

Assignment 1

Distributed Computing – Mohammad Hossein Mahrooghi – Università di Genova 2026

Introduction

- In this assignment, as required, we study the behavior of jobs executed in multiple queues and analyze their performance under different load balancing strategies, with a particular focus on the supermarket (power-of-d choices) method.
- We first analyze the system under the standard memoryless assumption, where job inter-arrival times and service times follow exponential distributions.
- The simulation results obtained under this setting are compared with the theoretical results presented in the course.
- We then extend the analysis by modeling job arrivals and service times using Weibull distributions, allowing us to study non-memoryless workloads and evaluate how different workload characteristics impact the system behavior.
- Finally, we analyze an additional extension introduced in the simulator and discuss its effects on system performance and trade-offs.

Implementation details and execution parameters are reported in the Appendix.

Experiments

Experimental evaluation under exponential (memoryless) workloads.

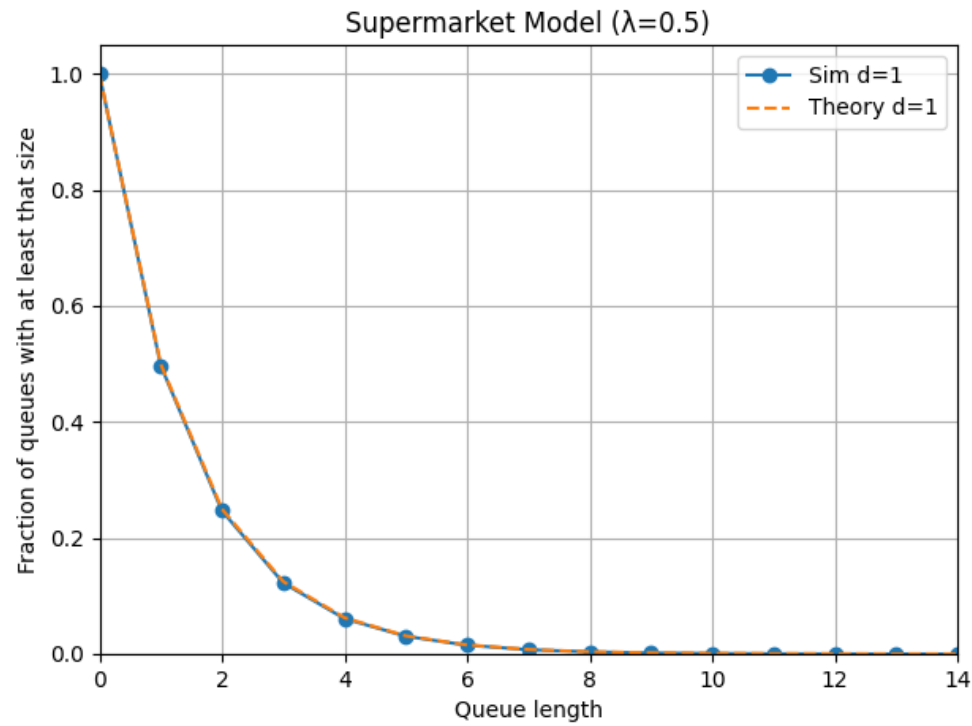
Following peer review feedback, we fixed the random seed (seed = 1) to ensure reproducibility of the experiments.

We first validate the simulator using $d = 1$, which corresponds to random assignment.

The experimentally observed queue length distribution closely matches the theoretical results.

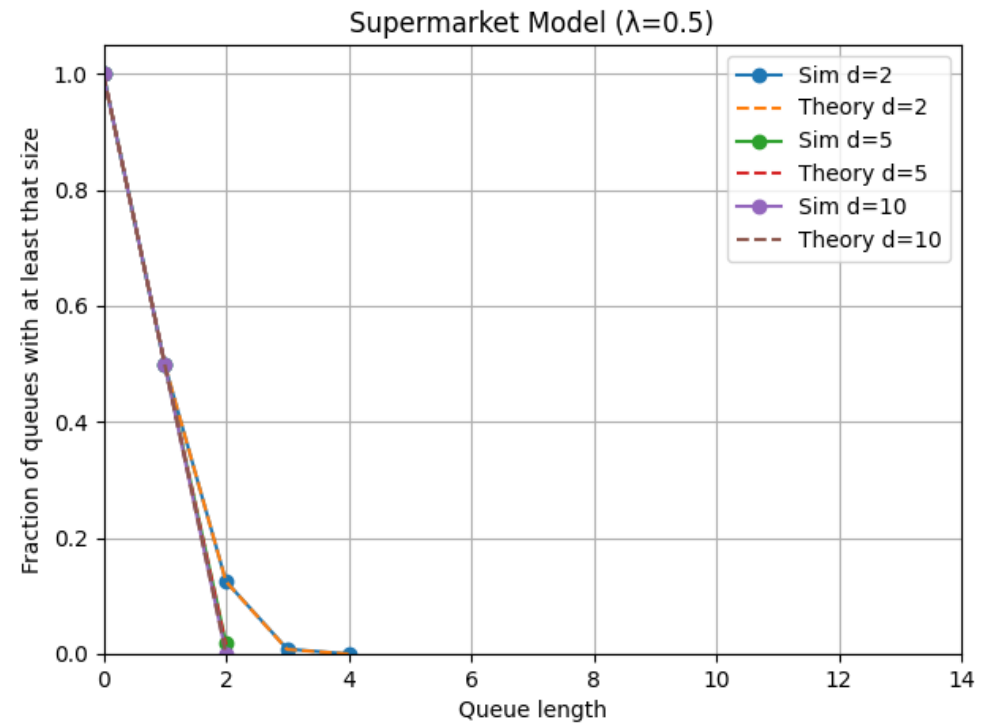
We then repeat the experiments for increasing values of d ($d = 2, 5, 10$).

As d increases, the probability of observing long queues decreases significantly,
confirming the effectiveness of the supermarket (power-of- d choices) method.



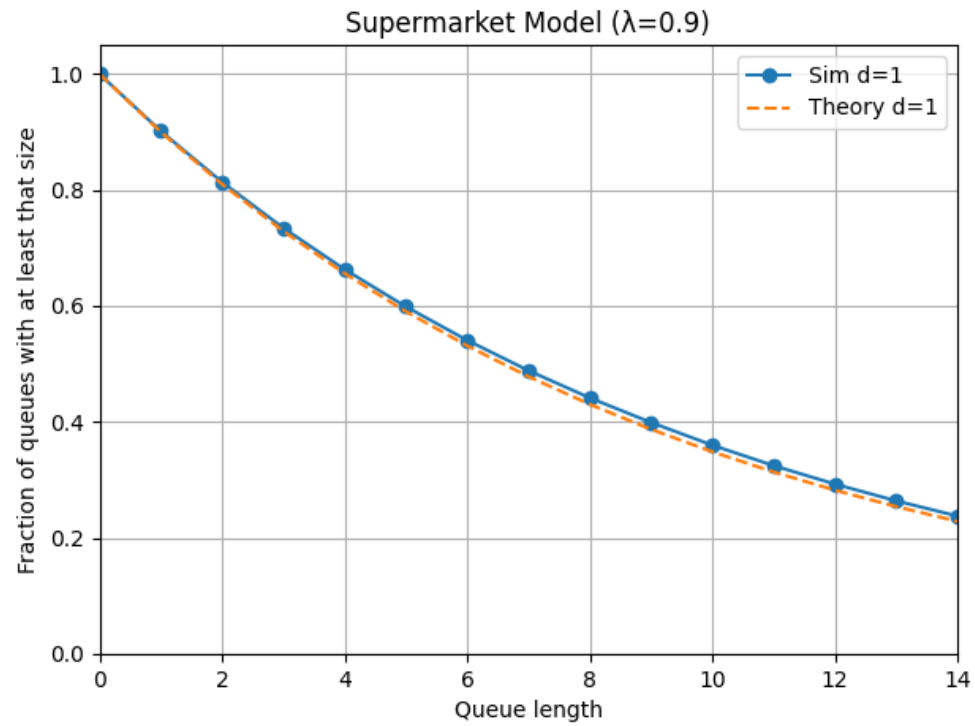
$d = 1$ (random assignment),

experimental results match the theoretical curve.



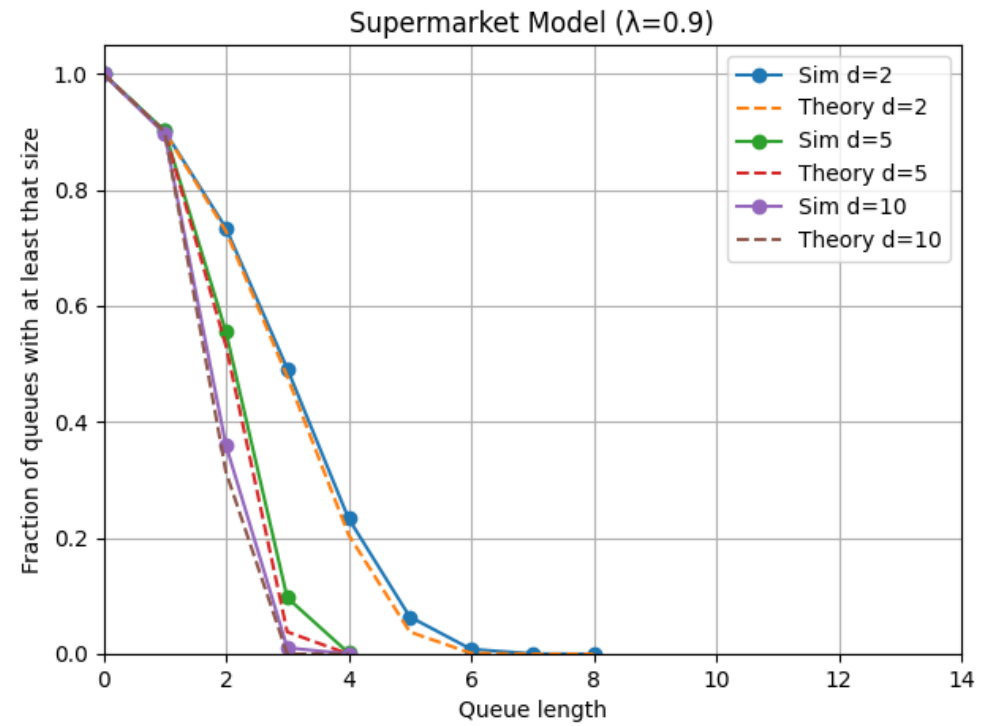
increasing d significantly reduces the probability of long queues.

Parameters: $\lambda = 0.5$, $\mu = 1$, $N = 50$, $MAXT = 20000$.



$d = 1$ (random assignment),

experimental results match the theoretical curve.



increasing d significantly reduces the probability of long queues.

Parameters: $\lambda = 0.9$, $\mu = 1$, $N = 50$, $MAXT = 20000$.

Weibull distributions

The exponential distribution assumes memoryless arrivals and service times, which is often unrealistic in real systems.

To model more realistic workloads, we replace exponential distributions with **Weibull distributions** for both job arrivals and service times.

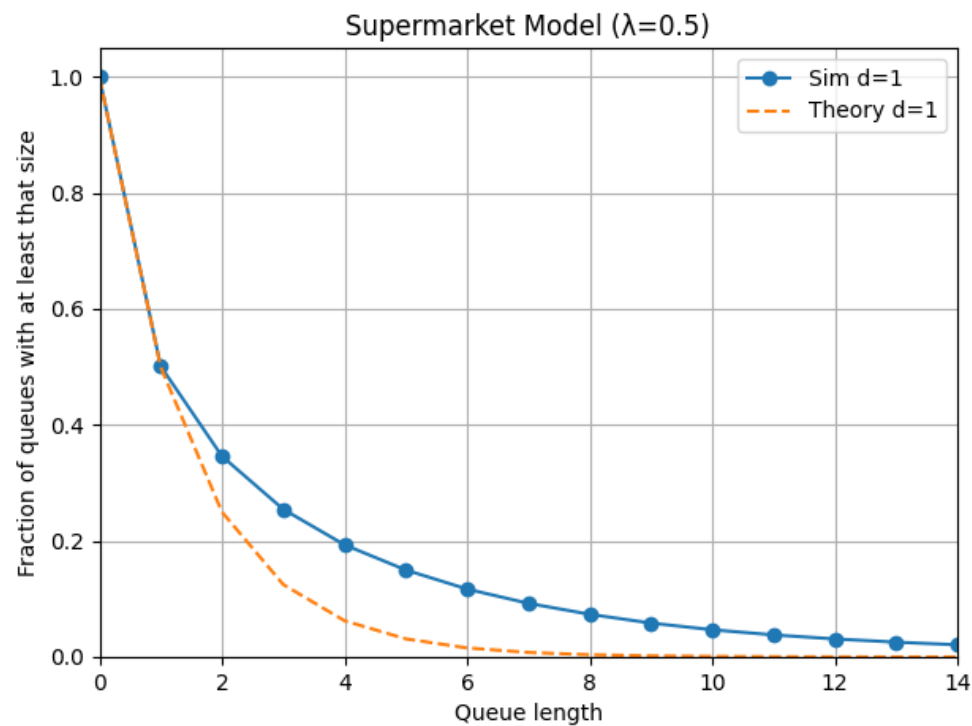
Impact of Weibull-distributed workloads on queue length distributions.

shape = 1 reproduces the exponential case.

shape < 1 leads to heavier tails and longer queues.

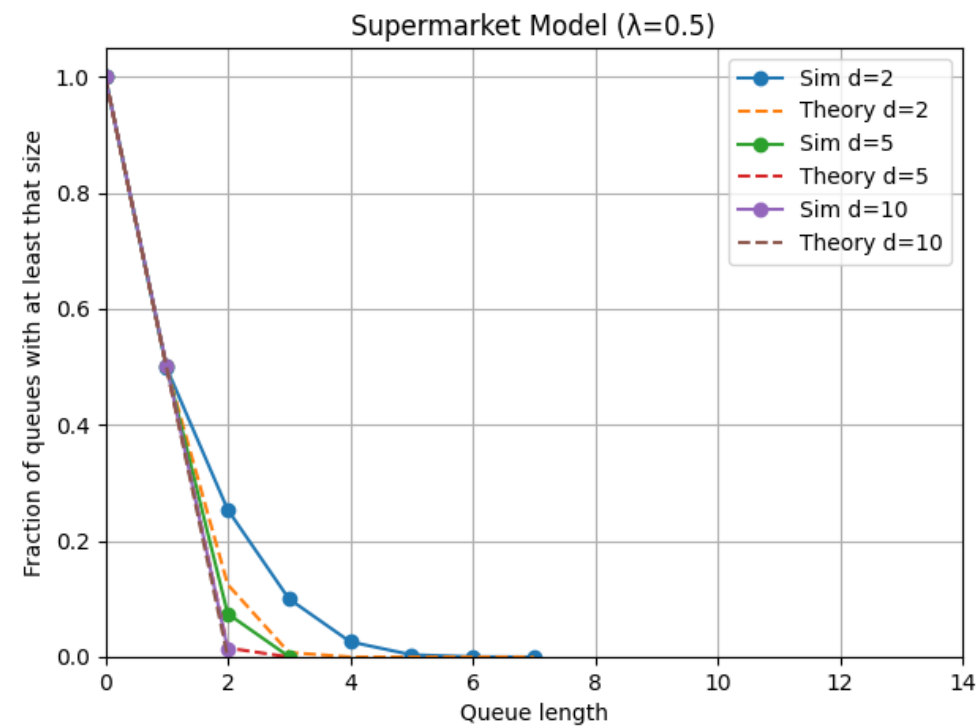
shape > 1 results in more regular behavior and shorter queues.

Shape $< 1 \Rightarrow$ Heavier tails



$d = 1$

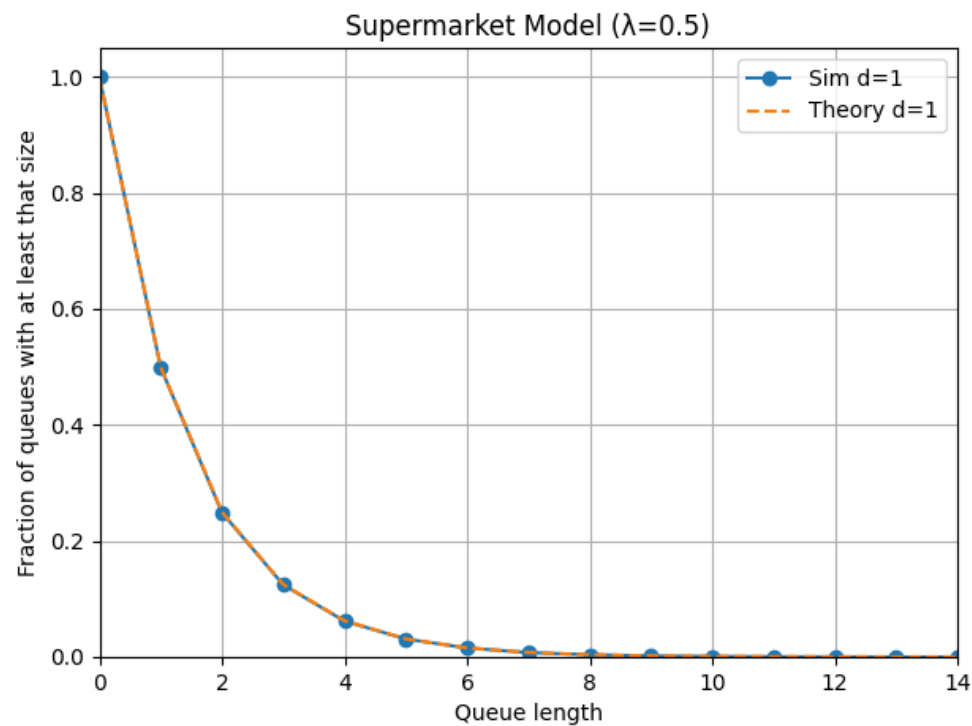
Parameters: $\lambda = 0.5$, $\mu = 1$, $N = 50$, MAXT = 20000



$d = 2, 5, 10$

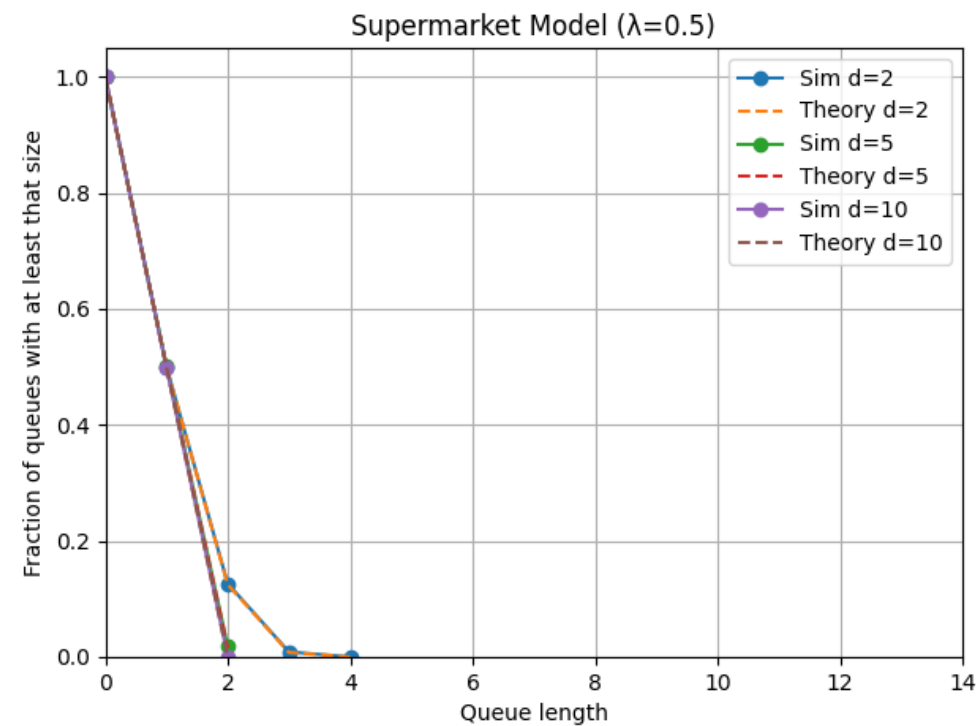
shape $< 1 \Rightarrow$ WSS = 0.5 , WAS = 0.5

Shape = 1 \Rightarrow exponential



$d = 1$

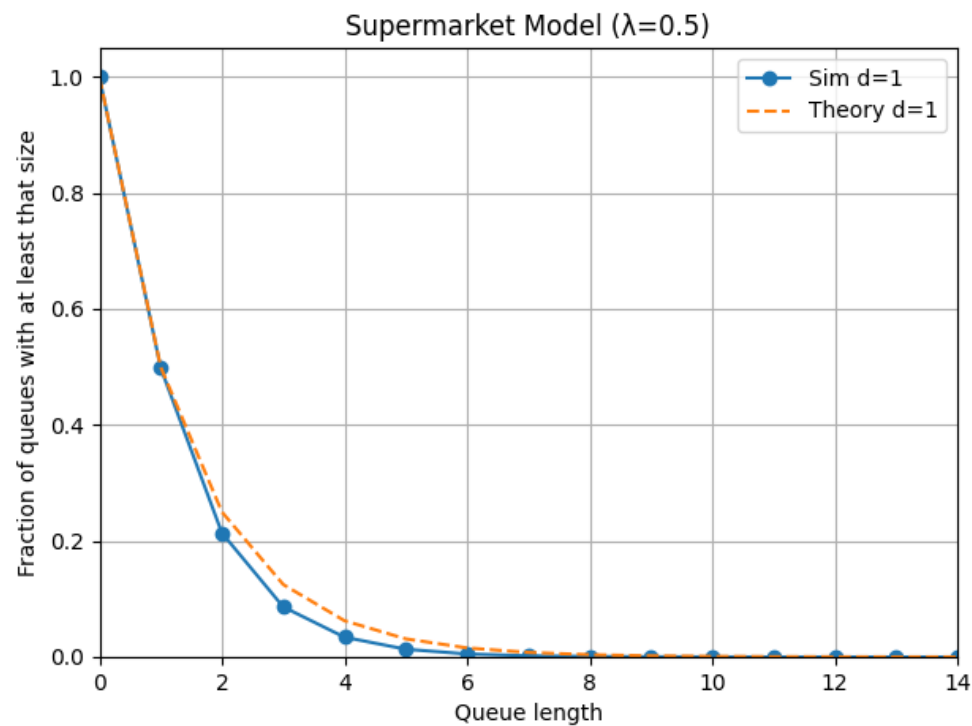
Parameters: $\lambda = 0.5$, $\mu = 1$, $N = 50$, MAXT = 20000



$d = 2, 5, 10$

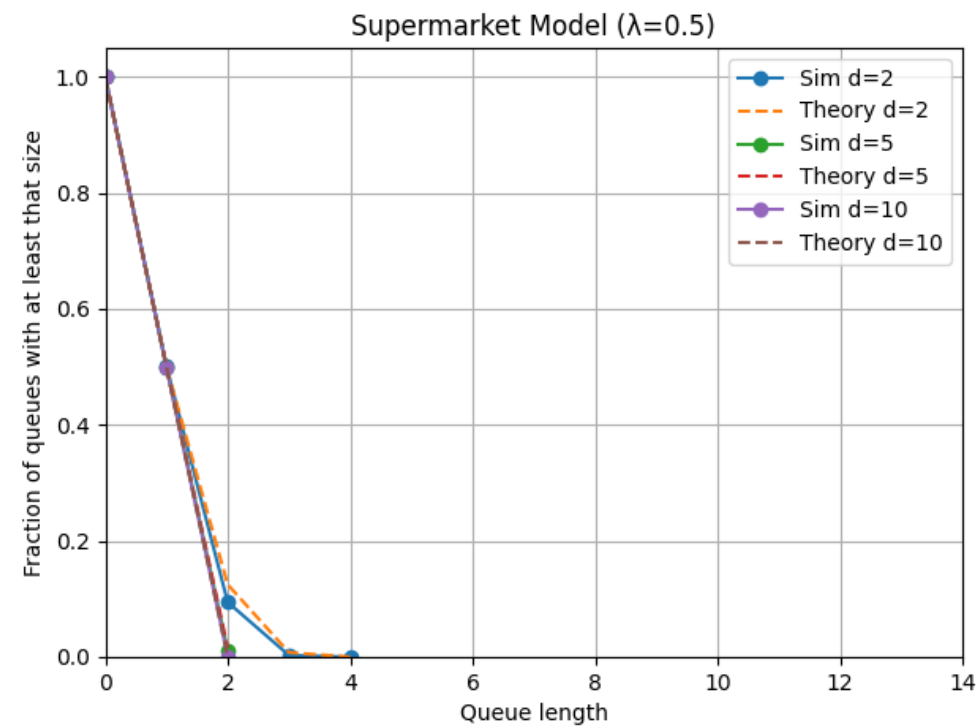
shape < 1 \Rightarrow WSS = 1, WAS = 1

Shape > 1 \Rightarrow shorter queues



d = 1

Parameters: $\lambda = 0.5$, $\mu = 1$, $N = 50$, MAXT = 20000



d = 2 , 5 , 10

shape < 1 \Rightarrow WSS = 1.5 , WAS = 1.5

Simulator Extension (Finite Queue)

In the assignment, queues are assumed to have infinite capacity, meaning that they can always accept new jobs.

In real systems, however, queues have limited capacity.

Once a queue becomes full, it cannot accept additional jobs.

For this reason, we extend the simulator with a configurable queue capacity and define how the system should behave when a queue is full.

Simulator Extension (Finite Queue)

When a queue reaches its maximum capacity, two behaviors are supported:

- drop: the incoming job is immediately discarded
- retry: the job tries to join another queue, up to a maximum number of attempts

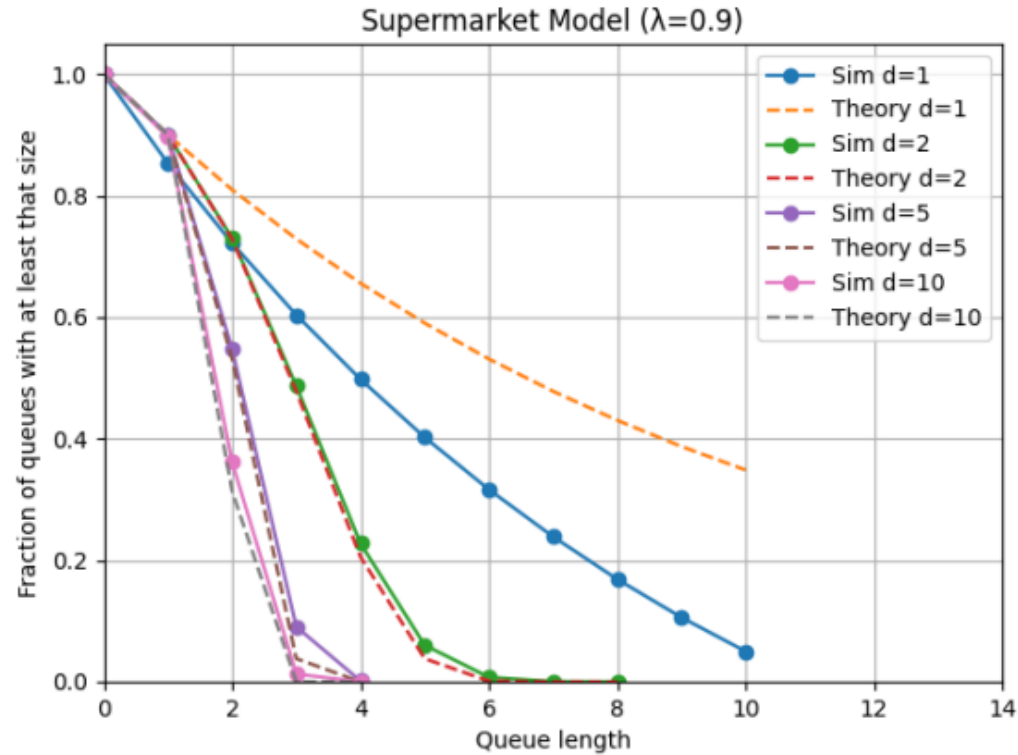
The behavior of the system is controlled through the following parameters:

- QUEUE_CAPACITY: maximum number of jobs which queues can hold (including the running job)
- OVERFLOW_BEHAVIOUR_OPTION: selects the policy to apply when a queue is full (drop or retry)

Example:

```
.\run_dc.ps1 -D 1,2,5,10 -LAMBDA 0.9 -MU 1 -N 50 -MAXT 20000 -QUEUE_CAPACITY 10 -OVERFLOW_BEHAVIOUR_OPTION retry -MAX_RETRIES 3
```

Experiments

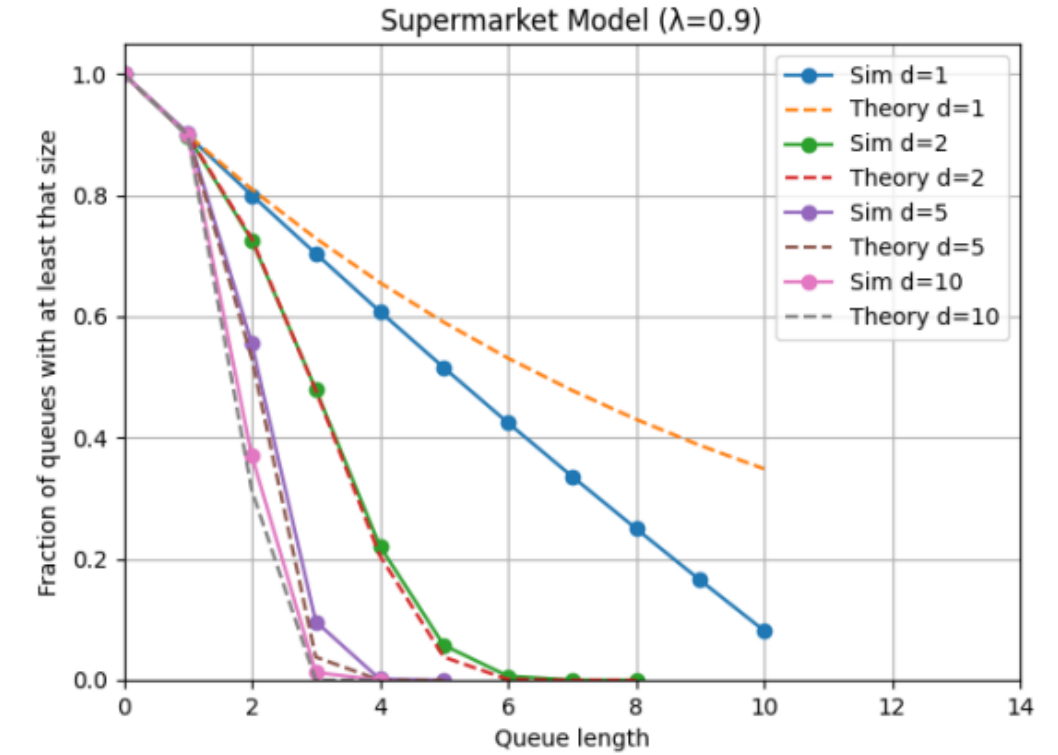


Drop

Dropped jobs:

d1 => 45061 / 899915 => drop rate is: 5%

d2, d5, d10 => 0, drop rate is :0%



Retry = 3

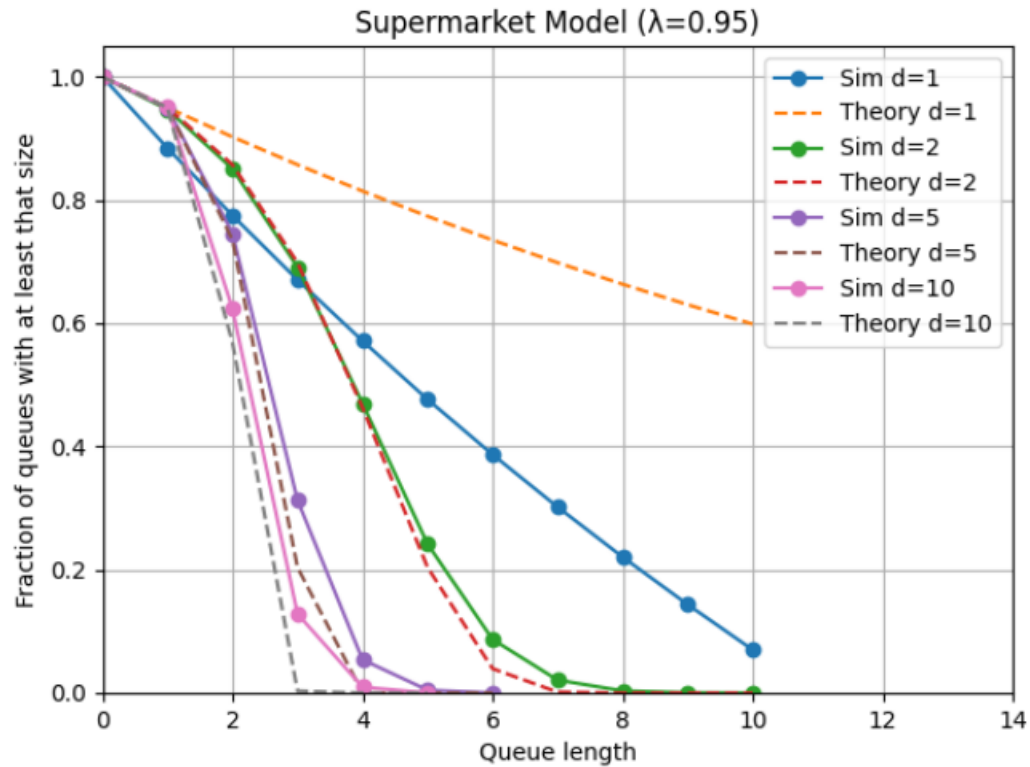
Dropped jobs:

168 / 899021 => drop rate is: 0.02%

d2, d5, d10 => 0, drop rate is :0%

Parameters: $\lambda = 0.9$, $\mu = 1$, $N = 50$, MAXT = 20000, QUEUE_CAPACITY = 10

Experiments

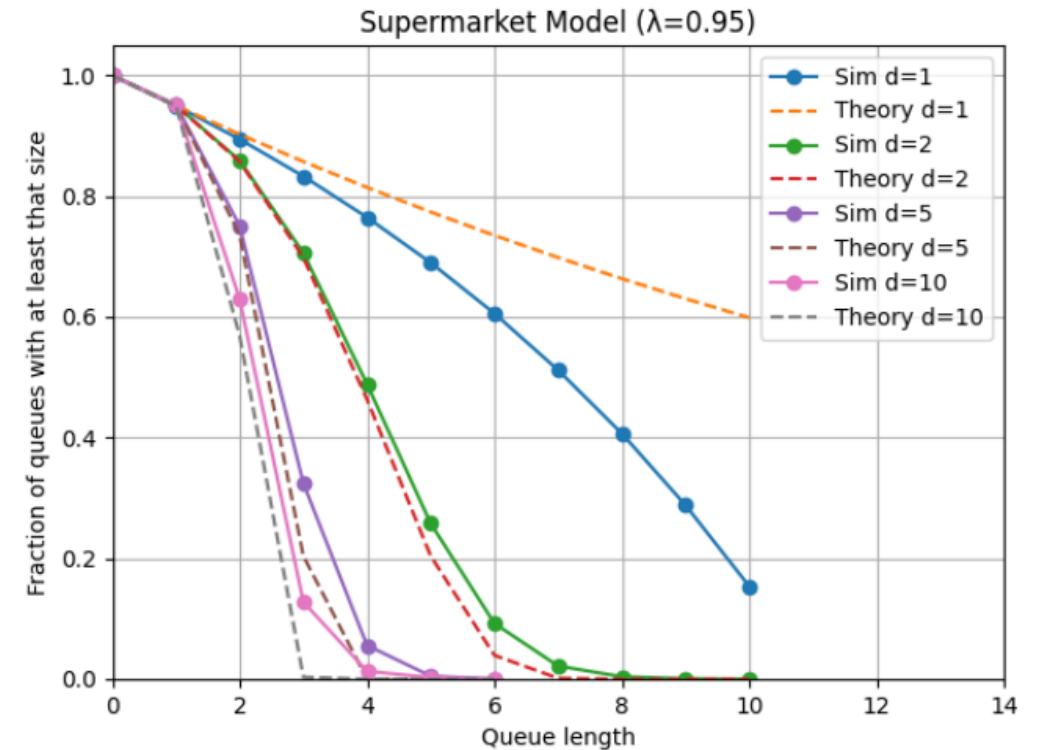


Drop

Dropped jobs:

d1 => 66353 / 949527 => drop rate is: 7%

d2, d5, d10 => 0, drop rate is :0%



Retry = 3

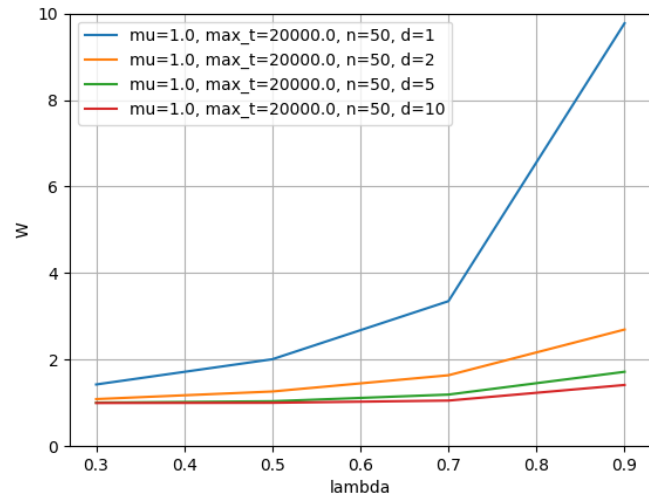
Dropped jobs:

1495 / 950906 => drop rate is: 0.16%

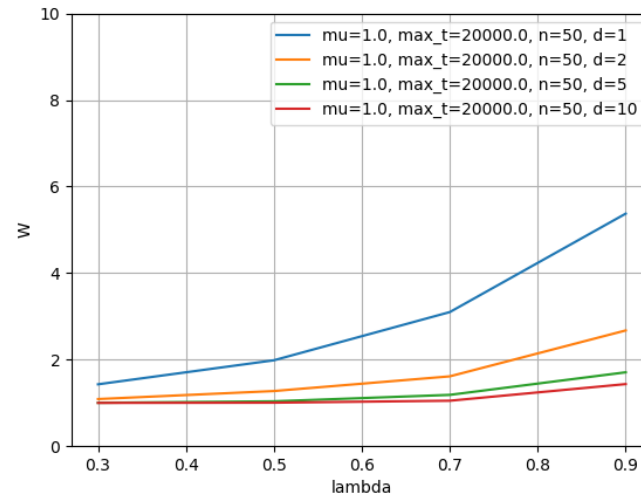
d2, d5, d10 => 0, drop rate is :0%

Parameters: $\lambda = 0.95$, $\mu = 1$, $N = 50$, MAXT = 20000, QUEUE_CAPACITY = 10

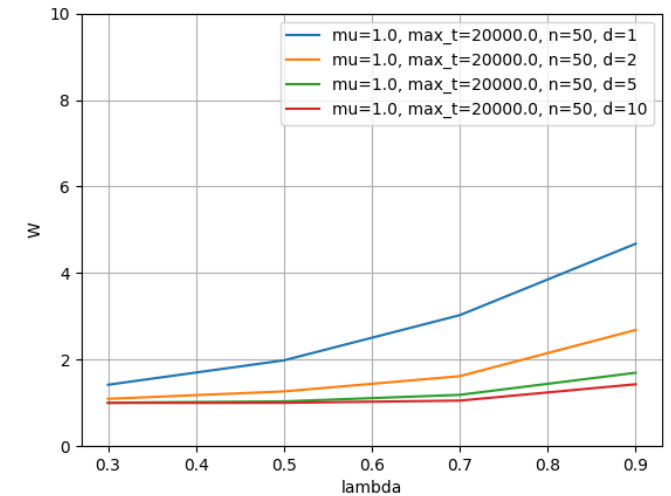
Average Time in System (W) – Effect of Overflow Policies



Infinite Queue (No Overflow Policy)



Finite Queue with Drop Policy



Finite Queue with Retry Policy
(Max Retries = 3)

Parameters: $\lambda = \{0.3, 0.5, 0.7, 0.9\}$, $\mu = 1$, $N = 50$, $MAXT = 20000$, $d = \{1, 2, 5, 10\}$

Average Time in System (W) – Effect of Overflow Policies

Following peer review feedback, we extended the evaluation by analyzing the average time spent in the system (W), including the impact of different overflow policies.

- Increasing d significantly reduces the average time spent in the system (W) in all configurations.
- The drop policy achieves the lowest W by discarding jobs when queues are full, at the cost of increased job loss.
- The retry policy (Max Retries = 3) introduces additional load but provides a good trade-off between latency and reliability, drastically reducing the drop rate with a moderate increase in W .

Effect of the Extension

- Introducing a finite queue capacity makes queues fill up faster, deviating from the idealized theoretical model but reflecting real systems.
- Even with limited queue capacity, the supermarket method has a strong impact: queue lengths are significantly reduced already for $d = 2$.
- Adding a retry mechanism increases the load on the system, but drastically reduces the job drop rate.
- Overall, combining finite queue capacity and retry with the supermarket model results in a realistic system with very good performance under high load.

Use of External Resources

- This assignment was completed individually.
- Large language models (ChatGPT) were extensively used as a support and learning tool, both to understand how the provided.
- simulator code works and to assist in the implementation of the assignment, including writing and modifying parts of the code.
- ChatGPT was also used to help with Python syntax, code structuring, and implementation ideas, due to limited prior experience with Python.
- In addition, ChatGPT was used to support the writing of the report, helping with structure, clarity, and grammatical correctness.
- Gemini was occasionally used to clarify specific theoretical concepts.
- All design decisions, simulator modifications and experimental setups were independently carried out by the author.





Project Repository

The source code of the simulator, execution scripts,
generated output files, and figures used in this report
are available at:

https://github.com/Argon2002/DC_Assignment1.git

Appendix – Implementation Details

Files overview

 d1.json	M
 d2.json	M
 d5.json	M
 d10.json	M
 discrete_event_sim.py	
 plot_queue_w.py	
 queue_sim.py	
 run_dc.ps1	
 sir.py	
 supermarket_plot.py	M
 workloads.py	

A quick look of the functionality of each file:

- discrete_event_sim.py: core discrete-event simulation engine (event queue and simulation time)
- queue_sim.py: implementation of the supermarket queueing model and simulation logic
- workloads.py: workload generators, including Weibull-distributed arrivals and service times
- supermarket_plot.py: comparison between experimental and theoretical queue length distributions
- plot_queue_w.py: visualization of average system time (W) under different configurations
- run_dc.ps1: automation script for running multiple experiments(using PowerShell)

Additionally, files such as d1.json, d2.json, d5.json, etc. are automatically generated while running the script. These files store the experimentally observed queue length distributions and are used to analyze and validate the simulation results.

run_dc.ps1 & execution

- The simulator is executed using a PowerShell script (run_dc.ps1), which requires the simulation parameters to be provided explicitly.
- A typical execution command is:

```
.\run_dc.ps1 -LAMBDA 0.5 -MU 1 -d 1,2,5,8,10 -N 50 -MAXT 20000
```

- The script runs the simulator with the specified parameters, executes multiple experiments with different values of d (1, 2, 5, 8, 10), and automatically generates output files containing the experimental results and after that, the corresponding plots comparing experimental and theoretical results are displayed automatically.

Parameters

The PowerShell script allows configuring the simulator through a set of parameters.

- λ (LAMBDA): job arrival rate
- μ (MU): service rate
- N: number of servers
- MAXT: maximum simulation time
- D: list of values for the number of sampled queues

Optional parameters:

- WEIBULL_SERVICE_SHAPE: shape parameter for service time distribution
- WEIBULL_ARRIVAL_SHAPE: shape parameter for inter-arrival time distribution
- QUEUE_CAPACITY: maximum queue size
- OVERFLOW_BEHAVIOUR_OPTION: drop or retry
- MAX_RETRIES: number of retries when retry is enabled

Thanks for your attention!

Mohammad Hossein Mahrooghi – S7499834



**Università
di Genova**