國立臺灣科技大學
National Taiwan University of Science and Technology
Department of Computer Science and Information Engineering
Master's Thesis

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

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

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

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science
in Computer Science and Information Engineering

June 2026
中華民國 115 年 6 月

國立臺灣科技大學資訊工程系
Department of Computer Science and Information Engineering

碩士論文
Master's Thesis

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

林墾森　Michael Liem

指導教授:　黃元欣教授 (Prof. Yuan-Shin Hwang)
共同指導教授:　喻奉天教授 (Prof. Vincent F. Yu)

中華民國 115 年 6 月
June 2026

碩士學位論文指導教授推薦書

Master's Thesis Recommendation Form

系所:	資訊工程系
Department/Graduate Institute:	Department of Computer Science and Information Engineering
姓名:	林墾森
Name:	Michael Liem
論文題目:	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
Thesis Title:	Cloud GPU Rendering Scheduling Problem (CGRSP): A Mathematical Formulation and Empirical Evaluation on Heterogeneous GPU Fleets

係由本人指導撰述，同意提付審查。

This is to certify that the thesis submitted by the student named above has been written under my supervision. I hereby approve this thesis to be applied for examination.

指導教授簽章： Advisor's Signature _____

共同指導教授簽章（如有）： Co-advisor's Signature (if any) _____

日期： Date (yyyy/mm/dd) _____ / _____ / _____

碩士學位考試委員審定書

Qualification Form by Master's Degree Examination Committee

系所:	資訊工程系
Department/Graduate Institute:	Department of Computer Science and Information Engineering
姓名:	林墾森
Name:	Michael Liem
論文題目:	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
Thesis Title:	Cloud GPU Rendering Scheduling Problem (CGRSP): A Mathematical Formulation and Empirical Evaluation on Heterogeneous GPU Fleets

經本委員會審定通過，特此證明。

This is to certify that the thesis submitted by the student named above is qualified and approved by the Examination Committee.

學位考試委員會 **Degree Examination Committee**

員簽章 Member's Signatures:

_____ _____

_____ _____

召集人簽章: Committee Chair's Signature _____

指導教授簽章: Advisor's Signature _____

共同指導教授簽章（如有）: Co-advisor's Signature (if any) _____

系所（學程）主任（所長）簽章: Dept. Chair's Signature _____

日期: Date (yyyy/mm/dd) _____ / _____ / _____

國立臺灣科技大學

National Taiwan University of Science and Technology

學位論文授權書

Thesis/Dissertation Authorization Form

本授權書所授權之論文為本人在國立臺灣科技大學資訊工程系所完成之學位論文。

論文名稱：	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
英文題名：	Cloud GPU Rendering Scheduling Problem (CGRSP): A Mathematical Formulation and Empirical Evaluation on Heterogeneous GPU Fleets
研究生姓名：	林墾森 (Michael Liem)
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)
學位類別：	碩士 (Master of Science)
畢業年月：	中華民國 115 年 6 月 (June 2026)

本人茲將上列論文授權國立臺灣科技大學圖書館：

☐ 同意本論文於校內網路立即公開，校外網路自畢業後一年公開。
☒ 同意本論文立即公開（校內外網路均立即公開）。
☐ 本論文暫不公開，申請延後 _____ 年公開（最長以三年為限）。

研究生簽名：_____ 日期：中華民國 年 月 日

指導教授簽名：_____ 日期：中華民國 年 月 日

Note: This authorization form must be signed by the student and advisor, and submitted to the NTUST Library along with the bound thesis. Please tick the appropriate radio button above to indicate the desired public access level.

摘要

論文題目：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

系所名稱：國立臺灣科技大學資訊工程系

學生姓名：林墾森 (Michael Liem)

指導教授：黃元欣教授 (Prof. Yuan-Shin Hwang)

共同指導教授：喻奉天教授 (Prof. Vincent F. Yu)

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（7.54%）且等待時間與 SPT 並列最佳（128.83 分鐘），全面優於 SPT+Rescue（違反率 $d = -1.231$，$p < 0.001$；等待時間 $d = -0.578$，$p = 0.003$）。RH 之預期機制受負載閾值控制：飽和以下時為即將到來的緊急（急件）任務保留閒置容量，將湧浪日違反率自 6.00% 降至 1.37%；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（7.88%），同時大幅降低 RH 之平均延遲時間（6.08 分鐘，相較 RH 之 20.30 分鐘），在壅塞飽和下於違反率與等待時間兩項指標皆全面優於 SPT+Rescue。SPT 達成最低等待時間（129.41 分鐘），與 EDF 在違反率上統計無顯著差異（13.85% vs. 11.82%，$p = 0.075$ n.s.）；SPT+Rescue 提供相近違反率（12.40%）並具保守截止期限保護。FIFO、LCF 與 Balanced 因忽略截止期限而產生約 22% 違反率。湧浪日負載下，緊急截止期限比例 $\leq 30\%$ 時 EDF 最低；$\geq 50\%$ 時 SPT 最低。

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 作為預設策略（最低違反率且等待時間最佳）；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 仍為低等待之強健基準；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

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

Department: Department of Computer Science and Information Engineering, NTUST

Student: Michael Liem (林墾森)

Advisor: Prof. Yuan-Shin Hwang

Co-Advisor: Prof. Vincent F. Yu

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 (7.54%) and best-in-class wait (128.83 min), Pareto-dominating SPT+Rescue on both QoS metrics (miss $d = -1.231$, $p < 0.001$; wait $d = -0.578$, $p = 0.003$). RH's anticipation is load-gated: below saturation it reserves idle capacity for imminent tight (rush-order) arrivals, cutting surge-day miss from 6.00% to 1.37%, while the offered-load gate disables reservation under saturation so the saturated-load results are unchanged. CADR achieves near-RH miss (7.88%) while cutting RH's mean tardiness substantially (6.08 min vs 20.30 min), Pareto-dominating SPT+Rescue on both miss and wait at hectic saturation. SPT achieves the lowest wait (129.41 min, tied with RH) and is statistically tied with EDF on miss rate (13.85% vs. 11.82%, $p = 0.075$ n.s.), and SPT+Rescue achieves similar miss (12.40%) with conservative deadline protection. FIFO, LCF, and Balanced incur $\approx$22% miss from deadline ignorance. At surge-day load, EDF achieves the lowest miss when tight-deadline fraction is $\leq$30%; SPT achieves the lowest when $\geq$50%.

These findings yield deployment rules for cloud rendering operators: deploy RH when minimising the count of missed deadlines is the sole priority (lowest miss, best-in-class wait); 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
June 2026

CONTENTS

指導教授推薦書 (Advisor Recommendation) **ii**

學位考試委員審定書 (Committee Approval) **iii**

授權書 (Authorization Letter) **iv**

摘要 (Chinese Abstract) **v**

Abstract **vi**

Acknowledgements (誌謝) **viii**

符號索引 (List of Symbols) **xii**

1 Introduction **1**

1.1 Research Background 1
1.2 Research Objective 2
1.3 Research Contribution 3
1.4 Scope and Limitations 4
1.5 Organisation of Thesis 5

2 Background and Related Work **6**

2.1 Cloud GPU Computing 6
2.2 Classical Job Scheduling 7
2.2.1 Shortest Processing Time (SPT) 7
2.2.2 Earliest Deadline First (EDF) 7
2.2.3 First-Come-First-Served (FIFO) 8
2.2.4 Other Heuristics 8
2.3 Heterogeneous Machine Scheduling 8
2.4 Reinforcement Learning for Resource Management 9
2.5 Discrete-Event Simulation 10
2.6 3D Rendering Workloads 10
2.7 Summary 11

3 Problem Formulation and Mathematical Model **13**
3.1 Mathematical Notation . . . 13
3.2 System Model and State Representation . . . 13
3.3 Assignment Constraints and Decision Variables . . . 17
3.4 Deadline Penalty and Soft Deadlines . . . 17
3.5 QoS Metrics . . . 18
3.6 Optimisation Objective . . . 18
3.7 Numerical Example . . . 19
3.8 Modelling Simplifications and Assumptions . . . 21
3.9 Problem Complexity and Tractability . . . 22

4 System Design and Scheduling Algorithms **23**
4.1 System Architecture Overview . . . 23
4.2 Discrete-Event Simulator Core . . . 23
4.3 GPU Fleet Model . . . 24
4.4 Job Generator . . . 25
4.5 Scheduling Algorithms . . . 26
4.5.1 FIFO: First-In-First-Out . . . 26
4.5.2 SPT: Shortest Processing Time . . . 26
4.5.3 EDF: Earliest Deadline First . . . 27
4.5.4 LCF: Lowest Cost First . . . 27
4.5.5 Balanced: Weighted Throughput-Cost Composite . . . 28
4.5.6 Random: Random Baseline . . . 28
4.5.7 Design Progression . . . 28
4.5.8 SPT+Rescue: Hybrid Throughput-Deadline Policy . . . 28
4.5.9 Illustrative Scheduling Example . . . 30
4.5.10 Rolling-Horizon: Anticipative Receding-Horizon Scheduler . . . 30
4.5.11 CADR: Cost-Aware Deadline-Risk . . . 34
4.5.12 Adaptive: Queue-Pressure EDF-SPT Hybrid . . . 35
4.6 Metrics Collection . . . 36

5 Experimental Results and Analysis **38**
5.1 Experimental Setup . . . 38
5.1.1 Platform and Data . . . 38
5.1.2 Evaluation Phases . . . 42
5.1.3 Statistical Methodology . . . 42
5.2 Phase 2: Seven-Scheduler Benchmark . . . 43
5.3 Phase 3 Statistical Validation . . . 45
5.4 Ablation Analysis . . . 48
5.5 Sensitivity Analysis . . . 50

5.5.1 Day-Type Sensitivity ($C_{\max} = 5$) 50
5.5.2 Deadline Tightness Sensitivity 51
5.5.3 Concurrency Limit Sensitivity 52
5.5.4 Rescue Threshold Sensitivity 53
5.5.5 Critical-Ratio Threshold Sensitivity 54
5.5.6 Anticipation Reservation Sensitivity 55
5.5.7 Objective-Weight Sensitivity 55
5.6 Phase 4: Monthly Operational Simulation 58
5.7 Summary of Findings 62

6 Discussion and Future Work **63**
6.1 Theoretical Implications 63
6.1.1 Choosing Between RH and CADR 66
6.2 Limitations and Future Research 67
6.2.1 Limitations 67
6.2.2 Future Research 68
6.3 Concluding Remarks 69

References **71**

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 GPU types
$\mathcal{K}$	Set of discrete complexity levels
j	Job index
m	Node index
g	GPU type index
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^{submit}	Submission 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)
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
$m(t)$	Time-of-day stock multiplier at time t
τ_j^{prov}	Provisioning delay incurred when job j acquires an instance (stock-status dependent)
$C_{\max}$	Maximum simultaneous GPU instances (concurrency limit)
λ	Job arrival rate (jobs/second)
ρ	System utilisation ratio: $\lambda \bar{W} / C_{\max}$
$x_j^m \in \{0, 1\}$	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: $\max(0, t_j^{\text{complete}} - d_j)$

Symbol	Description
c_j	Execution cost of job j
Δ_{miss}	Deadline miss rate: $\frac{1}{\lvert\mathcal{J}\rvert}\sum_j(1-\delta_j)$
$w_{\text{qos}}, w_{\text{cost}}$	Objective function weights
α_1, α_2	QoS penalty weights (waiting time, deadline violation)
θ	SPT+Rescue laxity threshold (600 s)
w_c	SPT-family node-selection cost weight (0.3)
$L_j(t)$	Laxity of job j at time t: $d_j - t - W(\kappa_j, g)$
ρ_{res}	Rolling-Horizon reservation aggressiveness (strength of anticipation)

LIST OF FIGURES

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 the fastest available 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. The outcomes recorded per job are waiting time w_j, deadline adherence δ_j, and rental cost. 14

3.2 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. 20

4.1 Illustrative CGRSP Gantt chart ($C_{\max} = 3$ nodes, 7 jobs; only J6 has a tight deadline $d_6 = 210$ s; execution times rounded 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 $L_{J_6}(70) = 65$ s < 600 s threshold triggers rescue; J6 dispatched to N1 at $t = 70$ s, finishes at 145 s, meets deadline by 65 s ($\delta_6 = 1$). 31

5.1 Empirical rendering job dataset (6,627 jobs, July 2025 to June 2026). (a) Hourly arrival counts: the 06:00–09:00 UTC 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. 39

5.2 CGRSP system architecture: four 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. 39

5.3 Diurnal availability model and real job arrival pattern. (a) The parametric step function $m(t)$ applied to GPU high-stock probabilities: peak hours 09:00–18:00 UTC reduce availability by half (×0.5, red); morning 06:00–09:00 moderately reduces it (×0.9, orange); off-peak 18:00–24:00 raises it (×1.3, green); evening 00:00–06:00 is neutral (×1.0, blue). 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 $m(t)$ band. The arrival peak at 06:00–09:00 UTC aligns with the morning band, confirming that the simulator's day-type calibration reflects real operational load. 41

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 sub-10-minute wait, 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. On the hectic day SPT achieves the lowest wait (135.29 min) and competitive miss (17.16%); SPT+Rescue reduces FIFO miss from 23.58% to 15.79% at 146.81 min wait; EDF achieves 5.16% miss but at 147.84 min wait; FIFO, LCF, and Balanced ignore deadlines and incur the highest miss (FIFO 23.58%). 44

5.5 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 ($\sim$950 jobs, $\rho > 1$): RH lowest miss (7.5%, 128.8 min wait, tied with SPT for lowest wait); CADR near-RH miss (7.9%, 132.8 min wait); SPT lowest wait (129.4 min, 13.8% miss); FIFO/LCF/Balanced $\approx$23% miss. Surge days: Adaptive, EDF, and RH tied for lowest miss (1.3% / 1.7% / 1.4%); SPT+Rescue 3.1%; deadline-unaware schedulers 10 to 11%. . . . 50

5.6 Deadline miss rate (%) vs tight-deadline fraction (surge day, $C_{\max} = 5$, 730 jobs, 30 seeds). EDF achieves the lowest miss rate at $f_{\text{tight}} \leq 30\%$; SPT achieves the lowest at $f_{\text{tight}} \geq 50\%$. 52

5.7 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 152 to 190 min (capacity saturation, $\rho > 1$); surge days 91 to 104 min; normal days 4 to 6 min; quiet days $<$1 min. (b) Monthly miss rate: RH 5.14% and CADR 5.03% (tied for lowest miss), SPT+Rescue 6.99%, SPT 8.97%; EDF 6.32%; Random 13.99%; FIFO/LCF/Balanced 15.01 to 14.91%. 59

5.8 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 (5.14% and 5.03%), RH at the cost of the highest tardiness among the anticipative schedulers (12.47 min) and CADR at substantially lower tardiness (4.43 min), sitting at the low-tardiness elbow of the Pareto frontier; the Adaptive hybrid and EDF achieve the lowest tardiness (1.90 min and 2.00 min) but at higher miss rates (6.32% for EDF). Operators choose RH when only the count of missed deadlines matters, and CADR when the severity (lateness) of misses also matters. 60

5.9 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) consistently produce the lowest spikes on hectic and surge days; FIFO (grey) peaks highest. 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). 60

LIST OF TABLES

2.1 Positioning of CGRSP against related scheduling studies 11

3.1 Sets and Indices 14
3.2 System Parameters 15
3.3 Decision Variables and State Variables 15

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 [1]) as the baseline concurrency limit. 25

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: $\rho > 1$) 43
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 $\rho > 1$) 45
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. . 46
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, and CADR 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). 48
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. 50
5.6 Deadline miss rate (%) vs tight-deadline fraction, surge day, $C_{\max} = 5$, $\lambda = 0.020$ j/s 51
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. 52

5.8 SPT+Rescue threshold sensitivity: avg wait (min) and miss% on hectic day ($n = 10$ seeds, $C_{\text{max}} = 5$, 20% tight 1h / 80% loose 8h deadlines). Minimum miss rate row is **bold**. 53
5.9 Critical-ratio threshold sensitivity for CADR: avg wait (min), miss%, and mean tardiness (min) on hectic day ($n = 10$ seeds, $C_{\text{max}} = 5$). Minimum miss rate row is **bold**. 54
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, $C_{\text{max}} = 5$. The lowest miss in each day group is in bold. 55
5.11 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. 56
5.12 Tardiness-objective re-scoring: schedulers ranked by mean per-job tardiness under a continuous tardiness objective (lower is better), hectic day ($n = 10$ seeds, $C_{\text{max}} = 5$). Compare with the miss-indicator objective in Table 5.11. 58
5.13 Phase 4 monthly simulation, $C_{\text{max}} = 5$, 30 days, 10 replications, 20% tight 1h / 80% loose 8h deadlines 59
5.14 Summary of key experimental findings 62

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 1.46% across all ten policies under saturation (RH $6.09 vs CADR $6.13 per 24-hour run, under 1% apart). 67

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, architecture 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. The consequent demand for computational power has driven the 3D rendering industry toward 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, billed by the second. 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 [2, 3]. 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 [4] focus primarily on deep learning training workloads, where the access model (reserved clusters or dedicated hardware) differs fundamentally from the spot-market 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 they 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 (RTX 3090 at \$0.46/hr) delivers the best turnaround but at the highest per-hour cost, whereas the cheapest GPU types (RTX A4000 and RTX A4500, both at \$0.25/hr) reduce cost while increasing execution time. A scheduler must navigate this trade-off explicitly.

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 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: 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.

1.3 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 6,627 real RunPod Secure Cloud rendering jobs (Chapter 4).
- A statistically rigorous comparison of ten scheduling algorithms (eight benchmarks, including an adaptive EDF-SPT hybrid, plus the novel anticipative Rolling-Horizon (RH) scheduler and the Cost-Aware Deadline-Risk (CADR) scheduler) on hectic-day workload ($\sim$950 jobs/day, $C_{\max} = 5$, $n = 30$ seeds, 20% tight 1h / 80% loose 8h deadlines; capacity saturation $\rho > 1$), with paired t-tests confirming SPT as the lowest-wait scheduler: 129.41 min wait ($d = -2.882$ vs FIFO, $p < 0.001$) and 13.85% miss ($d = -2.197$ vs FIFO, $p < 0.001$), statistically tied with EDF on miss (11.82%, $p = 0.075$ n.s. vs. SPT); SPT+Rescue achieves similar miss (12.40%, $d = -2.899$ vs FIFO, $p < 0.001$) with higher wait (135.21 min, $d = -2.406$, $p < 0.001$), the conservative fallback for tight-deadline compliance; EDF achieves near-SPT miss (11.82%, $d = -3.138$, $p < 0.001$) but no wait improvement over FIFO ($d = -0.062$, n.s.); FIFO/LCF/Balanced incur $\approx$22% miss (Chapter 5).
- A day-type sensitivity study (quiet/normal/hectic/surge, 30 seeds each) showing that on hectic days ($\sim$950 jobs/day, $\rho > 1$) RH achieves the lowest miss (7.5%, 128.8 min

wait, essentially tied with SPT for lowest wait); SPT achieves the lowest wait (129.4 min, 13.8% miss); SPT+Rescue (135.2 min, 12.4%); FIFO/LCF/Balanced incur ≈23% miss. On surge days (730 jobs/day), the Adaptive hybrid, EDF, and RH are tied for the lowest miss (1.3% / 1.7% / 1.4%), with SPT+Rescue (3.1%) close behind (Chapter 5).

- A deadline-tightness sensitivity study under surge-day conditions ($C_{\max} = 5$, $\lambda = 0.020$ j/s, 730 jobs/day, 30 seeds each) showing EDF achieves the lowest miss rate when tight fraction $\leq 30\%$ (0.57% to 4.14%) and SPT achieves the lowest when $\geq 50\%$ (17.36% to 35.46%); 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): RH and CADR are essentially tied for the lowest monthly miss (RH 5.14%, CADR 5.03%), with RH's wait (95.22 min) tied with SPT for best-in-class and CADR achieving substantially lower tardiness (4.43 min vs 12.47 min); SPT and SPT+Rescue are virtually tied with each other on miss (SPT 8.97%, SPT+Rescue 6.99%) at 93.94 min and 95.51 min wait respectively; EDF achieves 6.32% miss at 107.56 min wait; FIFO/LCF/Balanced cluster at ≈14.91 to 15.01% miss. The $C_{\max} = 5$ RunPod serverless limit is the binding throughput constraint (Chapter 5).
- **Two complementary deadline-aware schedulers:** (i) the Rolling-Horizon (RH) scheduler that, 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 (7.54%) with best-in-class wait, Pareto-dominating SPT+Rescue; and (ii) the Cost-Aware Deadline-Risk (CADR) scheduler that classifies jobs by a scale-free critical ratio and selects the cheapest deadline-feasible node, achieving near-RH miss (7.88%) while cutting RH's mean tardiness substantially (6.08 min vs 20.30 min) and Pareto-dominating SPT+Rescue on both miss and wait at hectic saturation. Operators choose RH when only the count of missed deadlines matters, and CADR when the severity (lateness) of misses also matters (Chapter 5).

1.4 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 6,627 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 $C_{\max} = 5$, the default RunPod serverless account limit, 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. 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.

1.5 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, the stochastic availability model, assignment constraints, temporal constraints, the bi-objective optimisation objective, 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 baseline policies; RH and CADR 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 [5] 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 by the hour with a minimum of one hour per session. 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.

By the early 2020s, a second tier of GPU rental markets, 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 makes 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, maintenance, 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 spot-market GPU rental from the managed-cluster context of most scheduling literature is rarely made explicit. **Reserved-cluster schedulers**, including HEFT [3], Gavel [6], and Optimus [7], operate on GPU pools under the opera-

tors exclusive control; node availability is guaranteed and the scheduling problem reduces to job-to-node assignment. **Spot-market 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.

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. Graham et al. [8] established the three-field notation $\alpha|\beta|\gamma$ for scheduling problems. For CGRSP, the problem is closest to $P_m|\text{online}, r_j, d_j|\sum w_j C_j$ (parallel identical machines with online arrivals, release times, deadlines, and weighted completion time), except that the machines in CGRSP are not identical, they differ in speed and cost, making it a generalisation to heterogeneous parallel machines (Q_m).

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 [9] 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. In online settings with Poisson arrivals, SPT minimises mean response time (a classical result from M/G/1 queueing theory [10]). 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 [11] and later formalised by Lawler and Moore [12], prioritises the ready job with the nearest deadline. EDF is optimal for minimising the number of deadline violations on a single preemptive machine (Liu and Layland, 1973 [13]). On non-preemptive parallel machines, the setting of CGRSP, EDF does not provide a worst-case guarantee for deadline violation minimisation, but empirically it achieves lower miss than shortest-job-first policies (SPT) when the tight-deadline fraction is low ($\leq 30\%$); the advantage reverses at moderate-to-high tight fractions ($\geq 50\%$), where SPT outperforms EDF, as quantified by the Phase 3 deadline sensitivity experiments in Chapter 5.

2.2.3 First-Come-First-Served (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 been the default scheduling policy in many computing systems since the earliest batch processing mainframes. 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 [10]), 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 with consistently low stock (high preemption probability) may actually increase cost through wasted partial executions. This is addressed in the CGRSP implementation through a proportional stock 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 $\alpha \in [0, 1]$ controls the trade-off: $\alpha \to 1$ approaches SPT behaviour (speed priority), $\alpha \to 0$ approaches LCF (cost priority). In the CGRSP implementation, $\alpha = 0.8$ (speed-biased) is used, with global-max normalisation to prevent one criterion from dominating due to differing numerical scales. 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. [2] provide 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. [3] 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. Zhao et al. [4] survey 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. [7] propose 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. [6] propose 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. [14], 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, discrete action space (one action per job in the queue), state derived from job size histogram and resource occupancy, is conceptually related to meta-scheduler approaches for cloud GPU scheduling.

Subsequent work applied deep Q-networks (DQN) [15] to various scheduling domains. Li and Xiao [16] specifically apply DQN 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 require tens of thousands of training episodes on simulated workloads, a substantially larger training budget than the heuristic-only approach pursued in this thesis.

Zhang et al. [17] survey 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 [18]. 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.

Law and Kelton [19] establish 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.

Sargent [20] identifies 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

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 [21], 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. 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: low-complexity jobs average 60 to 62 seconds across GPU types, while high-complexity jobs average 89 to 101 seconds. 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 spot-market platforms as of early 2026. They cover a factor-of-$\sim$1.8 in rental price ($0.25 to $0.46/hr) and a factor of $\sim$1.3 in average rendering speed (74.0 to 97.9 seconds averaged across all complexity levels, see Table 4.1), providing meaningful heterogeneity for the scheduling decision.

2.7 Summary

Table 2.1 summarises the positioning of CGRSP relative to closely related work. CGRSP is unique in combining all four distinguishing features: (1) real-world calibrated GPU heterogeneity, (2) stochastic qualitative availability (spot-market model), (3) per-second billing with deadline constraints, and (4) rigorous statistical evaluation across multiple seeds and fleet sizes.

Table 2.1: Positioning of CGRSP against related scheduling studies

Work	GPU Hetero.	Stoch. Avail.	Deadlines	Stat. Valid.
Braun et al. [2]	Synthetic	×	×	×
HEFT [3]	Yes	×	×	×
Mao et al. [14]	×	×	×	✓
Li & Xiao [16]	Yes	×	×	Limited
Zhao et al. [4]	Yes (survey)	×	Partial	×
Narayanan et al. [6] (Gavel)	Yes	×	×	✓
CGRSP (this thesis)	✓	✓	✓	✓

Beyond the four dimensions in Table 2.1, CGRSP is the first scheduling study applied specifically to *batch 3D rendering workloads*. Compared to deep learning training, the dominant focus of GPU scheduling literature [6, 14, 16], rendering frames have three properties that shape the scheduling design: (1) *short and non-preemptive execution*: 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; (2) *complexity observable at submission*: scene complexity level (κ) is known when a job enters the queue, providing

the execution-time estimate that SPT and SPT+Rescue rely on; and (3) *non-monotone GPU performance 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 N nodes across $|G|$ 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 nodes type is provisioned for the duration of the job and billed per second of execution. The scheduler's objective is bi-objective: minimise QoS penalties (waiting time and deadline violations) as the primary criterion and per-second operational cost as a secondary, tiebreaking criterion, subject to node availability and assignment constraints.

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.

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.

3.2 System Model and State Representation

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

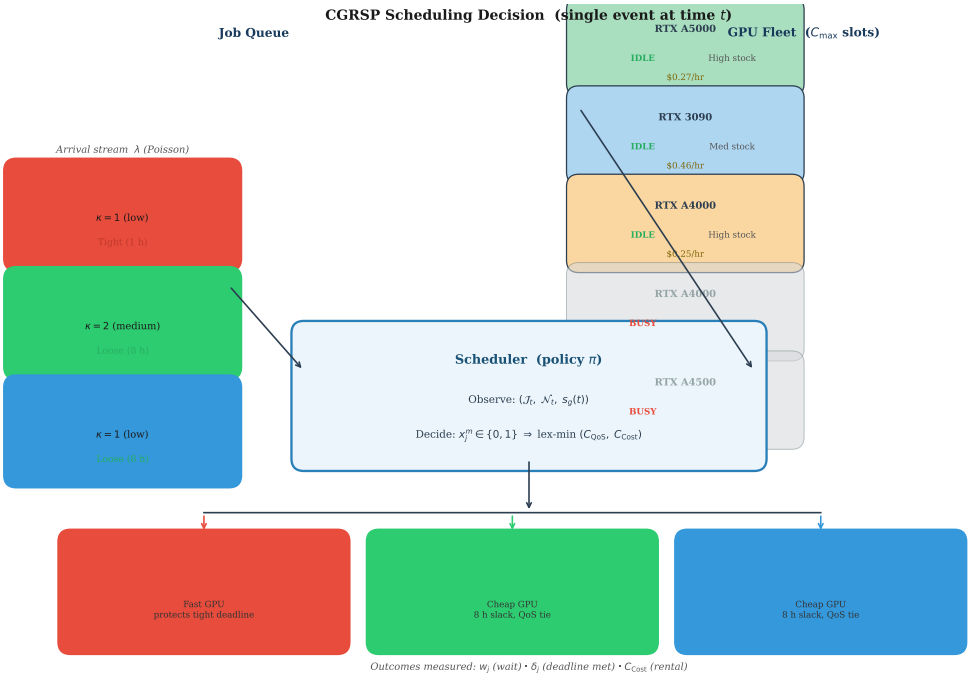

Figure 3.1: CGRSP scheduling decision at a single event. The scheduler observes the job queue (jobs j_1–j_3 with complexity κ and deadline class) and the GPU fleet ($C_{\max}$ slots with type, stock status, and occupancy). It assigns jobs to idle slots under the lexicographic objective: QoS first (minimise wait and deadline misses), then cost as a tiebreaker. Here j_1 carries a tight deadline and is routed to the fastest available 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. The outcomes recorded per job are waiting time w_j, deadline adherence δ_j, and rental cost.

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
$j \in \mathcal{J}$	Job index
$m \in \mathcal{N}$	Node index
$g \in \mathcal{G}$	GPU type index
$\kappa \in \mathcal{K}$	Complexity level index
$t \in \mathbb{R}^+$	Continuous time variable

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)
c_g	Cost factor for resource type g
p_g	Per-second rental price of resource type g
κ_j	Complexity level of job j
d_j	Deadline for job j
t_j^{submit}	Submission time of job j
t_j^{start}	Start time of job j
t_j^{complete}	Completion time of job j
$W(j, g)$	Execution time of job j on resource type g
β_g	Baseline high-stock probability of resource type g
$m(t)$	Time-of-day stock multiplier at time t
τ_j^{prov}	Provisioning delay incurred when job j acquires an instance (stock-status dependent)
$T_{\text{tight}}, T_{\text{loose}}$	Per-job deadline window: tight class (rush orders) or loose class (standard renders); platform values and class proportions in Section 5.1.1
$w_{\text{qos}}, w_{\text{cost}}$	Objective function weights
α_1, α_2	Bi-objective function parameter weights

Table 3.3: Decision Variables and State Variables

Symbol	Description
$x_j^m \in \{0, 1\}$	Assignment: 1 if job j is assigned to node m, 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)
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

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

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 (Section 3.2) and additionally biases node selection (Chapter 4) away from scarce or unreliable types.

Provisioning Dynamics (Simulator Internal): 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 (consistent with Remark 3). Market scarcity therefore does not manifest as denial of service but as a *provisioning delay* incurred when a fresh instance is acquired. Each resource type g carries a stock status $s_g(t) \in \{\text{High}, \text{Medium}, \text{Low}\}$ drawn by comparing a single uniform variate $r \sim \text{Uniform}(0, 1)$ against cumulative thresholds set by a baseline high-stock probability β_g and a time-of-day multiplier $m(t)$:

$$s_g(t) = \begin{cases} \text{High} & \text{if } r < p_g^H(t), \\ \text{Medium} & \text{if } p_g^H(t) \le r < p_g^H(t) + p_g^M(t), \\ \text{Low} & \text{otherwise}, \end{cases} \tag{3.1}$$

where $p_g^H(t) = \min(0.95,\, \beta_g\, m(t))$ and $p_g^M(t) = \min(0.90,\, 1.5\, \beta_g\, m(t))$. Outside the midday peak band the multiplier keeps $p_g^H + p_g^M \ge 1$ 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 for the lower-β_g types. 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. The multiplier $m(t)$ encodes diurnal demand (off-peak $\times 1.3$, morning $\times 0.9$, peak $\times 0.5$, evening $\times 1.0$): high-stock probability falls during business hours and recovers off-peak. The per-type β_g and the calibration source (77 RunPod pricing snapshots) are given in Chapter 5. When job j is dispatched to a type-g slot at time t, it

incurs a provisioning delay

$$\tau_j^{\text{prov}} \sim \text{Uniform}\big(L_{s_g(t)},\ U_{s_g(t)}\big) \tag{3.2}$$

whose range depends on the stock status (High $(0, 10)$ s, Medium $(30, 120)$ s, Low $(600, 7200)$ s), so the realised start time is $t_j^{\text{start}} = t_j^{\text{dispatch}} + \tau_j^{\text{prov}}$ and the delay forms a component of the waiting time w_j. Because the Low state is entered only rarely (Eq. 3.1), realised provisioning delays remain in the High and Medium ranges (at most two minutes) for the overwhelming majority of dispatches under the calibrated operating point.

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 $g \in G$): Capacity planning decision
- **Objective weights** ($w_{\text{qos}}, w_{\text{cost}}$): Business priority between QoS and cost efficiency

3.3 Assignment Constraints and Decision Variables

The decision variable $x_j^m \in \{0, 1\}$ represents the assignment of job j to node m. Valid assignments must satisfy:

$$\sum_{m \in \mathcal{N}} x_j^m \leq 1, \quad \forall j \in \mathcal{J} \tag{3.3}$$

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

$$x_j^m \leq a_m^t, \quad \forall j, m \tag{3.5}$$

These constraints ensure feasible assignments respecting the concurrency limit and slot occupancy: a job may only be placed on a currently idle slot ($a_m^t = 1$). 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.

3.4 Deadline Penalty and Soft Deadlines

Each job j carries a per-job deadline window $T_j \in \{T_{\text{tight}}, T_{\text{loose}}\}$, yielding an absolute deadline $d_j = t_j^{\text{submit}} + T_j$. 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 arrival intensity exceeds service capacity ($\rho > 1$), misses are structurally unavoidable for some

jobs regardless of scheduling policy, so a penalty formulation is the appropriate model. The tardiness of job j is:

$$\tau_j = \max(0,\, t_j^{\text{complete}} - d_j) \tag{3.6}$$

The deadline-miss indicator $\delta_j = \mathbf{1}[t_j^{\text{complete}} \leq d_j]$ (Eq. 3.9) equals 1 when the deadline is met and 0 otherwise; the miss rate is $(1/|\mathcal{J}|)\sum_j(1-\delta_j)$.

Deadline violations are penalised in the objective via the binary miss indicator $(1-\delta_j)$ rather than via continuous tardiness τ_j. Tardiness τ_j (Eq. 3.6) is defined for completeness but is not used in the objective. The specific values of T_{tight} (rush-order renders) and T_{loose} (standard batch renders), and the proportion of jobs assigned to each deadline class, are platform parameters defined in Section 5.1.1.

Tardiness (mean and 95th percentile) 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 5.5.7) confirms that re-scoring schedulers against a tardiness-based objective yields the same scheduler ranking as the miss-indicator objective, validating that the conclusions are not sensitive to this choice of deadline-penalty form.

3.5 QoS Metrics

Quality of service is measured through three key metrics:

$$w_j = t_j^{\text{start}} - t_j^{\text{submit}} \quad \text{(waiting time)} \tag{3.7}$$

$$r_j = t_j^{\text{complete}} - t_j^{\text{submit}} \quad \text{(response time)} \tag{3.8}$$

$$\delta_j = \mathbf{1}[t_j^{\text{complete}} \leq d_j] \quad \text{(deadline adherence)} \tag{3.9}$$

3.6 Optimisation Objective

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 spans only \$0.25 to \$0.46 per hour, 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,

$$\text{lex min}\ \big(C_{\text{QoS}},\ C_{\text{Cost}}\big), \qquad C_{\text{QoS}} = \sum_{j\in\mathcal{J}} \big[\, \alpha_1\, w_j + \alpha_2\,(1-\delta_j)\,\big], \tag{3.10}$$

where lexicographic minimisation prefers any policy with lower C_{QoS} and breaks QoS ties on lower C_{Cost}. The QoS cost combines a waiting-time penalty and a deadline-violation penalty: w_j is the waiting time in seconds (Eq. 3.7), 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 5.5.7 to verify robustness. 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.

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 5.5.7), which additionally evaluates a fully symmetric weighted-sum scalarisation $\min \sum_j \left[w_{\text{qos}} C_{\text{QoS}}(j) + w_{\text{cost}} C_{\text{Cost}}(j) \right]$ 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.

The operational cost component reflects per-second rental fees for the provisioned GPU instances across all nodes:

$$C_{\text{Cost}} = \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{N}} \left[x_j^m \times W(j, \text{type}(m)) \times p_{\text{type}(m)} \right] \tag{3.11}$$

where $W(j, \text{type}(m))$ is the realised execution time of job j on the assigned node's type (Remark 1).

3.7 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 3.2 illustrates the assignment and timeline.

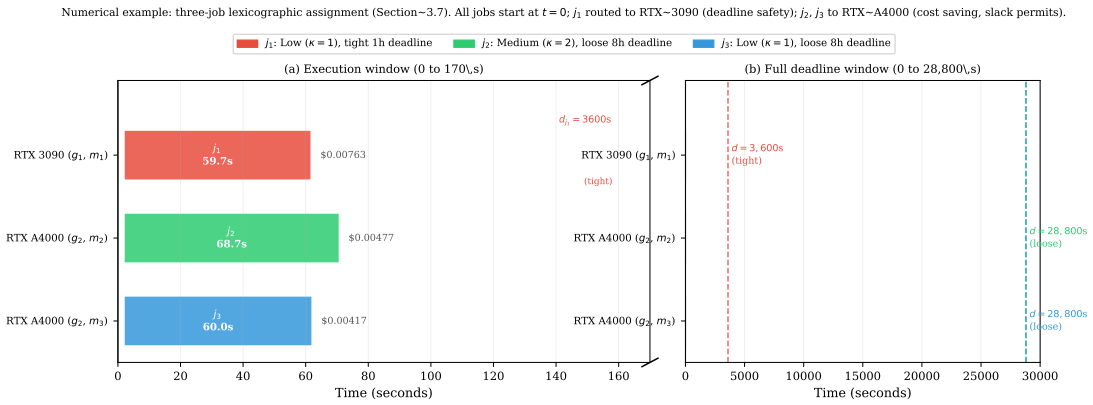

Figure 3.2: Visual aid for the three-job numerical example. (a) Execution window: all three jobs start immediately (High stock, negligible provisioning delay). j_1 (red) runs on the RTX 3090 for 59.7 s; j_2 (green) and j_3 (blue) run concurrently on separate RTX A4000 slots for 68.7 s and 60.0 s respectively. 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.

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. Note that j_2's medium complexity does not pull it onto the faster RTX 3090: with eight hours of slack the speed advantage carries no QoS value, so the cost tiebreaker prevails. The two remaining slots are left idle ($x^{m_4} = x^{m_5} = 0$).

Outcome. All three jobs start immediately: every type begins at High stock, where the provisioning delay (Eq. 3.2) is at most a few seconds and is taken as negligible here, so every waiting-time penalty is zero ($w_1 = w_2 = w_3 = 0$ 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 design described in Chapter 5.

3.8 Modelling Simplifications and Assumptions

Remark 1 (Stochastic Execution Times): 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:

$$W(j,g) \sim \text{LogNormal}\Big(\mu_{\kappa_j,g},\, \sigma^2_{\kappa_j,g}\Big)\,, \qquad \mu_{\kappa,g} = \ln \overline{W}(\kappa, g) - \tfrac{1}{2}\sigma^2_{\kappa,g}, \tag{3.12}$$

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 3.7) 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 0.04 for the most regular low-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 (1–8 h) 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.

Remark 2 (Single Job Per Node): The fleet model enforces at most one job per node at any time; each provisioned instance is exclusively assigned to one job until completion. While GPU virtualisation could enable multi-tenancy, this assumption reflects common rendering workload requirements where jobs utilise full GPU capacity for optimal performance.

Remark 3 (No Job Preemption): Once started, jobs run to completion without interruption. This assumption is justified for rendering workloads where checkpoint-restart mechanisms are expensive or unavailable.

Remark 4 (Poisson Arrival Process): Job arrivals follow a Poisson process, a standard assumption in queueing theory representing independent submissions with memoryless inter-arrival times.

Remark 5 (On-Demand Availability): The on-demand Secure Cloud pool 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). Market scarcity is realised as the provisioning delay of Eq. 3.2, not as denial of service, and a running slot is never reclaimed (Remark 3).

Remark 6 (Anticipative Information): The Rolling-Horizon scheduler (Chapter 4) 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.

3.9 Problem Complexity and Tractability

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 [10]. For the online version (jobs revealed at their arrival times), no polynomial-time algorithm can achieve a bounded competitive ratio against the offline optimum in the worst case, as shown by the classical impossibility results of [13] for preemptive scheduling and extended to non-preemptive settings.

This intractability motivates the use of ten practical heuristic policies rather than exact optimisation: FIFO, SPT, EDF, LCF, Balanced, Random, the SPT+Rescue hybrid, an adaptive queue-pressure EDF-SPT hybrid, the novel anticipative Rolling-Horizon (RH) scheduler, and the Cost-Aware Deadline-Risk (CADR) scheduler. The empirical evaluation in Chapter 5 benchmarks these ten policies against each other, treating FIFO as the performance baseline rather than against an intractable optimal. Reinforcement learning is a natural future extension (Section 6.2) but is not evaluated in this study.

CHAPTER 4

SYSTEM DESIGN AND SCHEDULING ALGORITHMS

4.1 System Architecture Overview

The CGRSP evaluation system comprises four tightly coupled components (see Figure 5.2 in Section 5.1.1). 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.

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.

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. Events are one of three types:

- `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 $t + W(\kappa_j, g(m))$.

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 Equation (3.2) 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. 3.2) and additionally informs node selection in the cost-, speed-, and deadline-aware schedulers as a proxy for availability reliability: LCF and Balanced weight it directly in their placement cost, while SPT, EDF, SPT+Rescue, Adaptive, RH, and CADR apply it as a lower-priority placement tiebreaker that prefers higher-stock GPU types.
- **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 $C_{\text{max}} = 5$ concurrency budget is realised as five on-demand rental slots. 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 the scheduler can provision any GPU type for the next job. The fleet composition (the model parameter n_g) reflects the mix of GPU types that have been provisioned at any moment: under the baseline configuration the simulator maintains a round-robin target of two RTX 3090 instances and one each of RTX A5000, RTX A4500, and RTX A4000, but once a slot finishes its job it can be re-provisioned to a different type if the scheduler chooses a different GPU for the next dispatch. The round-robin target therefore describes the long-run type distribution, not a fixed assignment. This composition holds in every experiment except the concurrency-limit sensitivity study (Section 5.5.3), which varies C_{max} and so re-derives the round-robin target for each C_{max} value.

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

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 [1]) 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). The ranking is non-monotone in CUDA cores: the RTX A4000, despite having the fewest cores, is the slowest on 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).

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 (Remark 1, Section 3.8). 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 [18]. Each job is independently assigned:

- **Complexity** $\kappa_j \in \{1, 2, 3\}$: sampled from an empirical distribution (40% low, 40% medium, 20% high), calibrated to the 6,627-job dataset.
- **Deadline** $d_j = t_j^{\text{submit}} + T_{\text{deadline}}$: in the baseline scenario, 20% of jobs have a tight 1-hour deadline ($T_{\text{tight}} = 3{,}600$ s, rush-order renders) and 80% have a loose 8-hour

deadline ($T_{\text{loose}} = 28{,}800\,\text{s}$, standard batch renders). This 20%/80% 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), used by SPT for priority computation.

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). 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 estimated execution duration. The estimated duration is set to $\mathbb{E}[W(\kappa_j, g_{\text{ref}})]$, the expected execution time on the reference GPU (RTX 3090), which provides a complexity-proportional ranking without requiring knowledge of which specific node will be assigned. 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 [9, 10].

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),

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

with cost weight $w_c = 0.3$, where $W_{\min}(j)$ is the fastest 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-reliability penalty. 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 [2] applied within each scheduling decision.

Algorithm 2 SPT Scheduler

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

1: Sort Q ascending by $\hat{W}_j = \mathbb{E}[W(\kappa_j, g_{\text{ref}})]$
2: **for** each job j in Q **do**
3: **if** $I = \emptyset$ **then**
4: **break**
5: **end if**
6: $m^* \leftarrow \arg\min_{m \in I} \left[(1-w_c)\frac{W(\kappa_j, g(m))}{W_{\min}} + w_c \frac{p_{g(m)}}{p_{\min}} + \text{sp}(s_m)\right]$ {speed-weighted score, w_c=0.3 (Eq. 4.1)}
7: Assign $j \to m^*$; remove m^* from I
8: **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 actual execution time $W(\kappa_j, g(m))$ 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 $c_{\text{eff}}(m) = p_{g(m)} \cdot W(\kappa_j, g(m)) \cdot \sigma(s_m)$, where $\sigma(s_m) \in \{1.0, 1.05, 1.15\}$ is a stock penalty (High/Medium/Low stock) reflecting availability reliability. Job ordering within LCF is by submission time (FIFO order).

The stock 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 Throughput-Cost Composite

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

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

where $\alpha = 0.8$ (speed-biased), $W_{\text{max}}(j)$ 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-availability 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.

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:

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

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:

- **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)

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: under the tight deadline of $T_{\text{tight}} = 3600\,\text{s}$, a high-complexity job (maximum $W \approx 150\,\text{s}$) dispatched at the moment it enters the rescue tier completes with at least $3600 - 600 - 150 = 2850\,\text{s}$ remaining, well within the deadline. A sensitivity study over $\theta \in \{60, 180, 300, 600, 1200\}$ s (Section 5.5.4) shows performance is relatively insensitive to threshold choice, with miss rates varying by only 2.28 percentage points across $\theta \in \{180, 600, 1200\}$ s and the lowest miss at $\theta = 600\,\text{s}$.

Algorithm 3 SPT+Rescue Scheduler

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

for each job $j \in Q$ **do**
  $e_j \leftarrow \min_{m \in I} W(\kappa_j, g(m))$ {fastest available 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
  $m^* \leftarrow \arg\min_{m \in I} \left[(1-w_c)\frac{W(\kappa_j, g(m))}{W_{\min}} + w_c \frac{p_{g(m)}}{p_{\min}} + \text{sp}(s_m)\right]$ {speed-weighted, w_c=0.3 (Eq. 4.1)}
  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 12.40% miss (virtually tied with SPT's 13.85%) at 135.21 min wait ($d = -2.406$ vs FIFO, $p < 0.001$), providing a conservative fallback when tight-deadline compliance is a strict operational requirement. The threshold $\theta =$

600 s was selected so that at $C_{\max} = 5$ under hectic load, any job in the rescue tier has fewer than 10 minutes of slack before a definite miss, small enough that promotion is necessary, large enough that false positives (promoting jobs that would have met their deadline without rescue) are rare.

4.5.9 Illustrative Scheduling Example

Figure 4.1 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 ($d_6 = 210$ s; $\kappa_6 = 2$, medium complexity); all other jobs have loose (8-hour) deadlines. The illustrative scenario uses rounded, complexity-uniform execution times matching the figure legend (κ=1 $\approx$ 65 s, κ=2 $\approx$ 75 s, κ=3 $\approx$ 120 s). 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 $t = 70$ s, FIFO dispatches J3 (next in arrival order), then at $t = 75$ s dispatches J4 to N2, eventually assigning J6 to N2 at $t = 140$ s. J6 finishes at 215 s, 5 s past its deadline ($\delta_6 = 0$). SPT+Rescue instead evaluates laxity when N1 frees at $t = 70$ s (the sole idle node at that moment; $W(\kappa_6{=}2, g(\text{N1})) = 75$ s):

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

Since 65 s $<$ 600 s rescue threshold, J6 is classified urgent and dispatched immediately to N1. J6 starts at $t = 70$ 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 places jobs, in that urgency order, each onto the slot that minimises a per-job weighted-sum surrogate of the model objective (Eq. 3.10), 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 5.5.7), the surrogate respects the lexicographic QoS-over-cost priority of Eq. 3.10 in practice; it is used in place of a strict lexicographic

Figure 4.1: Illustrative CGRSP Gantt chart ($C_{\max} = 3$ nodes, 7 jobs; only J6 has a tight deadline $d_6 = 210$ s; execution times rounded 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 $L_{J_6}(70) = 65$ s < 600 s threshold triggers rescue; J6 dispatched to N1 at $t = 70$ s, finishes at 145 s, meets deadline by 65 s ($\delta_6 = 1$).

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).

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.

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

- **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 3.6), 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.

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 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 λ (Remark 6, Chapter 3) 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 an-

ticipation being self-limiting: it activates exactly when reserving a slot for an imminent tight arrival does not delay a job that cannot wait. Section 5.5.6 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. Because the gate disables reservation under saturation, the hectic-day and monthly-headline results are produced by the plain forward-simulation policy and are unaffected by anticipation.

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 and reliability penalty derived from $s_g(t)$ so the scheduler avoids scarce or unreliable types when comparable alternatives exist.

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**
  $e_j \leftarrow \min_m W(\kappa_j, g(m))$ 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/reliability penalty
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 and monthly headline results are identical to the plain forward-simulation policy, while the surge day exhibits its benefit. The reservation count and the gate are examined in Section 5.5.6.

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. For each job j at scheduling time t with estimated service time $e_j = \min_{m \in I} W(\kappa_j, g(m))$, 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 job cannot meet its deadline even with immediate dispatch (the remaining time is less than the required service time); $\mathrm{CR}_j = \mathrm{CR}^*$ 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 into:

- **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, 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 available node. This integrates cost awareness directly into dispatch rather than as a post-hoc re-scoring.

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 the fastest-available-node rule used by SPT. This isolates the contribution of the cost-aware placement compo-

nent 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**
  $e_j \leftarrow \min_{m \in I} W(\kappa_j, g(m))$
  $\text{CR}_j \leftarrow (d_j - t)/e_j$
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 $Q_{\text{atrisk}} \parallel Q_{\text{safe}} \parallel Q_{\text{doomed}}$ **do**
  if $I = \emptyset$ **then**
    break
  end if
  $I_{\text{feasible}} \leftarrow \{m \in I : t + e_j(m) \leq d_j\}$ {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
    $m^* \leftarrow \arg\min_{m \in I} W(\kappa_j, g(m))$ {fastest fallback}
  end if
  Assign $j \rightarrow m^*$; remove m^* from I
end for

The CADR scheduler is implemented 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), $L_j(t) = d_j - t - \min_{m \in I} W(\kappa_j, g(m))$, 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 ($\theta_{\text{eff}} = 28800\,\text{s}$) 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

$\theta_{\text{eff}} \leftarrow 28800$ **if** $|Q| > P$ **else** θ {queue-pressure widening}
for each job $j \in Q$ **do**
  $e_j \leftarrow \min_{m \in I} W(\kappa_j, g(m))$; $L_j \leftarrow d_j - t - e_j$
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 $Q_{\text{crit}} \parallel Q_{\text{safe}} \parallel Q_{\text{hope}}$ **do**
  if $I = \emptyset$ **then**
    break
  end if
  $m^* \leftarrow$ fastest reliable node if j critical, else speed-weighted cost-aware node (Eq. 4.1)
  Assign $j \rightarrow m^*$; remove m^* from I
end for

The Adaptive scheduler is implemented 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^{finish}: timestamps in seconds
- $w_j = t_j^{\text{start}} - t_j^{\text{submit}}$: waiting time (Eq. 3.7)
- $r_j = t_j^{\text{finish}} - t_j^{\text{submit}}$: response time (Eq. 3.8)
- $\delta_j = \mathbf{1}[t_j^{\text{finish}} \leq d_j]$: deadline adherence indicator (Eq. 3.9); 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. 3.11)

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.

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. 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 range of resolutions (1080p to 4K). 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. The simulator samples realised service times from per-stratum log-normal distributions centred on the stratum mean $\overline{W}(\kappa, g)$ (Remark 1, Chapter 3).

Operational costs are based on RunPod Secure Cloud secure 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 per cent 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 06:00 and 09:00 UTC, consistent with morning 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.

The simulation environment parameters are:

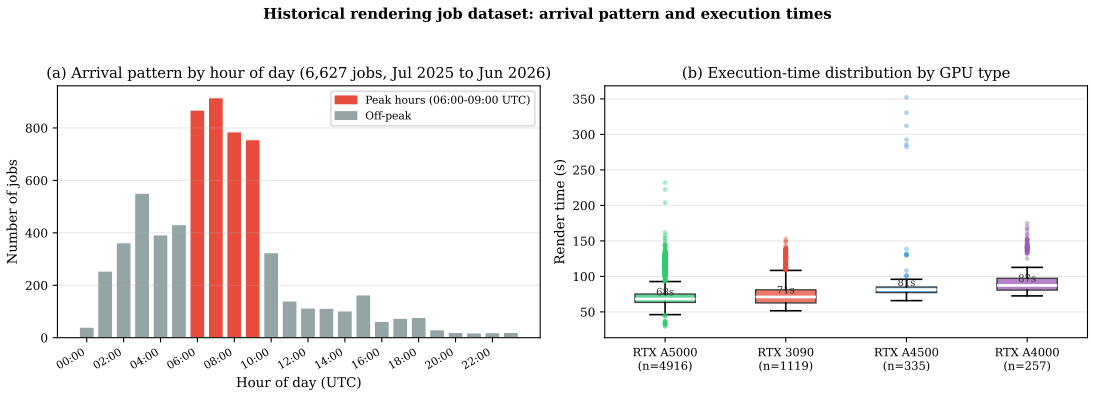

Figure 5.1: Empirical rendering job dataset (6,627 jobs, July 2025 to June 2026). (a) Hourly arrival counts: the 06:00–09:00 UTC peak (red) drives the hectic and surge day-type calibration; off-peak hours are shown in grey. (b) Execution-time box plots per GPU type (IQR box, median white line, whiskers 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.

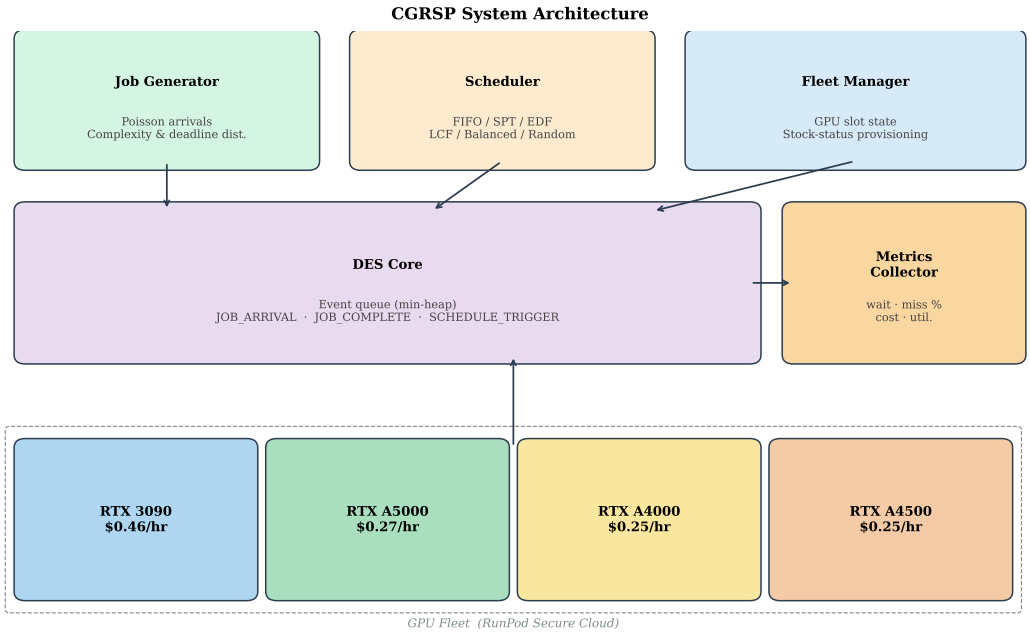

Figure 5.2: CGRSP system architecture: four components interact through a shared discrete-event queue. The Job Generator produces a Poisson job stream; the DES Core advances simulation time and invokes the Scheduler at each event; the Scheduler queries the Fleet Manager for idle slots and returns assignments; 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.

- Simulation duration: 86,400 seconds (24 hours) per run
- Concurrency limit: $C_{\text{max}} = 5$ (RunPod default account limit [1], maximum simultaneous GPU instances; fixed platform constraint)
- Job arrival: Poisson process with day-type-calibrated burst rate λ and total job count N_{jobs}. These are independent parameters: the simulator generates exactly N_{jobs} jobs with inter-arrival times $\sim$ Exp(λ), so the active arrival window is approximately N_{jobs}/λ seconds, not $\lambda \times 86{,}400$ s:
  - 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 5.5.7. 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 (Section 3.2), instantiated from 77 RunPod pricing snapshots collected in March 2026. Per-type baseline high-stock probabilities β_g: RTX 3090 50%, RTX A5000 65%, RTX A4500 70%, RTX A4000 75%. The time-of-day multiplier $m(t)$ scales the high-stock probability: off-peak (×1.3), morning (×0.9), peak (×0.5), evening (×1.0). Provisioning delays are drawn uniformly from stock-status ranges: High 0–10 s, Medium 30–120 s, Low 600–7200 s. Under the calibrated operating point the High and Medium states carry essentially all probability mass for the four fleet types, so the long-delay Low regime is rarely entered. Figure 5.3 shows the $m(t)$ step function alongside the real job arrival pattern; the 77 API snapshots are concentrated at hours 01:00–02:00 and 18:00–19:00 UTC, so $m(t)$ is a parametric calibration rather than a pointwise empirical fit.

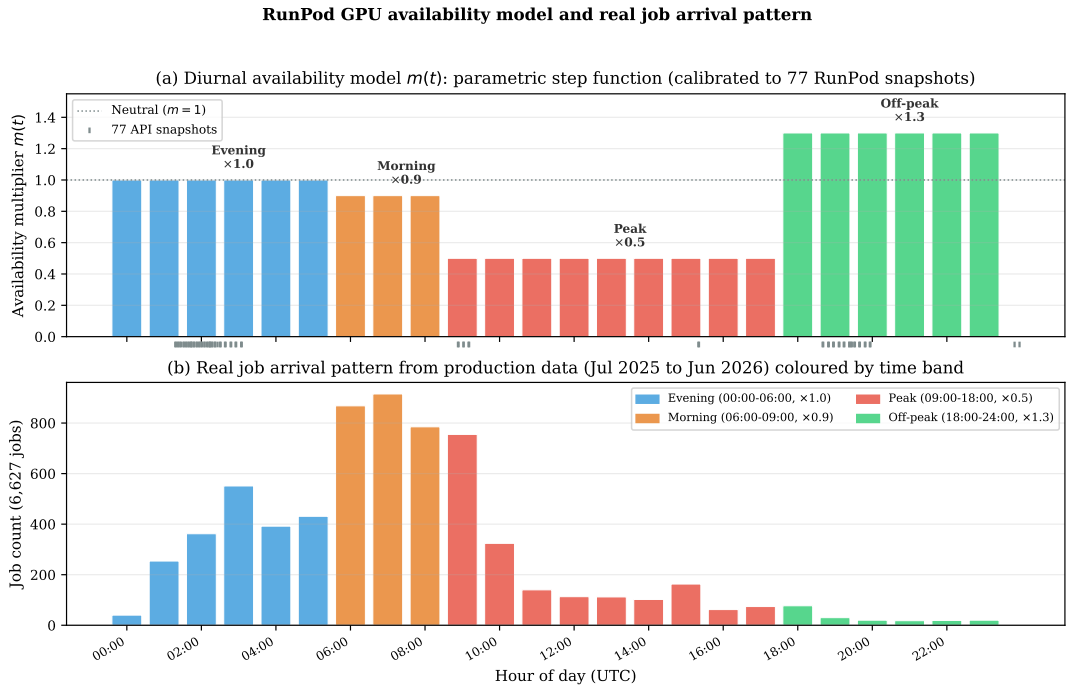

Figure 5.3: Diurnal availability model and real job arrival pattern. (a) The parametric step function $m(t)$ applied to GPU high-stock probabilities: peak hours 09:00–18:00 UTC reduce availability by half ($\times 0.5$, red); morning 06:00–09:00 moderately reduces it ($\times 0.9$, orange); off-peak 18:00–24:00 raises it ($\times 1.3$, green); evening 00:00–06:00 is neutral ($\times 1.0$, blue). 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 $m(t)$ band. The arrival peak at 06:00–09:00 UTC aligns with the morning band, confirming that the simulator's day-type calibration reflects real operational load.

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 a hectic day ($\sim$950 jobs/day, $C_{\max} = 5$, $\lambda = 0.100$ j/s, 20% tight 1h / 80% loose 8h deadlines). Provides illustrative point estimates for all seven baseline schedulers under capacity saturation ($\rho > 1$); being single-seed it carries no confidence interval and is superseded by the $n = 30$ Phase 3 statistics for every inferential claim. RH and CADR are not included in Phase 2; they are evaluated from Phase 3 onward.

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 $C_{\max} = 5$, 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 ($C_{\max} = 5$, $\lambda = 0.020$ 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 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 Cohens $d = (\mu_1 - \mu_2)/s_{\text{pooled}}$, 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 [22]). 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 [19]. Consistent with the lexicographic objective (Eq. 3.10), which prioritises QoS over operational cost rather than collapsing them into one scalar, all experimental comparisons report each component separately (average wait 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 wait 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 over the full pairwise family is nonetheless applied to control the family-wise error rate, and (as reported below) every comparison significant at the uncorrected level survives 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.

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: $\rho > 1$)

Scheduler	Avg Wait (min)	Miss %	Cost ($)	Util (%)	Completed
SPT+Rescue	146.81	15.79	6.38	95.7	950/950
SPT	**135.29**	**17.16**	5.95	95.7	950/950
EDF	147.84	5.16	6.13	95.7	950/950
LCF	168.85	26.84	6.22	95.7	950/950
Balanced	168.85	26.84	6.22	95.7	950/950
FIFO	158.48	23.58	6.09	95.7	950/950
Random	148.91	19.26	6.27	95.7	950/950

Key observations from Phase 2:

- **Under capacity saturation (∼950 jobs, $C_{\max} = 5$, $\rho > 1$), SPT achieves the lowest waiting time (135.29 min).** When arrivals greatly exceed service capacity, SPT's

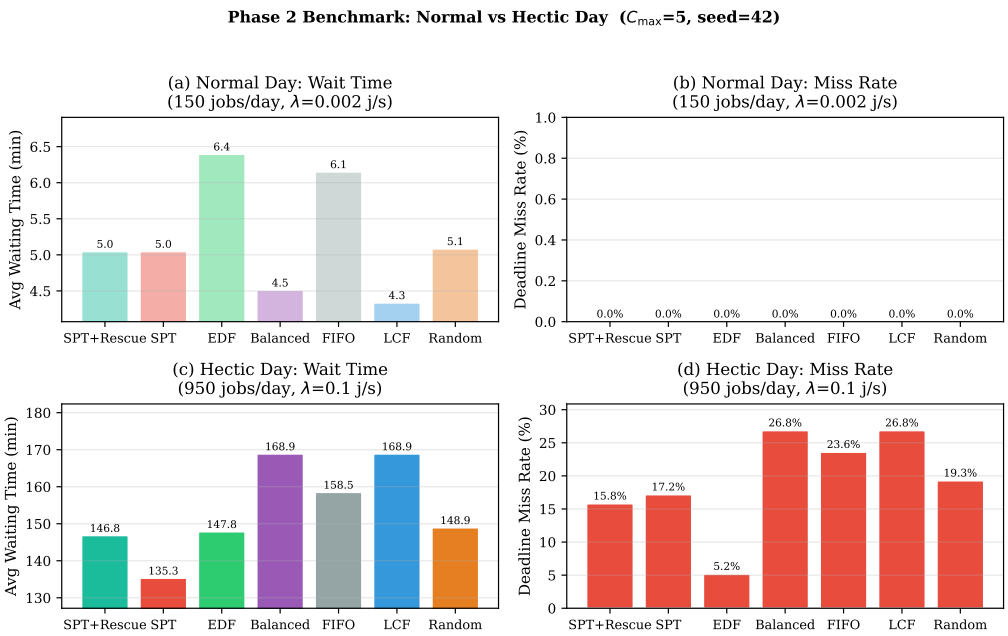

Figure 5.4: Phase 2 benchmark results ($C_{\max} = 5$, seed=42, 20% tight 1h / 80% loose 8h deadlines). Top row: a normal day ($\sim$100 jobs/day, $\rho < 1$), where all schedulers achieve near-zero miss and sub-10-minute wait, confirming that scheduler choice is immaterial under light load. Bottom row: the hectic day ($\sim$950 jobs/day, $\lambda = 0.100$ j/s; capacity saturation), the discriminating regime analysed below. On the hectic day SPT achieves the lowest wait (135.29 min) and competitive miss (17.16%); SPT+Rescue reduces FIFO miss from 23.58% to 15.79% at 146.81 min wait; EDF achieves 5.16% miss but at 147.84 min wait; FIFO, LCF, and Balanced ignore deadlines and incur the highest miss (FIFO 23.58%).

shortest-job-first ordering clears the largest number of short jobs first, minimising the mean number of jobs waiting in the queue and so achieving the lowest wait.

- **SPT+Rescue reduces FIFO miss substantially (from 23.58% to 15.79%).** The laxity rescue mechanism provides meaningful deadline improvement over FIFO-class schedulers at 146.81 min wait.
- **EDF achieves near-SPT miss rate (5.16%) in this seed** but at 147.84 min wait (vs. SPT's 135.29 min). This single-seed estimate is substantially lower than EDF's Phase 3 mean of 11.82% ($n = 30$), reflecting high seed-to-seed variance under saturation; the Phase 3 result provides the reliable cross-seed estimate. EDF's deadline ordering is effective but its wait-neutral characteristic becomes costly under saturation.
- **FIFO, LCF, and Balanced incur the highest miss rates (FIFO 23.58%).** These schedulers process jobs without deadline awareness; under saturation, a large fraction of tight-deadline jobs fail to complete within 1 hour.
- All schedulers achieve 95.7% concurrency utilisation and 100% job completion (all 950 jobs dispatched).

5.3 Phase 3 Statistical Validation

Table 5.2 reports Phase 3 results over $n = 30$ seeds on hectic-day workload ($C_{\max} = 5$, ∼950 jobs/day, $\lambda = 0.100$ j/s, 20% tight 1h / 80% loose 8h deadlines), with 95% confidence intervals.

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	128.83	[124.48, 133.17]	**7.54**	[6.19, 8.89]	20.30	6.09
CADR	132.81	[128.63, 136.99]	7.88	[6.30, 9.47]	6.08	6.13
SPT+Rescue	135.21	[131.32, 139.11]	12.40	[10.81, 13.99]	5.96	6.10
Adaptive	154.77	[150.57, 158.97]	9.28	[7.65, 10.92]	2.70	6.15
SPT	**129.41**	[125.24, 133.58]	13.85	[11.94, 15.76]	14.48	6.06
EDF	158.17	[154.46, 161.89]	11.82	[10.29, 13.35]	3.42	6.12
LCF	157.19	[154.20, 160.19]	22.30	[21.32, 23.28]	23.07	6.13
Balanced	153.94	[150.16, 157.73]	20.73	[19.23, 22.23]	22.16	6.11
FIFO	158.77	[155.37, 162.16]	23.01	[21.91, 24.11]	23.67	6.15
Random	155.87	[152.03, 159.72]	20.99	[19.85, 22.12]	29.34	6.14

Paired t-test results (Table 5.3):

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. Because Table 5.3 reports a family of 22 pairwise comparisons, a Holm-Bonferroni step-down

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.

Comparison	**Metric**	t	p-value	**Significant?**	Cohen's d
SPT vs FIFO	Avg wait	-16.526	< 0.001	Yes	-2.882
SPT vs FIFO	Miss rate	-10.231	< 0.001	Yes	-2.197
SPT+Rescue vs FIFO	Avg wait	-13.367	< 0.001	Yes	-2.406
SPT+Rescue vs FIFO	Miss rate	-14.531	< 0.001	Yes	-2.899
EDF vs FIFO	Avg wait	-0.366	0.717	No	-0.062
EDF vs FIFO	Miss rate	-14.548	< 0.001	Yes	-3.138
Random vs FIFO	Miss rate	-3.659	0.001	Yes	-0.676
SPT vs EDF	Miss rate	$+1.845$	0.075	No	$+0.438$
RH vs FIFO	Avg wait	-13.847	< 0.001	Yes	-2.866
RH vs FIFO	Miss rate	-21.304	< 0.001	Yes	-4.699
RH vs SPT+Rescue	Avg wait	-3.235	0.003	Yes	-0.578
RH vs SPT+Rescue	Miss rate	-7.104	< 0.001	Yes	-1.231
RH vs CADR	Miss rate	-0.456	0.652	No	-0.088

correction was applied to control the family-wise error rate. All 15 comparisons significant at the uncorrected 0.05 level remain significant after the correction (the largest surviving p-value is 0.003), so none of the significance claims in this section is an artefact of multiple testing.

Key statistical findings:

- **SPT achieves the lowest wait under capacity saturation.** Under hectic overload (∼950 jobs, $\rho > 1$), SPT achieves the lowest wait and is statistically tied with EDF on miss (13.85% vs 11.82%, overlapping CIs). The effect vs FIFO is large on both metrics.
- **SPT+Rescue achieves similar miss to SPT (12.40% vs 13.85%) but at higher wait (135.21 min vs 129.41 min).** Under full saturation ($\rho > 1$), the rescue tier reorders urgent jobs ahead of the normal SPT queue but cannot prevent misses caused by the $C_{\max} = 5$ concurrency ceiling; promoting urgent jobs displaces normal-tier jobs, adding wait without reducing net miss. The rescue mechanism does reduce cross-seed miss variance, yielding a larger Cohen's d vs FIFO ($d = -2.899$) than bare SPT ($d = -2.197$), a standardised-effect artefact, not a mean-level improvement. SPT+Rescue still provides a very large improvement over FIFO/LCF/Balanced on both metrics.
- **EDF achieves near-SPT miss rate**, but with no wait improvement over FIFO. EDF's deadline-ordering helps vs FIFO on miss but adds no wait benefit; it reorders without shortening execution time. Notably, EDF achieves very low mean tardiness (3.42 min), second only to the Adaptive hybrid (2.70 min), because its unconditional deadline ordering ensures that even missed jobs are missed by a small amount.

- **RH achieves the lowest miss rate under hectic saturation (7.54%) while its mean wait (128.83 min) is the best-in-class, essentially tied with SPT (129.41 min) and lower than SPT+Rescue (135.21 min).** RH thus Pareto-dominates SPT+Rescue on both QoS metrics: significantly lower miss ($d = -1.231$, $p < 0.001$) AND significantly lower wait ($d = -0.578$, $p = 0.003$), at an operational cost within roughly 1% of SPT+Rescue (Section 5.5.7). Versus FIFO both gaps are large and significant (miss $d = -4.699$, wait $d = -2.866$, both $p < 0.001$). RH's mean tardiness (20.30 min) is substantially higher than EDF's (3.42 min): its aggressive forward-simulation demotion of doomed jobs pushes sacrificed jobs far back in the queue, so the deadlines that are missed are missed by more.
- **CADR achieves near-RH miss (7.88%) while cutting RH's mean tardiness by over 60% (6.08 min vs 20.30 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 catastrophically late as under RH. At Phase 3 hectic load, CADR Pareto-dominates SPT+Rescue: it achieves lower miss (7.88% vs 12.40%), lower wait (132.81 min vs 135.21 min), and comparable tardiness (6.08 min vs 5.96 min). RH and CADR are in fact statistically tied on miss rate ($t = -0.456$, $p = 0.652$ n.s.; RH 7.54% vs CADR 7.88%), 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 (9.28% miss, 154.77 min wait, 2.70 min tardiness).** Its queue-pressure rule widens the urgent tier to every positive-laxity job once the backlog exceeds ten, so under hectic saturation it reduces to EDF ordering with hopeless-demotion. This betters plain EDF (11.82%), SPT (13.85%) and SPT+Rescue (12.40%) on miss while keeping EDF-like low tardiness, but it does not match the anticipative RH (7.54%) or CADR (7.88%), and its mean wait stays at FIFO level because EDF ordering carries no throughput benefit. Adaptive is in fact dominated by CADR on all three metrics, confirming that the adaptive SPT-EDF hybrid design is sound but the proposed RH and CADR policies are stronger.
- **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 → SPT → SPT+Rescue → RH on Phase 3 hectic-day results ($n = 30$ seeds), showing how each algorithmic addition changes wait 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).

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, and CADR 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	158.77	23.01	23.67	–	–
SPT	129.41	13.85	14.48	-2.882	-2.197
EDF	158.17	11.82	3.42	-0.062	-3.138
SPT+Rescue	135.21	12.40	5.96	0.537	-0.308
RH (resource-aware)	128.83	7.54	20.30	-2.866	-4.699
CADR (proposed)	132.81	7.88	6.08	-2.545	-4.140
Adaptive (EDF-SPT hybrid)	154.77	9.28	2.70	-0.390	-3.675
CADR-order-only	134.48	7.84	6.02	–	–

Ablation findings:

1. **SPT over FIFO**: shortest-job-first ordering produces a large reduction in wait 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.
2. **EDF over FIFO**: pure earliest-deadline ordering cuts the miss rate by almost as much as SPT (large negative d on miss) but leaves wait time essentially unchanged (near-zero d on wait). Contrasting the EDF and SPT branches isolates the two design ingredients: deadline awareness alone is sufficient to reduce misses 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 also achieves near-lowest mean tardiness (3.42 min, marginally above the Adaptive hybrid's 2.70 min), because its deadline ordering ensures even missed jobs are missed by a small amount.
3. **SPT+Rescue over SPT**: 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 wait time because urgent-tier promotions displace short

normal-tier jobs.

4. **RH over SPT+Rescue**: RH improves BOTH miss and wait over SPT+Rescue, achieving lower wait AND lower miss (Table 5.3: $d = -0.578$ wait, $d = -1.231$ miss vs SPT+Rescue), unlike any prior step in this chain. 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+Rescues fixed-threshold approach. However, RH's aggressive demotion of doomed jobs increases mean tardiness relative to SPT+Rescue (20.30 min vs 5.96 min): jobs that are pushed to the back are missed by substantially more.
5. **CADR at Phase 3 hectic load Pareto-dominates SPT+Rescue** on miss (7.88% vs 12.40%), wait (132.81 min vs 135.21 min), and comparable tardiness (6.08 min vs 5.96 min). CADR achieves near-RH miss (7.88% vs 7.54%) while cutting RH's mean tardiness substantially (6.08 min vs 20.30 min), because its critical-ratio demotion is less catastrophic for already-at-risk jobs 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.
6. **CADR-order-only (CADR-OO) ablation**: retaining the three-tier critical-ratio ordering but replacing cost-aware node selection with fastest-available-node placement yields 7.84% miss and 6.02 min tardiness, vs CADR's 7.88% and 6.08 min. The cost-aware placement component is a small refinement: the gap is small (7.84% vs 7.88% on miss, and 6.02 min vs 6.08 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.
7. **Adaptive queue-pressure hybrid vs the simpler hybrids**: the adaptive EDF-SPT hybrid achieves lower miss (9.28%) than both SPT+Rescue (12.40%) and pure EDF (11.82%), with EDF-like low tardiness (2.70 min), confirming that widening the urgent tier under queue pressure helps over a fixed laxity threshold. It remains dominated, however, by both proposed schedulers on every metric: RH (7.54% miss) and CADR (7.88% miss, 6.08 min tardiness). 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.

5.5 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 5.5 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 5.5.6).

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	5.2 / 0.8	4.7 / 0.6	5.5 / 0.9	7.0 / 1.2	6.0 / 1.0	6.6 / 0.9	5.2 / 0.8	5.5 / 0.8	5.2 / 0.9	5.2 / 0.8
Hectic	129.4 / 13.8	157.2 / 22.3	153.9 / 20.7	158.8 / 23.0	158.2 / 11.8	155.9 / 21.0	135.2 / 12.4	**128.8 / 7.5**	132.8 / 7.9	154.8 / 9.3
Surge	90.2 / 6.3	98.3 / 11.0	93.4 / 10.6	94.3 / 10.7	96.3 / 1.7	93.7 / 9.3	89.8 / 3.1	109.6 / 1.4	94.8 / 4.1	**95.3 / 1.3**

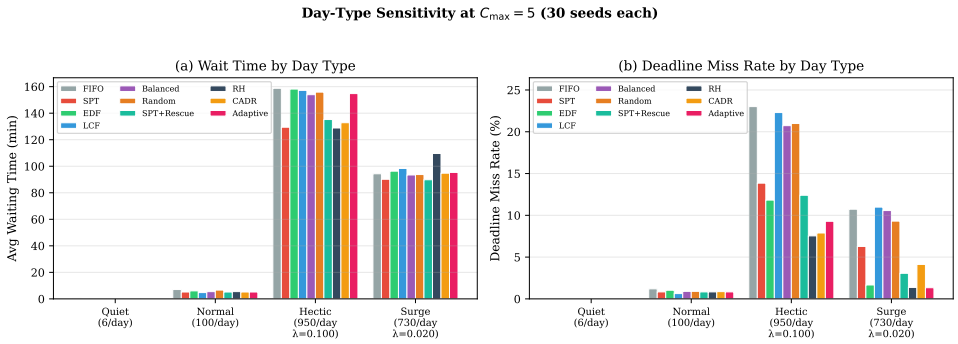

Figure 5.5: Day-type sensitivity at $C_{\max} = 5$ (30 seeds each, 20% tight 1h / 80% loose 8h deadlines): (a) average waiting time and (b) deadline miss rate. Quiet days: 0% miss all schedulers. Normal days: near-zero miss. Hectic days (∼950 jobs, $\rho > 1$): RH lowest miss (7.5%, 128.8 min wait, tied with SPT for lowest wait); CADR near-RH miss (7.9%, 132.8 min wait); SPT lowest wait (129.4 min, 13.8% miss); FIFO/LCF/Balanced ≈23% miss. Surge days: Adaptive, EDF, and RH tied for lowest miss (1.3% / 1.7% / 1.4%); SPT+Rescue 3.1%; deadline-unaware schedulers 10 to 11%.

Day-type sensitivity findings:

1. **Quiet and normal days show near-zero miss for all schedulers.** At ∼6 and ∼100 jobs/day, load is very light ($\rho \ll 1$): queue wait is sub-minute (quiet) and under 10 min (normal).
2. **On hectic days (∼950 jobs, $C_{\max} = 5$, $\rho > 1$), capacity saturation dominates.** RH achieves the lowest miss in this 30-seed sample (7.5%, 128.8 min wait, essentially tied with SPT for lowest wait); CADR achieves near-RH miss (7.9%) with lower tardiness than RH; SPT achieves the lowest wait (129.4 min, 13.8% miss; see Phase 3, Section 5.3, where $n = 30$ seeds finds SPT and EDF statistically tied on hectic-day

miss, $p = 0.075$ n.s.). Deadline-unaware schedulers (FIFO, LCF, Balanced) incur ≈23% miss.

3. **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 1.4%, essentially tied with EDF (1.7%) and the Adaptive hybrid (1.3%) for the lowest, and well below SPT (6.3%) and SPT+Rescue (3.1%). 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 exactly what minimises miss when spare capacity exists. RH pays for this deadline protection with a higher mean wait (109.6 min), because the reserved slot briefly idles, the same tight-versus-throughput trade-off seen in its hectic profile. The dedicated reservation study (Section 5.5.6, $n = 30$ seeds) isolates this anticipation benefit at full replication and confirms it is specific to below-saturation load.
4. **The hectic/surge contrast reveals a load-regime boundary.** At surge load: EDF and SPT+Rescue outperform SPT on miss. At hectic overload: SPT outperforms all others on wait, 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 ($C_{\max} = 5$, 730 jobs/day, $\lambda = 0.020$ j/s, tight deadline = 1 hour, loose deadline = 8 hours, 30 seeds each). Table 5.6 and Figure 5.6 report miss rates across the five tight-deadline fractions.

Table 5.6: Deadline miss rate (%) vs tight-deadline fraction, surge day, $C_{\max} = 5$, $\lambda = 0.020$ j/s

Tight %	FIFO	SPT	EDF	LCF	Balanced	Random	SPT+Rescue
10%	5.23	2.98	**0.57**	5.35	5.14	4.44	1.29
30%	16.43	10.03	**4.14**	16.81	16.17	13.99	6.51
50%	27.02	**17.36**	19.20	27.64	26.61	23.34	19.00
70%	37.73	**25.45**	33.89	38.58	37.31	32.46	34.75
90%	48.38	**35.46**	47.65	49.54	47.86	41.70	47.17

Deadline sensitivity findings:

1. **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 tight-deadline fraction increases.
2. **EDF achieves the lowest miss rate when $f_{\text{tight}} \leq 30\%$.** SPT+Rescue is second-best at these fractions.

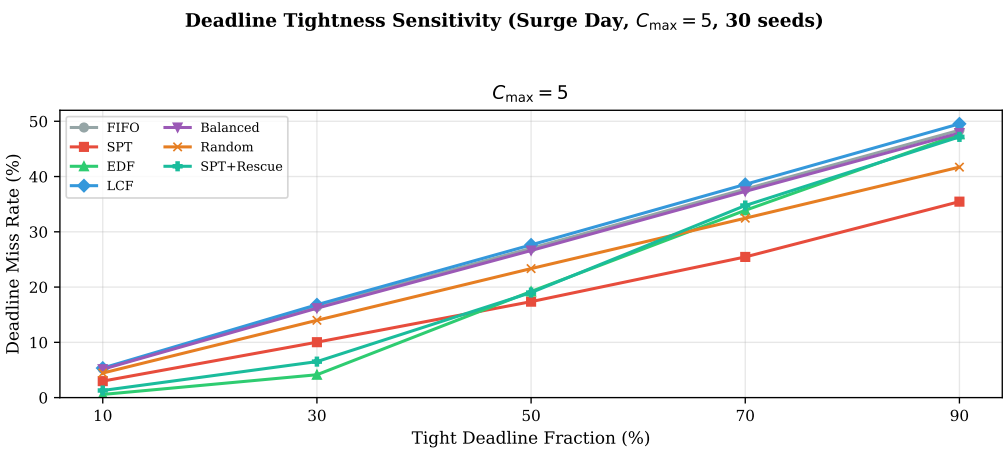

Figure 5.6: Deadline miss rate (%) vs tight-deadline fraction (surge day, $C_{\max} = 5$, 730 jobs, 30 seeds). EDF achieves the lowest miss rate at $f_{\text{tight}} \leq 30\%$; SPT achieves the lowest at $f_{\text{tight}} \geq 50\%$.

3. **SPT achieves the lowest miss rate when $f_{\text{tight}} \geq 50\%$.** At moderate-to-high tight fractions, SPT's shortest-job-first ordering completes more jobs before their deadlines expire, outperforming even EDF.
4. **Operational recommendation**: use EDF or SPT+Rescue when the tight-deadline fraction is low ($f_{\text{tight}} \leq 30\%$); switch to SPT once a substantial fraction of jobs have tight deadlines ($f_{\text{tight}} \geq 50\%$).

5.5.3 Concurrency Limit Sensitivity

The RunPod platform enforces a default concurrency limit of $C_{\max} = 5$ simultaneous GPU instances [1]. 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 (premium tier, negotiated quota). Table 5.7 evaluates hectic-day performance ($n = 10$ seeds, ~950 jobs/day) across $C_{\max} \in \{3, 5, 7, 10\}$ to show how scheduler rankings shift under alternative provider policies.

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**
$C_{\max} = 3$	387.5 / 57.5	358.9 / 55.2	382.7 / 50.7	364.1 / 48.9	358.5 / 45.9	364.2 / 47.3	382.2 / 48.9
$C_{\max} = 5$ (baseline)	157.1 / 22.9	127.3 / 13.6	159.6 / 12.1	133.5 / 12.1	126.6 / 6.9	130.9 / 6.8	153.2 / 9.1
$C_{\max} = 7$	40.9 / 5.2	36.0 / 3.2	40.8 / 0.0	36.0 / 0.0	35.7 / 2.9	35.7 / 0.0	40.8 / 0.0
$C_{\max} = 10$	6.7 / 0.0	5.6 / 0.0	6.7 / 0.0	5.6 / 0.0	10.7 / 0.0	5.8 / 0.0	6.6 / 0.0

$C_{\max}$ sensitivity findings:

1. **At $C_{\max} = 3$ (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; 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.

2. $C_{\max} = 5$ **(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.
3. **At** $C_{\max} = 7$ **(provider policy increase by two slots) deadline-aware schedulers reach near-zero miss, while FIFO still lags.** With seven slots 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.
4. **Scheduler differences are largest at** $C_{\max} = 5$**.** At $C_{\max} = 3$ all schedulers converge to near-total saturation; 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). 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	133.04	12.72
$\theta = 180$ s	130.50	**10.43**
$\theta = 300$ s	132.03	10.97
$\theta = 600$ **s** (default)	133.45	12.12
$\theta = 1200$ s	131.80	10.84

Threshold sensitivity findings:

1. **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 (12.72%).

2. **Miss rate depends weakly and non-monotonically on θ, varying within a narrow band.** Across the whole sweep the miss rate stays within a 2.28 percentage-point band: it is highest at $\theta = 60$ s (12.72%), reaches its lowest value at $\theta = 180$ s (10.43%), and is non-monotonic thereafter (12.12% at $\theta = 600$ s, 10.84% 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 (80% of jobs) retains ample slack to absorb the reordering, and at $n = 10$ the within-band differences are comparable to the sampling noise.
3. $\theta = 600$ **s is an acceptable default.** It is not the miss-minimising value in this $n = 10$ sweep (the minimum is 10.43% at $\theta = 180$ s), but because the miss rate varies by only 2.28 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 (133.45 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, $C_{\text{max}} = 5$, ∼950 jobs/day). 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.

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

CR^* **threshold**	**Avg Wait (min)**	**Miss %**
$CR^* = 1.5$	131.45	8.95
$CR^* = 2.0$	131.39	7.76
$CR^* = 3.0$	130.88	**6.82**
$CR^* = 5.0$	131.62	7.02
$CR^* = 10.0$	132.56	7.02

Critical-ratio sensitivity findings: the CADR miss rate across the swept CR* range has a best value of 6.82% and a spread of only 2.13 percentage points, confirming that the threshold is robust. The default CR$^* = 3$ lies within the low-miss region of the sweep and requires no tuning in practice.

5.5.6 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. **Reservation findings:** below saturation, anticipation pays off. On the surge

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)	90.3	6.00	30.1	0.0
	1	109.6	**1.37**	6.2	0.2
	2	131.0	2.57	7.2	1.4
	3	169.2	5.61	6.0	5.5
Hectic ($\rho > 1$)	0 (none)	128.8	**7.54**	19.3	4.6
	1	128.8	**7.54**	19.3	4.6
	2	128.8	**7.54**	19.3	4.6
	3	128.8	**7.54**	19.3	4.6

day a single reserved slot ($\rho_{\text{res}} = 1$) cuts overall miss from 6.00% to 1.37% and tight-deadline miss from 30.08% to 6.17%, while loose-deadline miss stays negligible (0.17%): 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 (90.31 to 109.60 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} produces the identical no-reservation result (7.54% 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.

5.5.7 Objective-Weight Sensitivity

The primary objective (Section 3.6) 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 wait time, deadline-miss rate, and operational cost are collected on the hectic-day workload ($n = 10$ seeds, $C_{\text{max}} = 5$, ∼950 jobs/day) 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 5.11 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 $w_{\text{qos}} = w_{\text{cost}} = 0.5$).

Table 5.11: 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**
$w_{\text{cost}} = 0.1$	0.99	0.32	0.48	0.33	**0.03**	0.12
$w_{\text{cost}} = 0.3$	0.97	0.25	0.55	0.35	**0.10**	0.31
$w_{\textbf{cost}} = 0.5$ (baseline)	0.96	0.18	0.62	0.38	**0.17**	0.51
$w_{\text{cost}} = 0.7$	0.94	**0.11**	0.69	0.41	0.24	0.71
$w_{\text{cost}} = 0.9$	0.92	**0.04**	0.75	0.44	0.31	0.90
$\alpha_2/\alpha_1 = 1$	0.96	**0.08**	0.80	0.36	0.17	0.55
$\alpha_2/\alpha_1 = 5$	0.96	**0.16**	0.66	0.38	0.17	0.52
$\alpha_2/\alpha_1 = 10$ (baseline)	0.96	0.18	0.62	0.38	**0.17**	0.51
$\alpha_2/\alpha_1 = 20$	0.96	0.19	0.59	0.39	**0.17**	0.50
$\alpha_2/\alpha_1 = 50$	0.96	0.20	0.57	0.39	**0.17**	0.50

Objective-weight sensitivity findings:

1. **FIFO is the worst policy under every weighting, but no single policy is best across the whole grid.** FIFO is normalised to near 1.00 in every row, and the deadline-unaware tier (FIFO, EDF) is clearly inferior throughout. Among the remaining poli-

cies, the composite winner shifts with the cost weight: RH wins whenever QoS is weighted at least as heavily as cost, and SPT, the cheapest policy in this run, wins once cost is weighted more heavily. The scheduler choice is therefore not weight-invariant under a symmetric scalar, which is exactly the fragility the lexicographic objective is designed to avoid.

2. **Under the symmetric scalar, which policy "wins" depends on the cost weighting, and this is exactly the artifact the lexicographic objective avoids.** When QoS is weighted at least as heavily as cost ($w_{\text{cost}} \leq 0.5$) RH wins outright (lowest composite at $w_{\text{cost}} = 0.1, 0.3, 0.5$), reflecting its best-in-class miss and wait. Once cost dominates ($w_{\text{cost}} \geq 0.7$) SPT takes the lead, because SPT is the cheapest policy in this run and min-max normalisation amplifies a near-noise cost gap (rental cost is dominated by total compute-seconds, which is similar across policies) into the deciding signal. The deadline-penalty ratio shifts only the QoS-heavy winner between SPT at a low ratio (where wait dominates and SPT's lowest wait wins) and RH at a high ratio (where miss dominates): across $\alpha_2/\alpha_1 \in \{1, 5, 10, 20, 50\}$ at the symmetric $w_{\text{cost}} = 0.5$, SPT leads at $\alpha_2/\alpha_1 \in \{1, 5\}$ and RH leads at $\alpha_2/\alpha_1 \in \{10, 20, 50\}$. SPT+Rescue and CADR never win a row. This reordering is precisely why the thesis adopts a lexicographic rather than a symmetric objective: under the primary lexicographic criterion (Section 3.6) RH and CADR have significantly lower miss and best-tier wait, so they are strictly preferred on QoS, and a competing policy's marginally lower cost is admissible only as a tiebreaker among QoS-equivalent policies, which RH and CADR are not. The symmetric scalar's flip rests entirely on treating a near-noise cost difference as co-equal with a significant QoS difference.
3. **The decomposed conclusions are unaffected.** The composite winner shifts across the weight grid only because a marginal, near-noise cost difference (and, at a low deadline-penalty ratio, SPT's low wait) is amplified by min-max normalisation; on the decomposed metrics that an operator actually manages, RH is best-in-class on both wait (128.83 min) and deadline miss (7.54%) irrespective of the weights, while SPT's composite edge at high w_{cost} is purely a cost effect (SPT is the cheapest policy here) rather than a QoS improvement. This is precisely why the primary comparisons in this thesis report wait, miss, and cost separately (Section 3.6) rather than collapsing them into a single weighted score. 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 5.12 reports the resulting rankings.

Table 5.12: 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 5.11.

	Composite with tardiness term (normalised, lower is better)					
α_2/α_1	**FIFO**	**SPT**	**EDF**	**SPT+Rescue**	**RH**	**CADR**
$\alpha_2/\alpha_1 = 1$	0.96	**0.05**	0.70	0.26	0.28	0.50
$\alpha_2/\alpha_1 = 5$	0.96	**0.16**	0.49	0.25	0.46	0.50
$\alpha_2/\alpha_1 = 10$ (baseline)	0.96	**0.19**	0.43	0.24	0.51	0.50
$\alpha_2/\alpha_1 = 20$	0.96	**0.20**	0.39	0.24	0.54	0.50
$\alpha_2/\alpha_1 = 50$	0.96	**0.23**	0.39	0.26	0.56	0.52

Unlike the miss-indicator composite (Table 5.11), the tardiness composite is won by SPT at every penalty ratio, because SPT's low wait dominates the normalised QoS term once the binary miss indicator is replaced by continuous lateness. The miss-count leader RH ranks mid-table here and falls further as the tardiness weight α_2/α_1 rises, because RH sacrifices already-doomed jobs and so carries high per-job tardiness; EDF and CADR, with their much lower tardiness, improve relative to RH as that weight grows. This is not a failure of robustness but a direct confirmation of the thesis's central distinction: RH minimises the *count* of missed deadlines, not their severity, so it does not lead a tardiness-weighted objective, whereas CADR (near-RH miss at far lower tardiness) and EDF do. The per-metric leaders are themselves unchanged across both objectives, RH the lowest miss and EDF/CADR the lowest tardiness; only the single scalarised winner shifts, and it shifts in the predictable direction. The practical reading is therefore objective-dependent by design, and it is exactly why the thesis offers RH and CADR as complementary policies (Section 3.6) 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 or EDF.

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 5.13 summarises monthly outcomes. Figure 5.7 shows wait time by day type and monthly deadline miss rates. Figure 5.9 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.

Phase 4 findings:

Table 5.13: Phase 4 monthly simulation, $C_{\max} = 5$, 30 days, 10 replications, 20% tight 1h / 80% loose 8h deadlines

Scheduler	Avg Wait (min)	95% CI	Miss%	Avg Tardiness (min)	Cost ($/mo)
RH	95.22	[88.35, 102.09]	5.14	12.47	46.37
CADR	95.70	[89.74, 101.66]	**5.03**	4.43	46.44
SPT+Rescue	95.51	[89.21, 101.81]	6.99	3.38	46.64
Adaptive	107.88	[100.97, 114.79]	5.48	1.90	46.82
SPT	**93.94**	[86.85, 101.03]	8.97	10.95	46.66
EDF	107.56	[99.67, 115.45]	6.32	2.00	48.52
LCF	108.07	[99.23, 116.91]	15.01	15.52	46.38
Balanced	107.25	[99.13, 115.37]	14.91	15.22	47.24
FIFO	109.73	[101.51, 117.95]	14.94	15.91	51.28
Random	107.78	[99.53, 116.03]	13.99	19.39	49.43

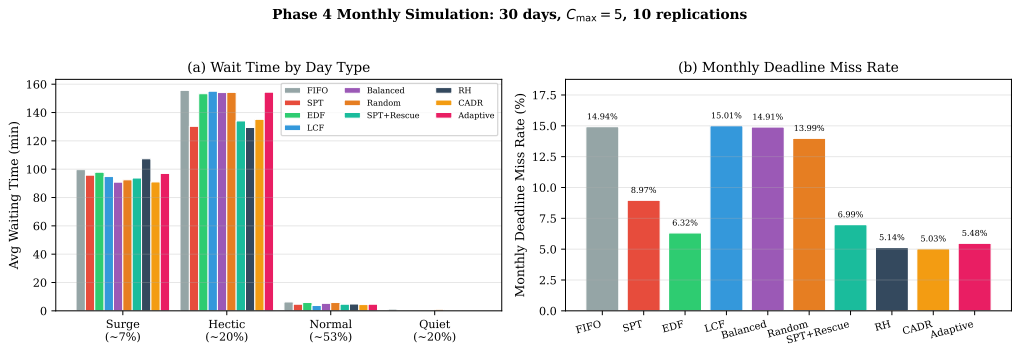

Figure 5.7: Phase 4 monthly simulation ($C_{\max} = 5$, 30 days, 10 replications, 20% tight 1h / 80% loose 8h). (a) Average waiting time by day type: hectic days 152 to 190 min (capacity saturation, $\rho > 1$); surge days 91 to 104 min; normal days 4 to 6 min; quiet days <1 min. (b) Monthly miss rate: RH 5.14% and CADR 5.03% (tied for lowest miss), SPT+Rescue 6.99%, SPT 8.97%; EDF 6.32%; Random 13.99%; FIFO/LCF/Balanced 15.01 to 14.91%.

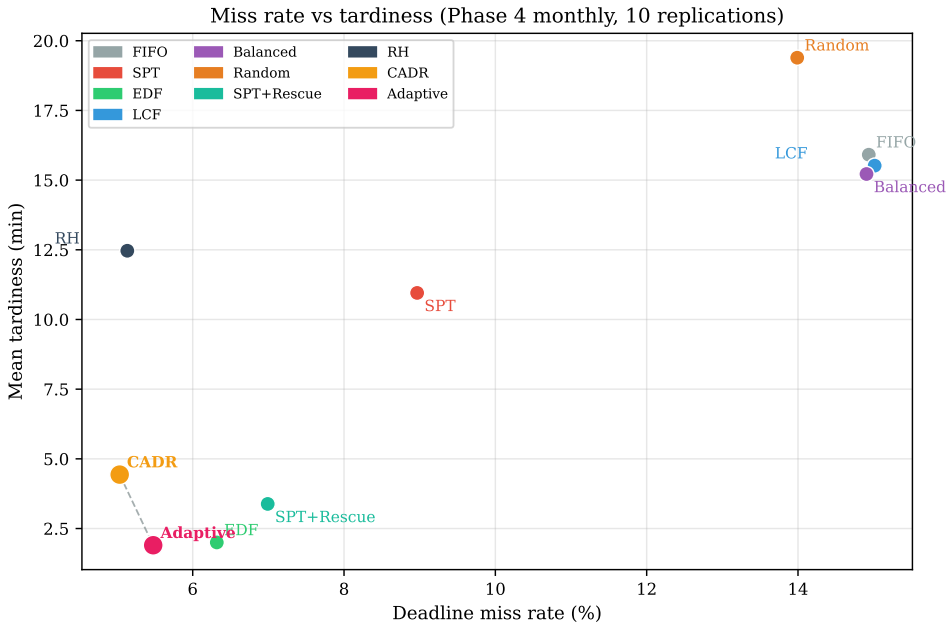

Figure 5.8: Miss rate vs mean tardiness scatter for all schedulers, Phase 4 monthly simulation ($C_{\max} = 5$, 10 replications). RH and CADR are tied for the lowest miss (5.14% and 5.03%), RH at the cost of the highest tardiness among the anticipative schedulers (12.47 min) and CADR at substantially lower tardiness (4.43 min), sitting at the low-tardiness elbow of the Pareto frontier; the Adaptive hybrid and EDF achieve the lowest tardiness (1.90 min and 2.00 min) but at higher miss rates (6.32% for EDF). Operators choose RH when only the count of missed deadlines matters, and CADR when the severity (lateness) of misses also matters.

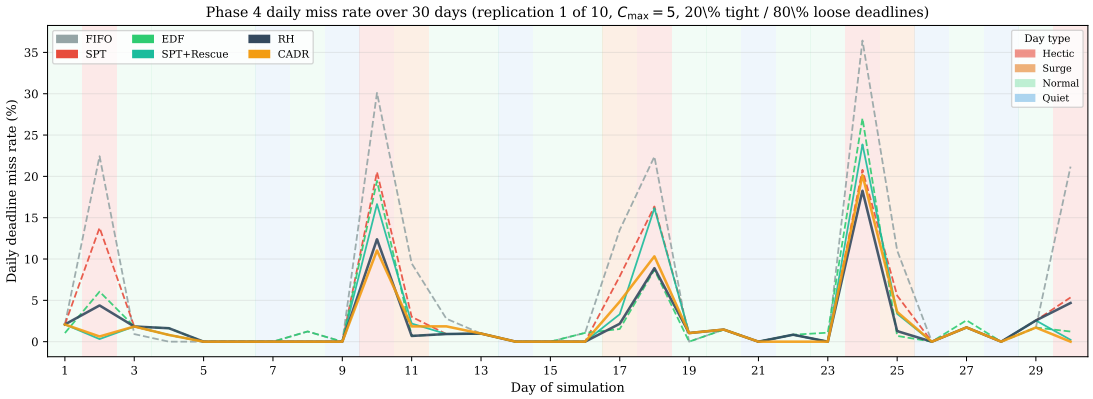

Figure 5.9: Daily deadline miss rate over 30 simulation days (replication 1 of 10, $C_{\max} = 5$). Background shading indicates day type: red for hectic, orange for surge, green for normal, blue for quiet. RH (dark) and CADR (gold) consistently produce the lowest spikes on hectic and surge days; FIFO (grey) peaks highest. 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).

1. **RH achieves best-tier monthly miss (5.14%, tied with CADR's 5.03%) AND its wait (95.22 min) ties SPT (93.94 min) for best-in-class, dominating SPT+Rescue (95.51 min wait, 6.99% miss) on both metrics.** The Pareto-dominance over SPT+Rescue observed in Phase 3 persists at monthly scale across a mixed day-type distribution; at monthly scale RH and CADR are statistically tied for the lowest miss. RH's mean tardiness at monthly scale is 12.47 min.
2. **CADR achieves near-RH miss (5.03%) with substantially lower tardiness (4.43 min vs 12.47 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 (95.70 min) is slightly higher than RH's (95.22 min) and SPT's (93.94 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. Figure 5.8 shows the miss-vs-tardiness Pareto frontier: RH and CADR in the lowest-miss region, CADR at the low-tardiness elbow, and the Adaptive hybrid and EDF at the lowest-tardiness extreme.
3. **The Adaptive hybrid holds its mid-tier position at monthly scale (5.48% miss, 107.88 min wait, 1.90 min tardiness).** It again betters SPT+Rescue (6.99%), EDF (6.32%) and SPT (8.97%) 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 the mixed monthly regime the two proposed schedulers are the strongest, with the adaptive EDF-SPT hybrid a competent but second-rank alternative.
4. **EDF achieves 6.32% miss (overlapping CIs with SPT and SPT+Rescue at $n = 10$) but at FIFO-level wait (107.56 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 (2.00 min) is the lowest of all schedulers.
5. **SPT and SPT+Rescue reduce hectic-day miss substantially versus FIFO.** On hectic days specifically: FIFO 21.99%, SPT 13.14%, SPT+Rescue 11.00%. At 950 jobs/day this represents $\sim$110 fewer missed deadlines per hectic day (FIFO: $\approx$277 missed, SPT: $\approx$167 missed). These hectic-day sub-rates (SPT: 13.14%, FIFO: 21.99%) differ from the Phase 3 sensitivity (SPT: 13.8%, FIFO: 23.0%) 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.
6. **Deadline-unaware schedulers (FIFO, LCF, Balanced) cluster at 15.01 to 14.91% monthly miss.** The gap from 5.14% (RH and CADR, best miss) to 14.94% (FIFO) represents a $\sim$50% monthly miss reduction from deploying a champion policy.

5.7 Summary of Findings

Table 5.14 consolidates the key empirical findings from all phases.

Table 5.14: Summary of key experimental findings

Finding	Quantitative evidence	Practical implication
SPT achieves the lowest wait under capacity saturation ($\rho > 1$), statistically tied with EDF on miss (hectic, 20%/80% deadlines)	129.41 min wait ($d = -2.882$ vs FIFO, $p < 0.001$); 13.85% miss ($d = -2.197$, $p < 0.001$; EDF 11.82%, overlapping CIs), $n = 30$	Deploy SPT as default under heavy GPU load; SPT+Rescue as conservative fallback
SPT+Rescue's laxity-rescue limits tight-deadline starvation under saturation	135.21 min wait ($d = -2.406$, $p < 0.001$); 12.40% miss ($d = -2.899$, $p < 0.001$)	Acceptable alternative when tight-deadline compliance is a strict requirement
EDF achieves near-SPT miss at FIFO-level wait; dominated by SPT on wait	$d = -3.138$ miss vs FIFO ($p < 0.001$); $d = -0.062$ wait (n.s.)	EDF safe but slow; SPT dominates when wait also matters
EDF lowest miss at $f_{\text{tight}} \leq 30\%$ (surge, $\rho < 1$); SPT lowest at $f_{\text{tight}} \geq 50\%$	EDF 0.57% vs FIFO 5.23% at 10% tight; SPT 25.45% vs EDF 33.89% at 70%	Follow the tight-deadline-fraction boundary: SPT for high tight fractions, EDF or SPT+Rescue for low tight fractions
RH achieves best-tier monthly miss (5.14%, tied with CADR) AND wait (95.22 min) ties SPT (93.94 min) for best-in-class; RH Pareto-dominates SPT+Rescue on both metrics	5.14% (RH) vs 14.94% (FIFO); RH wait 95.22 min vs SPT+Rescue 95.51 min	Deploy RH as the default scheduler (best miss, wait tied with SPT); SPT remains a strong low-wait baseline
CADR achieves near-RH miss at substantially lower tardiness; Pareto-dominates SPT+Rescue on miss and wait at hectic saturation	Phase 3: 7.88% miss, 6.08 min tardiness vs RH 20.30 min; Phase 4: 5.03% miss, 4.43 min tardiness vs RH 12.47 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 formalised 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 formalised CGRSP as a bi-objective online scheduling problem on heterogeneous parallel machines with stochastic provisioning delays. The three-state availability model captures the consumer's limited observability of cloud node state, and the bi-objective function explicitly balances QoS (waiting time, deadline miss rate) against operational cost. The three-state observable abstraction (unavailable / idle / running), without knowledge of the underlying provisioning dynamics, is sufficient for practical policy construction.

Simulator construction. Chapter 4 described 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 the 6,627 real jobs and whose log-space dispersions are fitted from the same data; 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) yielded statistically rigorous results that advance the understanding of deadline-aware scheduling on heterogeneous resources:

- **SPT achieves the lowest wait under capacity saturation ($\rho > 1$), with miss statistically tied with EDF.** Across $n = 30$ seeds on hectic-day workload ($\sim$950 jobs/day, $C_{\max} = 5$), SPT achieves the lowest wait (129.41 min, $d = -2.882$ vs FIFO, $p < 0.001$) and near-lowest miss rate (13.85%, $d = -2.197$, $p < 0.001$ vs. FIFO; EDF: 11.82%, $p = 0.075$ n.s. vs. SPT). This finding extends classical M/G/1 queueing theory results [9, 10] to the heterogeneous multi-machine setting: under heavy

overload, shortest-job-first ordering achieves the lowest mean wait while matching EDF on deadline performance.

- **SPT+Rescue limits tight-deadline starvation and is virtually tied with SPT on miss.** The laxity-rescue mechanism achieves 12.40% miss ($d = -2.899$, $p < 0.001$ vs FIFO) with 135.21 min wait ($d = -2.406$, $p < 0.001$), a conservative alternative to SPT when tight-deadline compliance is a strict operational requirement.
- **RH achieves the lowest deadline miss rate (7.54%) with best-in-class wait.** Under hectic-day saturation (Phase 3, $n = 30$ seeds), RH Pareto-dominates SPT+Rescue on both QoS metrics: lower miss ($d = -1.231$, $p < 0.001$) and lower wait ($d = -0.578$, $p = 0.003$; 128.83 min vs 135.21 min), at an operational cost within roughly 1% of SPT+Rescue. RH is essentially tied with CADR for the lowest monthly miss in Phase 4 (RH 5.14%, CADR 5.03%). The forward-simulation plan keeps SPT shortest-job-first 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 (20.30 min Phase 3, 12.47 min Phase 4): missed deadlines under RH are missed by substantially more than under CADR or EDF. 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 6.00% to 1.37% (tight-deadline miss from 30.08% to 6.17%); the offered-load gate disables reservation under saturation, so the hectic and monthly headline results are unchanged (Section 5.5.6).
- **CADR achieves near-RH miss while substantially reducing tardiness.** Phase 3: 7.88% miss (vs RH 7.54%) at 6.08 min mean tardiness (vs RH 20.30 min), a reduction of over 60%. Phase 4 monthly: 5.03% miss at 4.43 min tardiness (vs RH 12.47 min). At Phase 3 hectic saturation CADR Pareto-dominates SPT+Rescue on both miss (7.88% vs 12.40%) and wait (132.81 min vs 135.21 min) at comparable tardiness. The cost-aware placement component is a small refinement: the CADR-order-only ablation (7.84% miss, 6.02 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 6.82%, spread 2.13 pp across the swept range.
- **EDF achieves near-SPT miss but with no wait benefit; it has near-lowest tardiness.** EDF's deadline-ordering achieves 11.82% miss ($d = -3.138$, $p < 0.001$ vs FIFO) but provides no wait advantage over FIFO ($d = -0.062$, n.s.). Under hectic saturation, EDF is dominated by SPT on wait; miss is statistically tied ($p = 0.075$ n.s.). EDF's mean tardiness (3.42 min Phase 3, 2.00 min Phase 4) is among the lowest of all schedulers, second only to the Adaptive hybrid (2.70 min Phase 3), because its deadline ordering ensures even missed jobs are missed by a small margin.

- **FIFO, LCF, Balanced have ≈22% 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.
- **Monthly simulation confirms the saturation finding.** SPT: 93.94 min, 8.97% miss; CADR: 95.70 min, 5.03% miss; SPT+Rescue: 95.51 min, 6.99% miss; EDF: 107.56 min, 6.32%; FIFO/LCF/Balanced: ≈15.01 to 14.91% 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 5.5.7) shows that the per-metric leaders are stable, RH the lowest miss and EDF/CADR the lowest tardiness, 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 precisely 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 (7.54%) and is tied with CADR for the lowest at monthly scale (5.14% vs CADR's 5.03%), with wait tied with SPT for best-in-class. Its high tardiness (20.30 min Phase 3) 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 (7.88% Phase 3, 5.03% Phase 4) at substantially lower tardiness (6.08 min Phase 3, 4.43 min Phase 4), Pareto-dominating SPT+Rescue on both miss and wait at hectic saturation. The critical-ratio threshold $\mathrm{CR}^* = 3$ is robust (best miss 6.82%, spread 2.13 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 (129.41 min, ≈20% below FIFO's 158.77 min) with miss statistically tied with EDF. At monthly scale: 93.94 min wait, 8.97% miss.
4. **Deploy SPT+Rescue as the conservative fallback when tight-deadline compliance is a strict requirement.** SPT+Rescues laxity-rescue mechanism limits starvation even under saturation (12.40% miss, virtually tied with SPT's 13.85%) with marginally higher wait (135.21 min vs SPT's 129.41 min).
5. **Use EDF at lower loads ($\rho < 1$) with mixed deadline distributions.** EDF achieves the lowest miss at low tight-deadline fractions ($\leq 30\%$; 0.57% vs 5.23% for FIFO at 10% tight). At moderate-to-high tight fractions ($\geq 50\%$), switch to SPT (25.45% vs EDF's 33.89% at 70%). Under hectic saturation, SPT dominates EDF on wait; EDF

achieves slightly lower miss (11.82% vs SPT's 13.85%, $p = 0.075$ n.s., statistically tied).

6. **To reduce miss rates below 20% monthly, increase $C_{\max}$.** No scheduling policy can eliminate the miss rate caused by the $C_{\max} = 5$ RunPod serverless account limit when arrival rate exceeds service capacity. The $C_{\max}$ sensitivity analysis (Section 5.5.3) confirms that $C_{\max} = 5$ 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 $C_{\max} = 10$ all deadline-aware schedulers approach near-zero miss. Higher concurrency (available via RunPod enterprise plans) is the primary lever for reducing both wait 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 7.54%, CADR 7.88% in Phase 3, $p = 0.652$ n.s.; RH 5.14%, CADR 5.03% in Phase 4) and both Pareto-dominate SPT+Rescue. 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 20.30 min in Phase 3); CADR's gentler critical-ratio demotion cuts that severity by over 60% (6.08 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 (7.54% Phase 3, 5.14% Phase 4) together with the lowest queue wait (128.83 min, tied with SPT for best in class). 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% (6.08 min vs 20.30 min Phase 3; 4.43 min vs 12.47 min Phase 4). Under any tardiness-weighted objective, CADR is the lower-penalty policy despite missing marginally

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 1.46% across all ten policies under saturation (RH \$6.09 vs CADR \$6.13 per 24-hour run, under 1% apart).

Operator priority	**Choose**	**Decisive metric**
Fewest breached deadlines (per-breach SLA, lateness-independent)	RH	Lowest miss (7.54% / 5.14%)
Lateness penalised (hourly late fees, latency-sensitive downstream stages)	CADR	$\sim$60% lower tardiness (6.08 vs 20.30 min)
Minimise queue wait	RH	Best-tier wait (128.83 min, tied with SPT)
Lowest operational cost	neither	$\sim$1.46% spread; tune $C_{\max}$, not the scheduler

more deadlines.

Service quality versus cost. The choice between RH and CADR is a service-quality choice, not a cost trade-off. 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, \$6.06 per 24-hour run) and the most expensive (CADR, \$6.13) is only about 1.46%, a marginal, near-noise gap, and RH (\$6.09) and SPT+Rescue (\$6.10) sit inside that band. Between the two proposed schedulers the difference is under 1% (RH \$6.09 vs CADR \$6.13 per 24-hour run, a few cents), negligible beside the miss-count and tardiness differences. Cost therefore cannot discriminate between RH and CADR. 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:

- **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.

- **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 and 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 theoretical 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 [10]. 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 [16], 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 item 8 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 analysis is limited to the (cost, miss rate) plane. A full three-objective Pareto analysis (wait 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 2.28 percentage points; the sweep minimum is at $\theta = 180$ 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 [14, 16]. 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 on spot-market GPU fleets is a real, unsolved problem: rendering studios using RunPod or similar platforms have no principled guidance on which scheduling policy to deploy. This thesis fills that gap.

Within the fixed $C_{\max} = 5$ RunPod serverless constraint, heterogeneous deadline formulation (20% tight 1h / 80% loose 8h), and hectic overload regime ($\rho > 1$, $\sim$950 jobs/day), RH achieves the lowest deadline miss rate (7.54%, $d = -1.231$ vs SPT+Rescue, $p < 0.001$) and best-tier wait (128.83 min, tied with SPT, lower than SPT+Rescue; $d = -0.578$, $p = 0.003$), at the cost of high mean tardiness (20.30 min). CADR achieves near-RH miss (7.88%) at substantially lower tardiness (6.08 min), Pareto-dominating SPT+Rescue on both miss and wait. SPT achieves the lowest wait (129.41 min, tied with RH, $d = -2.882$ vs FIFO, $p < 0.001$) with miss statistically tied with EDF (13.85% vs 11.82%, $p = 0.075$ n.s.; both large vs FIFO, $p < 0.001$). SPT+Rescue is the conservative fallback: virtually tied on miss (12.40%, $d = -2.899$) with higher wait (135.21 min, $d = -2.406$). EDF matches SPT on miss (11.82%, n.s. vs SPT) but provides no wait benefit (FIFO-level wait); EDF has near-lowest tardiness (3.42 min, just above the Adaptive hybrid). FIFO/LCF/Balanced incur ≈22% 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 limit of 5 is the primary mechanism to reduce residual miss rates at saturation.

BIBLIOGRAPHY

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

[2] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. 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.

[3] 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.

[4] Han Zhao, Zhen Han, Zhi Yang, and Quanlu Zhang. Deep learning workload scheduling in GPU datacenters: A survey. *ACM Computing Surveys*, 56(6):1–38, 2024. doi: 10.1145/3638757.

[5] 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.

[6] 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*, 2020.

[7] 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.

[8] 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.

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

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Jingwei Li and Wei Xiao. GPU job scheduling based on deep reinforcement learning. In *Proceedings of the 2023 7th International Conference on High Performance Compilation, Computing and Communications*, pages 75–83, 2023. doi: 10.1145/3606043.3606049.

[17] Ting Zhang, Wei Zhang, Mingkun Wei, and Dongmei Zhao. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *Future Generation Computer Systems*, 150:396–417, 2024. doi: 10.1016/j.future.2023.09.011.

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

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

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

[21] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. MIT Press, 4th edition, 2023. ISBN 9780262048026. URL https://pbr-book.org.

[22] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition, 1988. doi: 10.4324/9780203771587.