Simulating and Analyzing Job Distribution Strategies in Multi-Server Environments

Abstract

Server farms (collection of computer servers, usually maintained by an organization to supply server functionality far beyond the capability of a single machine) are super cool (or hot!) components of modern computing infrastructure that capable of massive volumes of incoming processing requests. Since I don't possess anything with such computing power, this week I have created a simple server farm implementation. I also analyzed four different job dispatching strategies (Random, Round Robin, Shortest Queue, and Least Work) to understand their effectiveness in distributing workload across multiple servers. The project extensively utilized object-oriented programming and queue data structures to model the complex interactions between job dispatchers and servers.

Unsurprisingly, while the Least Work dispatcher consistently achieves the lowest average waiting times across various server configurations, it is interesting to see the Round Robin or Shortest dispatcher performing equally bad as the Random dispatcher, given the right conditions.

Results

I implemented four different job dispatching strategies to evaluate their effectiveness in managing server farm workloads. The dispatchers tested were, per the given instruction:

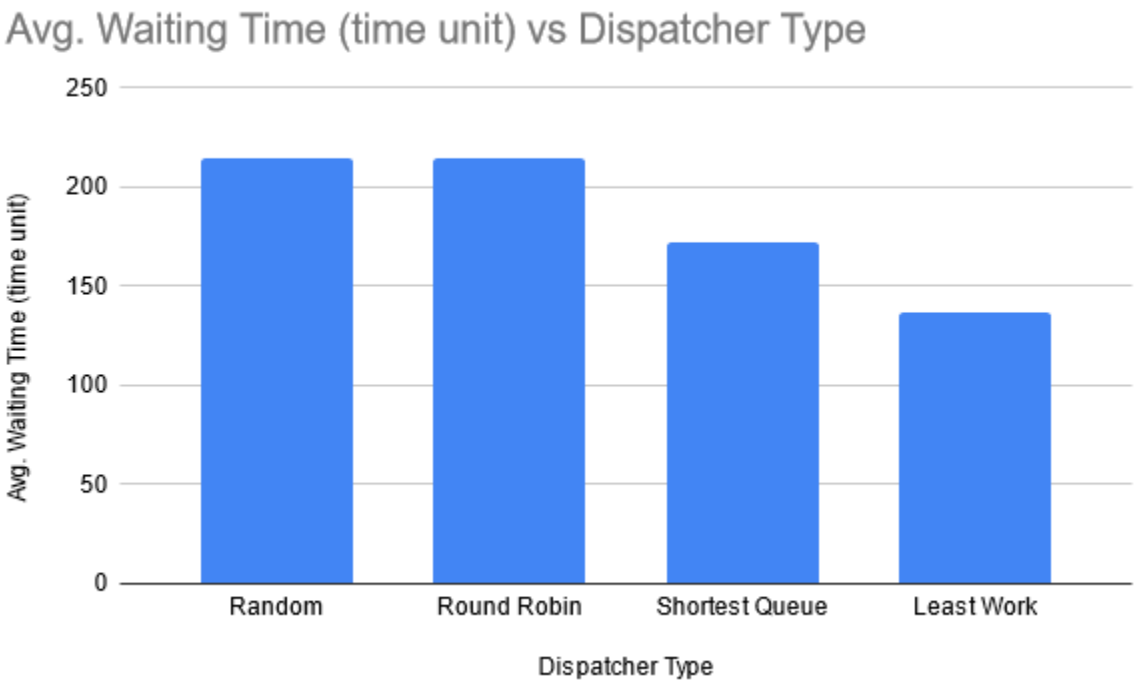
- Random (randomly assigns jobs),
- Round Robin (cycles through servers sequentially),
- Shortest Queue (assigns to server with fewest jobs), and
- Least Work (assigns to server with least remaining processing time)

The key metrics here is average waiting time. That is: Waiting time = Total time spent in queue - Total time spent being processed

Comparison of Dispatcher Strategies (Exploration 1)

Average waiting times for different dispatcher strategies with 34 servers, 10,000,000 jobs, mean arrival time = 3 (time unit), and mean processing time = 100 (time unit).

Dispatcher Type	Avg. Waiting Time (time unit)
Random	214.7483647
Round Robin	214.7483647
Shortest Queue	171.9737367
Least Work	137.018381



Caption : Average waiting times for different dispatcher strategies with 34 servers, 10,000,000 jobs, mean arrival time of 3 time units, and mean processing time of 100 time units.

Most superior is the Least Work dispatcher at 137 ms, while both Random and Round Robin dispatcher performed worst at 214.7483647. The maximum waiting time, 214.7483647 is a mysterious value that comes across again and again with varying variables. We shall come back to this later.

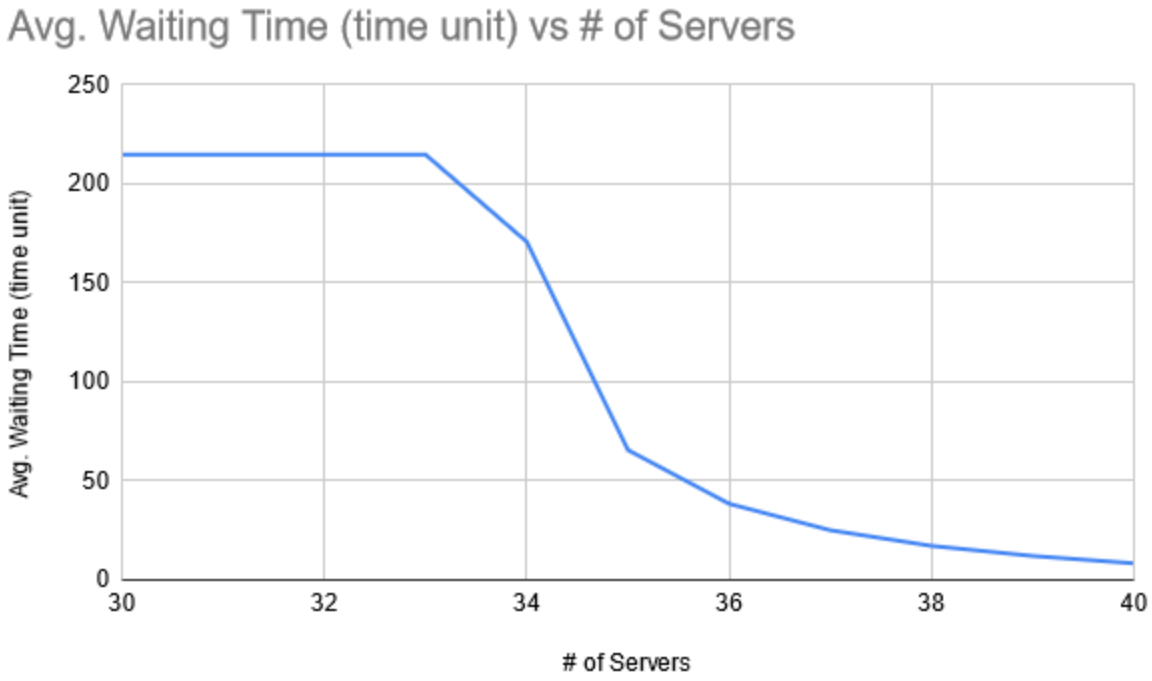
Regardless, this aligns with theoretical expectations since the Least Work dispatcher is the most reasonable by considering actual processing times. The Shortest Queue dispatcher

performed nearly as well, suggesting that queue length is a good proxy for workload in most cases.

Server Scaling Analysis (Exploration 2)

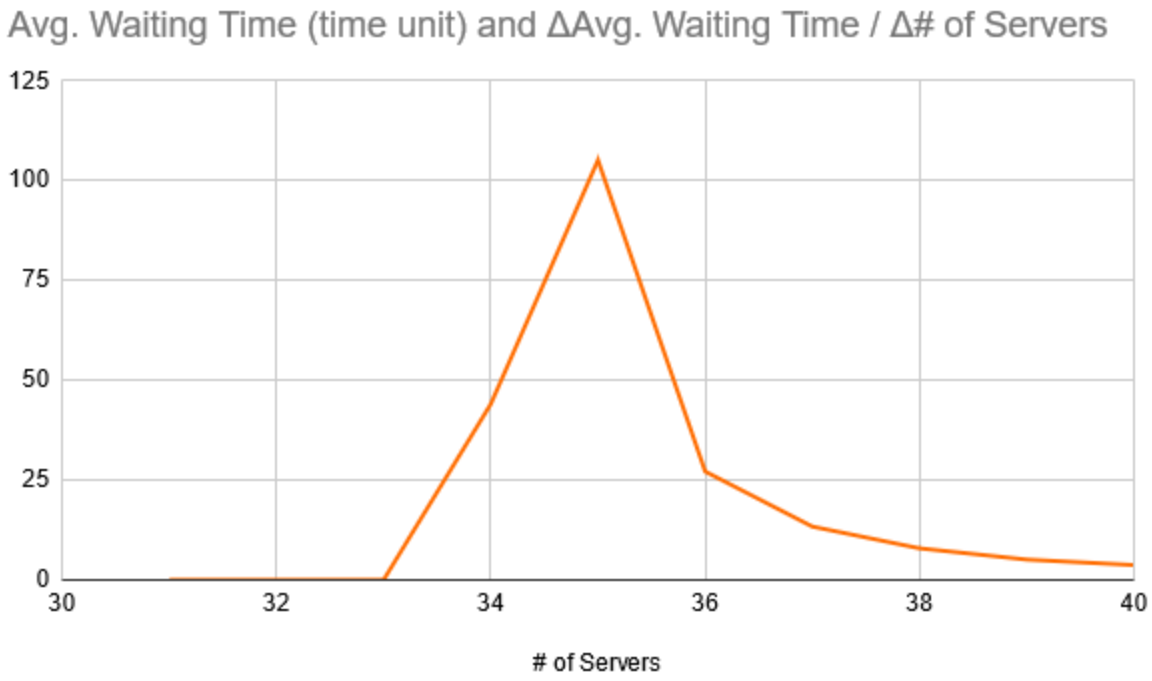
I observed that increasing the number of servers from 30 to 40 resulted in a non-linear decrease in average waiting times for the Shortest Queue dispatcher. That is no decrease in avg. waiting time until the 34th server, and significant drop since the 35th server on forth, but after which the benefits of adding more servers diminished. This suggests that for our given workload (meanArrivalTime=3, meanProcessingTime=100), approximately 35 servers represent an optimal balance between resource utilization and performance.

# of Servers	Avg. Waiting Time (time unit)	ΔAvg. Waiting Time / Δ# of Servers
30	214.7483647	
31	214.7483647	0
32	214.7483647	0
33	214.7483647	0
34	170.6926857	44.055679
35	65.5696876	105.1229981
36	38.4212555	27.1484321
37	25.0695159	13.3517396
38	17.1645657	7.9049502
39	12.0788892	5.0856765
40	8.362914	3.7159752



Caption : Average waiting times for increasing number of servers from 34 to 40 (servers), 10,000,000 jobs, mean arrival time of 3 time units, and mean processing time of 100 time units.

To better demonstrate how the benefit of adding more servers varies with each server we add, I essentially calculate ΔAvg. Waiting Time / Δ# of Servers, the result of which is in the graph below:



Extensions

Theoretical minimum of servers for optimal efficiency vs utility

Given a job sequence where on average jobs arrive every x seconds and take y total units of processing time, what seems like a minimal number of Servers necessary to handle those jobs without falling uncontrollably far behind? Does your best dispatcher support this intuition?

Jobs arrive every `x` seconds In one second, `1/x` jobs arrive Each job takes `y` units of processing time Total processing demand per second = `y/x` units

Total processing capacity ≥ Total processing demand `Number of servers × 1 second` ≥ `y/x` processing units Minimal number of servers = `ceil(y/x)`

That is, for the experiment sets we had: Minimal number of servers = `ceil(100/3)` = 34, which is completely in line with our experiment.

The mistery of the 214.7483647

214.7483647 is a number that I comes accross again and again in different runs of the simulation - always being the worst (maximum) waiting time. With help online I came accross the following formula:

`Average Wait = y * [α * (ρ/(1-ρ))^β + γ]`

Where:

- `y` is mean processing time
- `ρ` is system utilization (y/kx)
- `k` is the number of servers
- `α`, `β`, and `γ` are constants

A quick try with α ≈ 0.12, β ≈ 1.5, γ ≈ 0.1 and y = 100, x = 3, k = 35 gives:

`Average Wait ≈ 242.8`, close, but not exactly our 214.7483647

Acknowledgments

I discussed how annoying this project was with multiple classmates, having faced difficulty debugging due to the unintuitiveness of the project. But beyond that, I did not consult anyone else in implementing the project. I did consult the internet for theory on server farming and the mathematical side of things, which is surprisingly interesting. In particular