Assignment 2
COMP8110: Distributed Systems, S2, 2025
SID: 48599506, Name: Samip Pant

1 Introduction

Job/resource scheduling decides which job runs where and when in a distributed system (e.g., clusters and data centers). Good scheduling improves user-perceived latency and throughput while controlling infrastructure cost and keeping resources well utilised. Classic results show that policies biased toward earlier completion reduce mean response/turnaround time , and earliest-finish/list-scheduling ideas are a strong basis for practical heuristics [1, 2]. Large-scale production systems deploy lightweight, robust schedulers for heterogeneity and scale. [3, 4].
In this assignment, I design and implement an **EFT+ (Earliest-Finish-Time with boot/cost awareness)** scheduler for the `ds-sim` environment. The objective function is to **minimize the average turnaround time**, with secondary aim to **maintain or improve resource utilization** and **reduce total rental cost** compared to baselines.
The report is organized as follows. Section 2 provides problem defination. Section 3 describes the algorithm, the principle behind it and an example scenario. Section 4 describes the implementation of algorithm using python. Section 5 describes the result of our algorithm and evaluates with baseline algorithms. Section 6 concludes the report.

2 Problem definition

We consider online job scheduling on a heterogeneous cluster. Each job arrives with 3 requirements (cores, memory, disk) and an estimated runtime (from the simulator). Servers are of various types, each with total capacities and an operational state (inactive/booting/idle/active). Servers also have boot times and hourly rental rates.
The `ds-sim` protocol provides [5]:

- `GETS Capable c m d` returns the set of servers that can run a job needing the given cores, memory, and disk.
- Each server reports its current number of waiting and running jobs (wJobs, rJobs),
- The simulator emits job lifecycle events (`JOBN`, `JCPL`, `NONE`) as the system progresses.

Goal: Assign each arriving job to a specific server (type, id) to minimise mean turnaround time while not sacrificing utilisation and rent cost.

3 Algorithm description

3.1 EFT+: Earliest-Finish with boot/cost awareness

The algorithm maintains, for each server, a tiny predictive timeline of jobs already placed on that server. It then schedules each new job to the candidate server that finishes the job earliest, with two pragmatic refinements:

1. **Fast path for instant runs.** If any active/idle capable server has no queue and enough available cores now, schedule there immediately, preferring the smallest total-core server that fits (curbs cost and fragmentation).
2. **General EFT scoring with boot & cost shaping.** If fast path does not apply, then evaluate every capalble server as follows:
   (a) Earliest start time on that server from the predicted timeline (multi-core aware)
   (b) Boot penalty: if the server is not active/idle, add its bootup time (from `ds-system.xml`, else a small fallback)
   (c) Finish time: start + estimated runtime
   (d) Cost bias (tiny): add a small tie-breaker proportional to hourlyRate $\times$ runtime so that, when EFT ties occur, the algorithm leans toward cheaper servers

The algorithm chooses the server with the lexicographically smallest key
`(finish, start, rate_bias, cores_total, type, id)`.

3.2 Principles

- **EFT principle.** HEFT/EFT-style list scheduling tends to decrease mean turnaround by looking ahead to when cores are actually free rather than naively picking the first / 'best fit' server [2].
- **Short-job friendliness.** Preferring earlier finish indirectly benefits shorter jobs [1].
- **Multi-resource sanity.** Candidate filtering via `GETS Capable` respects cores/mem/disk feasibility [6].
- **Practicality.** Large systems (Borg, Paragon) succeed with lightweight and robust heuristics that combine speed, heterogeneity, and simple cost/availability signals [4, 3].

3.3 Scheduling Scenario

Servers (from `ds-system.xml`) :

- **S-4:** 4 cores, boot: 60 s, rate: $0.5/h, id {0,1,2}, states initially active for {0,1}, inactive for {2}.
- **L-8:** 8 cores, boot: 60 s, rate: $1.0/h, id {0}, state active.

Jobs arrive (t in seconds; est in seconds):

1. `J1 @ t=0`: 2 cores, 2 GB, 10 GB, est=300 The Fast path finds S-4(0) active with no queue and 2 free cores, so EFT+ schedules J1 on S-4(0) immediately and expects it to finish at t=300
2. `J2 @ t=10`: 6 cores, 4 GB, 10 GB, est=600 S-4s cannot fit 6 cores concurrently so L-8(0) is the only feasible choice. . Its predicted earliest start is t=10 (no queue), and finish at t=610.
3. `J3 @ t=20`: ⟨2 cores, 1 GB, 1 GB, est=120⟩ The fast path sees S-4(1) is idle and can run now, so EFT+ places J3 on S-4(1) with a predicted finish at t=140.

This demonstrate why short jobs often start immediately on small/cheaper servers while large jobs use the bigger machine. This separation avoids blocking and typically improves turnaround and cost.

General EFT scoring with boot & cost shaping : When no instant candidate exists (i.e., the fast path set is empty)

Scenario 1 Boot vs Queue trade-off :

1. `J4 @ t=50`: ⟨4 cores, 2 GB, 5 GB, est=200⟩
   - S-4(0) is running J1 (finishes at t=300), so earliest start on S-4(0) is 300 (no boot penalty); finish would be 300 + 200 = 500.
   - S-4(1) is running J3 (finishes at t=140). Even though it is currently busy, at t=140 all 4 cores become free, so earliest start 140; finish 140 + 200 = 340.
   - L-8(0) is running J2 (finishes at t=610); earliest start 610, finish 810.
   - S-4(2) is inactive (no queue), so earliest start is now + boot = 50 + 60 = 110; finish 110 + 200 = 310.

Decision: EFT scoring compares finishes and chooses S-4(2) because 310 is the smallest finish time, even though that requires paying a 60 s boot penalty. This example shows how the algorithm may prefer booting a small server to avoid long queue waits, thereby reducing turnaround.

Scenario 2 Cost tie-break when finishes tie :

1. `J5 @ t=100`: ⟨2 cores, 1 GB, 1 GB, est=240⟩
   - Suppose both S-4(0) and L-8(0) will become free at t=100 (e.g., J1 and J2 both just completed)
   - Predicted start is 100 on either; predicted finish 340 on either.
   - Since `finish` and `start` tie, EFT+ uses a tiny cost bias (`rate_bias =  × hourlyRate × est`) as a tie-breaker.
   - S-4 has $0.5/h while L-8 has $1.0/h, so S-4 yields a smaller `rate_bias`.

Decision: Schedule J5 on S-4. The bias is intentionally tiny so it never overrides a real finish-time advantage, it only resolves ties toward the cheaper type.

4 Implementation

The algorithm was implemented on a file `final.py` .

Language & I/O [5]: The algorithm is implemented on python. It communicates with `ds-sim` line-based protocol over a TCP socket. Commands are newline-terminated and flushed immediately to ensure protocol compliance.

- **Handshake:** `HELO`, `AUTH`.
- **Polling:** repeatedly send `REDY`, parse events (`JOBN`, `JCPL`, `NONE`).
- **Discovery:** `GETS All` once (initial cluster view), then `GETS Capable` per job.

Data structures :

- `ServerSched`: per-server **min-heap** of `(end_time, cores_used)` to model the server's predicted future.
  - `earliest_start_for(now, c_need)` prunes finished entries, sums running cores, then walks future release times to find the first time enough cores are free (multi-core aware) [2] .
  - Complexity per decision is small: heaps are short (only jobs placed on that server), and we evaluate only the capable set returned by the simulator.

Boot/cost metadata: `ds-system.xml` is parsed (if present) for bootupTime, hourlyRate, and cores per server type. If missing, the code falls back to safe defaults (e.g., small boot penalty).
Scheduling loop:

1. If instant candidates exist (active/idle, zero queue, enough cores now): pick the smallest total-core server among them and schedule immediately.
2. Else, compute `(finish, start, rate, bias, ...)` as in §3.1 for each capable server and pick the minimum.
3. On `JCPL`, prune finished tasks for that server (keeps the prediction fresh).

Additional details: The client uses buffered `makefile()` streams (`utf-8`, `\n`) and explicit `flush()` on every write to avoid newline desynchronisation. Tie-breaking is deterministic for reproducibility. The socket is closed in a `finally` block for robustness.

5 Evaluation

5.1 Simulation settings

- **Test cases:** 20 configs covering short/medium/long mixes, low/medium/high loads. Provided in the `/TestConfig` folder. The config were like chosen to stress short-job latency, probe heterogeneous capacity and queueing effects , and challenge cost/boot-time trade-offs under heavier mixes .
- **Baselines:** ATL, FF, BF, FC, FAFC (as provided with the simulator).
- **Metrics:** average turnaround, average resource utilisation, and total rental cost
  - Turnaround time: completion time - arrival time
  - Resource utilization (%): time-weighted fraction of active resources used
  - Total rental cost: simulation's accumulated server rental charges
- `ds_test.py` script was used to test for all 20 config in one pass.

5.2 Results

Using the `ds_test.py`.

Final results:
Handshake: 1/1
Scheduled All Jobs: 2/2
Average Performance: 2/2

Turnaround Performance: 8/10

The algorithm (`final.py`) successfully completed the protocol handshake , scheduled all jobs across both required end-to-end evaluations, satisfied the script's average-performance verification against the baselines, and achieved a score of 8/10 on the turnaround-time assessment.

Algorithm	Turnaround time	Utilisation	Rental cost
ATL	294261.95	100.00	391.03
FF	1830.30	71.53	551.55
BF	2247.70	67.54	550.03
FC	328410.70	97.67	378.15
FAFC	1462.85	71.86	551.35
My algo (EFT+)	1450.10	73.53	522.75

Table 1: Average Performance.

Observations:

- **Turnaround:** EFT+ achieves the best average (1,450.10), slightly beating FAFC and substantially beating FF/BF. The instant-run fast path plus EFT look-ahead trims queuing delay on busy mixes, especially for short-heavy configs (e.g., `16-short-high`, `40-short-high`).
- **Utilisation:** EFT+ improves average utilisation to 73.53% (+1.7 pp vs FAFC, +2.0 pp vs FF). The small-server preference in the fast path reduces fragmentation and keeps small nodes busy.
- **Cost:** EFT+ reduces total rental cost to 522.75, around 5.2% improvement vs FAFC (551.35) and FF (551.55). The gentle cost tiebreak and small-server first choice in instant placements contribute[7, 8].

Pros and Cons of each algorithm:

- **ATL**
  - Pros: Simple, maximal utilisation.
  - Cons: Catastrophic turnaround/cost in heterogeneous mixes.
- **FF**
  - Pros: Fast, decent latency on some mixes.
  - Cons: Can queue on unlucky servers, moderate cost.
- **BF**
  - Pros: Packs tightly.
  - Cons: Over-queues "best-fit" boxes, lower utilisation and higher latency.
- **FC**
  - Pros: Low cost.
  - Cons: Very poor turnaround, insensitive to queuing/fit quality.
- **FAFC**
  - Pros: Active-first improves latency vs FC/FF, strong baseline for turnaround.
  - Cons: Cost is high and utilization is low
- **EFT+**
  - Pro: Best average turnaround, higher utilisation vs FAFC/FF, lower cost vs FAFC/FF.
  - Cons: More expensive than FC, tie-break weights could be tuned further

5.3 Discussion

- **Estimate sensitivity:** EFT+ relies on the simulator's runtime estimates. In real systems, adding feedback-based correction (e.g., exponential smoothing of job types) would stabilise decisions [2, 3, 4].
- **Boot-time awareness:** Using `ds-system.xml` boot durations is effective; future work could learn per-type cold/warm start distributions.
- **Fairness & tail:** Like SRPT/EFT, mean turnaround improves, but fairness/tail may suffer under skew. Incorporating a bounded slowdown or wait-time boost can limit starvation [1, 6].
- **Cost shaping:** The current cost bias is deliberately small (tie-breaker). For cost-critical workloads, a bi-objective score (finish time + ·cost) could be exposed as a knob, trading a few % of latency for double-digit cost savings[8, 7].
- **Backfilling variant:** A conservative backfill variant could further lift utilisation without harming predicted start times of queued jobs.

6 Conclusion

EFT+ combines a fast instant-run path with a look-ahead earliest-finish selection, lightly shaped by boot time and rental rate. Across 20 diverse configurations, it achieves the best average turnaround (1,450.10), improves utilisation (73.53% vs 71–72%) and reduces cost relative to strong baselines (5% vs FAFC/FF). The approach is simple, fast, and robust, well aligned with practice in large-scale cluster managers while still leaving room for tunable fairness and cost-latency trade-offs.

References

[1] N. Bansal and M. Harchol-Balter, "Analysis of srpt scheduling: Investigating unfairness," in *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '01, (New York, NY, USA), pp. 279–290, Association for Computing Machinery, 2001.

[2] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[3] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, (New York, NY, USA), pp. 77–88, Association for Computing Machinery, 2013.

[4] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the 10th European Conference on Computer Systems*, EuroSys '15, (New York, NY, USA), pp. 1–17, Association for Computing Machinery, 2015.

[5] School of Computing, Macquarie University, Sydney, NSW, Australia, *ds-sim User Guide*, 2025.

[6] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '11, (Berkeley, CA, USA), pp. 323–336, USENIX Association, 2011.

[7] N. A. Hussein, E. M. Khalid, and M. A. Fakhreldin, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, vol. 142, p. 102539, 2019.

[8] A. Khan and I. Ahmad, "A heuristic-based cost-effective task scheduling model for grid computing," *Journal of Parallel and Distributed Computing*, vol. 67, no. 8, pp. 924–937, 2007.