

Evaluating Three Predefined Scheduling and Batching Configurations in Recommendation Systems: A Simulation-Based Study

Runze Li

Hong Kong Metropolitan University, Hong Kong, 999077, China
S1342047@live.hkmu.edu.hk

ABSTRACT

This study evaluates the impact of three predefined scheduling and batching configurations on recommendation-system serving performance using a custom Python-based simulation framework. The framework compares FIFO Scheduling with Fixed Batching, Priority Scheduling with Fixed Batching, and Priority Scheduling with Elastic Batching. The evaluation is limited to these three configurations rather than a complete two-by-two factorial design. Experiments use request features derived from the MovieLens-1M dataset and synthetic Poisson arrivals under low-, medium-, and high-load conditions.

Performance is evaluated using throughput, P95 latency, P99 latency, latency-distribution plots, timeout rate, and Welch's t-tests on request-level latency observations. The results show that the evaluated strategies behave similarly under low load. Under medium load, Priority Scheduling with Fixed Batching remains close to the FIFO baseline, while Priority Scheduling with Elastic Batching produces substantially higher P95 and P99 latency. Under high load, Priority Scheduling with Elastic Batching achieves the highest throughput but still has much worse P95 and P99 latency than the FIFO baseline.

The findings show mixed latency outcomes for elastic batching. The simulation results indicate a trade-off in which throughput-oriented batching can increase latency variability under heavier workloads. The study does not include deployed-system testing, hardware-utilization measurement, or recommendation-accuracy evaluation.

Keywords: Recommendation Systems, Scheduling, Batching, Tail Latency, Throughput, Simulation

1. INTRODUCTION

1.1 Research Background

Recommendation systems are widely used in online services to select and rank content for users ^[1-4]. Although recommendation accuracy is important, serving performance also affects user experience because recommendations are often requested during interactive sessions. Latency therefore becomes a system-level concern in addition to an algorithmic concern.

Average latency alone can hide slow responses experienced by a minority of requests. For this reason, tail-latency metrics such as P95 and P99 are useful when evaluating online serving systems ^[5,6]. This thesis focuses on request-processing behavior rather than recommendation accuracy.

1.2 Research Problem

Scheduling and batching are common request-management mechanisms, but their effect on latency distributions can vary by workload. A policy that improves throughput may also increase waiting time for some requests. The research problem is therefore to determine how selected scheduling and batching strategies affect throughput and tail-latency behavior in a controlled recommendation-serving simulation ^[7].

1.3 Research Objectives

Compare FIFO Scheduling with Fixed Batching and Priority Scheduling with Fixed Batching under different workload conditions.

Compare the three predefined configurations evaluated in this study: `fifo_fixed`, `priority_fixed`, and `priority_elastic`.

Evaluate throughput, P95 latency, P99 latency, timeout rate, and latency distributions.

Use Welch's t-test to compare request-level latency observations between configurations ^[8].

Identify trade-offs between throughput and tail-latency behavior within the three-configuration design, without assuming that one metric implies improvement in another.

1.4 Research Contributions

The study contributes a reproducible simulation framework, workload-specific comparisons across three predefined processing configurations, and an evidence-based discussion of throughput and tail-latency trade-offs. The contribution is limited to the evaluated configurations: `fifo_fixed`, `priority_fixed`, and `priority_elastic`.

2. LITERATURE REVIEW

2.1 Tail Latency and Recommendation Serving

Tail latency has been identified as an important concern in large-scale online systems because slow requests can affect perceived service quality even when average latency is acceptable [5]. Recommendation systems are commonly studied using public datasets such as MovieLens, which support transparent and reproducible experimentation [9].

General recommender-system literature describes a wide range of recommendation tasks and evaluation concerns [1-4]. This thesis does not evaluate recommendation accuracy. Instead, MovieLens-1M is used to create request features for a serving-layer simulation [9].

2.2 Statistical Evaluation

Welch's t-test is used because it does not require equal variances between samples [8]. In this study, the test is applied to request-level latency observations within each workload. The test results are interpreted alongside P95, P99, throughput, and timeout rate rather than treated as a substitute for tail-latency analysis.

3. METHODOLOGY

3.1 Simulation Framework

A Python discrete-event simulation framework was implemented to model request generation, scheduling, batch formation, service execution, and metric collection. The framework uses MovieLens-1M ratings to derive request features, including user tier, item popularity, and freshness [9]. Recent real-time recommendation-system work further shows why scalable serving architecture remains important for online recommendation scenarios [10]. Request arrivals are generated using exponential inter-arrival times with fixed random seeds.

3.2 Scheduling Model

The scheduler is either FIFO or strict priority. FIFO processes requests in arrival order. Priority scheduling computes a score as 0.4 times normalized user tier plus 0.3 times popularity plus 0.3 times freshness, and then processes higher-scoring requests first, breaking ties by arrival time.

3.3 Batching and Service Model

Fixed batching uses a batch size of 15. Elastic batching uses a batch size of 5 when queue length is below 50, 10 when queue length is from 50 to below 100, and 20 when queue length is 100 or greater. For a selected batch B with n requests, the simulator computes service time as $T(B) = \beta_0 + \beta_1 \times n^{\alpha} - \gamma \times \text{mean}(s_i)$, where $T(B)$ is service time in milliseconds, β_0 is the fixed base service time, β_1 is the per-request service coefficient, α is the batching exponent, γ is the priority speedup coefficient, and s_i is the priority score of request i in the batch. The parameter values are $\beta_0 = 20$ ms, $\beta_1 = 6$ ms, $\alpha = 0.85$, and $\gamma = 2.5$ ms. These values come from the fixed experimental configuration in `src/config.py` and are design parameters of the simulation rather than measurements fitted from production traces. No sensitivity sweep was executed, so this thesis does not report sensitivity ranges for these parameters. Hardware utilization is not evaluated.

4. EXPERIMENTAL SETUP

4.1 Dataset and Workload Configuration

The experiments use three workload levels and three processing configurations. The evaluated configurations are `fifo_fixed`, `priority_fixed`, and `priority_elastic`, as shown in Table 1.

Table 1. Experimental Configurations.

Workload	Request count	Arrival rate	Seed	Configurations	Batch settings
Low	500	50 requests/s	101	Evaluated configurations: fifo_fixed; priority_fixed; priority_elastic	Fixed = 15; elastic policy used only in priority_elastic
Medium	5,000	150 requests/s	202	Evaluated configurations: fifo_fixed; priority_fixed; priority_elastic	Fixed = 15; elastic policy used only in priority_elastic
High	10,000	230 requests/s	303	Evaluated configurations: fifo_fixed; priority_fixed; priority_elastic	Fixed = 15; elastic policy used only in priority_elastic

4.2 Metrics and Output Files

Metrics are computed from request-level observations. In the simulator, all generated requests continue to be processed until completion. A timeout is therefore a measurement label rather than a cancellation event: requests with latency greater than 1000 ms are counted in the timeout rate, but they are still included in the processed-request count and in throughput. Throughput is calculated as processed requests divided by simulated duration, where simulated duration is the difference between the maximum completion time and the first arrival time.

5. RESULTS AND DISCUSSION

5.1 Tail-Latency Results

This subsection presents the P95 and P99 latency results for each workload and processing method, as shown in Table 2.

Table 2. P95/P99 Results.

Workload	Method	P95 latency (ms)	P99 latency (ms)
low	fifo_fixed	63.81	70.19
low	priority_fixed	63.81	70.19
low	priority_elastic	63.81	71.58
medium	fifo_fixed	135.51	163.86
medium	priority_fixed	133.16	156.24
medium	priority_elastic	531.27	9720.20
high	fifo_fixed	9142.20	9503.84
high	priority_fixed	35123.89	45860.17
high	priority_elastic	24853.79	41689.86

Note. Values are copied from results/p95_p99_summary.csv and rounded to two decimals for presentation.

Under low load, all configurations produce similar P95 latency, and Priority Scheduling with Elastic Batching has only a slightly higher P99 latency than the two fixed-batch configurations. Under medium load, Priority Scheduling with Fixed

Batching remains close to FIFO, while Priority Scheduling with Elastic Batching increases P95 to 531.27 ms and P99 to 9720.20 ms.

Under high load, FIFO with Fixed Batching has P95 latency of 9142.20 ms and P99 latency of 9503.84 ms. Priority Scheduling with Fixed Batching has P95 latency of 35123.89 ms and P99 latency of 45860.17 ms. Priority Scheduling with Elastic Batching has lower tail latency than Priority Scheduling with Fixed Batching under high load, but remains substantially worse than the FIFO baseline with P95 latency of 24853.79 ms and P99 latency of 41689.86 ms.

5.2 Throughput Results

This subsection presents the throughput results for each workload and processing method, as shown in Table 3.

Table 3. Throughput Results.

Workload	Method	Throughput (requests/s)
low	fifo_fixed	47.89
low	priority_fixed	47.89
low	priority_elastic	47.89
medium	fifo_fixed	151.06
medium	priority_fixed	151.05
medium	priority_elastic	149.28
high	fifo_fixed	189.63
high	priority_fixed	189.62
high	priority_elastic	206.44

Note. Values are copied from results/throughput_summary.csv and rounded to two decimals for presentation.

Throughput is nearly identical under low load. Under medium load, Priority Scheduling with Elastic Batching has slightly lower throughput than the two fixed-batch configurations. Under high load, Priority Scheduling with Elastic Batching has the highest throughput at 206.44 requests per second, while FIFO with Fixed Batching and Priority Scheduling with Fixed Batching process approximately 189.62 requests per second.

5.3 Statistical Results

This subsection presents Welch's t-test results computed from request-level latency observations, as shown in Table 4.

Table 4. Welch's t-test Results.

Workload	Method A	Method B	t statistic	p value	Significant at .05
high	fifo_fixed	priority_elastic	23.791	2.478e-122	Yes
high	fifo_fixed	priority_fixed	0.231	0.818	No
high	priority_elastic	priority_fixed	-14.773	4.192e-49	Yes
low	fifo_fixed	priority_elastic	-0.113	0.910	No
low	fifo_fixed	priority_fixed	0.000	1.000	No
low	priority_elastic	priority_fixed	0.113	0.910	No
medium	fifo_fixed	priority_elastic	-10.250	2.053e-24	Yes
medium	fifo_fixed	priority_fixed	-0.019	0.985	No
medium	priority_elastic	priority_fixed	10.250	2.066e-24	Yes

Note. Welch's t-tests compare request-level latency samples within each workload. These tests do not directly test P95 or P99 values.

The low-load comparisons are not statistically significant at the .05 level. Under medium load, comparisons involving Priority Scheduling with Elastic Batching are statistically significant, while FIFO with Fixed Batching and Priority Scheduling with Fixed Batching are not significantly different. Under high load, FIFO with Fixed Batching and Priority Scheduling with Elastic Batching differ significantly, and Priority Scheduling with Elastic Batching and Priority Scheduling with Fixed Batching also differ significantly. The comparison between FIFO with Fixed Batching and Priority Scheduling with Fixed Batching is not significant according to the Welch test, even though their tail percentiles differ substantially.

The complete descriptive metrics for all workload and method combinations are summarized in Table 5.

Table 5. Full Results Summary.

Workload	Method	Request count	Average latency (ms)	P50 latency (ms)	P95 latency (ms)	P99 latency (ms)	Throughput (requests/s)	Timeout rate
low	fifo_fixed	500	42.71	43.13	63.81	70.19	47.89	0.00
low	priority_fixed	500	42.71	43.13	63.81	70.19	47.89	0.00
low	priority_elastic	500	42.80	43.22	63.81	71.58	47.89	0.00
medium	fifo_fixed	5000	90.09	87.37	135.51	163.86	151.06	0.00
medium	priority_fixed	5000	90.10	86.83	133.16	156.24	151.05	0.00
medium	priority_elastic	5000	392.31	91.61	531.27	9720.20	149.28	0.04
high	fifo_fixed	10000	4853.72	4806.43	9142.20	9503.84	189.63	0.91
high	priority_fixed	10000	4827.15	135.30	35123.89	45860.17	189.62	0.21
high	priority_elastic	10000	2768.75	156.80	24853.79	41689.86	206.44	0.14

5.4 Discussion

The strongest supported conclusion is that throughput and tail latency behaved differently in the simulation. Priority Scheduling with Elastic Batching produced the highest high-load throughput, but this did not translate into lower P95 or P99 latency relative to the FIFO baseline.

6. CONCLUSION

6.1 Summary of Findings

This thesis evaluated three predefined scheduling and batching configurations in a simulation-based recommendation-serving environment. The results show that request-management strategies had limited effect under low load. Under medium load, Priority Scheduling with Elastic Batching substantially increased P95 and P99 latency. Under high load, Priority Scheduling with Elastic Batching achieved the highest throughput but still had much higher P95 and P99 latency than FIFO with Fixed Batching.

The study therefore concludes that throughput optimization and tail-latency control should be treated as separate objectives. Hardware utilization, deployed-system behavior, and recommendation quality are outside the measured scope of this experiment.

6.2 Limitations and Future Work

The study is limited to a simulation environment, three processing configurations, a CPU-only service model, and synthetic arrivals derived from MovieLens-1M features. Future work could evaluate additional scheduling policies, fairness mechanisms, service models, and deployment settings.

REFERENCES

- [1] Aggarwal, C. C. (2016). *Recommender systems: The textbook*. Springer. <https://doi.org/10.1007/978-3-319-29659-3>
- [2] Ricci, F., Rokach, L., & Shapira, B. (Eds.). (2022). *Recommender systems handbook* (3rd ed.). Springer. <https://doi.org/10.1007/978-1-0716-2197-4>
- [3] Raza, S., Rahman, M., Kamawal, S., Toroghi, A., Raval, A., Navah, F., & Kazemeini, A. (2024). A comprehensive review of recommender systems: Transitioning from theory to practice. *arXiv*. <https://arxiv.org/abs/2407.13699>
- [4] Deldjoo, Y., He, Z., McAuley, J., Korikov, A., Sanner, S., Ramisa, A., Vidal, R., Sathiamoorthy, M., Kasirzadeh, A., & Milano, S. (2024). A review of modern recommender systems using generative models (Gen-RecSys). *arXiv*. <https://arxiv.org/abs/2404.00579>
- [5] Dean, J., & Barroso, L. A. (2013). The tail at scale. *Communications of the ACM*, 56(2), 74-80. <https://doi.org/10.1145/2408776.2408794>
- [6] Gupta, U., Hsia, S., Zhang, J., Wilkening, M., Pombra, J., Lee, H.-H. S., Wei, G.-Y., Wu, C.-J., & Brooks, D. (2021). RecPipe: Co-designing models and hardware to jointly optimize recommendation quality and performance. *arXiv*. <https://arxiv.org/abs/2105.08820>
- [7] Kossmann, F., Wu, Z., Turk, A., Tatbul, N., Cao, L., & Madden, S. (2024). CascadeServe: Unlocking model cascades for inference serving. *arXiv*. <https://arxiv.org/abs/2406.14424>
- [8] Welch, B. L. (1947). The generalization of Student's problem when several different population variances are involved. *Biometrika*, 34(1-2), 28-35. <https://doi.org/10.1093/biomet/34.1-2.28>
- [9] Harper, F. M., & Konstan, J. A. (2015). The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), Article 19. <https://doi.org/10.1145/2827872>
- [10] Liu, Z., Zou, L., Zou, X., Wang, C., Zhang, B., Tang, D., Zhu, B., Zhu, Y., Wu, P., Wang, K., & Cheng, Y. (2022). Monolith: Real time recommendation system with collisionless embedding table. *arXiv*. <https://arxiv.org/abs/2209.07663>