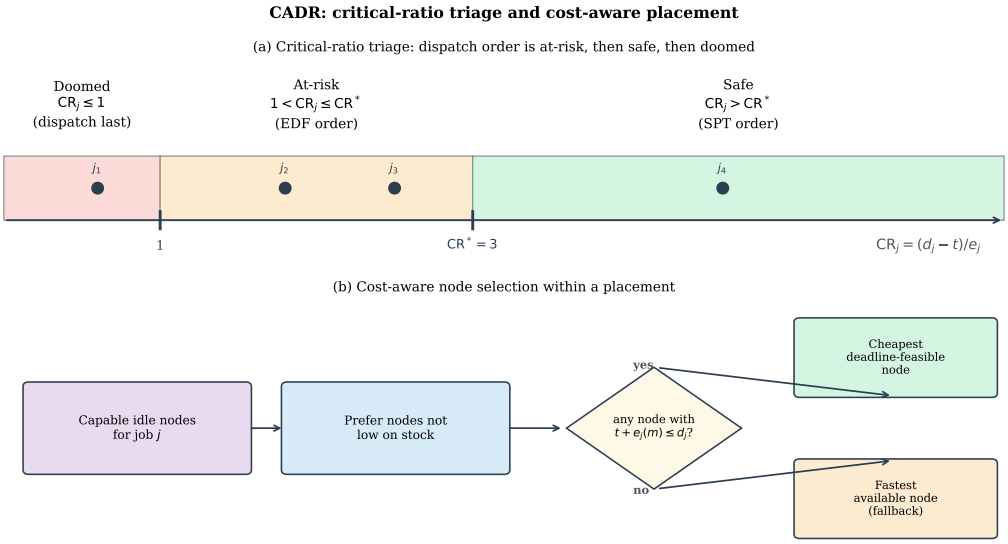

Figure 4.6: CADR scheduling mechanics. Panel (a): the critical ratio $\mathrm{CR}_j = (d_j - t)/e_j$ places every queued job on a scale-free axis partitioned into three tiers: doomed ($\mathrm{CR}_j \leq 1$, dispatched last), at-risk ($1 < \mathrm{CR}_j \leq \mathrm{CR}^*$, EDF order), and safe ($\mathrm{CR}_j > \mathrm{CR}^*$, SPT order); example jobs j_1 to j_4 illustrate the triage. Panel (b): within each placement, node selection prefers non-low-stock nodes and picks the cheapest deadline-feasible node, falling back to the fastest available node when no node can meet the deadline.

Why CADR has lower tardiness than RH. RH's forward-simulation demotes jobs to the "hopeless" tier as soon as the timeline shows they will miss, which can push those jobs far back in the queue, resulting in very large lateness for the jobs that do miss. CADR's critical-ratio demotion is less aggressive: a doomed job ($\mathrm{CR}_j \leq 1$) is dispatched last, but its relative position is determined only by the current instant, not by a multi-step forward plan. As a result, missed deadlines under CADR are missed by less, giving substantially lower mean tardiness than RH while keeping the miss count nearly as low.

Ablation variant. The CADR-order-only (CADR-OO) variant retains the three-tier critical-ratio ordering but replaces cost-aware node selection with fastest-available-node placement within the stock-filtered pool. This isolates the contribution of the cost-aware placement component from the ordering component (Section 5.4).

Algorithm 5 CADR Scheduler

Require: Queue Q; idle node set I; current time t; threshold $\text{CR}^* = 3$

for each job $j \in Q$ **do**
  $e_j \leftarrow \min_{m \in I} \overline{W}(\kappa_j, g(m))$
  $\text{CR}_j \leftarrow (d_j - t)/e_j$
end for
$Q_{\text{doomed}} \leftarrow \{j \in Q : \text{CR}_j \leq 1\}$, ordered last
$Q_{\text{atrisk}} \leftarrow \{j \in Q : 1 < \text{CR}_j \leq \text{CR}^*\}$, sorted by d_j ascending (EDF)
$Q_{\text{safe}} \leftarrow \{j \in Q : \text{CR}_j > \text{CR}^*\}$, sorted by e_j ascending (SPT)
for each job j in $Q_{\text{atrisk}} \,\|\, Q_{\text{safe}} \,\|\, Q_{\text{doomed}}$ **do**
  if $I = \emptyset$ **then**
    break
  end if
  $I' \leftarrow \{m \in I : s_{g(m)}(t) \neq \text{Low}\}$; **if** $I' = \emptyset$ **then** $I' \leftarrow I$ {stock-filtered pool}
  $I_{\text{feasible}} \leftarrow \{m \in I' : t + \overline{W}(\kappa_j, g(m)) \leq d_j\}$ {deadline-feasible nodes}
  if $I_{\text{feasible}} \neq \emptyset$ **then**
    $m^* \leftarrow \arg\min_{m \in I_{\text{feasible}}} p_{g(m)}$ {cheapest per-second rate, ties by exec time}
  else
    $m^* \leftarrow \arg\min_{m \in I'} \overline{W}(\kappa_j, g(m))$ {fastest fallback in the stock-filtered pool}
  end if
  Assign $j \rightarrow m^*$; remove m^* from I
end for

Algorithm 5 summarises the full procedure; the implementation is in `CGRSP/cgrsp/schedulers/cost_aware_deadline_risk.py`. The critical-ratio threshold $\text{CR}^* = 3$ is the baseline; its sensitivity is examined in Section 5.5.5.

4.5.12 Adaptive: Queue-Pressure EDF-SPT Hybrid

The Adaptive scheduler is a third hybrid policy, evaluated alongside RH and CADR to test the alternative "adaptive SPT-EDF hybrid" design. Like SPT+Rescue it uses a laxity

threshold to separate urgent from non-urgent jobs, but it adds two refinements: a *hopeless* tier that demotes already-doomed jobs (as in RH and CADR), and a *queue-pressure* rule that widens the urgent tier as the backlog grows, so the policy interpolates between SPT under light load and EDF under heavy load.

At each scheduling event the laxity of job j is computed exactly as in SPT+Rescue (Eq. 4.3), $L_j(t) = d_j - t - \min_{m \in I} \overline{W}(\kappa_j, g(m))$, and the queue is partitioned into three tiers against an *effective* threshold θ_{eff}:

- **Critical** ($0 \leq L_j < \theta_{\text{eff}}$): dispatched first, in earliest-deadline (EDF) order.
- **Safe** ($L_j \geq \theta_{\text{eff}}$): dispatched next, in shortest-processing-time (SPT) order.
- **Hopeless** ($L_j < 0$): a miss is already unavoidable; dispatched last (EDF order within the tier) so these jobs do not displace savable work.

The queue-pressure adaptation sets $\theta_{\text{eff}} = \theta$ (baseline $600\,\text{s}$) under normal load, but expands it to the full loose-deadline window ($\theta_{\text{eff}} = T_{\text{loose}} = 28{,}800\,\text{s}$, a fixed constant in the implementation) whenever the waiting queue exceeds a pressure threshold P (baseline 10 jobs). Under that expansion every job with positive laxity falls into the critical tier, so the policy reduces to EDF ordering with hopeless-demotion exactly when the system is congested, the regime in which deadline ordering is most valuable; when the backlog is short it behaves as SPT for the safe majority. Critical-tier jobs are placed on the fastest deadline-reliable node (with a stock and provisioning-reliability penalty), while safe-tier jobs use the same speed-weighted cost-aware node score as SPT (Eq. 4.1).

Algorithm 6 Adaptive EDF-SPT Hybrid Scheduler

Require: Queue Q; idle node set I; current time t; threshold θ; pressure P

$\theta_{\text{eff}} \leftarrow T_{\text{loose}}$ **if** $|Q| > P$ **else** θ {queue-pressure widening}
for each job $j \in Q$ **do**
    $e_j \leftarrow \min_{m \in I} \overline{W}(\kappa_j, g(m))$; $L_j \leftarrow d_j - t - e_j$
end for
$Q_{\text{crit}} \leftarrow \{j : 0 \leq L_j < \theta_{\text{eff}}\}$ sorted by d_j (EDF)
$Q_{\text{safe}} \leftarrow \{j : L_j \geq \theta_{\text{eff}}\}$ sorted by e_j (SPT)
$Q_{\text{hope}} \leftarrow \{j : L_j < 0\}$ sorted by d_j, ordered last
for each job j in $Q_{\text{crit}} \,\|\, Q_{\text{safe}} \,\|\, Q_{\text{hope}}$ **do**
    if $I = \emptyset$ **then**
        break
    end if
    $m^* \leftarrow$ fastest reliable node if j critical (exec time plus stock penalty $\{0, 50, 200\}$ s and per-type reliability prior {A4000 0, A4500 5, A5000 15, 3090 40} s), else speed-weighted cost-aware node (Eq. 4.1)
    Assign $j \rightarrow m^*$; remove m^* from I
end for

Algorithm 6 summarises the procedure; the implementation is in `CGRSP/cgrsp/schedulers/adaptive_scheduler.py`. It differs from RH in that its tier boundary is driven by an instantaneous laxity-and-queue-length rule rather than a forward-simulation timeline, and from CADR in that it uses an absolute laxity threshold rather than a scale-free critical ratio. Its empirical behaviour relative to RH and CADR is reported in Chapter 5.

4.6 Metrics Collection

The simulator records per-job metrics at the time of job completion:

- t_j^{submit}, t_j^{start}, t_j^{complete}: timestamps in seconds

- $w_j = t_j^{\text{start}} - t_j^{\text{submit}}$: waiting time (Eq. 3.11)
- $r_j = t_j^{\text{complete}} - t_j^{\text{submit}}$: response time (Eq. 3.12), tracked as a diagnostic; Chapter 5 reports it through its two components rather than as a separate table column, since r_j differs from the tabulated waiting time w_j only by the realised execution time, whose per-policy differences enter through the cost metric
- $\delta_j = \mathbf{1}[t_j^{\text{complete}} \leq d_j]$: deadline adherence indicator (Eq. 3.14); the miss indicator is $1 - \delta_j$, and the miss rate is $1 - \frac{1}{|\mathcal{J}|}\sum_j \delta_j$
- $c_j = W(\kappa_j, g(m_j)) \cdot p_{g(m_j)}$: job execution cost (Eq. 3.15)

Fleet-level metrics (utilisation per node type, total idle time, total provisioning delay) are accumulated continuously. All metrics are reported at the end of the simulation via `get_metrics_summary()`, which returns a structured dictionary used both for JSON export and for the PDF report generator.

The verification half of the V&V framework adopted in Chapter 2 is executed rather than merely asserted. A constraint verifier runs as part of the main experiment and rechecks each completed run against three model constraints, recomputing each one from the recorded per-job data rather than trusting the accumulators that produced the summary: that every job's deadline-adherence flag agrees with its own recorded completion time and deadline, and that the reported miss count agrees with those flags; that each job is assigned to at most one node and each node runs at most one job at a time; and that each job's cost equals $W(\kappa_j, g(m_j)) \cdot p_{g(m_j)}$. The verifier writes its findings to a `verification` block in the run's `report.json`, which records all three constraints as checked and satisfied, with zero violations, for every scheduler in the run.

Chapter 5 Experimental Results and Analysis

5.1 Experimental Setup

5.1.1 Platform and Data

All experiments are conducted using the CGRSP simulator with per-stratum mean execution times from RunPod Secure Cloud derived from 6,627 real rendering jobs collected from the platform between July 2025 and June 2026. Job timestamps are recorded in UTC by the originating AWS Lambda handler (verified against live CloudWatch logs) and are converted to Taiwan time (UTC+8) throughout this thesis for readability. The jobs span four GPU types (RTX 3090: 1,119 jobs, RTX A5000: 4,916 jobs, RTX A4000: 257 jobs, RTX A4500: 335 jobs), three scene complexity levels, and a wide range of output resolutions (from small preview renders of a few thousand pixels to 48-megapixel stills). In the raw dataset, execution times range from 30 seconds to 352 seconds, with an aggregate coefficient of variation of ≈0.28 across the whole dataset and a smaller within-stratum dispersion (pooled $\sigma_{\ln} \approx 0.11$) once complexity and GPU type are fixed. Because the production logs record no scene metadata (polygon counts, materials), the complexity level of each job in the calibration dataset was assigned by render-time terciles of the empirical distribution; the simulator then treats κ_j as an observable job attribute, corresponding to a deployment in which the studio classifies complexity from scene statistics known before rendering.

The realised execution time of job j on GPU type g is modelled as a log-normal random variable conditioned on the job's complexity level $\kappa_j \in \{1, 2, 3\}$ and the GPU type g:

$$W(j,g) \sim \text{LogNormal}\left(\mu_{\kappa_j,g},\, \sigma^2_{\kappa_j,g}\right), \qquad \mu_{\kappa,g} = \ln \overline{W}(\kappa, g) - \tfrac{1}{2}\sigma^2_{\kappa,g}, \tag{5.1}$$

where $\overline{W}(\kappa, g)$ is the empirical mean execution time of the (κ, g) stratum in the 6,627-job dataset (Table 4.1). The location parameter $\mu_{\kappa,g}$ is centred so that $\mathbb{E}[W(j, g)] = \overline{W}(\kappa, g)$; consequently the mean execution times reported throughout this thesis and used in the numerical example (Section 3.5) are the expected values $\mathbb{E}[W(\kappa, g)]$. The log-normal form is appropriate because render times are strictly positive and right-skewed. The dispersion $\sigma_{\kappa,g}$ is fitted per stratum as the standard deviation of ln(render time) within that stratum, ranging from 0.03 for the most regular medium-complexity strata to 0.23 for the most variable high-complexity stratum, with an n-weighted pooled within-stratum value of $\sigma_{\ln} \approx 0.11$ (a within-stratum coefficient of variation of about 0.11); strata with fewer than five observations use the pooled value. (The larger aggregate coefficient of variation of ≈ 0.28 measured across the whole dataset is dominated by between-complexity differences in the mean and is therefore not the relevant conditional dispersion.) Schedulers plan with the deterministic estimate $\overline{W}(\kappa_j, g)$, that is, the expected service time e_j; only the realised completion time is stochastic, mirroring real operation where the dispatcher cannot observe a job's exact walltime in advance. Because render walltimes (tens to a few hundred seconds) are small relative to deadline windows (T_{tight} to T_{loose}) and to queueing delays under saturation, this service-time stochasticity has only a minor effect on the primary saturated-load metrics: the 30-seed Phase 3 hectic-day means change by under 2% relative to a mean-based run. The effect can be slightly larger in lightly loaded, few-seed sub-experiments (for example, the surge-day deadline-tightness sweep), where outcomes near a crossover are more sensitive to the realised service times.

Operational costs are based on RunPod Secure Cloud pricing from a snapshot taken on 10 March 2026, treated as fixed deterministic constants throughout all simulations (RunPod prices change over time; these values represent a point-in-time snapshot): RTX A5000 \$0.27/hr, RTX 3090 \$0.46/hr, RTX A4500 \$0.25/hr, RTX A4000 \$0.25/hr. The RTX 3090 is the most expensive GPU in the fleet at \$0.46/hr, approximately 70% more expensive than the A5000, yet is the second-fastest on average (78.1 s vs 74.0 s for A5000); this cost-speed trade-off is the primary driver of the cost-aware scheduler designs (RH, CADR, Balanced). The A4500 and A4000 share the same hourly rate (\$0.25/hr) at this snapshot date, so cost differentiation between those two types is absent; the scheduler can exploit the speed difference between them without any cost penalty.

Figure 5.1 summarises the empirical dataset. The arrival pattern (panel a) shows a pronounced peak between 14:00 and 17:00 Taiwan time, consistent with afternoon rendering bursts; the four day-type arrival rates (λ) are calibrated to this pattern. The execution-time distributions (panel b) confirm right-skewed, GPU-dependent walltimes with medians ranging from 68 s (RTX A5000) to 87 s (RTX A4000), motivating the per-stratum log-normal service model.

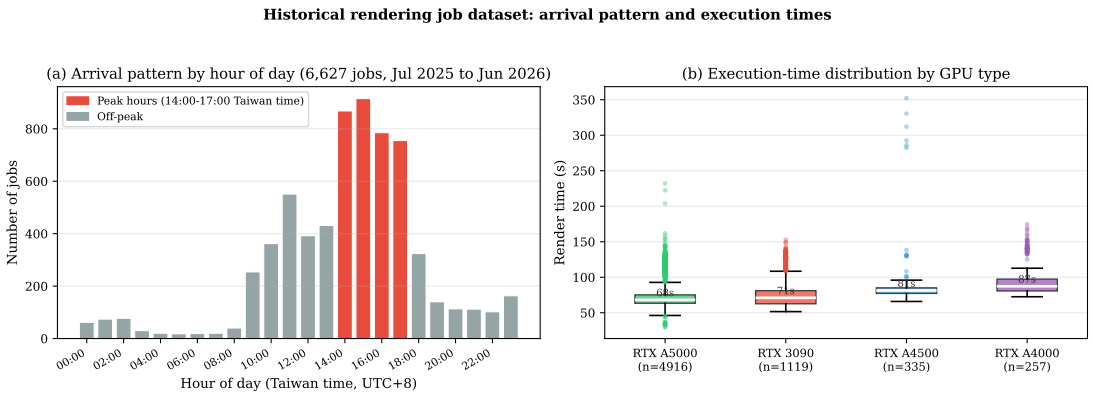

Figure 5.1: Empirical rendering job dataset (6,627 jobs, July 2025 to June 2026). (a) Hourly arrival counts: the 14:00–17:00 Taiwan-time peak (red) drives the hectic and surge day-type calibration; off-peak hours are shown in grey. (b) Execution-time box plots per GPU type (IQR box, median white line, whiskers at 1.5×IQR, outliers as dots): the RTX A5000 is fastest (median 68 s) and the RTX A4000 slowest (median 87 s), with right skew in all strata justifying the log-normal service model.

The simulation environment parameters are:

- Simulation duration: 86,400 seconds (24 hours) per run
- Concurrency limit: $C_{\max} = 5$ (RunPod default account limit [30], maximum simultaneous GPU instances; fixed platform constraint)
- Job arrival: Poisson process with day-type-calibrated burst rate λ and total job count N_{jobs}, a standard queueing-theory assumption representing independent submissions with memoryless inter-arrival times. These are independent parameters: the simulator generates exactly N_{jobs} jobs with inter-arrival times $\sim \text{Exp}(\lambda)$, so the active arrival window is approximately N_{jobs}/λ seconds; the job count is N_{jobs}, not $\lambda \times 86{,}400$:
  - Quiet: $\lambda = 0.0008$ j/s, 6 jobs/day (arrival window ≈2.1 hr)
  - Normal: $\lambda = 0.002$ j/s, ∼100 jobs/day (arrival window ≈13.9 hr)

  - Hectic: $\lambda = 0.100$ j/s, ∼950 jobs/day (arrival window ≈2.6 hr; capacity saturation $\rho > 1$)
  - Surge: $\lambda = 0.020$ j/s, 730 jobs/day (arrival window ≈10.1 hr)

- Deadline: heterogeneous distribution: 20% of jobs have a tight 1-hour deadline ($T_{\text{tight}} = 3{,}600$ s, rush orders) and 80% have a loose 8-hour deadline ($T_{\text{loose}} = 28{,}800$ s, standard renders). Objective weights: $\alpha_1 = 1$ (penalty per second of wait) and $\alpha_2 = 10$ (penalty per missed deadline); the ratio $\alpha_2/\alpha_1 = 10$ is swept in Section 5.5.9. This split is a modelling assumption reflecting a rendering studio context in which a minority of frames are time-critical (client delivery, real-time preview) while the majority are standard batch renders with flexible delivery windows. A uniform deadline distribution would fail to reveal the starvation-deadline trade-off between SPT and SPT+Rescue. The sensitivity of all results to this assumption is assessed explicitly in Section 5.5.2, which sweeps the tight-deadline fraction from 10% to 90%.

- Availability: each rental request incurs a stock-status-dependent provisioning delay τ_j^{prov} (formalised in Section 3.3, Eq. 3.8). The studied platform is an on-demand Secure Cloud pool: every slot is rentable at any time and a running instance is never reclaimed by the provider, so market scarcity does not manifest as denial of service but purely as this provisioning delay. The availability model is informed by RunPod pricing-API snapshots collected between 10 March and 20 April 2026 (5,294 raw snapshots; 77 daily distribution summaries spanning that same 10 March–20 April collection window were used during calibration): the snapshots supply the secure-cloud hourly prices, the qualitative stock-status vocabulary (High/Medium/Low), and the diurnal demand shape. Snapshot timestamps are tagged UTC by the scraper's own code, but the scraping machine's system clock was in fact running on GMT+7; snapshot hours are therefore shifted by $+1$ h to Taiwan time (UTC+8) throughout this thesis, consistent with the job-arrival timestamps (which are genuine UTC, shifted by $+8$ h). The per-type baseline high-stock probabilities β_g (RTX 3090 50%, RTX A5000 65%, RTX A4500 70%, RTX A4000 75%) are *modelling assumptions* for the on-demand secure pool rather than empirical frequencies from the snapshot record: the stock-status field published by the pricing API refers to the shared spot

marketplace pool, in which the four consumer-tier fleet types are chronically low-stock or unlisted (Figure 5.3), and it does not directly measure the probability that an on-demand secure rental is fulfilled promptly, which is what β_g models. The time-of-day multiplier $\phi(t)$ scales the high-stock probability and is expressed on the simulator's own local clock, which is Taiwan time throughout this project: off-peak, 00:00–06:00 ($\times$1.3); morning, 06:00–12:00 ($\times$0.9); peak, 12:00–18:00 ($\times$0.5); evening, 18:00–24:00 ($\times$1.0); the observed record supports this shape qualitatively, with marketplace availability of the fleet types visibly worse during the business-hours peak (Figure 5.3b). Simulation time zero is anchored to 08:00 on the modelled day, matching a studio working day that opens in the morning rather than at midnight, so every run starts in the morning band; the offset is applied arithmetically rather than read from the host machine's clock, so the sequence of bands a run traverses does not depend on the timezone it happens to execute under. Outside the peak band the multiplier keeps $p_g^H + p_g^M \geq 1$ for all four fleet types, so the Low state is then unreachable; only during the peak band (multiplier $\times 0.5$) does it become reachable at all, and there for all four types, with per-draw probability largest for the low-β_g types (37.5% for the RTX 3090 down to 6.25% for the RTX A4000). Because the peak occupies a minority of the day, the time-weighted Low probability is small and the long-delay Low regime is rarely entered in the calibrated runs: realised provisioning delays remain in the High and Medium ranges (at most two minutes) for the overwhelming majority of dispatches. Provisioning delays are drawn uniformly from stock-status ranges: High 0–10 s, Medium 30–120 s, Low 600–7200 s. (Implementation note: the status the scheduler observes and the status realised at acquisition are two independent draws from the same time-varying model, so the observation behaves like a marketplace API poll moments before renting rather than a binding quote.) Figure 5.2 shows the $\phi(t)$ step function alongside the real job arrival pattern; the 77 calibration snapshots are concentrated at hours 02:00–03:00 and 19:00–20:00 Taiwan time, so $\phi(t)$ is a parametric calibration rather than a pointwise empirical fit.

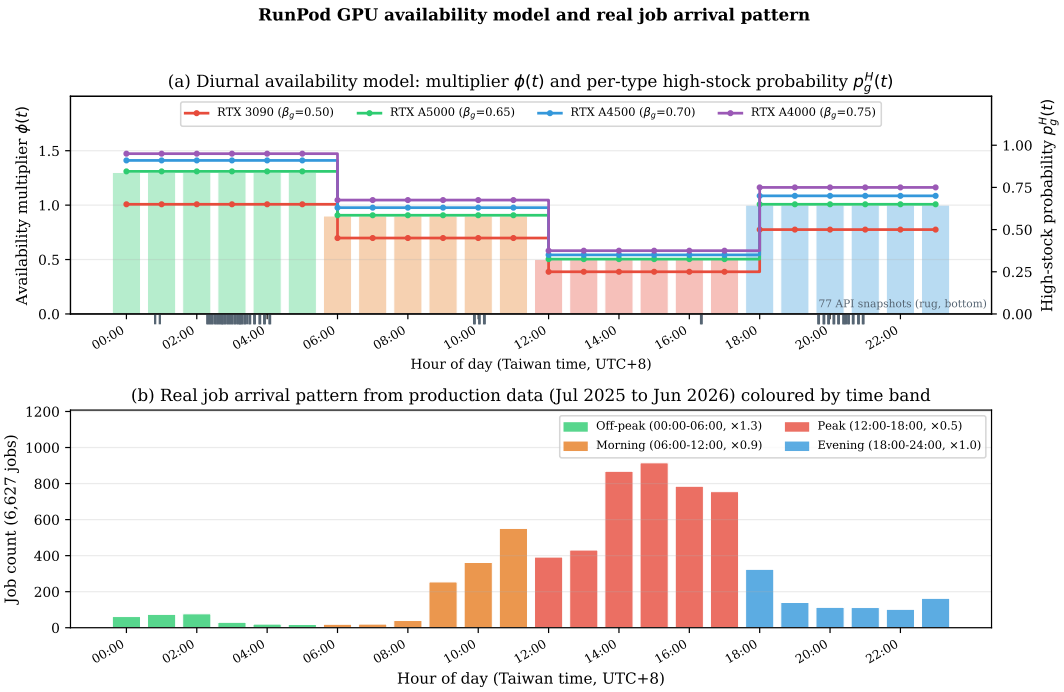

Figure 5.2: Diurnal availability model and real job arrival pattern, all times Taiwan time (UTC+8), matching the simulator's own local clock (Section 3.3). (a) Bars (left axis) show the parametric step function $\phi(t)$: off-peak 00:00–06:00 raises availability ($\times$1.3, green); morning 06:00–12:00 moderately reduces it ($\times$0.9, orange); peak 12:00–18:00 reduces availability by half ($\times$0.5, red); evening 18:00–24:00 is neutral ($\times$1.0, blue). Lines (right axis) show the resulting per-type high-stock probability $p_g^H(t) = \min(0.95,\ \beta_g\,\phi(t))$ for each of the four fleet GPUs, using each type's baseline β_g (RTX 3090 50%, RTX A5000 65%, RTX A4500 70%, RTX A4000 75%): the four curves share $\phi(t)$'s step shape but sit at different levels set by β_g, and the RTX A4000 curve flattens against the 0.95 ceiling during the off-peak band ($0.75 \times 1.3 = 0.975 \rightarrow 0.95$). Rug marks at the bottom show the 77 RunPod API snapshot times, which are concentrated in two narrow windows. (b) Actual hourly job count from 6,627 production rendering jobs (July 2025 to June 2026), coloured by the corresponding $\phi(t)$ band. The historical arrival peak at 14:00–17:00 Taiwan time falls inside the peak band, so the busiest hours of the production record coincide with the model's lowest-availability window. The simulated day is anchored differently: simulation time zero corresponds to 08:00 Taiwan time (Section 5.1.1), so a run opens in the morning band ($\times$0.9), and the hectic day's arrival window of roughly 2.6 hours lies wholly inside it; only the longer surge and normal arrival windows extend into the afternoon peak at all. Panel (b) therefore describes when the studio historically submitted work, not the availability band in which the saturated scheduler comparison is run.

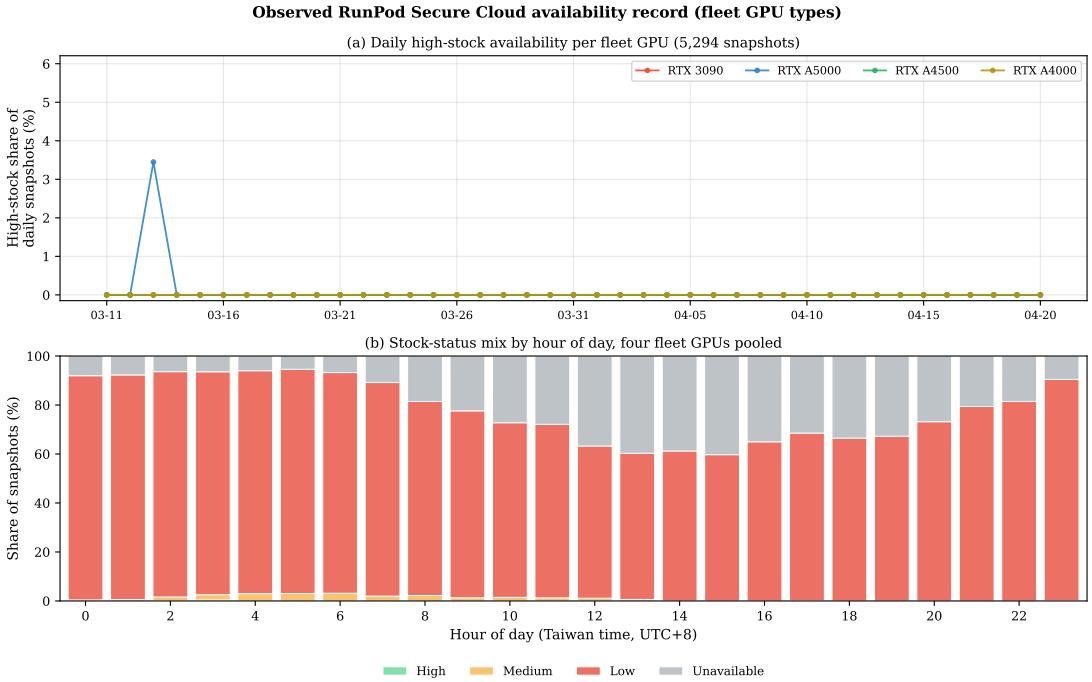

Figure 5.3: Observed RunPod availability record for the four fleet GPU types, 10 March to 20 April 2026 (5,294 pricing-API snapshots). (a) Daily share of snapshots reporting High stock: essentially zero throughout the collection window. (b) Stock-status mix by hour of day (fleet types pooled): the spot marketplace pool for these consumer-tier types is chronically Low-stock, and the share of snapshots in which a type is unlisted entirely rises during the business-hours window, qualitatively consistent with the peak-demand reduction encoded in $\phi(t)$. Because this published stock-status field tracks the spot marketplace rather than on-demand secure-rental fulfilment, the baseline high-stock probabilities β_g are treated as modelling assumptions for the on-demand pool; the snapshot record contributes the fleet prices, the stock-status vocabulary, and the diurnal demand shape.

5.1.2 Evaluation Phases

Four phases of evaluation are conducted (phases are numbered from 2; Phase 1 comprised initial exploratory calibration runs not reported here):

Phase 2 Single-seed benchmark (seed=42) on the hectic-day configuration of Section 5.1.1. Provides illustrative point estimates for the seven non-hybrid baseline schedulers under capacity saturation ($\rho > 1$; RH, CADR, and the Adaptive hybrid enter at Phase 3); being single-seed it carries no confidence interval and is superseded by the $n = 30$ Phase 3 statistics for every inferential claim.

Phase 3 Statistical 30 independent replications (seeds 0 to 29) on hectic-day workload ($\sim$950 jobs/day). Computes 95% confidence intervals and paired t-tests (with a Wilcoxon signed-rank backup) and Cohen's d for pairwise comparisons. Includes RH, CADR, and the CADR-order-only (CADR-OO) ablation variant. Tardiness (mean minutes past deadline) is reported as a diagnostic alongside miss rate.

Phase 3 Sensitivity Compares all schedulers across four operationally calibrated day types (quiet/normal/hectic/surge) at $C_{\text{max}} = 5$, with 30 seeds per day type (1,200 total runs). Identifies the load regime where scheduler differences become operationally significant.

Phase 4 30-day monthly simulation (10 replications, empirically calibrated day-type distribution: quiet/normal/hectic/surge). Aggregates scheduler performance across monthly workload variation to assess operational readiness. Ten replications rather than 30 were used because each replication simulates 30 days of continuous operation; 10 replications yield 95% CIs with $t_9^* = 2.262$, providing adequate precision for the monthly comparison objective.

A **deadline tightness sensitivity** study sweeps tight-deadline fraction $f_{\text{tight}} \in \{10\%, 30\%, 50\%, 70\%, 90\%\}$ under surge-day conditions ($C_{\text{max}} = 5$, $\lambda = 0.020$ j/s, 730 jobs/day, 30 seeds each) to identify the load regime where EDF's deadline-safety advantage over SPT is maximised.

5.1.3 Statistical Methodology

All statistical comparisons use a paired t-test on the per-seed averages, with a Wilcoxon signed-rank test [31] as a distribution-free backup. Every scheduler is evaluated on the same 30 seeds with identical generated jobs per seed, so the per-seed observations are matched (a randomised block design); the paired test is therefore both valid and more powerful than an unpaired alternative, as it removes the between-seed variance that the schedulers share. Effect sizes are reported as Cohen's $d = (\mu_1 - \mu_2)/s_{\text{pooled}}$, where s_{pooled} is the pooled standard deviation. The threshold $|d| \geq 0.8$ is used as the criterion for a practically meaningful effect ("large" by Cohen's convention [32]). Confidence intervals

are computed as $\bar{x} \pm t^*_{n-1} \cdot (s/\sqrt{n})$ with $t^*_{29} = 2.045$ for $n = 30$, $\alpha = 0.05$. The choice of $n = 30$ independent replications satisfies the standard requirement for the central limit theorem to yield approximately normal sampling distributions of per-seed means, enabling valid parametric inference [28]. Consistent with the lexicographic objective (Eq. 3.1), which prioritises QoS over operational cost rather than collapsing them into one scalar, all experimental comparisons report each component separately (average waiting time, deadline miss rate, operational cost) rather than as an aggregate. Decomposed reporting enables direct identification of which dimension drives scheduler differences: a scheduler that reduces miss rate by increasing waiting time would appear identically weighted in a scalar objective but the trade-off is transparent in the decomposed view. Each pairwise comparison in Table 5.3 tests a pre-specified directional hypothesis grounded in scheduling theory (e.g. SPT reduces wait by shortest-job-first ordering; EDF reduces miss by deadline ordering) rather than a data-driven hypothesis selection; a Holm-Bonferroni step-down correction [33] over the full pairwise family is nonetheless applied to control the family-wise error rate; as reported below, most but not all comparisons significant at the uncorrected level survive it.

5.2 Phase 2: Seven-Scheduler Benchmark

Table 5.1 and Figure 5.4 report Phase 2 results for all seven schedulers on a hectic day; Figure 5.4 additionally contrasts a normal day (top row), where all schedulers achieve near-zero miss, to show that the saturated hectic day is the regime in which scheduler choice matters.

Four observations follow from Phase 2, though as a single seed it carries no confidence interval and several of the following patterns do not survive to the robust $n = 30$ Phase 3 mean; that comparison is made explicit below. First, in this seed SPT+Rescue achieves both the lowest waiting time (137.49 min) and the lowest miss rate (9.37%) of the seven baseline schedulers: the laxity rescue mechanism performs very well on this particular seed. This is not the pattern that holds at $n = 30$ (Section 5.3), where SPT and RH are statistically tied for the lowest wait and SPT+Rescue trails both; the divergence

Table 5.1: Phase 2 benchmark: hectic day (∼950 jobs/day), $C_{\max} = 5$, seed=42, $\lambda = 0.100$ j/s, 20% tight 1h / 80% loose 8h deadlines (capacity saturation: $\rho > 1$). Util is event-sampled (mean of per-event fleet-occupancy samples over the active window), not a time-weighted day average.

Scheduler	Avg Wait (min)	Miss %	Cost ($)	Util (%)	Completed
SPT+Rescue	**137.49**	**9.37**	6.33	95.7	950/950
SPT	141.82	16.21	6.48	95.7	950/950
EDF	177.78	18.32	6.56	95.7	950/950
LCF	172.33	21.47	6.36	95.7	950/950
Balanced	172.33	21.47	6.36	95.7	950/950
FIFO	184.65	28.00	6.35	95.7	950/950
Random	178.76	25.05	6.58	95.7	950/950

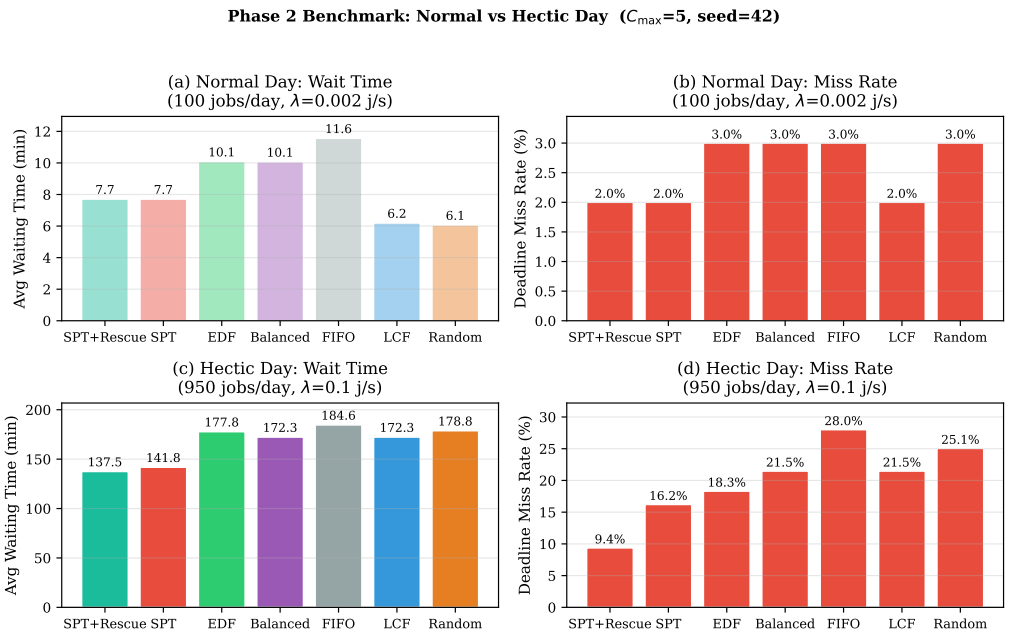

Figure 5.4: Phase 2 benchmark results ($C_{\max} = 5$, seed=42, 20% tight 1h / 80% loose 8h deadlines). Top row: a normal day (∼100 jobs/day, $\rho < 1$), where all schedulers achieve near-zero miss and wait under about 12 minutes, confirming that scheduler choice is immaterial under light load. Bottom row: the hectic day (∼950 jobs/day, $\lambda = 0.100$ j/s; capacity saturation), the discriminating regime analysed below. In this seed SPT+Rescue achieves both the lowest wait (137.49 min) and the lowest miss (9.37%) of the seven baseline schedulers; SPT is a close second on both (141.82 min wait, 16.21% miss); EDF's miss (18.32%) is markedly higher here than its 30-seed Phase 3 mean, illustrating single-seed variance under saturation; FIFO, LCF, and Balanced ignore deadlines, and FIFO incurs both the highest wait and the highest miss of all seven schedulers in this seed (28.00%).

illustrates why Phase 3's 30-seed replication, not this single seed, is the basis for every inferential claim in this thesis. Second, SPT itself is a close second on both metrics in this seed (141.82 min wait, 16.21% miss), consistent with shortest-job-first ordering benefiting both metrics simultaneously under saturation. Third, EDF's miss rate in this seed (18.32%) is markedly higher than its Phase 3 mean of 15.81% ($n = 30$), reflecting high seed-to-seed variance under saturation, so the Phase 3 result provides the reliable cross-seed estimate rather than this seed's unfavourable draw. Fourth, FIFO processes jobs without deadline awareness and incurs both the highest wait and the highest miss rate of all seven schedulers in this seed (28.00%): under saturation, a large fraction of tight-deadline jobs fail to complete within 1 hour. Across all policies, the schedulers achieve 95.7% event-sampled concurrency utilisation and 100% job completion (all 950 jobs dispatched); the utilisation figure is the mean of per-event samples, which concentrate on the busy active window, and is not a time-weighted average over the full 24-hour day, most of which lies after the backlog drains.

5.3 Phase 3 Statistical Validation

Table 5.2 reports Phase 3 results over $n = 30$ seeds on hectic-day workload (configuration in Section 5.1.1), with 95% confidence intervals. Figure 5.5 plots the same intervals graphically, making the statistical groupings directly visible: which scheduler pairs overlap (and are therefore statistically indistinguishable) and which are cleanly separated.

Figure 5.6 complements the interval view with the full per-seed distributions, showing that the group separations hold across individual seeds rather than only in the aggregate. Table 5.3 reports the pairwise paired t-test results.

Figure 5.7 presents the same comparisons graphically. The p-values are computed from the exact Student t-distribution for the paired differences (df $= n-1$), so they match an independent recomputation (`scipy.stats.ttest_rel`); a Wilcoxon signed-rank test agrees with every significant result, confirming the conclusions are not artefacts of the normality assumption. The full pairwise family comprises 24 comparisons, of which Table 5.3 prints the pre-specified subset; the Holm-Bonferroni step-down correction is applied over

Table 5.2: Phase 3 statistical validation: $n = 30$ seeds, hectic day ($C_{\max} = 5$, ~950 jobs/day, $\lambda = 0.100$ j/s, 20% tight 1h / 80% loose 8h deadlines; capacity saturation $\rho > 1$)

Scheduler	Avg Wait (min)	95% CI	Miss %	95% CI	Avg Tardiness (min)	Cost ($)
RH	**140.24**	[136.28, 144.20]	**9.50**	[8.33, 10.66]	30.86	6.48
CADR	145.70	[141.83, 149.56]	10.09	[8.44, 11.73]	6.43	6.47
SPT+Rescue	145.85	[142.40, 149.29]	12.50	[10.83, 14.17]	6.64	6.49
Adaptive	169.64	[166.49, 172.79]	12.49	[11.32, 13.66]	4.45	6.48
SPT	141.03	[138.28, 143.78]	18.60	[17.44, 19.76]	11.99	6.45
EDF	174.16	[170.66, 177.65]	15.81	[14.18, 17.45]	5.74	6.48
LCF	173.36	[170.50, 176.23]	26.14	[25.26, 27.01]	28.50	6.53
Balanced	173.24	[169.98, 176.51]	26.39	[25.49, 27.30]	28.30	6.52
FIFO	169.32	[165.29, 173.35]	24.85	[23.44, 26.25]	26.69	6.50
Random	172.80	[169.03, 176.56]	24.04	[23.05, 25.04]	35.18	6.50

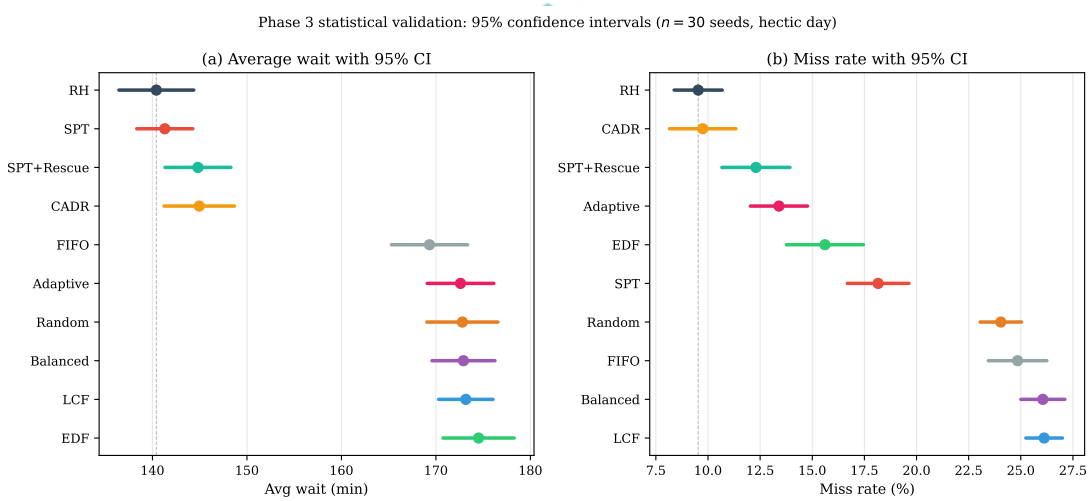

Figure 5.5: Phase 3 confidence interval plot: 95% CIs for average wait (a) and miss rate (b) for all ten schedulers, $n = 30$ seeds, hectic day. Schedulers are ordered by mean value within each panel. These are marginal intervals summarising each scheduler's own across-seed variability, so they describe the spread of the ten distributions and are not themselves a test of any pair. Because every scheduler is run on the same 30 seeds (a randomised block design, Section 5.1.3), the paired tests of Table 5.3 are the authoritative verdict, and overlap here need not agree with them in either direction: EDF's and FIFO's wait intervals overlap although that pair separates at the uncorrected level, and EDF's and SPT's miss intervals overlap by a hair although the paired test separates them clearly ($p = 0.002$). Key groupings: RH and CADR are statistically tied for the lowest miss rate (paired $t = -0.710$, $p = 0.484$); SPT and RH are tied for the lowest wait (paired $t = -0.429$, $p = 0.671$); EDF achieves significantly lower miss than SPT despite much higher wait, underscoring that deadline-awareness and throughput are separate design levers.

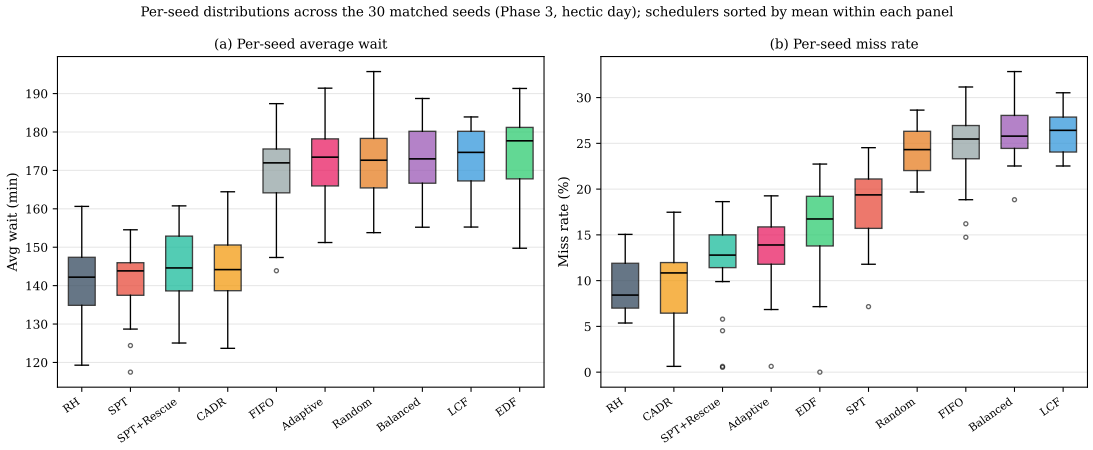

Figure 5.6: Per-seed distributions behind the Phase 3 means: box plots of average wait (a) and miss rate (b) across the 30 matched seeds, schedulers sorted by mean within each panel. The box spans the interquartile range with the median line; whiskers extend to 1.5×IQR and dots mark outlier seeds. The clear vertical separation between the deadline-aware and deadline-oblivious groups in panel (b) holds seed by seed, not just on average, which is what the paired tests in Table 5.3 formalise.

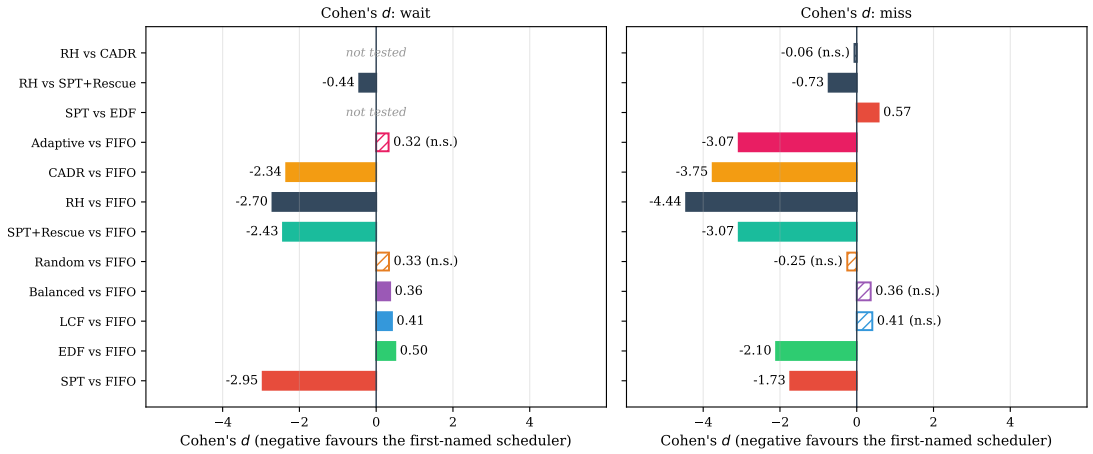

Figure 5.7: Effect sizes for the whole pairwise family, of which Table 5.3 prints the pre-specified subset: Cohen's d for average wait (left) and miss rate (right). Negative bars favour the first-named scheduler; hatched bars are not significant at the raw, uncorrected $p < 0.05$ level, and pairs without a pre-specified hypothesis for a metric are marked "not tested". The headline tie is directly visible: RH vs CADR on miss is non-significant, while RH vs FIFO shows the largest miss effect in the study. The shading is the uncorrected verdict, so six bars render solid without surviving the Holm-Bonferroni family-wise correction: RH's wait advantage over SPT+Rescue ($d = -0.564$, raw $p = 0.005$), EDF's higher wait than FIFO ($d = 0.479$, raw $p = 0.023$), and four LCF and Balanced comparisons against FIFO that Table 5.3 does not print and this thesis makes no claim about. Defer to the table, not this figure, for every corrected verdict.

Table 5.3: Pairwise paired t-test results, Phase 3 ($n = 30$, $C_{\max} = 5$, hectic day ∼950 jobs/day, 20% tight 1h / 80% loose 8h deadlines). Each comparison is matched on seed; a Wilcoxon signed-rank test agrees with every entry. "Significant?" reflects the Holm-Bonferroni-corrected verdict at family-wise $\alpha = 0.05$ (Section 5.1.3), not the raw uncorrected p-value; †marks the two comparisons whose raw $p < 0.05$ but which do not survive the correction.

Comparison	**Metric**	t	p-value	**Significant?**	Cohen's d
SPT vs FIFO	Avg wait	-14.523	< 0.001	Yes	-3.062
SPT vs FIFO	Miss rate	-7.611	< 0.001	Yes	-1.807
SPT+Rescue vs FIFO	Avg wait	-9.766	< 0.001	Yes	-2.337
SPT+Rescue vs FIFO	Miss rate	-11.472	< 0.001	Yes	-2.987
EDF vs FIFO	Avg wait	2.392	0.023	No†	0.479
EDF vs FIFO	Miss rate	-9.558	< 0.001	Yes	-2.211
Random vs FIFO	Miss rate	-1.289	0.208	No	-0.246
SPT vs EDF	Miss rate	$+3.501$	0.002	Yes	$+0.733$
RH vs FIFO	Avg wait	-12.933	< 0.001	Yes	-2.717
RH vs FIFO	Miss rate	-19.416	< 0.001	Yes	-4.435
RH vs SPT+Rescue	Avg wait	-3.055	0.005	No†	-0.564
RH vs SPT+Rescue	Miss rate	-3.392	0.002	Yes	-0.778
RH vs CADR	Miss rate	-0.710	0.484	No	-0.154

the whole family, not over the printed rows, to control the family-wise error rate. Of the 18 comparisons significant at the uncorrected 0.05 level, 12 remain significant after the correction (the largest surviving p-value is 0.002), so the correction removes six of them. The correction governs this pre-specified family and nothing else, and every comparison reported as significant from it is the corrected verdict: the two affected comparisons that Table 5.3 prints, RH's wait advantage over SPT+Rescue and EDF's higher wait than FIFO, are marked there with † rather than reported as headline findings; the remaining four, LCF's and Balanced's wait and miss differences against FIFO, are deadline-oblivious baselines outside the thesis's pre-specified hypothesis set, are not printed in Table 5.3, and no claim about any of them is made anywhere in this thesis. Tests reported elsewhere in this chapter are neither members of this family nor included in these counts, and are stated uncorrected. That applies in particular to the three significant tests drawn from the avail-

ability and prior sweeps (Section 5.5.8), each of which carries a p-value well below the smallest threshold a correction over a family of this size could impose, so no conclusion drawn from them turns on the distinction.

The statistical comparison supports the following conclusions.

SPT achieves the lowest wait among the baseline policies under capacity saturation, statistically tied with RH. Under hectic overload ($\sim$950 jobs, $\rho > 1$), SPT achieves the lowest baseline wait (RH's mean is marginally lower, and the paired test over the same 30 seeds cannot separate the two: $t = -0.429$, $p = 0.671$, $d = -0.086$) but is now significantly outperformed by EDF on miss (18.60% vs 15.81%, $d = 0.733$, $p = 0.002$): the two are no longer statistically tied on miss. The effect vs FIFO is large on both metrics.

SPT+Rescue achieves a notably lower miss rate than SPT (12.50% vs 18.60%) but at higher wait (145.85 min vs 141.03 min). Under full saturation ($\rho > 1$), the rescue tier reorders urgent jobs ahead of the normal SPT queue but cannot prevent all misses caused by the $C_{\text{max}} = 5$ concurrency ceiling; promoting urgent jobs displaces normal-tier jobs, adding wait while still reducing net miss relative to bare SPT. The rescue mechanism also reduces cross-seed miss variance, yielding a larger Cohen's d vs FIFO ($d = -2.987$) than bare SPT ($d = -1.807$). SPT+Rescue still provides a very large improvement over FIFO/LCF/Balanced on both metrics.

EDF now achieves significantly lower miss than SPT ($d = 0.733$, $p = 0.002$) and carries a higher wait than FIFO that clears the uncorrected threshold but not the family-wise one ($d = 0.479$, raw $p = 0.023$, marked † in Table 5.3): its deadline ordering helps on miss, while reordering the queue around deadlines rather than job length buys no wait advantage at all over the deadline-oblivious baseline. EDF's mean tardiness (5.74 min) remains low relative to the deadline-oblivious policies, though it no longer leads the deadline-aware group: Adaptive, SPT+Rescue, and CADR all now post lower tardiness (Section 5.4).

RH achieves the lowest miss rate under hectic saturation (9.50%) while its mean wait (140.24 min) is in the lowest tier, statistically tied with SPT (141.03 min; $t = -0.429$, $p = 0.671$) and numerically lower than SPT+Rescue (145.85 min). RH's miss advantage over SPT+Rescue survives Holm correction ($d = -0.778$, $p = 0.002$); its wait advantage,

while numerically favourable, has a raw $p = 0.005$ that does *not* survive the family-wise correction (Table 5.3), so RH is best described as significantly better than SPT+Rescue on miss and comparable to it on wait, rather than dominant on both metrics. Versus FIFO both gaps are large and significant (miss $d = -4.435$, wait $d = -2.717$, both $p < 0.001$), and RH's operational cost is within roughly 1% of SPT+Rescue (Section 5.5.9). RH's mean tardiness (30.86 min) is not only substantially higher than EDF's (5.74 min) but now exceeds even the deadline-oblivious FIFO baseline (26.69 min): its aggressive forward-simulation demotion of doomed jobs pushes sacrificed jobs far back in the queue, so under this workload the jobs RH misses are missed by more than FIFO's, even though RH misses far fewer jobs overall.

CADR achieves near-RH miss (10.09%) while cutting RH's mean tardiness by over 60% (6.43 min vs 30.86 min). CADR's critical-ratio triage is less aggressive than RH's forward-simulation demotion: doomed jobs are dispatched last but are not pushed as far past their deadlines as under RH. At Phase 3 hectic load, CADR achieves substantially lower miss than SPT+Rescue (10.09% vs 12.50%) at a wait the paired test cannot separate (145.70 min vs 145.85 min; $t = -0.092$, $p = 0.927$, $d = -0.015$) and comparable tardiness (6.43 min vs 6.64 min): CADR's advantage over SPT+Rescue is concentrated on miss rate rather than a joint improvement on both QoS metrics. RH and CADR are in fact statistically tied on miss rate ($t = -0.710$, $p = 0.484$ n.s.; RH 9.50% vs CADR 10.09%), so the choice between the two proposed schedulers turns on tardiness rather than on a significant miss rate difference: RH holds the lowest miss point estimate (and the fewest missed jobs overall), while CADR matches it at substantially lower lateness. Operators who care about the severity of misses, not only their count, therefore prefer CADR over RH; operators who care only about minimising the count of missed deadlines prefer RH.

The Adaptive queue-pressure EDF-SPT hybrid lands in the mid-tier (12.49% miss, 169.64 min wait, 4.45 min tardiness). Its queue-pressure rule widens the urgent tier to every positive-laxity job once the backlog exceeds the pressure threshold P, so under hectic saturation it reduces to EDF ordering with hopeless-demotion. This betters plain EDF (15.81%) and SPT (18.60%) on miss, though SPT+Rescue (12.50%) now edges ahead of it; its mean wait sits slightly above FIFO's rather than matching it, because EDF ordering

carries no throughput benefit. Adaptive is in fact dominated by CADR on miss and wait at comparable cost, though it holds the lowest mean tardiness of all ten policies (4.45 min), confirming that the adaptive SPT-EDF hybrid design is sound but the proposed RH and CADR policies are stronger on the primary QoS metrics.

Finally, FIFO, LCF, and Balanced incur the highest miss rates. These schedulers carry no deadline information; under saturation approximately one-third of tight-deadline jobs miss their 1-hour window.

5.4 Ablation Analysis

Table 5.4 traces the evolutionary chain FIFO $\rightarrow$ SPT $\rightarrow$ SPT+Rescue $\rightarrow$ RH on Phase 3 hectic-day results ($n = 30$ seeds), showing how each algorithmic addition changes waiting time, deadline miss rate, and mean tardiness relative to the previous step. EDF is included as an alternative single-criterion branch off FIFO (a deadline-driven sibling of the shortest-job-first SPT), isolating which design ingredient, shortest-job-first ordering or deadline awareness, drives each metric. CADR and the CADR-order-only (CADR-OO) ablation variant are included to show the contribution of the cost-aware placement component, and the Adaptive queue-pressure EDF-SPT hybrid is included as an alternative hybrid branch. Cohen's d values measure the effect size of each step: positive d indicates an increase (worse) relative to the comparison policy, negative d indicates a decrease (better). Figure 5.8 plots the three metrics across the chain, making the per-step trade-offs visible at a glance.

The ablation chain isolates the contribution of each design ingredient. The first step, SPT over FIFO, shows that shortest-job-first ordering produces a large reduction in waiting time (d vs FIFO from Table 5.3) and a statistically significant miss reduction, confirming that shortest-job-first ordering benefits both metrics simultaneously under saturation. The alternative branch, EDF over FIFO, shows that pure earliest-deadline ordering cuts the miss rate even more than SPT does ($d = -2.211$ vs $d = -1.807$) but leaves waiting time essentially unchanged ($d = 0.479$). Contrasting the EDF and SPT branches therefore isolates the two design ingredients: deadline awareness alone is sufficient to reduce misses

Table 5.4: Ablation analysis: evolutionary chain on Phase 3 hectic day ($n = 30$ seeds, $C_{\max} = 5$, ∼950 jobs/day). Cohen's d for SPT, EDF, RH, CADR, and Adaptive is relative to FIFO; for SPT+Rescue it is relative to SPT. The RH and CADR comparisons against SPT+Rescue are reported in Table 5.3 and the Phase 3 statistics (Section 5.3).

Scheduler	**Avg Wait (min)**	**Miss %**	**Avg Tardiness (min)**	**Cohen d (wait)**	**Cohen d (miss)**
FIFO	169.32	24.85	26.69	–	–
SPT	141.03	18.60	11.99	-3.062	-1.807
EDF	174.16	15.81	5.74	0.479	-2.211
SPT+Rescue	145.85	12.50	6.64	0.577	-1.584
RH (proposed)	140.24	9.50	30.86	-2.717	-4.435
CADR (proposed)	145.70	10.09	6.43	-2.233	-3.602
Adaptive (EDF-SPT hybrid)	169.64	12.49	4.45	0.033	-3.568
CADR-order-only	146.50	10.28	6.38	–	–

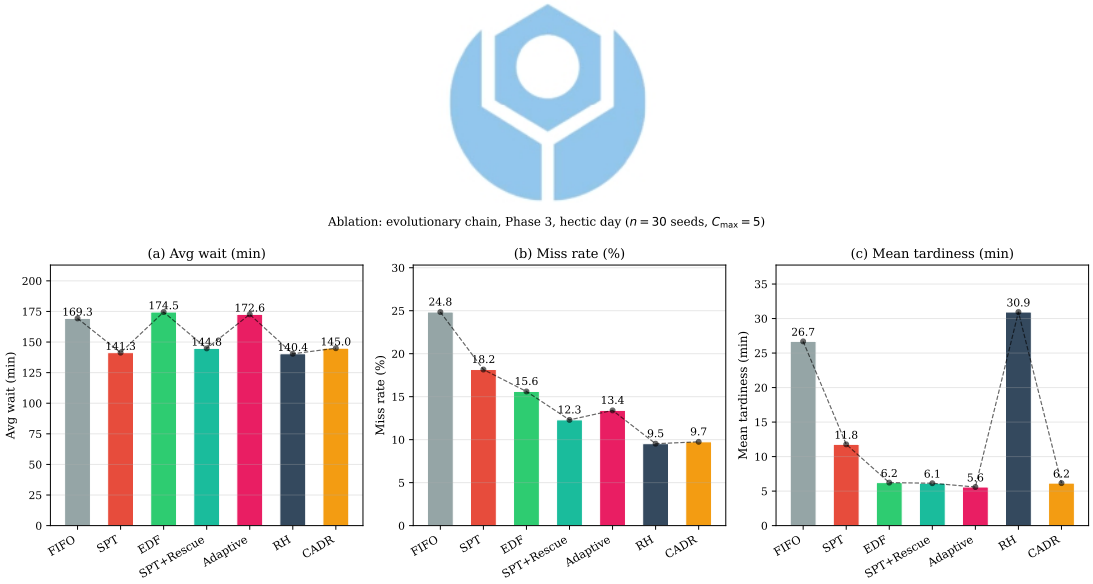

Figure 5.8: Ablation waterfall: average wait (a), miss rate (b), and mean tardiness (c) for each scheduler in the evolutionary chain. Each step along the chain adds one algorithmic ingredient. SPT improves both wait and miss over FIFO; EDF cuts miss and tardiness but leaves wait at FIFO level; SPT+Rescue recovers deadline awareness while retaining most of SPT's wait advantage; the Adaptive hybrid (plotted alongside the main chain) widens the rescue idea's urgent tier under queue pressure, cutting miss further at EDF-like tardiness; RH then improves both miss and wait over SPT+Rescue simultaneously, at the price of high tardiness on the jobs it sacrifices; CADR trades a marginal wait increase for a large tardiness reduction (30.86 to 6.43 min) at near-identical miss.

under saturation, whereas the wait-time benefit comes specifically from shortest-job-first ordering. This motivates the hybrid designs, which seek both effects at once. EDF's mean tardiness (5.74 min) remains low relative to the deadline-oblivious policies, because its deadline ordering ensures even missed jobs are missed by a small amount, though it no longer leads the deadline-aware group on this metric (Adaptive, SPT+Rescue, and CADR all post lower tardiness here).

The step from SPT to SPT+Rescue shows that the laxity rescue mechanism reduces miss rate variance across seeds (reflected in the Cohen d vs FIFO being larger for SPT+Rescue than for SPT) but increases mean waiting time because urgent-tier promotions displace short normal-tier jobs. The step from SPT+Rescue to RH improves *both* miss and wait (Table 5.3: $d = -0.564$ wait, $d = -0.778$ miss vs SPT+Rescue), the only hybrid step in this chain to advance both metrics at once (plain SPT also improves both relative to FIFO, but at a far higher miss level). Replacing the fixed laxity threshold with timeline-derived urgency restores SPT shortest-job-first ordering for jobs with slack, cutting wait back to SPT-level; simultaneously, hopeless-demotion (blocking already-doomed jobs from displacing savable ones) combined with precise forward-simulation urgency cuts miss below SPT+Rescue's fixed-threshold approach. However, RH's aggressive demotion of doomed jobs increases mean tardiness relative to SPT+Rescue (30.86 min vs 6.64 min): jobs that are pushed to the back are missed by substantially more.

CADR at Phase 3 hectic load achieves substantially lower miss than SPT+Rescue (10.09% vs 12.50%) at essentially the same wait (145.70 min vs 145.85 min) and comparable tardiness (6.43 min vs 6.64 min); the advantage over SPT+Rescue is concentrated on miss rate rather than a joint improvement on both metrics. CADR achieves near-RH miss (10.09% vs 9.50%) while cutting RH's mean tardiness substantially (6.43 min vs 30.86 min), because its critical-ratio demotion penalises already-at-risk jobs far less severely than RH's forward-simulation. RH retains the lowest miss count; CADR minimises miss count subject to bounded tardiness. These are complementary policies, not a linear "each step better" chain: the right choice depends on whether the operator cares more about the number of missed deadlines or the severity of the misses.

The CADR-order-only (CADR-OO) ablation retains the three-tier critical-ratio ordering but replaces cost-aware node selection with fastest-available-node placement, yielding 10.28% miss and 6.38 min tardiness, vs CADR's 10.09% and 6.43 min. The cost-aware placement component is thus a small refinement (10.28% vs 10.09% on miss, and 6.38 min vs 6.43 min on tardiness). This is expected: operational cost headroom is structurally small in this fleet (a marginal, near-noise gap across policies), so optimising node selection for cost has only a minor effect on QoS metrics.

Lastly, the Adaptive queue-pressure hybrid achieves lower miss (12.49%) than pure EDF (15.81%), though SPT+Rescue (12.50%) now edges ahead of it, with EDF-like low tardiness (4.45 min), confirming that widening the urgent tier under queue pressure helps over a fixed laxity threshold, if not as much as the forward-simulation or critical-ratio designs. It remains beaten, however, by both proposed schedulers on miss and on wait: RH (9.50% miss) and CADR (10.09% miss). Tardiness runs the other way, with Adaptive lowest of the ten (4.45 min). This isolates the value of the two novel ingredients: queue-pressure tier widening alone improves miss over SPT+Rescue, but the forward-simulation timeline (RH) and the scale-free critical-ratio triage with cost-aware placement (CADR) are what reach the lowest miss.

5.4.1 Capacity Lower Bound

To situate the Phase 3 miss rates against what any policy could achieve, two brackets are computed on the expected-service hectic instances (the same 30 job streams, each job granted its fastest GPU's mean execution time and zero provisioning delay). First, a certified window-capacity bound: for any window $[s, t]$, the jobs submitted at or after s with deadlines at or before t must fit their total work into $C_{\max}(t - s)$ machine-seconds, so any excess implies unavoidable misses under *every* policy, clairvoyant or not. Across all 30 seeds this bound certifies 0.00% unavoidable misses (maximum 0 jobs in any seed): the tight-deadline stream alone never exceeds fleet capacity, and the loose eight-hour windows spread the remaining demand far below it. Second, a constructive clairvoyant relaxation: non-preemptive EDF with full hindsight on the same relaxed instances schedules every job on time in every seed (0.00% miss). The certificate is one-sided and deterministic: it

bounds the expected-service instance, not individual stochastic realisations, whose provisioning and service draws can and do produce misses; that residual is precisely what the simulator measures. The offline-relaxed optimum is therefore exactly zero misses, and no reachable-optimum gap can be quoted against it; instead, the observed miss rates measure the *price of real operation*: online arrivals (Section 1.2), non-preemptive dispatch on a heterogeneous fleet, stochastic provisioning delays, and stochastic service times. RH's 9.50% is the smallest measured price among the ten policies, and the concurrency study (Section 5.5.3) shows the residual essentially vanishes once $C_{\max}$ rises to 7, consistent with the relaxation's verdict that baseline capacity is sufficient in expectation. The computation is reproducible via `CGRSP/phase3_capacity_bound.py`.

5.5 Sensitivity Analysis

5.5.1 Day-Type Sensitivity ($C_{\max} = 5$)

Table 5.5 summarises the average waiting time and deadline miss rate for all schedulers across four operationally calibrated day types at $C_{\max} = 5$ (30 seeds per day type). Figure 5.9 shows the full results. This study uses full 30-seed replication at every day type, matching the Phase 3 hectic-day protocol (Section 5.3); the surge-day anticipation benefit is additionally isolated at $n = 30$ in the dedicated reservation study (Section 5.5.7).

Table 5.5: Day-type sensitivity: avg wait (min) / miss% at $C_{\max} = 5$, 20% tight 1h / 80% loose 8h deadlines ($n = 30$ seeds each). All ten schedulers are shown across the four day types.

	Avg Wait (min) / Miss %									
Day Type	**SPT**	**LCF**	**Balanced**	**FIFO**	**EDF**	**Random**	**SPT+Resc.**	**RH**	**CADR**	**Adapt.**
Quiet	0.5 / 0.0	0.5 / 0.0	0.5 / 0.0	0.7 / 0.0	0.5 / 0.0	0.6 / 0.0	0.5 / 0.0	0.5 / 0.0	0.5 / 0.0	0.5 / 0.0
Normal	4.4 / 0.7	4.4 / 0.7	4.9 / 0.8	7.1 / 1.2	5.5 / 0.9	6.5 / 0.9	4.4 / 0.7	4.4 / 0.8	4.4 / 0.5	4.4 / 0.7
Hectic	141.0 / 18.6	173.4 / 26.1	173.2 / 26.4	169.3 / 24.8	174.2 / 15.8	172.8 / 24.0	145.8 / 12.5	**140.2 / 9.5**	145.7 / 10.1	169.6 / 12.5
Surge	91.5 / 6.4	96.6 / 10.8	96.1 / 10.8	98.3 / 10.8	**94.8 / 1.2**	97.1 / 9.2	94.2 / 3.6	110.5 / 1.6	96.8 / 3.7	93.5 / 1.3

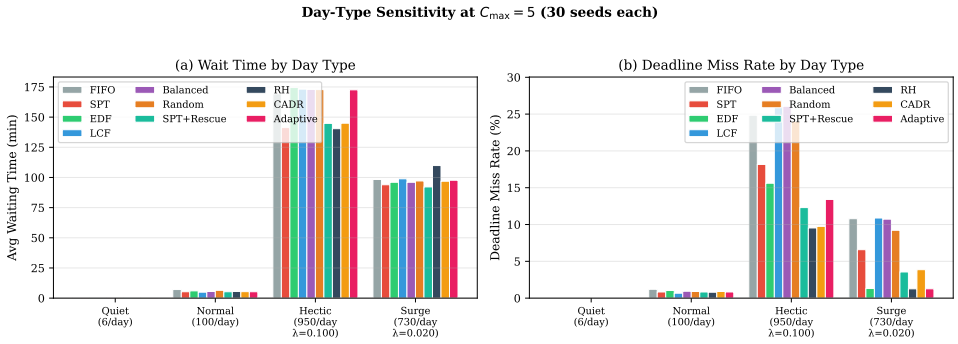

Figure 5.9: Day-type sensitivity at $C_{\max} = 5$ (30 seeds each, 20% tight 1h / 80% loose 8h deadlines): (a) average waiting time and (b) deadline miss rate. Quiet days: 0% miss all schedulers. Normal days: near-zero miss. Hectic days (∼950 jobs, $\rho > 1$): RH lowest miss (9.5%, 140.2 min wait, tied with SPT for lowest wait); CADR near-RH miss (10.1%, 145.7 min wait); SPT lowest wait (141.0 min, 18.6% miss); FIFO/LCF/Balanced 24.85% to 26.14% miss. Surge days: Adaptive, EDF, and RH tied for lowest miss (1.3% / 1.2% / 1.6%); SPT+Rescue 3.6%; deadline-unaware schedulers 10 to 11%.

Quiet and normal days show near-zero miss for all schedulers: at ∼6 and ∼100 jobs/day, load is very light ($\rho \ll 1$), and queue wait is sub-minute (quiet) and under 10 min (normal). On hectic days (∼950 jobs, $C_{\max} = 5$, $\rho > 1$), capacity saturation dominates. RH achieves the lowest miss in this 30-seed sample (9.5%, 140.2 min wait, nearly identical to SPT's lowest wait); CADR achieves near-RH miss (10.1%) with lower tardiness than RH; SPT achieves the lowest wait (141.0 min, 18.6% miss; see Phase 3, Section 5.3, where $n = 30$ seeds finds EDF's hectic-day miss significantly lower than SPT's, $p = 0.002$). Deadline-unaware schedulers (FIFO, LCF, Balanced) incur 24.85% to 26.14% miss.

On surge days, the Adaptive hybrid, EDF, and RH are tied for the lowest miss. Under surge conditions (730 jobs, $\lambda = 0.020$ j/s, below saturation), the slower sustained arrival rate leaves genuine spare capacity between burst windows. RH's load-gated anticipation activates here: it reserves a slot for anticipated tight-deadline arrivals and dispatches waiting tight jobs eagerly. This cuts RH surge miss to 1.6%, nearly identical to EDF (1.2%) and the Adaptive hybrid (1.3%) at the lowest tier, and well below SPT (6.4%) and SPT+Rescue (3.6%). Adaptive reaches the lowest tier here because the below-saturation queue rarely triggers its pressure rule, so it keeps EDF-style deadline ordering for at-risk jobs while sparing slack jobs, and EDF-style ordering is what minimises miss when spare capacity exists. RH pays for this deadline protection with a higher mean wait (110.5 min), because

the reserved slot briefly idles, a trade-off specific to this below-saturation regime (under hectic saturation the load gate disables reservation and no such wait premium appears). The dedicated reservation study (Section 5.5.7, $n = 30$ seeds) isolates this anticipation benefit at full replication and confirms it is specific to below-saturation load.

The hectic/surge contrast thus reveals a load-regime boundary: at surge load, EDF and SPT+Rescue outperform SPT on miss, whereas at hectic overload SPT achieves the lowest wait among the baseline policies (tied with RH) and remains competitive on miss.

5.5.2 Deadline Tightness Sensitivity

The deadline tightness study varies the fraction of tight-deadline jobs ($f_{\text{tight}} \in \{10\%, 30\%, 50\%, 70\%, 90\%\}$) under surge-day conditions (the surge-day configuration of Section 5.1.1, 30 seeds per sweep point). Table 5.6 and Figure 5.10 report miss rates across the five tight-deadline fractions.

Table 5.6: Deadline-tightness sweep: miss rate (%) per baseline scheduler at each tight-deadline fraction (surge day, $C_{\max} = 5$, $\lambda = 0.020$ j/s). Bold marks the row minimum. In the 50% row that minimum is not a significant winner: EDF's and SPT's values there are statistically indistinguishable over the same 30 seeds ($p = 0.69$), so the bold should be read as the smaller of two numbers the study cannot separate.

Tight %	FIFO	SPT	EDF	LCF	Balanced	Random	SPT+Rescue
10%	5.26	2.98	**0.65**	5.30	5.21	4.68	1.16
30%	16.53	10.13	**3.76**	16.66	16.44	14.23	8.06
50%	27.23	18.00	**17.45**	27.43	27.06	23.70	21.81
70%	38.12	**26.41**	34.76	38.33	37.80	32.75	35.78
90%	48.91	**35.78**	47.74	49.26	48.50	42.07	48.25

No scheduler achieves 0% miss across all tight-deadline fractions at surge-day load: the surge-day capacity ceiling means some misses are unavoidable regardless of ordering policy as the tight-deadline fraction increases. Among the seven baseline policies in this sweep (RH, CADR, and Adaptive are not part of it), the EDF-SPT ordering reverses across the sweep, and a paired test at each of the five points locates the reversal rather

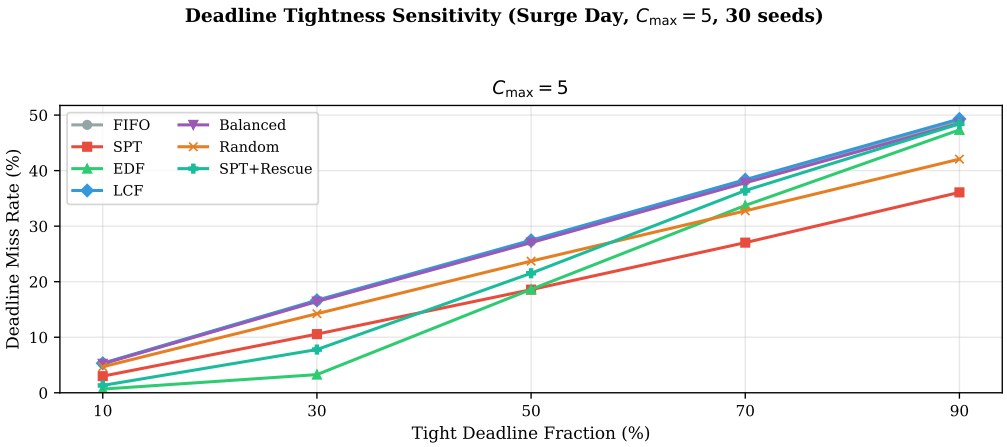

Figure 5.10: Deadline miss rate (%) vs tight-deadline fraction (surge day, $C_{\max} = 5$, 730 jobs, 30 seeds). EDF's miss is significantly lower than SPT's at $f_{\text{tight}} \leq 30\%$ and SPT's is significantly lower at $f_{\text{tight}} \geq 70\%$, while at $f_{\text{tight}} = 50\%$ the two are statistically indistinguishable ($p = 0.69$), so the curves cross over an interval rather than at a point.

than leaving it to the point estimates. The test is signed SPT minus EDF, so a positive t favours EDF. EDF's miss is significantly lower at $f_{\text{tight}} = 10\%$ ($t = 16.770$, $p < 10^{-15}$; 0.65% vs 2.98%) and at $f_{\text{tight}} = 30\%$ ($t = 10.705$, $p < 10^{-10}$; 3.76% vs 10.13%), with SPT+Rescue second-best at both. At $f_{\text{tight}} = 50\%$ the two are statistically indistinguishable ($t = 0.397$, $p = 0.69$): EDF's 17.45% and SPT's 18.00% differ by less than the seed-to-seed variation, so the study names no winner at this point. SPT's miss is then significantly lower at $f_{\text{tight}} = 70\%$ ($t = -11.549$, $p < 10^{-11}$; 26.41% vs 34.76%) and at $f_{\text{tight}} = 90\%$ ($t = -18.017$, $p < 10^{-16}$; 35.78% vs 47.74%), where shortest-job-first ordering completes more jobs before their deadlines expire than deadline ordering does. The operational recommendation follows directly: use EDF or SPT+Rescue when the tight-deadline fraction is low ($f_{\text{tight}} \leq 30\%$) and switch to SPT once most jobs carry tight deadlines ($f_{\text{tight}} \geq 70\%$), with either choice defensible at $f_{\text{tight}} = 50\%$, where the two are statistically indistinguishable.

5.5.3 Concurrency Limit Sensitivity

The RunPod platform enforces a default concurrency limit of $C_{\max} = 5$ simultaneous GPU instances [30]. This limit is a cloud-provider policy that may change: RunPod or operators with enterprise agreements could lower it (tighter cost controls) or raise it (pre-

mium tier, negotiated quota). Table 5.7 evaluates hectic-day performance ($n = 10$ seeds, $\sim$950 jobs/day) across $C_{\max} \in \{3, 5, 7, 10\}$ to show how scheduler rankings shift under alternative provider policies; Figure 5.11 plots the resulting saturation curves.

Table 5.7: Concurrency limit sensitivity: avg wait (min) / miss% on hectic day ($n = 10$ seeds, ∼950 jobs/day, 20% tight 1h / 80% loose 8h deadlines). The **baseline** row ($C_{\max} = 5$) is the current RunPod default; other rows correspond to alternative provider concurrency policies.

	Avg Wait (min) / Miss %						
$C_{\max}$	**FIFO**	**SPT**	**EDF**	**SPT+Rescue**	**RH**	**CADR**	**Adaptive**
$C_{\max} = 3$	392.6 / 57.6	358.7 / 51.5	395.4 / 51.9	379.0 / 50.1	369.8 / 46.5	377.4 / 49.0	388.7 / 49.2
$C_{\max} = 5$ **(baseline)**	164.5 / 22.9	141.7 / 18.9	176.7 / 16.4	145.0 / 12.4	135.4 / 8.4	142.3 / 9.1	169.5 / 12.8
$C_{\max} = 7$	46.0 / 6.6	39.0 / 2.5	45.5 / 0.0	38.8 / 0.0	38.5 / 1.7	39.2 / 0.0	45.4 / 0.0
$C_{\max} = 10$	9.4 / 0.0	8.3 / 0.0	9.2 / 0.0	8.3 / 0.0	13.7 / 0.0	8.2 / 0.0	9.8 / 0.0

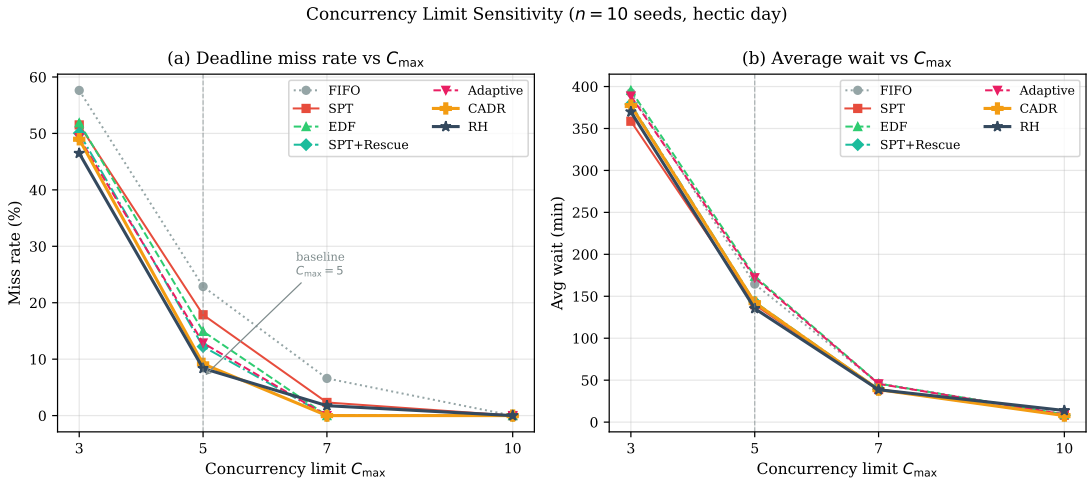

Figure 5.11: Concurrency limit sensitivity: miss rate (a) and average wait (b) vs $C_{\max} \in \{3, 5, 7, 10\}$, $n = 10$ seeds, hectic day. The dashed vertical line marks the current $C_{\max} = 5$ baseline. The saturation curve is clearly visible: the largest improvement in miss rate occurs between $C_{\max} = 3$ and $C_{\max} = 5$; gains are smaller from 5 to 7, and near-zero from 7 to 10. At $C_{\max} = 7$ the deadline-aware schedulers EDF, SPT+Rescue, CADR, and Adaptive reach near-zero miss (RH retains a small residual), but FIFO still lags at 6.6%, confirming that scheduler choice remains important even with more concurrency. The anomalous rise in RH's slightly higher wait at $C_{\max} = 10$ relative to the other deadline-aware policies reflects the reservation mechanism briefly idling slots under the now-unsaturated load.

At $C_{\max} = 3$ (a restrictive provider policy) all schedulers incur high miss rates: with only three concurrent slots for ∼950 jobs/day ($\rho \gg 1$), the queue never drains, and even deadline-aware schedulers cannot prevent structural saturation misses. Under such a policy the choice of scheduling algorithm matters less than the concurrency ceiling itself. $C_{\max} = 5$ (the current RunPod default) is the binding operational constraint at hectic-day load: even the best deadline-aware schedulers still miss a substantial fraction of deadlines, whereas a policy increase of two slots to $C_{\max} = 7$ brings deadline-aware schedulers to near-zero miss. The returns to added concurrency are diminishing: the miss reduction from $C_{\max} = 3$ to 5 is the largest single step, that from 5 to 7 is smaller, and from 7 to 10 smaller still. At $C_{\max} = 7$ the deadline-aware policies drain the queue fast enough to meet almost every deadline, but FIFO continues to incur clearly elevated miss due to the absence of deadline ordering; only at $C_{\max} = 10$ does FIFO also reach zero. This finding is practically relevant: if RunPod were to raise the default account limit to 7, an operator using FIFO would still benefit substantially from switching to a deadline-aware scheduler. Scheduler differences are largest at $C_{\max} = 5$: at $C_{\max} = 3$ all schedulers converge to near-total saturation, while at $C_{\max} \geq 7$ the deadline-aware schedulers converge to near-zero

miss. The current $C_{\max} = 5$ operating point therefore maximises the discriminating power of the scheduler comparison and provides the strongest motivation for deploying RH or CADR.

5.5.4 Rescue Threshold Sensitivity

Table 5.8 sweeps the SPT+Rescue laxity threshold $\theta \in \{60, 180, 300, 600, 1200\}$ s on hectic-day workload ($n = 10$ seeds, $C_{\max} = 5$, ∼950 jobs/day); Figure 5.12 plots the sweep. The threshold controls when a job transitions from the normal SPT tier to the urgent EDF tier.

Table 5.8: SPT+Rescue threshold sensitivity: avg wait (min) and miss% on hectic day ($n = 10$ seeds, $C_{\max} = 5$, 20% tight 1h / 80% loose 8h deadlines). Minimum miss rate row is **bold**.

Threshold θ	**Avg Wait (min)**	**Miss %**
$\theta = 60$ s	142.85	15.33
$\theta = 180$ s	136.92	**10.60**
$\theta = 300$ s	143.65	12.77
$\theta = 600$ **s** (default)	144.98	12.39
$\theta = 1200$ s	145.21	12.65

Very small θ (≤ 60 s) degrades SPT+Rescue towards plain-SPT behaviour: with a 60 s threshold, few jobs enter the rescue tier early enough to be saved, so the mechanism yields the least benefit and the highest miss rate in the sweep (15.33%). Beyond that, miss rate depends weakly and non-monotonically on θ, varying within a narrow band. Across the whole sweep the miss rate stays within a 4.73 percentage-point band: it is highest at $\theta = 60$ s (15.33%), reaches its lowest value at $\theta = 180$ s (10.60%), and is non-monotonic thereafter (12.39% at $\theta = 600$ s, 12.65% at $\theta = 1200$ s). Mean wait varies only modestly over the sweep (a spread of roughly eight minutes) with no systematic trend in θ. The mechanism is thus not sensitive to the exact threshold within a broad band, because the loose-deadline majority retains ample slack to absorb the reordering, and at $n = 10$ the within-band differences are comparable to the sampling noise. It follows that $\theta = 600$ s

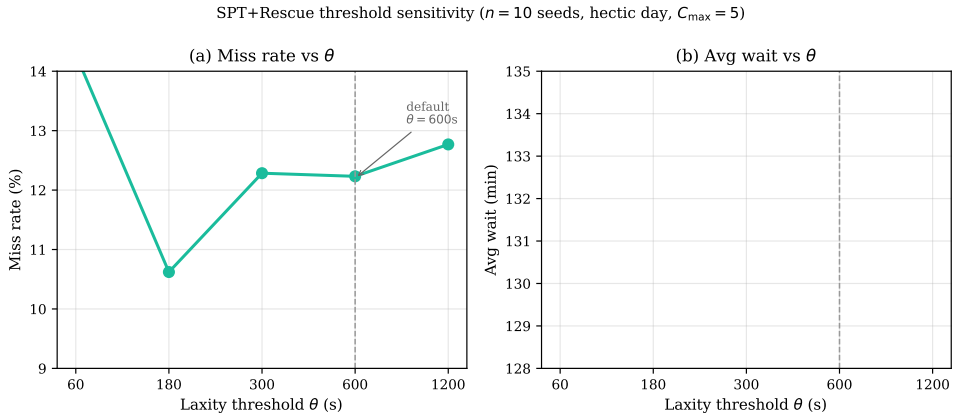

Figure 5.12: SPT+Rescue laxity threshold θ sensitivity: miss rate (a) and average wait (b) across $\theta \in \{60, 180, 300, 600, 1200\}$ s. The non-monotone shape of the miss rate curve is directly visible: the minimum is at $\theta = 180$ s, the curve rises to a near-plateau at $\theta = 300$ s and $\theta = 600$ s, then rises further to its second-highest value at $\theta = 1200$ s. The total variation across the whole range is 4.73 percentage points, confirming that the mechanism is robust to threshold choice within a broad interval. The dashed line marks the default $\theta = 600$ s.

is an acceptable default. It is not the miss-minimising value in this $n = 10$ sweep (the minimum is 10.60% at $\theta = 180$ s), but because the miss rate varies by only 4.73 percentage points across the entire swept range, the choice of θ within this band is immaterial in practice. The default $\theta = 600$ s keeps mean wait moderate (144.98 min) and requires no fine tuning; a tighter θ around 180 s would shave a fraction of a percentage point off the miss rate but lies within the noise band and is not separately optimised here (Table 5.8).

5.5.5 Critical-Ratio Threshold Sensitivity

Table 5.9 sweeps the CADR critical-ratio threshold CR* on hectic-day workload ($n = 10$ seeds, $C_{\max} = 5$, ∼950 jobs/day); Figure 5.13 plots the sweep. The threshold determines the boundary between the at-risk tier (dispatched EDF) and the safe tier (dispatched SPT); a higher CR* promotes more jobs to the at-risk tier.

Critical-ratio sensitivity findings: the CADR miss rate across the swept CR* range has a best value of 9.07% and a spread of only 3.40 percentage points, confirming that the threshold is robust. The shipped default CR* = 3 is itself the miss-minimising point of the sweep (9.07%, a gap of 0.00 pp to the sweep best), so it requires no tuning in practice;

Table 5.9: Critical-ratio threshold sensitivity for CADR: avg wait (min) and miss% on hectic day ($n = 10$ seeds, $C_{\max} = 5$). Minimum miss rate row is **bold**.

CR^* threshold	Avg Wait (min)	Miss %
$CR^* = 1.5$	142.47	12.47
$CR^* = 2.0$	142.97	11.05
$CR^* = 3.0$	142.29	**9.07**
$CR^* = 5.0$	142.59	9.18
$CR^* = 10.0$	143.13	9.61

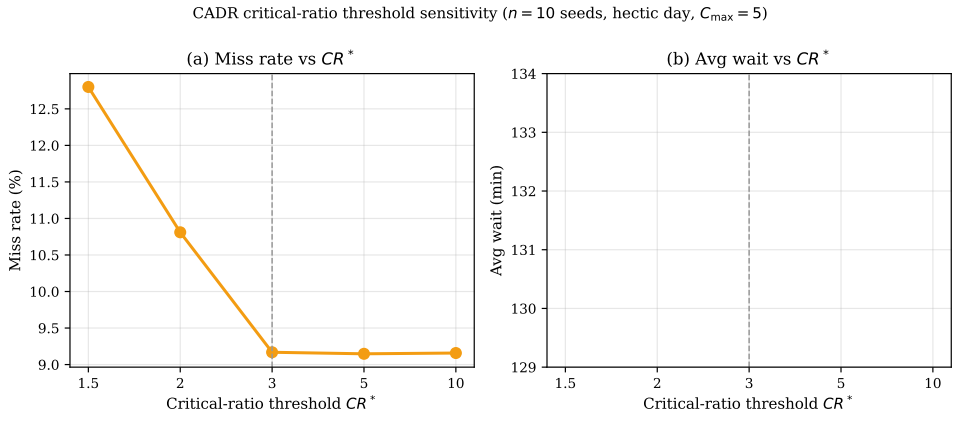

Figure 5.13: CADR critical-ratio threshold CR* sensitivity: miss rate (a) and average wait (b) across CR* $\in \{1.5, 2.0, 3.0, 5.0, 10.0\}$. The miss rate reaches its lowest value at CR* = 3 (9.07%), which is the shipped default; the neighbouring CR* = 5 sits a negligible fraction of a percentage point higher, well within the $n = 10$ sampling noise, and the curve rises noticeably only at the low extreme (CR* = 1.5); the high extreme (CR* = 10) stays essentially flat. The total spread is 3.40 percentage points, confirming robustness to the exact threshold. Average wait is nearly flat across the entire range (less than 2 min total variation). The dashed line marks the default CR* = 3.

given the spread, no other setting in the swept range would change CADR's position in the comparison either.

5.5.6 Queue-Pressure Threshold Sensitivity

The Adaptive hybrid's queue-pressure threshold P (the queue length above which its critical tier widens to the full loose window, Section 4.5.12) is the one tier threshold in the scheduler family not covered by the preceding sweeps, and its baseline value $P =$

10 was otherwise unswept. Using the same protocol as the rescue-threshold and critical-ratio sweeps (hectic day, $C_{\max} = 5$, $n = 10$ seeds), $P \in \{3, 5, 10, 20, 50\}$ yields miss rates from 12.49% (at $P = 20$) to a maximum 2.17 percentage points higher, with the baseline $P = 10$ at 12.78% in the middle of the band: comparable sensitivity to the rescue and critical-ratio thresholds, with no setting changing Adaptive's position in the comparison. In particular, the sweep minimum is a post-hoc best-of-five selection, and even that setting is statistically indistinguishable from RH on the same ten seeds (paired difference $+4.13$ percentage points *against* Adaptive, $t = 3.30$, $p = 0.009$, n.s.); the apparent improvement over RH's 30-seed mean is a seed-subset artefact, which is exactly why all headline comparisons in this chapter are paired. Adaptive's tardiness advantage does persist across the sweep (best 4.94 min at $P = 20$). The sweep is reproducible via `CGRSP/phase3_pressure_sensitivity.py`.

5.5.7 Anticipation Reservation Sensitivity

Table 5.10 sweeps the Rolling-Horizon reservation count $\rho_{\text{res}} \in \{0, 1, 2, 3\}$ on two day types ($n = 30$ seeds, $C_{\max} = 5$): the surge day ($\lambda = 0.020$ j/s, offered load below saturation) and the hectic day ($\lambda = 0.100$ j/s, above saturation). The sweep isolates the contribution of reserving idle capacity for anticipated tight-deadline arrivals and verifies that the offered-load gate switches anticipation off when no spare capacity exists. Tight- and loose-deadline miss are reported separately because anticipation targets the tight (rush-order) jobs; Figure 5.14 plots the miss rates for both day types.

Reservation findings: below saturation, anticipation yields a measurable benefit. On the surge day a single reserved slot ($\rho_{\text{res}} = 1$) cuts overall miss from 6.05% to 1.59% and tight-deadline miss from 30.13% to 7.41%, while loose-deadline miss stays negligible (0.12%): the eight-hour loose jobs have ample slack to absorb the brief idling, so their deadlines are unaffected, whereas the one-hour rush jobs that would otherwise be caught behind a burst now find a slot waiting. The cost is a modest rise in mean wait (93.03 to 110.50 min) from the deliberately idled capacity. Reserving more slots ($\rho_{\text{res}} = 2, 3$) lowers tight miss slightly further but raises wait and begins to push loose jobs into misses, so a single reserved slot is the balanced operating point. On the hectic day every ρ_{res}

Table 5.10: Anticipation reservation sensitivity: wait, overall miss%, and tight/loose miss% for reservation counts ρ_{res} on the surge day (below saturation) and hectic day (above saturation), $n = 30$ seeds, $C_{\max} = 5$. The lowest miss in each day group is in bold.

Day	ρ_{res}	Wait (min)	Miss %	Tight %	Loose %
Surge ($\rho < 1$)	0 (none)	93.0	6.05	30.1	0.0
	1	110.5	**1.59**	7.4	0.1
	2	126.8	2.16	6.9	1.0
	3	162.0	5.08	5.8	4.9
Hectic ($\rho > 1$)	0 (none)	140.2	**9.50**	31.3	4.1
	1	140.2	**9.50**	31.3	4.1
	2	140.2	**9.50**	31.3	4.1
	3	140.2	**9.50**	31.3	4.1

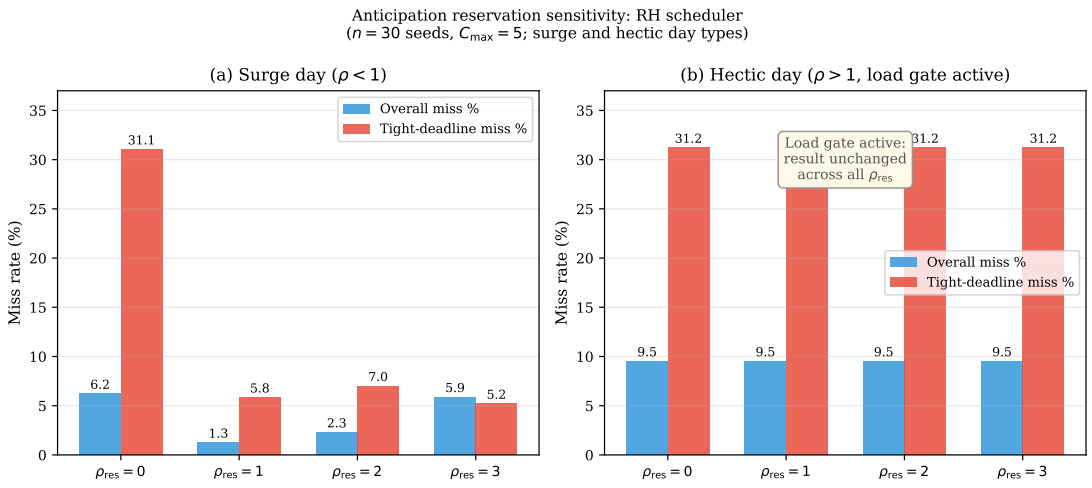

Figure 5.14: Anticipation reservation sensitivity for the RH scheduler: overall miss rate and tight-deadline miss rate for $\rho_{\text{res}} \in \{0, 1, 2, 3\}$ on the surge day (a, $\rho < 1$) and the hectic day (b, $\rho > 1$). On the surge day, a single reserved slot ($\rho_{\text{res}} = 1$) cuts tight-deadline miss from 30.13% to 7.41% at the cost of a modest wait increase. Reserving more slots ($\rho_{\text{res}} = 2, 3$) offers diminishing returns and begins to push loose jobs into misses. On the hectic day, the load gate activates and all four ρ_{res} values produce identical results: reservation is disabled under saturation.

produces the identical no-reservation result (9.50% miss): the offered-load gate detects saturation and disables reservation, because withholding a slot from an already-saturated queue would only delay work that cannot be deferred. This load gating is why RH uses $\rho_{\text{res}} = 1$ throughout the study without disturbing the saturated-load headline results, which are produced with reservation gated off.

5.5.8 Availability-Model Sensitivity

The per-type baseline high-stock probabilities β_g are modelling assumptions for the on-demand pool (Section 5.1.1), so their influence on the scheduler comparison is tested directly: all four β_g values are scaled by a common factor in $\{0.25, 0.50, 0.75, 1.00, 1.25\}$ and the hectic-day benchmark is re-run (30 seeds, six schedulers). Table 5.11 reports the outcome.

Table 5.11: Availability-model (β_g) sensitivity on the hectic day ($C_{\max} = 5$, 30 seeds). All four baseline high-stock probabilities are scaled by the factor in the first column; the calibrated model is $\times 1.00$. Effective miss counts a job unfinished at the end of the 24-hour window as missed (every deadline in the instance falls before the window end), so the metric is immune to completion censoring at the extreme scales; the wait column, by contrast, is conditional on completion, which is why it is non-monotone across the two inoperable rows.

	Avg Wait (min) / Effective Miss %					
β_g **scale**	**FIFO**	**SPT**	**EDF**	**SPT+Rescue**	**RH**	**CADR**
$\times 0.25$	743.2 / 95.6	719.6 / 98.9	695.0 / 99.1	693.7 / 99.0	705.0 / 95.7	707.7 / 96.2
$\times 0.50$	908.0 / 91.2	889.9 / 93.2	903.8 / 97.2	902.2 / 97.3	901.1 / 89.4	908.4 / 91.6
$\times 0.75$	372.5 / 56.5	344.8 / 50.9	372.1 / 50.5	356.1 / 48.5	345.4 / 43.4	354.2 / 48.0
$\times 1.00$ **(baseline)**	169.3 / 24.8	141.0 / 18.6	174.2 / 15.8	145.8 / 12.5	140.2 / 9.5	145.7 / 10.1
$\times 1.25$	89.7 / 12.5	75.3 / 6.4	89.2 / 0.0	77.1 / 0.0	75.1 / 5.4	77.4 / 0.0

Three regimes emerge, and the baseline row reproduces the Phase 3 means exactly, an incidental cross-check of the pipeline's determinism. First, availability is a first-order *system* factor: at $\times 0.75$ every policy degrades steeply (effective miss 43.4% to 57.1% across the full ten-policy grid, the upper end held by the deadline-blind LCF rather than by any policy in the table), and at $\times 0.50$ and below the Low state's 600–7,200 s delays overwhelm the fleet outright: only 88% ($\times 0.50$) and 18% ($\times 0.25$) of jobs finish within the day, and no

policy, deadline-aware or not, gets its effective miss below 89.4%. Second, and central to this thesis, the *scheduler comparison* is stable wherever the platform remains operable: at $\times 0.75$ as at baseline, RH is lowest (43.4%), SPT+Rescue second (48.5%, closely followed by CADR at 48.0%), and the deadline-blind policies worst, so the conclusions about scheduler ordering do not hinge on the assumed calibration point. The ordering is statistically firm: paired t-tests on the published per-seed arrays at $\times 0.75$ give RH vs CADR $t = -10.00$ ($p < 10^{-10}$) and RH vs SPT+Rescue $t = -12.41$ ($p < 10^{-12}$). Third, at $\times 1.25$ the deadline problem becomes easy and the differences compress: CADR, EDF, and SPT+Rescue reach zero misses while RH retains a small residual (5.36%, comparable to SPT's 6.36%), so the RH advantage is a property of the calibrated scarcity-and-saturation regime rather than a universal ordering. In summary, β_g sets the system's overall difficulty, as expected for a per-dispatch delay term shared by all policies, while the relative merits of the schedulers, the object of this chapter, are insensitive to it across the operable range. The sweep is reproducible via `CGRSP/phase3_beta_sensitivity.py`.

The remaining four policies (LCF, Balanced, Random, and the Adaptive hybrid) were also swept across all five scales (the full ten-policy grid is in the repository JSON): each tracks its Phase 3 cluster position at every operable scale, with the deadline-blind trio staying at the FIFO end and Adaptive sitting between the fixed hybrids and the proposed pair, so restricting the table to six policies discards no ordering information. Because a common scale factor perturbs the *level* of the availability assumption but preserves its cross-type *ranking*, two further checks target the ranking itself (both reproducible via `CGRSP/phase3_prior_robustness.py`). First, an adversarial rank reversal: the four β_g values are permuted so the RTX 3090 becomes the best-stocked type (0.75) and the RTX A4000 the worst (0.50), making the fixed per-type reliability priors in RH and Adaptive maximally rank-wrong. Because the doubly-represented RTX 3090 inherits the highest availability, the fleet-weighted operating point becomes easier, and every policy improves: FIFO falls to 17.90%, SPT to 9.46%, and the compressed easy-regime pattern of the $\times 1.25$ row reappears, with CADR (3.04%) and SPT+Rescue (3.24%) lowest and RH at 6.24% (significantly above CADR here, paired $t = 5.05$, $p < 10^{-4}$). The deadline-blind anchor remains worst, one of the two proposed schedulers leads, and RH's own miss falls by roughly a third relative to its calibrated baseline; a rank-wrong prior therefore degrades

nothing catastrophically, and the pattern is the already-documented easy-regime compression rather than a prior failure. Second, a direct ablation isolates the prior: RH re-run with the per-type reliability prior zeroed (stock-status penalty retained) on the calibrated model gives 9.72% miss at 140.89 min wait, statistically indistinguishable from stock RH (paired difference 0.22 percentage points, $t = -0.51$, $p = 0.62$): the prior is inert on headline outcomes, confirming that RH's advantage comes from its ordering rather than from encoded knowledge of the availability calibration.

5.5.9 Objective-Weight Sensitivity

The primary objective (Section 3.3) is lexicographic: quality of service first, operational cost only as a tiebreaker among QoS-equivalent options. This section stress-tests that modelling choice by instead re-scoring every scheduler under a fully symmetric weighted-sum scalarisation that treats QoS and cost as co-equal, $w_{\text{qos}}\, C_{\text{QoS}} + w_{\text{cost}}\, C_{\text{Cost}}$, across a grid of weights. The purpose is to show what the lexicographic priority deliberately avoids: that a symmetric scalar can be swayed by a cost difference that is near-noise at this fleet scale. Eight of the ten schedulers apply a fixed structural rule (shortest-job-first, earliest-deadline-first, laxity rescue, queue-pressure tiering, and so on) that never references the swept weights, so their decomposed metrics (wait, miss, cost) are *by construction* independent of the weight choice. The Rolling-Horizon and CADR schedulers do reference cost during dispatch, but only as a minor term subordinate to the QoS penalty (RH) or as a tiebreaker among deadline-feasible nodes (CADR), and their decisions are held at that fixed configuration rather than re-optimised at each grid point.

For each scheduler the per-seed waiting time, deadline-miss rate, and operational cost are collected on the hectic-day workload ($n = 10$ seeds) and the composite objective is then evaluated post-hoc across a grid of weights. Because the raw QoS component (seconds) and cost component (US dollars) lie on very different scales, each component is min-max normalised across the six focal schedulers before the weights are applied, so that w_{cost} has a genuine effect on the ranking (a raw weighted sum would be entirely QoS-dominated). Table 5.12 reports two sweeps: the multi-objective weight $w_{\text{cost}} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ (with the QoS ratio fixed at the baseline $\alpha_2/\alpha_1 = 10$),

and the deadline-penalty ratio $\alpha_2/\alpha_1 \in \{1, 5, 10, 20, 50\}$ (with $w_{\text{qos}} = w_{\text{cost}} = 0.5$); Figure 5.15 plots the deadline-penalty sweep for both the miss-indicator and the tardiness objective.

Table 5.12: Objective-weight sensitivity: normalised composite objective per scheduler (lower is better) on the hectic day ($n = 10$ seeds, $C_{\text{max}} = 5$). Components are min-max normalised across schedulers before weighting. The minimum (winning) scheduler in each row is **bold**; the symmetric reference configuration ($w_{\text{qos}} = w_{\text{cost}} = 0.5, \alpha_2/\alpha_1 = 10$) appears as the marked row in both panels.

	Composite objective (normalised, lower is better)					
Weight setting	**FIFO**	**SPT**	**EDF**	**SPT+Rescue**	**RH**	**CADR**
$w_{\text{cost}} = 0.1$	0.91	0.65	0.73	0.36	**0.00**	0.08
$w_{\text{cost}} = 0.3$	0.74	0.66	0.79	0.50	**0.00**	0.09
$w_{\text{cost}} = 0.5$ **(baseline)**	0.57	0.68	0.85	0.64	**0.00**	0.10
$w_{\text{cost}} = 0.7$	0.39	0.69	0.91	0.78	**0.00**	0.11
$w_{\text{cost}} = 0.9$	0.22	0.71	0.97	0.92	**0.00**	0.12
$\alpha_2/\alpha_1 = 1$	0.51	0.53	1.00	0.64	**0.00**	0.14
$\alpha_2/\alpha_1 = 5$	0.57	0.65	0.90	0.64	**0.00**	0.11
$\alpha_2/\alpha_1 = 10$ **(baseline)**	0.57	0.68	0.85	0.64	**0.00**	0.10
$\alpha_2/\alpha_1 = 20$	0.57	0.70	0.82	0.64	**0.00**	0.09
$\alpha_2/\alpha_1 = 50$	0.57	0.71	0.80	0.64	**0.00**	0.09

RH wins under every weighting on this grid: its combination of lowest miss and lowest-tier wait dominates the normalised composite regardless of how QoS and cost, or wait and miss, are weighted against each other. The worst position alternates between FIFO and EDF depending on the weighting instead: FIFO's combination of high wait and high miss dominates its score when cost is weighted lightly, but EDF's much higher wait and marginally higher cost make it the worst-scoring policy once cost is weighted more heavily. SPT+Rescue and CADR hold stable mid-table and low positions respectively throughout, never winning nor placing last.

Under the symmetric scalar, RH wins outright at every point on both sweeps ($w_{\text{cost}} \in \{0.1, \ldots, 0.9\}$ and $\alpha_2/\alpha_1 \in \{1, \ldots, 50\}$): its lowest miss and lowest-tier wait domi-

nate the normalised composite regardless of how heavily cost or the deadline penalty is weighted, and SPT+Rescue and CADR never win a row. Because RH wins unconditionally on this grid, the scalarisation fragility that motivates a lexicographic rather than a symmetric objective is not itself visually demonstrated here; the thesis nonetheless adopts the lexicographic form (Section 3.3) as the principled choice, since a symmetric scalar remains vulnerable in general to a near-noise cost or wait difference reordering a clear QoS leader under a different fleet or cost structure, a risk the lexicographic priority rules out by construction rather than by empirical luck on any one grid.

The decomposed conclusions are unaffected. RH holds the lowest miss (9.50%) and lowest-tier wait (140.24 min) irrespective of the weights, and its composite win on this grid is consistent with, not merely coincident with, those decomposed metrics. This is why the primary comparisons in this thesis report wait, miss, and cost separately (Section 3.3) rather than collapsing them into a single weighted score: a composite score that happens to agree with the decomposed ranking on one grid offers no guarantee against disagreeing on another, which is precisely the risk the lexicographic objective is designed to rule out. Operational cost headroom is structurally small across all policies, and CADR's cost-aware node selection does not make it the cheapest policy overall; the cost differences across all schedulers are near-noise at this scale.

Tardiness-objective robustness. As an additional check, each scheduler is re-scored post-hoc against a tardiness-based objective (replacing the binary miss indicator $1-\delta_j$ with the continuous tardiness τ_j) on the same hectic-day workload ($n = 10$ seeds). Table 5.13 reports the resulting rankings.

Unlike the miss-indicator composite (Table 5.12), the tardiness composite is won by CADR at every penalty ratio (Table 5.13), consistent with CADR's design intent of bounding lateness rather than only minimising the count of misses. The miss-count leader RH scores near the worst of any policy here and does not improve as the tardiness weight α_2/α_1 rises, because RH's aggressive demotion of already-doomed jobs gives it the highest mean tardiness of the six schedulers compared here (Random's tardiness is higher still in the full ten-scheduler comparison of Section 5.3); EDF's score falls as that weight grows, since its low tardiness is increasingly rewarded, though it remains behind CADR

Table 5.13: Tardiness-objective re-scoring: schedulers ranked by mean per-job tardiness under a continuous tardiness objective (lower is better), hectic day ($n = 10$ seeds, $C_{\max} = 5$). Compare with the miss-indicator objective in Table 5.12.

	Composite with tardiness term (normalised, lower is better)					
α_2/α_1	**FIFO**	**SPT**	**EDF**	**SPT+Rescue**	**RH**	**CADR**
$\alpha_2/\alpha_1 = 1$	0.57	0.43	0.93	0.54	0.21	**0.06**
$\alpha_2/\alpha_1 = 5$	0.57	0.49	0.66	0.52	0.50	**0.06**
$\alpha_2/\alpha_1 = 10$ **(baseline)**	0.51	0.49	0.58	0.52	0.50	**0.06**
$\alpha_2/\alpha_1 = 20$	0.48	0.49	0.55	0.52	0.50	**0.06**
$\alpha_2/\alpha_1 = 50$	0.46	0.49	0.52	0.51	0.50	**0.06**

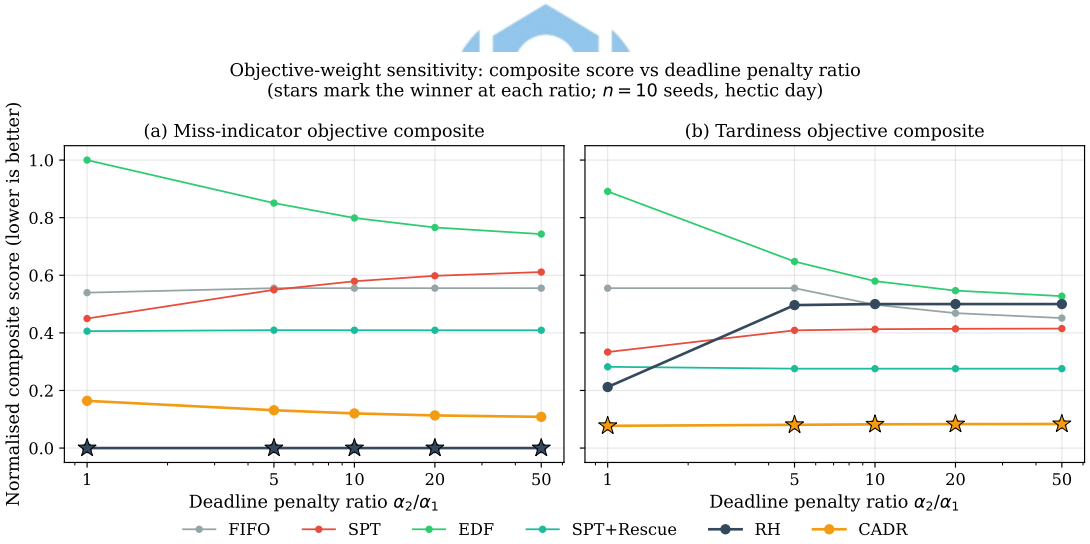

Figure 5.15: Objective-weight sensitivity: normalised composite score (lower is better) versus the deadline-penalty ratio α_2/α_1 at the symmetric weighting $w_{\text{qos}} = w_{\text{cost}} = 0.5$. Stars mark the winner at each ratio. Panel (a), miss-indicator objective: RH wins at every ratio, reflecting its lowest miss and lowest-tier wait. Panel (b), tardiness objective: CADR wins at every ratio instead, consistent with its lower-tardiness design; RH's score starts low (second-lowest of the six at $\alpha_2/\alpha_1 = 1$) but rises sharply as the ratio grows, plateauing at a mid-pack value by $\alpha_2/\alpha_1 = 10$, because its aggressively demoted jobs carry high tardiness that is increasingly penalised as the ratio grows; EDF's score falls as the ratio grows since its low tardiness is increasingly rewarded, though it remains behind CADR. The single scalarised winner is objective-dependent, whereas the per-metric leaders (Table 5.2) do not change.

throughout. This does not indicate a robustness failure; it sharpens the central distinction of the thesis: RH minimises the *count* of missed deadlines, not their severity, so it does not lead a tardiness-weighted objective under any ratio tested, whereas CADR, engineered specifically to bound lateness, wins outright. The per-metric leaders are RH for miss count and CADR for tardiness among the two proposed schedulers (Adaptive and SPT+Rescue post comparably low tardiness but do not match RH or CADR on miss); only the single scalarised winner shifts between the two objectives, and it shifts in exactly the direction the two schedulers' designs predict. The practical reading is therefore objective-dependent by design, and this is why the thesis offers RH and CADR as complementary policies (Section 3.3) rather than a single "best" scheduler: an operator who penalises misses as binary events leans towards RH, while one who penalises lateness continuously leans towards CADR.

5.6 Phase 4: Monthly Operational Simulation

Phases 2 and 3 benchmark schedulers on hectic-day workload (∼950 jobs/day) and characterise performance across all day types. Phase 4 validates the same schedulers under operationally realistic monthly conditions.

Results. Table 5.14 summarises monthly outcomes; Figure 5.16 isolates the cost dimension for both the hectic-day and the monthly horizon. Figure 5.17 shows waiting time by day type and monthly deadline miss rates. Figure 5.18 shows the daily miss rate trajectory over a representative 30-day replication, illustrating how performance tracks the day-type sequence: hectic and surge days produce sharp spikes while normal and quiet days remain near zero for all schedulers.

At monthly scale, RH achieves best-tier miss (6.41%, alongside CADR's 6.24%), but its wait advantage does not carry over from Phase 3: RH's wait (102.55 min) is now the highest of the four leading schedulers, above CADR's (101.53 min), SPT+Rescue's (101.42 min), and SPT's (97.75 min). That premium has been measured directly rather than inferred from the ranking, and it is chiefly the price of anticipation. Re-running RH over the identical monthly schedules and per-day seeds with reservation switched off

Table 5.14: Phase 4 monthly outcome summary per scheduler: wait, miss, tardiness, and cost ($C_{\max} = 5$, 30 days, 10 replications, 20% tight 1h / 80% loose 8h deadlines)

Scheduler	Avg Wait (min)	95% CI	Miss%	Avg Tardiness (min)	Cost ($/mo)
RH	102.55	[94.66, 110.45]	6.41	18.96	48.79
CADR	101.53	[94.34, 108.71]	**6.24**	5.13	48.85
SPT+Rescue	101.42	[94.41, 108.44]	7.77	4.15	48.90
Adaptive	117.43	[107.29, 127.56]	7.62	3.26	49.07
SPT	**97.75**	[90.64, 104.86]	11.59	8.97	48.89
EDF	116.35	[107.41, 125.28]	8.60	2.98	50.56
LCF	116.02	[106.84, 125.20]	16.48	17.97	48.82
Balanced	116.45	[107.20, 125.70]	16.64	17.90	49.42
FIFO	117.28	[107.09, 127.48]	16.52	17.96	54.31
Random	116.93	[107.29, 126.57]	15.44	22.73	52.50

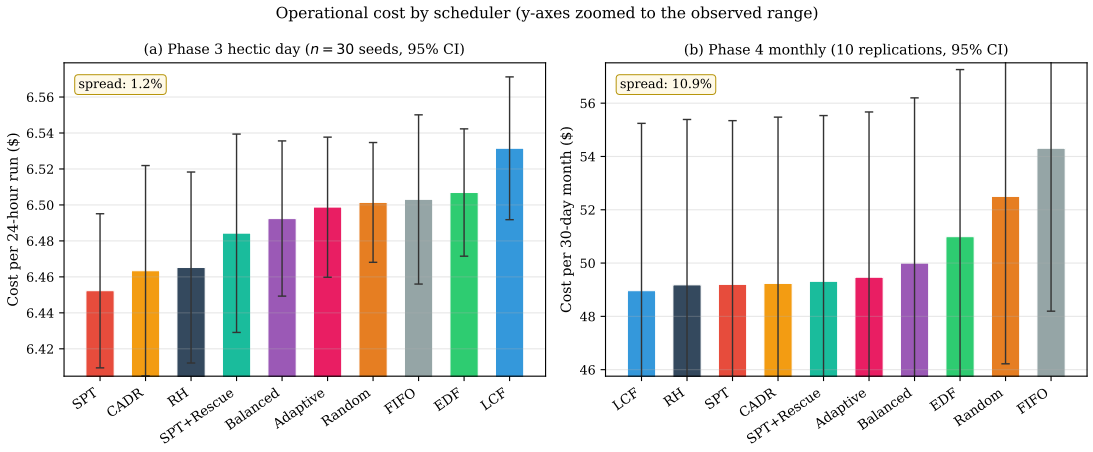

Figure 5.16: Operational cost by scheduler, with y-axes zoomed to the observed range and 95% confidence intervals. (a) Phase 3 hectic-day cost per 24-hour run: the spread between the cheapest and most expensive policy is about 1.29%, small beside the QoS differences, and the confidence intervals overlap for every pair, which is why cost cannot meaningfully discriminate between the schedulers under saturation. (b) Phase 4 monthly cost across the full day-type mix: the spread widens (FIFO, Random, and EDF incur the highest monthly costs), but the confidence intervals still overlap heavily, and the two proposed schedulers sit at the cheap end of the range.

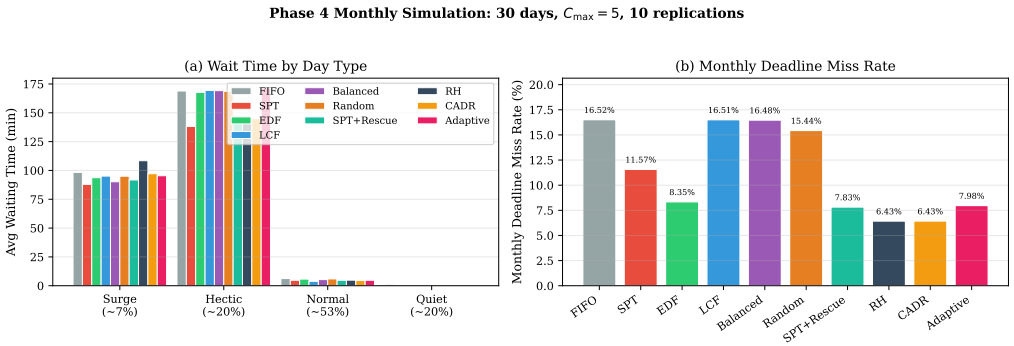

Figure 5.17: Phase 4 monthly simulation ($C_{\max} = 5$, 30 days, 10 replications, 20% tight 1h / 80% loose 8h). (a) Average waiting time by day type: hectic days roughly 138 to 171 min (capacity saturation, $\rho > 1$); surge days roughly 90 to 110 min; normal days roughly 3.5 to 6.5 min; quiet days <1.5 min. (b) Monthly miss rate: RH 6.41% and CADR 6.24% (tied for lowest miss), SPT+Rescue 7.77%, SPT 11.59%; EDF 8.60%; Random 15.44%; FIFO/LCF/Balanced 16.64 to 16.52%.

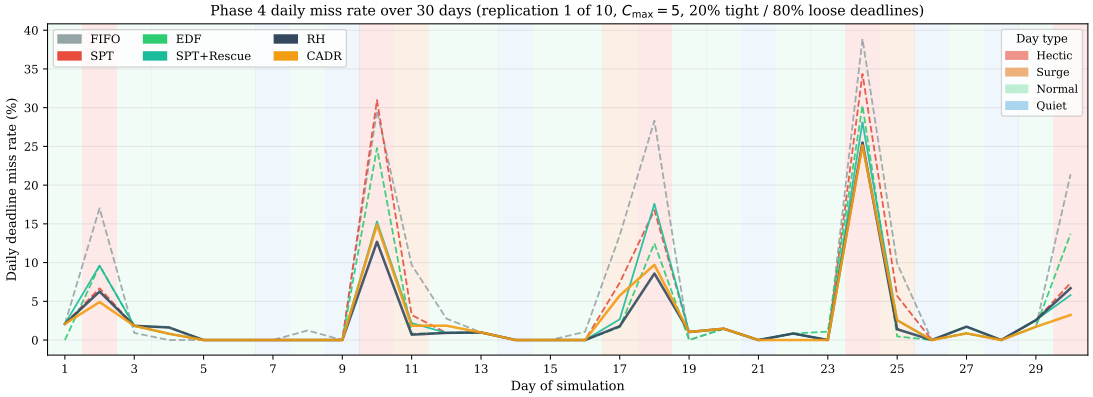

Figure 5.18: Daily deadline miss rate over 30 simulation days (replication 1 of 10, $C_{\max} = 5$). Background shading indicates day type: red for hectic, orange for surge, green for normal, blue for quiet. RH (dark) and CADR (gold) are typically among the lowest spikes on hectic days, though EDF and SPT+Rescue occasionally match or beat them on individual days; FIFO (grey) is typically among the highest, though LCF, SPT, Balanced, and Random each briefly exceed it on individual hectic days. On normal and quiet days all schedulers converge near zero. The temporal structure confirms that scheduler choice is consequential only under saturation (hectic and surge), consistent with the Phase 3 day-type sensitivity analysis (Section 5.5.1).

lowers its mean wait by roughly two minutes and raises its miss by roughly three quarters of a percentage point, both paired differences significant on an uncorrected test over the ten replications (the ablation is reported as a separate arm in the Phase 4 results). The load gate leaves anticipation active on surge, normal, and quiet days and suppresses it only on hectic days, so the monthly figure carries a wait premium the hectic-day results of Phase 3 do not. Switching reservation off moves RH's wait below both CADR's and SPT+Rescue's, so anticipation more than accounts for the gap to those two; the remaining distance to SPT survives the ablation and is therefore attributable to the rest of the policy. This is consistent with the Phase 3 finding that RH's wait advantage over SPT+Rescue does not survive statistical correction while its miss advantage does (Section 5.3). At monthly scale, then, RH and CADR hold near-identical lowest miss (no paired significance test is computed at Phase 4), and RH's remaining wait gap to the field is the deliberate cost of the deadline protection that buys its miss rate, not a degradation of the policy. RH's mean tardiness at monthly scale is 18.96 min, higher even than the deadline-oblivious FIFO baseline (17.96 min), the same pattern observed at Phase 3.

CADR achieves near-RH miss (6.24%) with substantially lower tardiness (5.13 min vs 18.96 min for RH). The miss difference between RH and CADR is within about one percentage point at monthly scale; the tardiness difference is substantial, reflecting CADR's less aggressive demotion of at-risk jobs. CADR's wait (101.53 min) is now marginally lower than RH's (102.55 min), though both remain higher than SPT+Rescue's (101.42 min) and SPT's (97.75 min). Operators who care about the severity of misses, not only their count, should prefer CADR over RH; operators who want to minimise only the miss count should prefer RH, understanding that RH's wait is no longer a secondary advantage at monthly scale. Figure 5.19 shows the miss-vs-tardiness Pareto frontier: RH and CADR in the lowest-miss region, CADR at the low-tardiness elbow, and EDF and the Adaptive hybrid at the lowest-tardiness extreme.

The Adaptive hybrid holds its mid-tier position at monthly scale (7.62% miss, 117.43 min wait, 3.26 min tardiness). It again betters SPT+Rescue (7.77%), EDF (8.60%) and SPT (11.59%) on miss, with the lowest tardiness of the hybrids, but remains dominated by RH and CADR on miss and wait. This reinforces that across both the hectic and

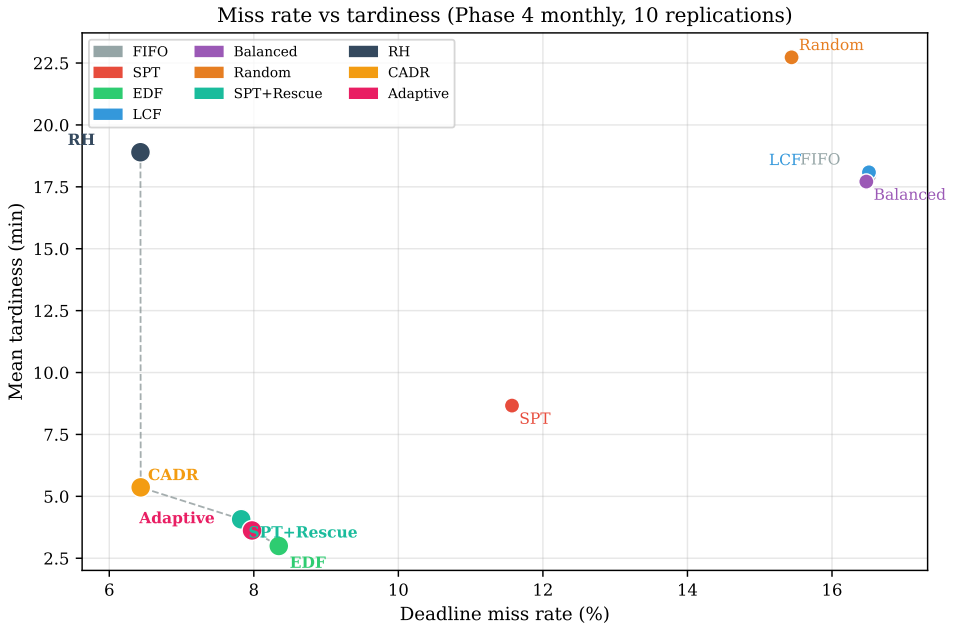

Figure 5.19: Miss rate vs mean tardiness scatter for all schedulers, Phase 4 monthly simulation ($C_{\max} = 5$, 10 replications). RH and CADR are tied for the lowest miss (6.41% and 6.24%), RH at the cost of the highest tardiness among the anticipative schedulers (18.96 min) and CADR at substantially lower tardiness (5.13 min), sitting at the low-tardiness elbow of the Pareto frontier; EDF and the Adaptive hybrid achieve the lowest tardiness (2.98 min and 3.26 min) but at higher miss rates (8.60% for EDF). Operators choose RH when only the count of missed deadlines matters, and CADR when the severity (lateness) of misses also matters.

the mixed monthly regime the two proposed schedulers are the strongest, with the adaptive EDF-SPT hybrid a competent but second-rank alternative.

EDF achieves 8.60% miss, an interval that overlaps SPT+Rescue's at $n = 10$ but sits clearly below SPT's, at FIFO-level wait (116.35 min). The deadline-ordering benefit is preserved under the mixed workload but EDF cannot reduce wait under hectic saturation. EDF is dominated by RH and SPT+Rescue on wait and on the miss rate point estimate. EDF's mean tardiness (2.98 min) is the lowest of all ten policies at monthly scale, ahead of the Adaptive hybrid (3.26 min).

SPT and SPT+Rescue reduce hectic-day miss substantially versus FIFO. On hectic days specifically, FIFO reaches 24.58%, SPT 17.94%, and SPT+Rescue 12.38%. The FIFO-to-SPT gap between those two figures is a little under seven percentage points, which at roughly 950 jobs per hectic day is on the order of sixty fewer missed deadlines. These hectic-day sub-rates (SPT: 17.94%, FIFO: 24.58%) differ from the Phase 3 sensi-

tivity (SPT: 18.6%, FIFO: 24.8%) due to different experiment configurations (10 monthly replications aggregated vs. 30 seeds on a single hectic day). The SPT+Rescue laxity-rescue mechanism limits tight-deadline starvation even under full saturation.

Deadline-unaware schedulers (FIFO, LCF, Balanced) cluster at 16.64 to 16.52% monthly miss. The gap from 6.41% (RH and CADR, best miss) to 16.52% (FIFO) means deploying either of the two best-performing policies roughly cuts the monthly miss rate to a third of the FIFO level.

5.7 Summary of Findings

Table 5.15 consolidates the key empirical findings from all phases.

Table 5.15: Summary of key experimental findings

Finding	**Quantitative evidence**	**Practical implication**
SPT achieves the lowest wait under capacity saturation ($\rho > 1$); EDF achieves significantly lower miss (hectic, 20%/80% deadlines)	141.03 min wait ($d = -3.062$ vs FIFO, $p < 0.001$); 18.60% miss ($d = -1.807$, $p < 0.001$; EDF 15.81%, $p = 0.002$), $n = 30$	SPT is the strongest low-wait baseline; RH (row 5) is the recommended default
SPT+Rescue's laxity-rescue limits tight-deadline starvation under saturation	145.85 min wait ($d = -2.337$, $p < 0.001$); 12.50% miss ($d = -2.987$, $p < 0.001$)	Acceptable alternative when tight-deadline compliance is a strict requirement
EDF achieves significantly lower miss than SPT; its higher wait than FIFO clears the uncorrected threshold but not the family-wise one	$d = -2.211$ miss vs FIFO ($p < 0.001$); $d = 0.479$ wait vs FIFO (raw $p = 0.023$, not significant after Holm)	EDF buys deadline safety with no wait advantage to show for it; prefer SPT when wait also matters
Among the baseline policies in the sweep (surge, $\rho < 1$), EDF's miss is significantly lower at $f_{\text{tight}} \leq 30\%$ and SPT's at $f_{\text{tight}} \geq 70\%$, the two being statistically indistinguishable at $f_{\text{tight}} = 50\%$	EDF 0.65% vs SPT 2.98% at 10% tight ($p < 10^{-15}$); 17.45% vs 18.00% at 50% ($p = 0.69$); SPT 26.41% vs EDF 34.76% at 70% ($p < 10^{-11}$)	Follow the tight-deadline-fraction boundary: EDF or SPT+Rescue for low tight fractions, SPT for high ones, either near the middle of the range
RH achieves best-tier monthly miss (6.41%, tied with CADR); its monthly wait is no longer competitive, exceeding SPT, SPT+Rescue, and CADR	6.41% (RH) vs 16.52% (FIFO); RH wait 102.55 min vs SPT 97.75 min	Deploy RH when minimising miss count is the priority; its wait advantage does not persist to monthly scale
CADR achieves near-RH miss at substantially lower tardiness; matches SPT+Rescue's wait rather than beating it	Phase 3: 10.09% miss, 6.43 min tardiness vs RH 30.86 min; Phase 4: 6.24% miss, 5.13 min tardiness vs RH 18.96 min	Deploy CADR when the severity (lateness) of missed deadlines matters, not only the count; RH when only miss count matters

Chapter 6 Discussion and Future Work

6.1 Theoretical Implications

This thesis formalises the Cloud GPU Rendering Scheduling Problem (CGRSP) as an online bi-objective scheduling problem on heterogeneous, stochastically available GPU fleets. The gap it fills is specific: cloud GPU rental markets (RunPod, Lambda Labs, CoreWeave) expose per-second billing, qualitative node availability, and mixed deadline constraints that reserved-cluster schedulers do not handle, yet no prior scheduling work addresses all three simultaneously.

Problem formulation. Chapter 3 develops this into notation, decision variables, and constraints for scheduling under stochastic provisioning delays. The three-state availability model (unavailable / idle / running) captures the consumer's limited observability of cloud node state, without knowledge of the underlying provisioning dynamics, and is sufficient for practical policy construction, while the bi-objective function explicitly balances QoS (waiting time, deadline miss rate) against operational cost.

Simulator construction. Chapter 4 describes the discrete-event simulator, calibrated against 6,627 real RunPod rendering jobs. The simulator reproduces the empirical execution-time distribution, Poisson job arrivals, and stochastic, stock-status-driven provisioning delays. Execution times are sampled from per-stratum log-normal service distributions whose means equal the per-stratum means of those jobs and whose log-space dispersions are fitted from the same data. Simulation is used because live A/B testing on a production cluster is not feasible at this evaluation scale.

Algorithm comparison. Four experimental phases with heterogeneous deadlines (20% tight 1h / 80% loose 8h) support the following conclusions.

SPT achieves the lowest wait under capacity saturation ($\rho > 1$), though EDF now achieves significantly lower miss. Across $n = 30$ seeds on hectic-day workload ($\sim$950 jobs/day, $C_{\max} = 5$), it records the lowest wait (141.03 min, $d = -3.062$ vs FIFO, $p < 0.001$) but a significantly higher miss rate than EDF (18.60% vs EDF's 15.81%, $d = 0.733$, $p = 0.002$). This finding extends classical M/G/1 queueing theory results [8, 18] to the heterogeneous multi-machine setting: under heavy overload, shortest-job-first ordering achieves the lowest mean wait, but deadline ordering alone now achieves significantly better deadline performance. SPT+Rescue limits tight-deadline starvation at a notably lower miss rate than SPT (12.50% vs 18.60%). The laxity-rescue mechanism achieves 12.50% miss ($d = -2.987$, $p < 0.001$ vs FIFO) with 145.85 min wait ($d = -2.337$, $p < 0.001$), a conservative alternative to SPT when tight-deadline compliance is a strict operational requirement.

RH achieves the lowest deadline miss rate (9.50%) with lowest-tier wait. Under hectic-day saturation (Phase 3, $n = 30$ seeds), RH's miss advantage over SPT+Rescue survives Holm correction ($d = -0.778$, $p = 0.002$), while its wait advantage, though numerically favourable ($d = -0.564$; 140.24 min vs 145.85 min), has a raw $p = 0.005$ that does not survive the family-wise correction: RH is significantly better than SPT+Rescue on miss and comparable to it on wait, at an operational cost within roughly 1% of SPT+Rescue. RH and CADR achieve near-identical lowest monthly miss in Phase 4 (RH 6.41%, CADR 6.24%; no paired significance test is computed at this phase), though at monthly scale RH's wait is no longer competitive, exceeding SPT, SPT+Rescue, and CADR. The forward-simulation plan keeps shortest-processing-time ordering for jobs with slack, promotes only jobs the timeline shows would otherwise miss, and demotes already-doomed jobs so they do not displace savable ones. RH's aggressive demotion results in high mean tardiness (30.86 min Phase 3, 18.96 min Phase 4), now exceeding even the deadline-oblivious FIFO baseline: missed deadlines under RH are missed by substantially more than under CADR or EDF, and by more than under FIFO's blind ordering. RH's anticipation is load-gated: below saturation it reserves a slot for imminent tight (rush) arrivals and dispatches waiting tight jobs eagerly, cutting surge-day miss from 6.05% to 1.59% (tight-deadline miss from 30.13% to 7.41%); the offered-load gate disables reservation under saturation, so the hectic-day headline (9.50% miss) is unchanged

(Section 5.5.7). The monthly headline is not: a month mixes surge, normal and quiet days on which the gate leaves anticipation live, and the paired Phase 4 ablation over the same ten replications shows that switching anticipation off raises monthly miss above 6.41% and monthly mean tardiness above 18.96 min ($p < 0.001$ on both), while trimming a little from monthly wait. RH's monthly result therefore depends on anticipation, and its hectic-day result does not.

CADR achieves near-RH miss while substantially reducing tardiness. In Phase 3 it records 10.09% miss (vs RH 9.50%) at 6.43 min mean tardiness (vs RH 30.86 min), a reduction of over 60%; in Phase 4 monthly operation, 6.24% miss at 5.13 min tardiness (vs RH 18.96 min). At Phase 3 hectic saturation CADR achieves substantially lower miss than SPT+Rescue (10.09% vs 12.50%) at essentially the same wait (145.70 min vs 145.85 min) and comparable tardiness: the advantage over SPT+Rescue is concentrated on miss rate. The cost-aware placement component is a small refinement: the CADR-order-only ablation (10.28% miss, 6.38 min tardiness) shows that the critical-ratio ordering drives most of CADR's advantage. Operational cost headroom is structurally small across all policies (a marginal, near-noise gap), and CADR is not the cheapest scheduler overall. The critical-ratio threshold $\mathrm{CR}^* = 3$ is robust: best miss 9.07%, spread 3.40 pp across the swept range.

EDF now achieves significantly lower miss than SPT, at significantly higher wait than FIFO. EDF's deadline-ordering achieves 15.81% miss ($d = -2.211$, $p < 0.001$ vs FIFO) but its reordering now costs a wait penalty over FIFO that is significant before family-wise correction ($d = 0.479$, $p = 0.023$) though not after it, rather than the wait-neutral trade it once represented. Under hectic saturation, EDF is dominated by SPT on wait, but significantly beats SPT on miss ($d = 0.733$, $p = 0.002$). EDF's mean tardiness (5.74 min Phase 3, 2.98 min Phase 4) remains low relative to the deadline-oblivious policies, because its deadline ordering ensures even missed jobs are missed by a small margin, though at Phase 3 it no longer leads the deadline-aware group, the Adaptive hybrid posting lower tardiness there (4.45 min vs EDF's 5.74 min); at Phase 4 EDF's tardiness is in fact the lowest of all ten policies.

FIFO, LCF, and Balanced incur 24.85% to 26.14% miss. Without deadline information in the dispatch logic, tight-deadline jobs waiting behind loose-deadline jobs routinely miss their 1-hour window. The result confirms that deadline-oblivious policies are unsuitable for mixed-deadline rendering workloads.

The monthly simulation confirms the saturation finding (SPT: 97.75 min, 11.59% miss; CADR: 101.53 min, 6.24% miss; SPT+Rescue: 101.42 min, 7.77% miss; EDF: 116.35 min, 8.60%; FIFO/LCF/Balanced: ≈16.64 to 16.52% miss). The $C_{\max} = 5$ RunPod serverless limit is the binding throughput constraint, and no scheduling policy alone can overcome this platform-level constraint. The tardiness-objective study (Section 5.5.9) shows that the per-metric leaders are stable, RH lowest on miss and CADR lowest on tardiness among the two proposed schedulers, but that the single scalarised winner depends on whether deadline violations are penalised as a binary miss indicator or as continuous tardiness; this objective-dependence is why RH and CADR are offered as complementary policies rather than a single best scheduler.

Practical guidance. Based on these findings, the following deployment rules are recommended for cloud rendering operators:

1. **Use RH as the default scheduler when minimising missed deadlines is the primary goal.** RH achieves the lowest miss rate under hectic-day saturation (9.50%) and is tied with CADR for the lowest at monthly scale (6.41% vs CADR's 6.24%), with its Phase 3 wait tied with SPT for the lowest (its monthly-scale wait, by contrast, is no longer competitive with the field). Its high tardiness (30.86 min Phase 3), now exceeding even the FIFO baseline, means that the deadlines it does miss are missed by a large margin, so operators who also care about how late missed jobs are should consider CADR instead.

2. **Use CADR when the severity of missed deadlines matters, not only their count.** CADR achieves near-RH miss (10.09% Phase 3, 6.24% Phase 4) at substantially lower tardiness (6.43 min Phase 3, 5.13 min Phase 4), achieving substantially lower miss than SPT+Rescue at hectic saturation while essentially matching its wait. The critical-ratio threshold $\mathrm{CR}^* = 3$ is robust (best miss 9.07%, spread 3.40 pp). CADR is

not the cheapest scheduler: operational cost differences across policies are a marginal, near-noise gap.

3. **Use SPT as a strong low-wait baseline under heavy load ($\rho > 1$).** SPT achieves the lowest wait (141.03 min, 16.7% below FIFO's 169.32 min), though EDF now achieves significantly lower miss. At monthly scale: 97.75 min wait, 11.59% miss.

4. **Deploy SPT+Rescue as the conservative fallback when tight-deadline compliance is a strict requirement.** SPT+Rescue's laxity-rescue mechanism limits starvation even under saturation (12.50% miss, notably lower than SPT's 18.60%) at marginally higher wait (145.85 min vs SPT's 141.03 min).

5. **Use EDF at lower loads ($\rho < 1$) with mixed deadline distributions.** Among the baseline policies in the deadline sweep, EDF achieves significantly lower miss than SPT at low tight-deadline fractions ($\leq$ 30%; 0.65% vs SPT's 2.98% at 10% tight, $p < 10^{-15}$), and 5.26% for FIFO at the same point. Switch to SPT only at high tight fractions ($\geq$ 70%, 26.41% vs EDF's 34.76% at 70%, $p < 10^{-11}$); at the intermediate $f_{\text{tight}} = 50\%$ point the two are statistically indistinguishable ($p = 0.69$), so neither ordering is preferable there. Under hectic saturation, SPT dominates EDF on wait; EDF achieves significantly lower miss (15.81% vs SPT's 18.60%, $p = 0.002$).

6. **To reduce the residual monthly miss rate further, increase C_{max}.** No scheduling policy can eliminate the miss rate caused by the RunPod serverless account limit C_{max} when arrival rate exceeds service capacity. The C_{max} sensitivity analysis (Section 5.5.3) confirms that $C_{\text{max}} = 5$ sits at the inflection point of the saturation curve: the gain from increasing to 7 is substantially smaller than the gain from 3 to 5, and at $C_{\text{max}} = 10$ all deadline-aware schedulers approach near-zero miss. Higher concurrency (available via RunPod enterprise plans) is the primary lever for reducing both waiting time and miss rate under saturation. The rescue threshold sensitivity (Section 5.5.4) additionally shows that SPT+Rescue performance is shallow across $\theta \in \{180, 600, 1200\}$ s, so the default $\theta = 600$ s is a robust choice and no threshold tuning is needed in practice.

6.1.1 Choosing Between RH and CADR

RH and CADR are the two strongest schedulers in this study, and an operator deploying one of them faces a single, well-defined choice. Both achieve statistically indistinguishable deadline miss rates (RH 9.50%, CADR 10.09% in Phase 3, $p = 0.484$ n.s.; RH 6.41%, CADR 6.24% in Phase 4), and at Phase 3 hectic saturation both post substantially lower miss than SPT+Rescue, though neither dominates it on wait: RH's wait advantage over SPT+Rescue does not survive statistical correction, and CADR's wait is essentially tied with it. Crucially, they themselves form the (miss rate, tardiness) Pareto frontier: neither dominates the other, because they diverge on one quality axis, the *severity* of the deadlines they do miss. RH demotes already-doomed jobs aggressively, so the jobs it misses are missed by a large margin (mean tardiness 30.86 min in Phase 3); CADR's gentler critical-ratio demotion cuts that severity by over 60% (6.43 min) while missing a statistically indistinguishable number of deadlines. The decision therefore reduces to how the operator's service-level agreement (SLA) penalises a missed deadline, summarised in Table 6.1.

- **Prefer RH when the penalty is per-breach and lateness-independent.** If the SLA counts each missed deadline equally regardless of how late the job finishes (for example, a fixed credit per breached rush order), RH is the correct default: it produces the fewest breaches (9.50% Phase 3, 6.41% Phase 4) together with the lowest queue wait at hectic saturation (140.24 min, tied with SPT, $p = 0.671$; at monthly scale RH's wait trails SPT, SPT+Rescue and CADR). The large tardiness of its few misses carries no extra penalty under such an SLA.

- **Prefer CADR when the penalty scales with lateness.** If late jobs incur escalating cost, for example hourly late fees or a downstream pipeline stage (compositing, review) that degrades the longer a frame is late, CADR is preferable: it holds the miss count within half a percentage point of RH while cutting mean lateness by over 60% (6.43 min vs 30.86 min Phase 3; 5.13 min vs 18.96 min Phase 4). Under any tardiness-weighted objective, CADR is the lower-penalty policy despite missing marginally more deadlines.

Table 6.1: Decision guidance for choosing between the two proposed schedulers. The choice is a service-quality choice (miss count versus miss severity); it is not a cost trade-off, because operational cost differs by only about 1.29% across all ten policies under saturation (RH $6.48 vs CADR $6.47 per 24-hour run, under 1% apart).

Operator priority	**Choose**	**Decisive metric**
Fewest breached deadlines (per-breach SLA, lateness-independent)	RH	Lowest miss (9.50% / 6.41%)
Lateness penalised (hourly late fees, latency-sensitive downstream stages)	CADR	Over 60% lower tardiness (6.43 vs 30.86 min)
Minimise queue wait at hectic saturation	RH (CADR monthly)	Best-tier Phase 3 wait (140.24 min, tied with SPT); at monthly scale choose CADR instead (101.53 min vs RH's 102.55 min)
Lowest operational cost	neither	$\sim$1.29% spread; tune $C_{\max}$, not the scheduler

Service quality versus cost. As Table 6.1 summarises, cost cannot break the tie between RH and CADR. Under the saturated hectic-day regime ($\rho > 1$), every work-conserving policy keeps the GPU fleet at the same high utilisation, so total operational cost is nearly fixed across all schedulers: the spread between the cheapest policy (SPT, $6.45 per 24-hour run) and the most expensive (LCF, $6.53) is only about 1.29%, a marginal, near-noise gap, and RH ($6.48) and SPT+Rescue ($6.49) sit inside that band. Between the two proposed schedulers the difference is under 1% (RH $6.48 vs CADR $6.47 per 24-hour run, a few cents), negligible beside the miss-count and tardiness differences. Cost therefore cannot discriminate between RH and CADR; Figure 5.16 in Chapter 5 visualises this compressed cost range for both the hectic-day and the monthly horizon. The genuine service-quality-versus-cost trade-off in this system is not located in the scheduler at all but in the concurrency limit $C_{\max}$: raising $C_{\max}$ provisions more parallel capacity, lowering

both wait and miss but raising spend (Section 5.5.3). An operator therefore tunes *quality at fixed cost* by choosing the scheduler (RH for fewest breaches, CADR for least lateness), and trades *cost against quality* by choosing $C_{\max}$.

6.2 Limitations and Future Research

6.2.1 Limitations

The following limitations bound the scope of the conclusions.

Simulation-only evaluation. Every result reported here is produced by the discrete-event simulator; no scheduler was run against the live RunPod platform, so no figure in this thesis is a field measurement. The point bites hardest on the availability model, which reproduces qualitative platform patterns rather than measured provisioning latencies (see the next limitation), so the absolute miss, wait and tardiness levels quoted throughout are outputs of that model and not observed platform behaviour. What the study claims is the *relative ordering* of the ten policies under one transparent, fully specified model, and the deployment rules of Section 6.1 should be read on those terms: an operator adopting them should expect the ranking to carry over, not the numbers. Live validation on the platform is the fourth item of the future work below, and it is the item that would turn these rules from model-supported into field-supported.

Single-source workload model. The job model rests on the submission history of one studio's rendering pipeline, and two of its ingredients inherit that pipeline's shape directly. The complexity levels κ are assigned by render-time terciles of that history, so the boundary between low, medium and high complexity is that studio's asset mix rather than a general one; and the four day-type arrival rates are set to reproduce its 14:00 to 17:00 submission peak, which is what places the busiest hours inside the model's lowest-availability band. A studio with heavier scenes, a different render farm, or an overnight submission habit would shift both, changing the tight/loose mix a scheduler faces and when the queue builds. The deadline split (20% tight / 80% loose) is a modelling assumption on top of

that, swept in Section 5.5.2 but never observed. Reproducing the comparison on a second studio's history is the cheapest available test of how far the ordering generalises.

Availability model simplification. The availability model is calibrated to qualitative empirical patterns rather than exact RunPod platform measurements, which are not publicly available. The model captures the correct qualitative behaviour (diurnal variation in stock and provisioning delay) but may not reproduce exact quantitative provisioning latencies. The availability-model sensitivity sweep (Section 5.5.8) bounds this concern: across the operable range of the sweep ($\times 0.75$ to $\times 1.25$ of the baseline high-stock probabilities) the scheduler comparison is stable, so the conclusions do not hinge on the exact calibration point. At $\times 0.50$ and below the platform becomes inoperable for every policy alike, and at $\times 1.25$ the eased regime lets several policies reach zero miss, so the RH advantage is a property of the calibrated scarcity-and-saturation regime rather than a universal ordering.

Non-preemptive assumption. CGRSP models non-preemptive scheduling. Some cloud platforms do permit early termination (spot preemption). Extending the formulation to preemptive scheduling would require tracking partial completion and migration overhead.

Single-job-per-frame model. This study models each rendering frame as a single indivisible job. In practice, some renderers support tile-based rendering (splitting a frame into spatial tiles rendered in parallel on multiple GPUs), which would require a DAG-structured job model.

Sparse strata. The 6,627-job calibration dataset is real but unevenly distributed across the twelve (κ, g) strata: the RTX A5000 dominates (4,916 jobs) and the RTX A4000/A4500 are sampled almost entirely at high complexity. Three of the twelve strata are too sparse for a direct per-stratum fit and instead use a relative-performance estimate anchored on the same GPU's well-sampled cells: low-complexity A4000 (no jobs), low-complexity A4500 (no jobs), and medium-complexity A4000 (three jobs). The practical impact is limited because these strata carry little workload weight, while the strata that receive the majority of dispatches (A5000 and RTX 3090 at all complexities, and the high-complexity A4000/A4500 cells) are each fitted directly from hundreds to thousands of observations.

No nontrivial optimality bound. The CGRSP instance sizes studied ($|\mathcal{J}| \approx 950$, $C_{\max} = 5$, $n = 30$ replications) make exact integer linear programming intractable at the required evaluation scale; no polynomial-time optimal algorithm is known for the general heterogeneous parallel machine scheduling problem with deadlines [8]. The capacity analysis of Section 5.4.1 certifies only that the offline-relaxed optimum on the expected-service instances is zero misses, so it provides only a trivial (zero) benchmark, not a meaningful gap against which the policies could be scored. Scheduler performance is therefore evaluated relative to the FIFO baseline and relative to each other. SPT's near-M/G/1-optimal mean-wait behaviour under saturation provides an indirect argument for near-optimality in the wait dimension; no such bound is available for miss rate.

Reinforcement learning not evaluated. RL-based policies (DQN, policy gradient) were not included in the comparison for three reasons: (1) training overhead, typically tens of thousands of environment interactions [24], substantially exceeds the heuristic implementation cost; (2) the scheduling environment changes frequently as RunPod updates GPU types and pricing, requiring periodic retraining; and (3) interpretable deployment rules are more operationally actionable for rendering studios than black-box policies. RL remains a direction for future work (see the reinforcement-learning item below).

6.2.2 Future Research

Eight directions for future work follow directly from the limitations above:

1. **Preemptive scheduling extension.** Incorporating checkpoint-and-resume (available in some cloud renderers) would allow modelling preemption events and evaluating EDF's theoretical guarantees in a preemptive setting.

2. **Multi-objective Pareto exploration.** The current Pareto analyses are limited to the (miss, wait) and (miss, tardiness) planes. A full three-objective Pareto analysis (waiting time, miss rate, cost) using NSGA-II or similar multi-objective evolutionary algorithms would provide a richer trade-off characterisation.

3. **Tile-based rendering job model.** Extending CGRSP to DAG-structured jobs (where a frame is decomposed into tiles) would enable scheduling research on job-level parallelism, relevant to real-time rendering pipelines.

4. **Real-platform integration.** Deploying the scheduler as a middleware layer on top of the RunPod API (using the platform's job submission endpoint) would enable live validation of the simulation results in production conditions.

5. **Newer GPU types.** The experimental fleet is limited to four GPU types available on RunPod Secure Cloud in March 2026. Extending to more recent GPU generations (H100, RTX 5090) and investigating the effect of higher $C_{\max}$ limits (available via RunPod enterprise plans) would characterise how the concurrency constraint scales beyond $C_{\max} = 5$.

6. **Deadline distribution assumptions.** The deadline distribution used in this study (20%/80% tight/loose split) is a modelling assumption. A study using real production schedules from a rendering studio would validate the choice of threshold and quantify the benefit of EDF in practice.

7. **Load-adaptive threshold policy.** The threshold sensitivity study in Section 5.5.4 shows SPT+Rescue performance is relatively insensitive to θ across the swept range (miss within 4.73 percentage points; the sweep minimum is at $\theta = 180\,\text{s}$, but the within-band differences are comparable to the sampling noise at $n = 10$). Future work could explore a load-adaptive threshold policy that adjusts θ dynamically based on current queue pressure and GPU utilisation.

8. **Reinforcement learning scheduling policies.** Deep RL methods (DQN, actor-critic) have shown promise for GPU job scheduling in data-centre contexts [22, 24]. Applying RL to CGRSP (with a state space encoding queue laxity distribution, GPU stock status, and time-of-day) could discover non-myopic dispatch policies that outperform the greedy heuristics studied here. The validated CGRSP simulator provides a ready training environment.

6.3 Concluding Remarks

Scheduling 3D rendering over on-demand GPU fleets with marketplace-driven availability has not previously been formulated or evaluated: rendering studios using RunPod or similar platforms have had no principled guidance on which scheduling policy to deploy. This thesis provides that formulation and the accompanying empirical evidence.

Within the fixed RunPod serverless concurrency constraint, the heterogeneous tight/loose deadline formulation, and the hectic overload regime ($\rho > 1$), RH achieves the lowest deadline miss rate (9.50%, $d = -0.778$ vs SPT+Rescue, $p = 0.002$) and best-tier wait (140.24 min, tied with SPT at $p = 0.671$, numerically lower than SPT+Rescue though that specific gap does not survive statistical correction), at the cost of high mean tardiness (30.86 min), now exceeding even the FIFO baseline. CADR achieves near-RH miss (10.09%) at substantially lower tardiness (6.43 min); its advantage over SPT+Rescue is concentrated on miss rate, with wait essentially tied ($p = 0.927$). SPT achieves the lowest wait (141.03 min, tied with RH, $d = -3.062$ vs FIFO, $p < 0.001$) but a significantly higher miss rate than EDF (18.60% vs 15.81%, $p = 0.002$; both effects large relative to FIFO, $p < 0.001$). SPT+Rescue is the conservative fallback: a notably lower miss rate than FIFO ($d = -2.987$) at a lower wait than FIFO as well (145.85 min vs 169.32 min, $d = -2.337$), its wait premium being over SPT rather than over FIFO. EDF significantly beats SPT on miss but now incurs a higher wait than FIFO, significant before family-wise correction ($p = 0.023$) though not after it; EDF's tardiness remains low relative to the deadline-oblivious policies, though at Phase 3 it sits second to the Adaptive hybrid within that group. FIFO/LCF/Balanced incur 24.85% to 26.14% miss and should not be

deployed when deadline compliance matters. Operators choose RH when only the count of missed deadlines matters, and CADR when the severity (lateness) of misses also matters. Increasing $C_{\max}$ beyond the serverless concurrency limit is the primary mechanism to reduce residual miss rates at saturation.

References

[1] Per H. Christensen and Wojciech Jarosz. The path to path-traced movies. *Foundations and Trends in Computer Graphics and Vision*, 10(2):103–175, 2016. doi: 10.1561/0600000073.

[2] RunPod. Pods pricing. https://docs.runpod.io/pods/pricing, 2026. Accessed: 2026-07-19.

[3] CoreWeave. Cloud pricing. https://www.coreweave.com/pricing, 2026. Accessed: 2026-07-19.

[4] Lambda. GPU cloud pricing. https://lambda.ai/pricing, 2026. Accessed: 2026-07-19.

[5] Tracy D. Braun, Howard Jay Siegel, Noah Beck, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001. doi: 10.1006/jpdc.2000.1714.

[6] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002. doi: 10.1109/71.993206.

[7] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in GPU datacenters: A survey. *ACM Computing Surveys*, 56(6):1–38, 2024. doi: 10.1145/3638757.

[8] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 5th edition, 2016. doi: 10.1007/978-3-319-26580-3.

[9] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi: 10.1145/321738.321743.

[10] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008. doi: 10.1145/1365490.1365500.

[11] Jeff Barr. New – per-second billing for EC2 instances and EBS volumes. https://aws.amazon.com/blogs/aws/new-per-second-billing-for-ec2-instances-and-ebs-volumes/, 2017. AWS News Blog, published 2017-09-18. Accessed: 2026-07-19.

[12] Sangho Yi, Derrick Kondo, and Artur Andrzejak. Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 236–243, 2010. doi: 10.1109/CLOUD.2010.35.

[13] Supreeth Subramanya, Tian Guo, Prateek Sharma, David Irwin, and Prashant Shenoy. SpotOn: A batch computing service for the spot market. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 329–341, 2015. doi: 10.1145/2806777.2806851.

[14] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, and Xinyu Wang. How to bid the cloud. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pages 71–84, 2015. doi: 10.1145/2785956.2787473.

[15] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation*, pages 481–498, 2020.

[16] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018. doi: 10.1145/3190508.3190517.

[17] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979. doi: 10.1016/S0167-5060(08)70356-X.

[18] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1–2):59–66, 1956. doi: 10.1002/nav.3800030106.

[19] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

[20] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969. doi: 10.1287/mnsc.16.1.77.

[21] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley, 10th edition, 2018. ISBN 978-1-119-32091-3.

[22] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016. doi: 10.1145/3005745.3005750.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

[24] Kexuan Chen and Jiayi Chen. GPU job scheduling based on deep reinforcement learning. In *Proceedings of the 2023 7th International Conference on High Performance Compilation, Computing and Communications*, pages 34–45, 2023. doi: 10.1145/3606043.3606049.

[25] Guangyao Zhou, Wenhong Tian, Rajkumar Buyya, Ruini Xue, and Liang Song. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *Artificial Intelligence Review*, 57(5):124, 2024. doi: 10.1007/s10462-024-10756-9.

[26] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Pearson, 5th edition, 2010.

[27] Robert G. Sargent. Verification and validation of simulation models. *Journal of Simulation*, 7(1):12–24, 2013. doi: 10.1057/jos.2012.20.

[28] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 3rd edition, 2000.

[29] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. MIT Press, 4th edition, 2023.

[30] RunPod. Serverless workers overview: Max worker limits. https://docs.runpod.io/serverless/workers/overview#max-worker-limits, 2026. Accessed: 2026-03-01.

[31] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi: 10.2307/3001968.

[32] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition, 1988.

[33] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.