Report - Distributed Computing 2024/25

Federico Conti - Nicolas Falcone

Assignment 1

Queue Simulator

The simulator utilized in this assignment is based on **Discrete Event Simulation (DES)** principles. It operates as a queue of events, where each event is organized chronologically by its simulated time of occurrence, rather than real-world elapsed time.

The workflow of the simulation is as follows:

1. The simulation retrieves the first event from the queue and updates its status to **running**.
2. This event may trigger additional events, which are then inserted back into the queue, maintaining chronological order.
3. The process continues iteratively until a predefined maximum simulated time is reached, at which point the simulation concludes.

The core components of the simulation are two primary files:

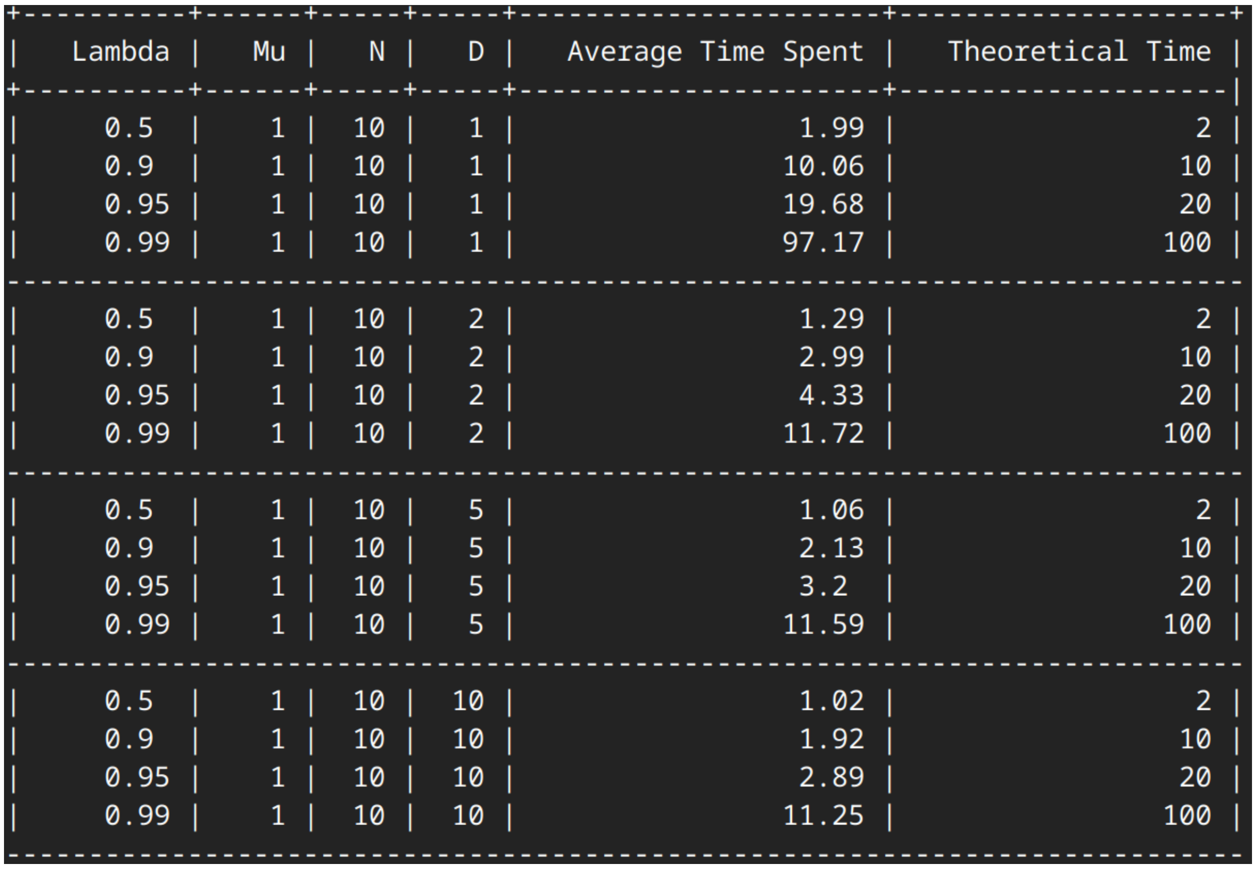
- **discrete_event_sim.py**: A library containing the fundamental functions and structures necessary to support the simulation.
- **queue_sim.py**: The main simulator script, responsible for executing the simulation process.

After having completed them as requested, we ran our first simulation.

The script **queue_sim.py** allows for parameterization to facilitate various experiments. The key parameters are as follows:

- **Lambda (λ)** = The arrival rate of processes into the system.
- **Mu (μ)** = The service rate of each server (queue).
- **Max_t** = The maximum simulated time at which the simulation terminates.
- **N** = The total number of servers in the system.
- **D** = The number of servers from which a queue is selected. The queue chosen is the one with the fewest processes among these D servers.

Using **Max_t** = 1.000.000 and **N** = 10, the results of the first simulation are as follows:



```
+----------+------+-----+-----+---------------------+--------------------+
|   Lambda |   Mu |   N |   D |   Average Time Spent |   Theoretical Time |
+----------+------+-----+-----+---------------------+--------------------|
|     0.5  |    1 |  10 |   1 |                1.99 |                  2 |
|     0.9  |    1 |  10 |   1 |               10.06 |                 10 |
|     0.95 |    1 |  10 |   1 |               19.68 |                 20 |
|     0.99 |    1 |  10 |   1 |               97.17 |                100 |
--------------------------------------------------------------------------
|     0.5  |    1 |  10 |   2 |                1.29 |                  2 |
|     0.9  |    1 |  10 |   2 |                2.99 |                 10 |
|     0.95 |    1 |  10 |   2 |                4.33 |                 20 |
|     0.99 |    1 |  10 |   2 |               11.72 |                100 |
--------------------------------------------------------------------------
|     0.5  |    1 |  10 |   5 |                1.06 |                  2 |
|     0.9  |    1 |  10 |   5 |                2.13 |                 10 |
|     0.95 |    1 |  10 |   5 |                3.2  |                 20 |
|     0.99 |    1 |  10 |   5 |               11.59 |                100 |
--------------------------------------------------------------------------
|     0.5  |    1 |  10 |  10 |                1.02 |                  2 |
|     0.9  |    1 |  10 |  10 |                1.92 |                 10 |
|     0.95 |    1 |  10 |  10 |                2.89 |                 20 |
|     0.99 |    1 |  10 |  10 |               11.25 |                100 |
--------------------------------------------------------------------------
```

Theoretical Time Reference (D = 1): The theoretical time corresponds to the edge case where **D=1**, representing the minimum number of server choices. As observed, the empirical average time spent closely aligns with the theoretical predictions in this scenario, validating the accuracy of the simulation.

$$W=\frac{L}{\lambda}=\frac{(\frac{\lambda}{1-\lambda})}{\lambda}=\frac{1}{1-\lambda}$$

Impact of Increasing D: As expected, increasing **D** (the number of server choices) leads to a reduction in the average time spent in the system. This improvement is due to the more balanced load distribution enabled by selecting queues with fewer processes.

Optimization Trade-offs: However, the results also reveal that the difference in Average Time Spent between **D = 5** and **D = 10** (the other edge case, where **D=N**, the total number of servers) is minimal. Given the potential overhead associated with further increasing **D**, it is reasonable to consider **D = 5** as the optimal choice among **D** = **1,2,5** and **10**, balancing performance gains and computational efficiency.

To visualize the simulation data, we implemented a **monitoring process** that periodically captures snapshots of the number of processes in each of the **N** queues at regular intervals.

To schedule the monitoring process at regular intervals in the queue of events, we derived and implemented the following formula:

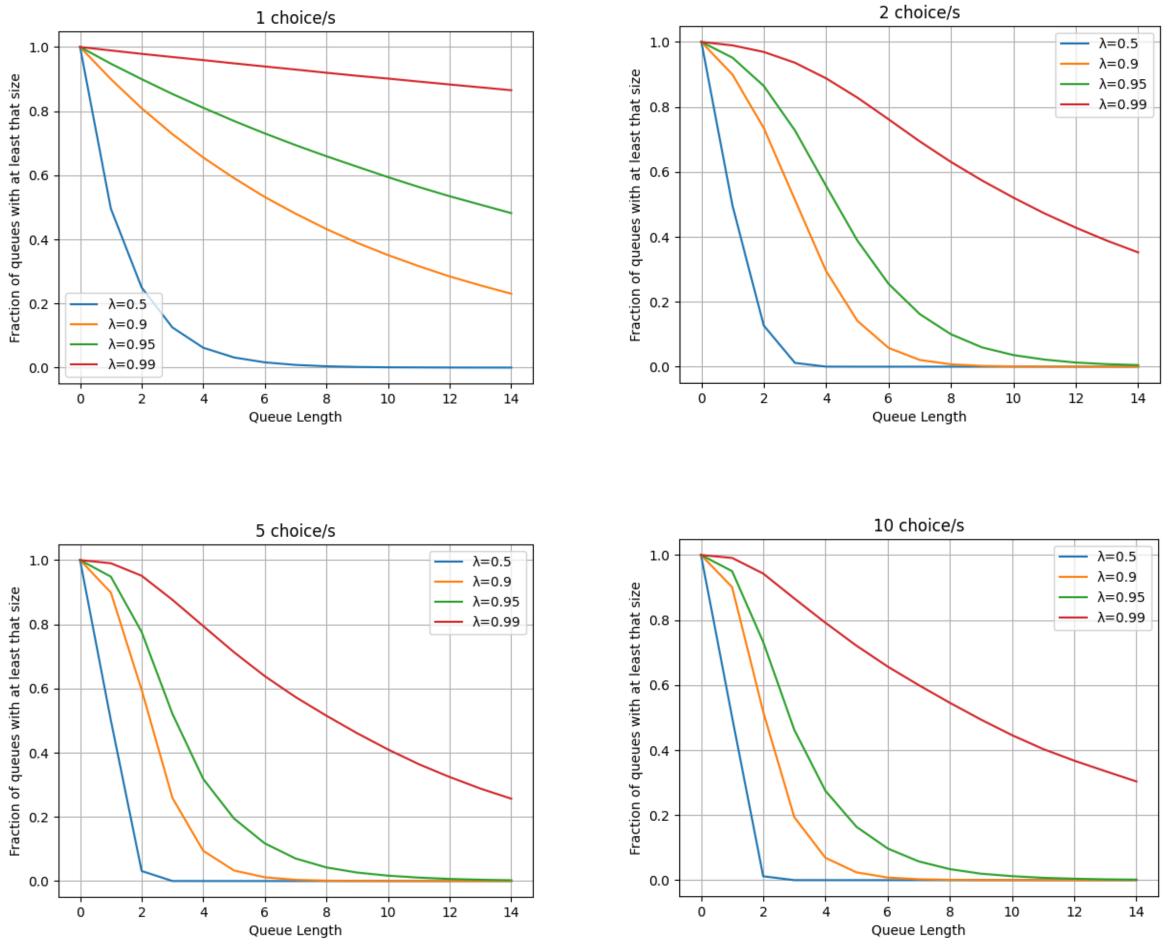
```
monitor_delay = (args.max_t*0.001)/(args.n*args.lambd)
```

This formula dynamically adjusts the frequency of the monitoring process based on key simulation parameters:

- It ensures regular snapshots across varying system sizes (**N**) and load conditions (**λ**).
- It avoids overloading the simulation with frequent monitoring for high **λ**, while maintaining enough granularity to observe system behavior under low **λ**.
- By normalizing using **args.max_t**, the monitoring frequency scales naturally with the duration of the simulation.

(Refer to the implementation details in the source code for specifics.)

After successfully integrating the monitoring process, we generated plots to illustrate the results of the initial simulation.



The resulting plots validated our earlier observations, confirming that the average time spent in queues decreases as **D** increases, with diminishing returns beyond **D=5**.

To further enhance our analysis, we compared these results to those obtained from a simulation using the **Weibull distribution**. This modification allowed us to introduce greater variability in the job arrival and service times, offering a more real-based view of the queuing system.

Weibull Distribution in Queueing Context

In the original implementation of the simulator, the **`expovariate`** function was used to generate random numbers based on the exponential distribution, which assumes a constant arrival rate (**λ**) and service rate (**μ**). This approach is used in memoryless services, so that the probability of an event occurring is independent of past events.

With the modification, the Weibull distribution replaced the exponential distribution. The **Weibull distribution** is defined by two parameters:

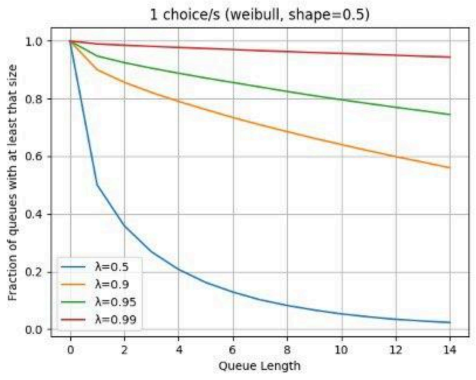
1. **Shape parameter (k)**: Determines the nature of the distribution. When k=1, the Weibull distribution is equivalent to the exponential distribution. For k<1, the probability of arrivals decreases over time, and for k>1, the probability increases.
2. **Scale parameter (λ)**: Adjusts the spread of the distribution, controlling the average inter-arrival or service time.

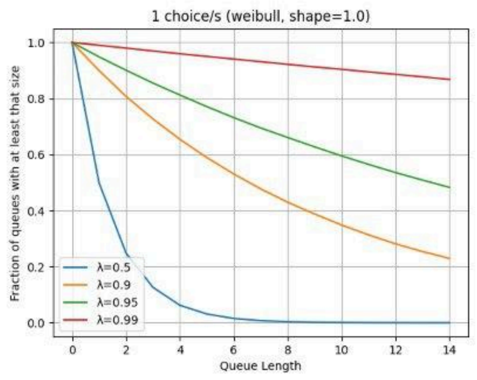
By adopting the Weibull distribution, the simulator achieves a more realistic representation of real-world queuing systems, where both arrival and service rates are rarely uniform.

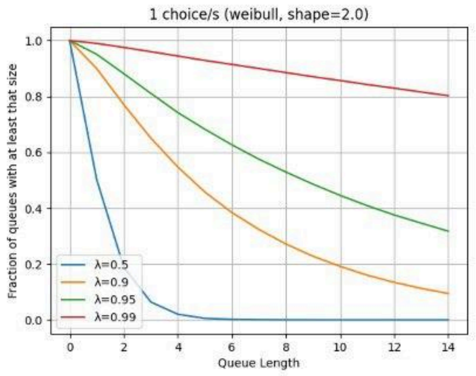
```
def weibull_generator(shape, mean):
    """Returns a callable that outputs random variables with a Weibull
distribution having the given shape and mean."""

    return functools.partial(random.weibullvariate, mean / math.gamma(1 + 1
/ shape), shape)
```

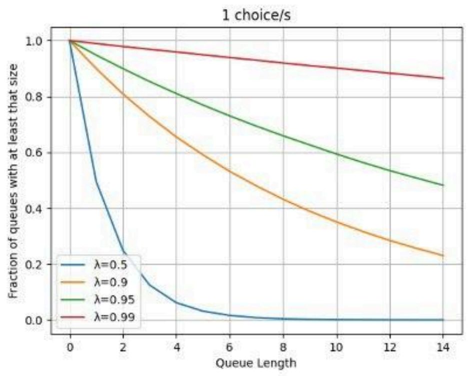
With **shape=1.0**, the gamma function simplifies to **Γ(2)=1**, making the scale parameter equal to the specified mean. Consequently, the Weibull distribution reduces to the exponential distribution, ensuring that the mean remains unchanged and the memoryless property is preserved. This behavior validates the equivalence of the Weibull generator with **shape=1.0** to the expovariate function.

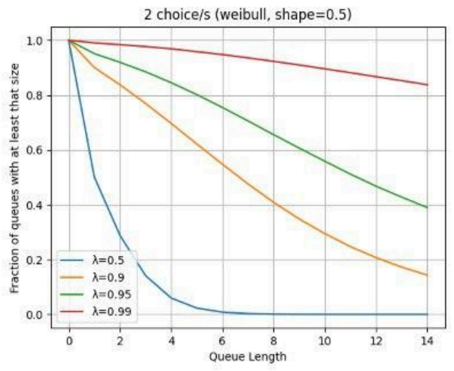
Here all the plots are compared, assuming that the Weibull Distribution with shape 1.0 must be equal to the memoryless one.

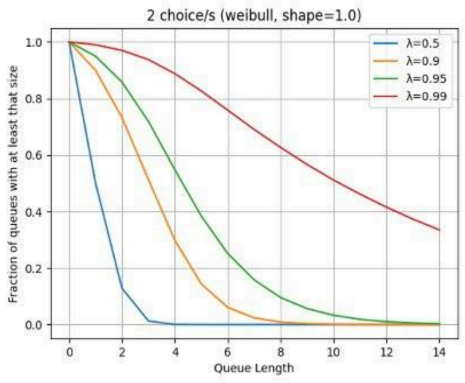

1 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

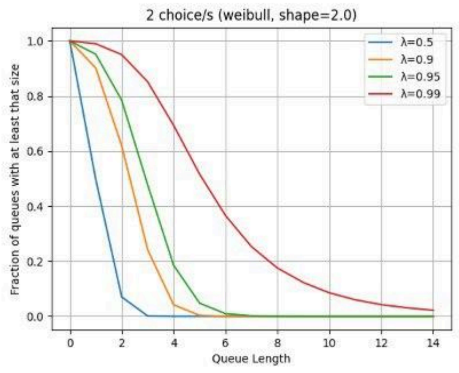

1 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

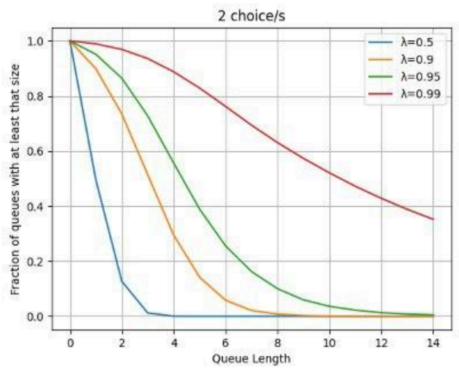

1 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

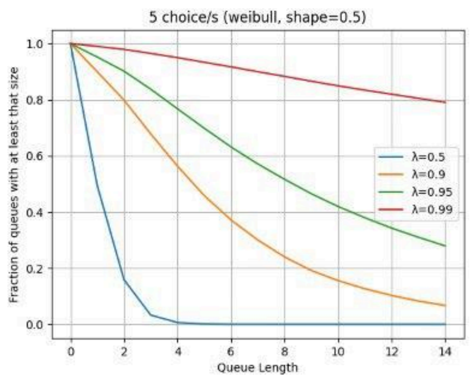
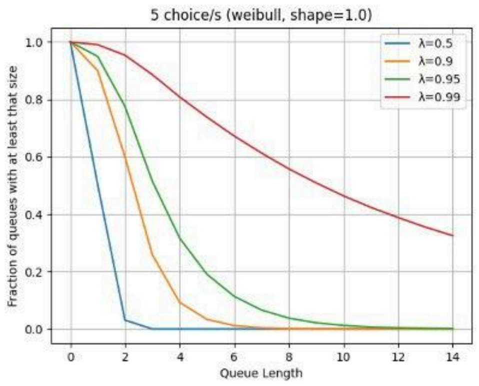
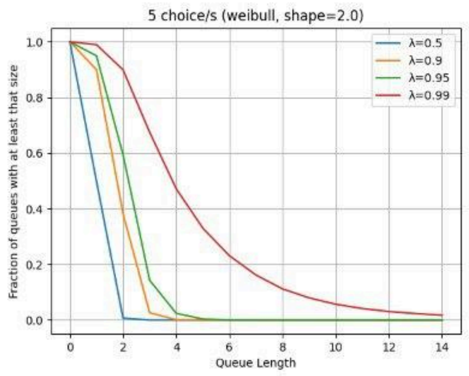
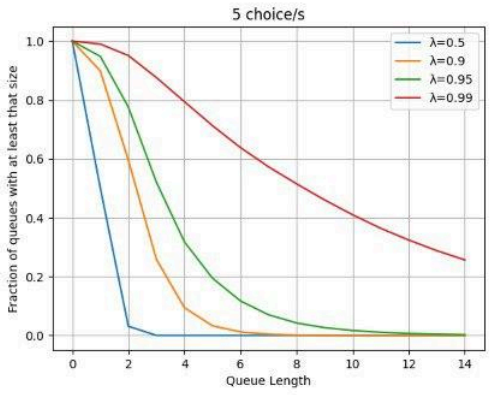
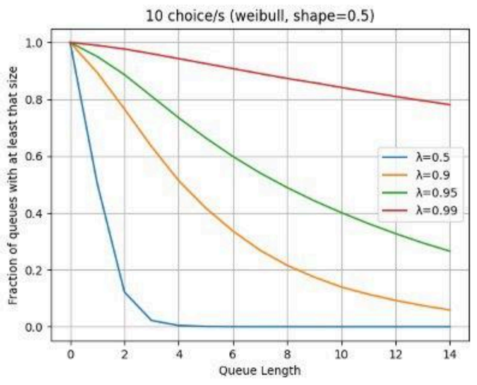
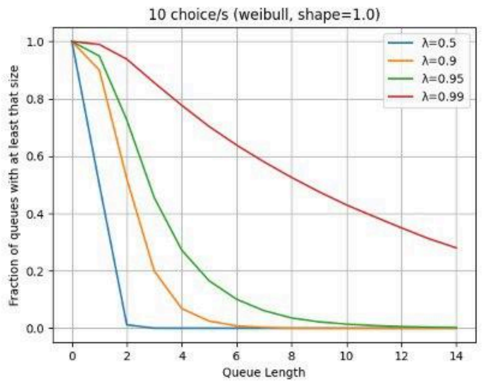
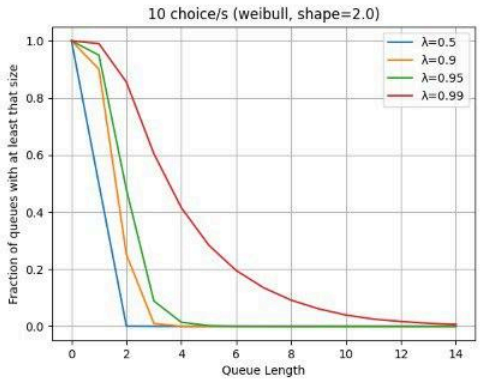
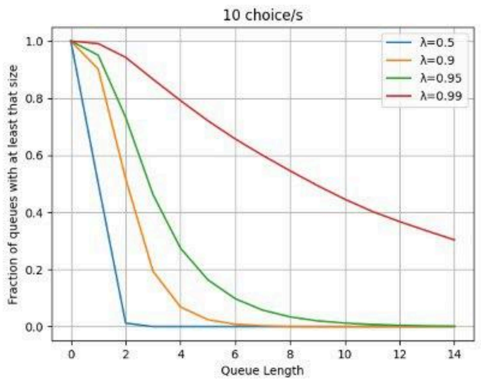

1 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


2 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


2 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


2 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


2 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


5 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

5 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

5 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

5 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

10 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

10 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

10 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

10 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

The resulting plots highlighted the impact of the **shape parameter (k)** of the Weibull distribution on the performance of the queuing system. The findings demonstrated distinct performance characteristics for different values of k:

1. Shape k=1.0:

   When the shape parameter is **k=1.0**, the Weibull distribution simplifies to the exponential distribution, as assumed in theory. Consequently, the plots showed that the queue lengths remained consistent with the theoretical predictions, confirming the validity of the simulator under this configuration.

2. Shape k=0.5:

   With **k=0.5**, the Weibull distribution exhibits a decreasing failure rate, meaning that smaller inter-arrival times are more likely. This results in **bursty arrivals** (a higher probability of multiple jobs arriving within a short time frame). The resulting plots indicated significantly worse performance under these conditions, as the queues experience uneven loads and increased congestion.

   The bursty nature of arrivals leads to temporary overloads, where the queues cannot clear tasks quickly enough, causing queue lengths to grow and response times to degrade. This is particularly problematic in scenarios with a **limited number of servers (N = 10)**.

3. Shape k=2.0:

   When **k=2.0**, the Weibull distribution exhibits an **increasing failure rate**, meaning longer inter-arrival times are more likely, and arrivals are more evenly spread out. The plots showed that this configuration resulted in the best performance overall, as the smoother arrival patterns allowed the queues to operate more efficiently without experiencing overloads.

   An increasing failure rate ensures that jobs arrive more predictably, preventing queue congestion and maintaining steady processing rates. This balanced load distribution improves system throughput and reduces delays.

Impact of D on Performance with Weibull k=2.0

The Weibull distribution with **k=2.0** provided the most significant performance improvements when **D≥2**. For **D=1**, the improvement over the exponential distribution was minimal. However, with **D≥2**, the combination of increasing server choices and smoother arrival patterns led to a marked reduction in average queue lengths and response times.

Conclusion

The results clearly indicate that the Weibull distribution with a shape parameter of **k=2.0** is the optimal configuration for the simulated queuing system with a limited number of servers. This highlights the importance of choosing appropriate distribution parameters and server-selection mechanisms to optimize system performance.

Simulator SRPT (Extension)

The decision to extend the simulator with the **Shortest Remaining Processing Time (SRPT)** algorithm was driven by its potential to significantly enhance performance in queueing systems. **SRPT** works by prioritizing jobs with the shortest remaining processing time, which helps minimize delays for smaller tasks and reduces overall response times.

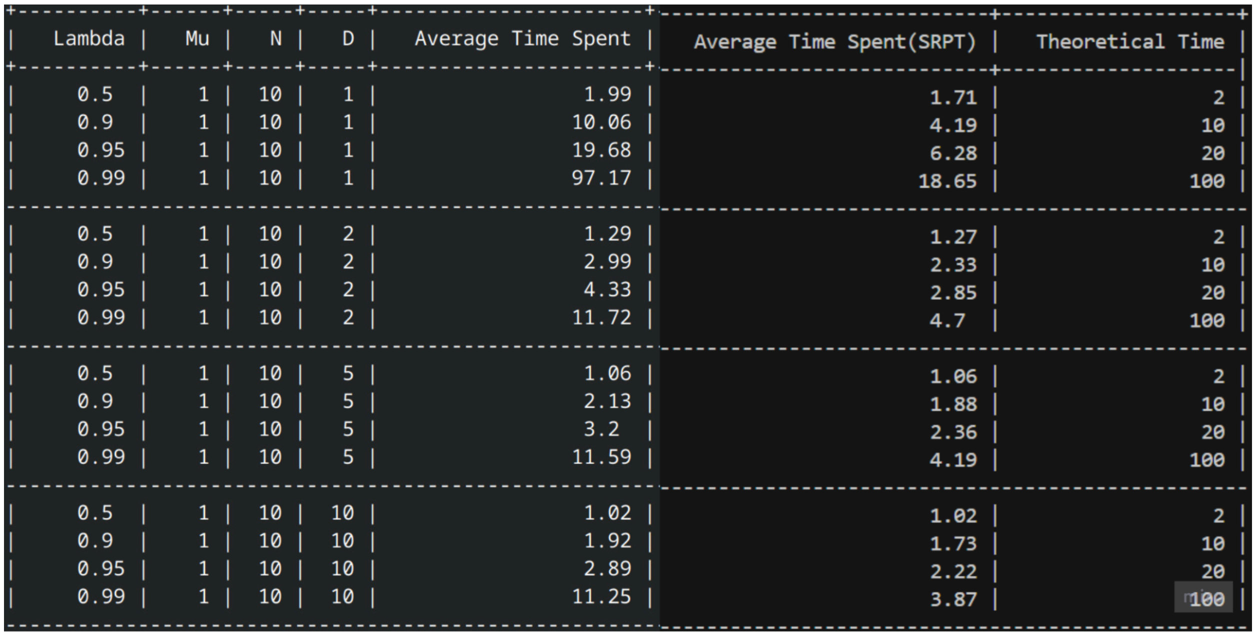
This extension is especially relevant given the simulation setup, which assumes **N=10**, a limited number of servers. **SRPT** enhances resource efficiency by ensuring that shorter tasks are not delayed by longer ones.This leads to shorter queues and faster processing times overall.

By including **SRPT**, we aim to gain deeper insights into how advanced scheduling strategies can improve system performance, particularly under realistic constraints.

To implement the Shortest Remaining Processing Time (SRPT) algorithm, we introduced several modifications to the simulator to improve its efficiency and functionality. The first change involved creating a **Job** class with a `remaining_time` attribute. This adjustment was necessary because, in the original simulator, jobs were only characterized by their total time spent in the system (including delay and processing time). By adding `remaining_time`, the simulator can directly manage job priorities based on how much work remains for each task.

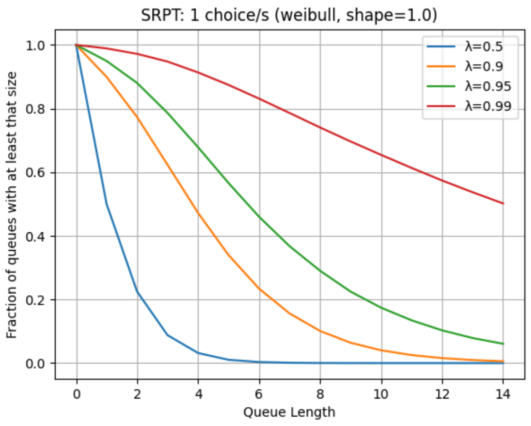
To further optimize the system, we replaced the traditional queue structure used to simulate servers with a **heap data structure**. This choice allows us to leverage the `remaining_time` attribute to efficiently prioritize jobs. The heap ensures that the job with the shortest remaining time is always on top, simplifying the process of selecting and removing the next job to run on the server. Each time a job is completed, the simulator pops the next shortest job from the heap, maintaining the **SRPT** logic.

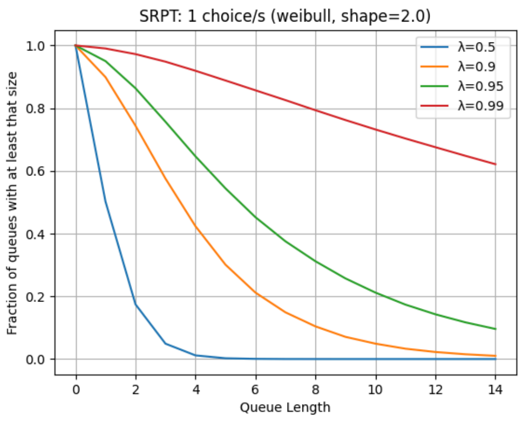
Using **Max_t** = 1.000.000 and **N** = 10, the results of the extension (SRPT), compared to the first simulation, are as follows:

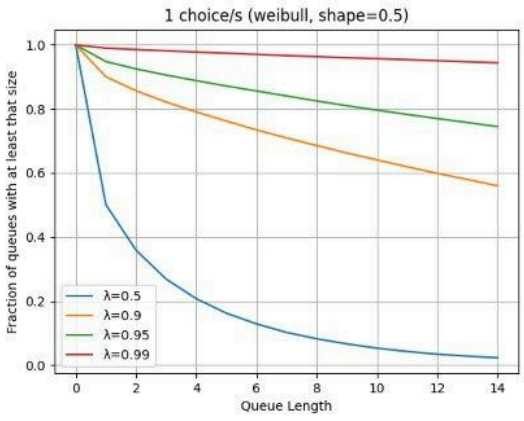


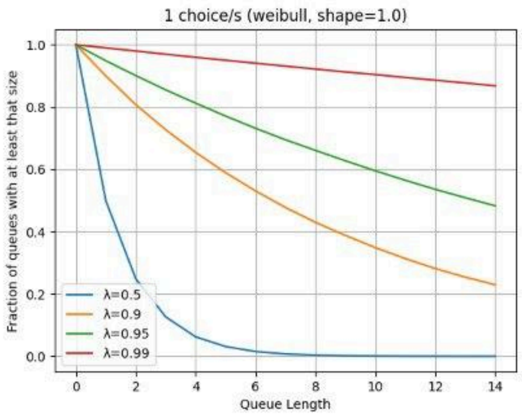
```
+----------+------+-----+-----+---------------------+-------------------------+-------------------+
|   Lambda |   Mu |   N |   D |  Average Time Spent |  Average Time Spent(SRPT) |  Theoretical Time |
+----------+------+-----+-----+---------------------+-------------------------+-------------------|
|     0.5  |    1 |  10 |   1 |                1.99 |                    1.71 |                 2 |
|     0.9  |    1 |  10 |   1 |               10.06 |                    4.19 |                10 |
|     0.95 |    1 |  10 |   1 |               19.68 |                    6.28 |                20 |
|     0.99 |    1 |  10 |   1 |               97.17 |                   18.65 |               100 |
---------------------------------------------------------------------------------------------------
|     0.5  |    1 |  10 |   2 |                1.29 |                    1.27 |                 2 |
|     0.9  |    1 |  10 |   2 |                2.99 |                    2.33 |                10 |
|     0.95 |    1 |  10 |   2 |                4.33 |                    2.85 |                20 |
|     0.99 |    1 |  10 |   2 |               11.72 |                    4.7  |               100 |
---------------------------------------------------------------------------------------------------
|     0.5  |    1 |  10 |   5 |                1.06 |                    1.06 |                 2 |
|     0.9  |    1 |  10 |   5 |                2.13 |                    1.88 |                10 |
|     0.95 |    1 |  10 |   5 |                3.2  |                    2.36 |                20 |
|     0.99 |    1 |  10 |   5 |               11.59 |                    4.19 |               100 |
---------------------------------------------------------------------------------------------------
|     0.5  |    1 |  10 |  10 |                1.02 |                    1.02 |                 2 |
|     0.9  |    1 |  10 |  10 |                1.92 |                    1.73 |                10 |
|     0.95 |    1 |  10 |  10 |                2.89 |                    2.22 |                20 |
|     0.99 |    1 |  10 |  10 |               11.25 |                    3.87 |               100 |
---------------------------------------------------------------------------------------------------
```

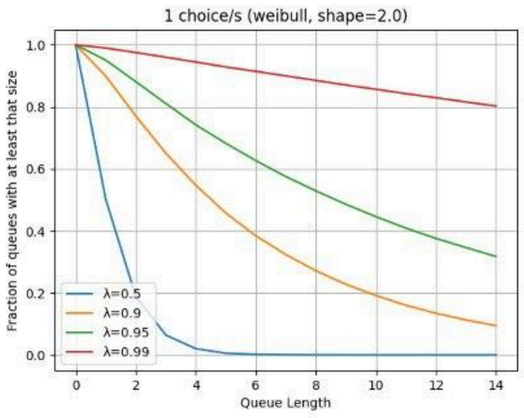
The following sections present a comparison of all generated plots (Default, Weibull, SRPT), organized and categorized based on the number of server choices (**D**).

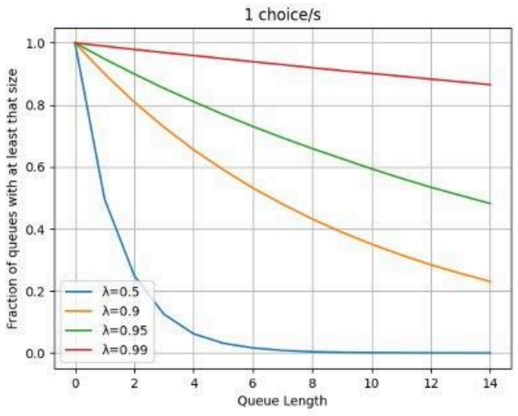

SRPT: 1 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

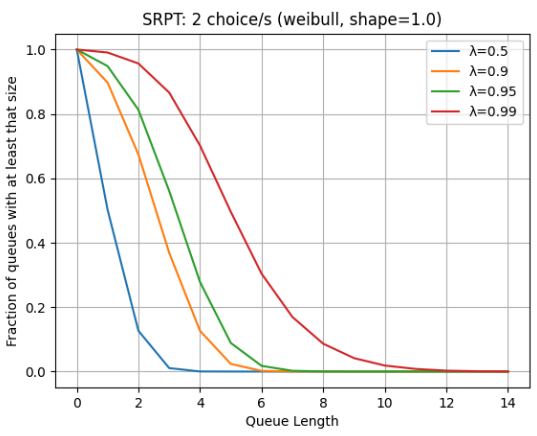

SRPT: 1 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

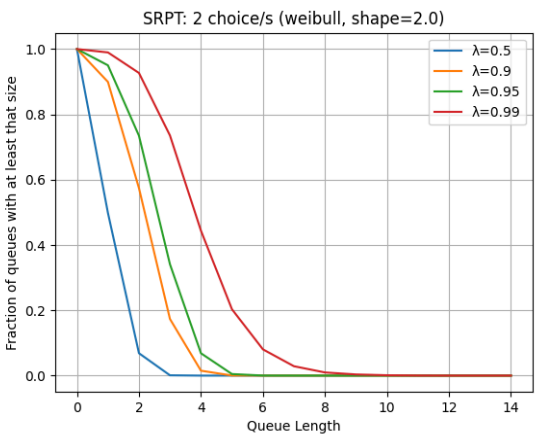

1 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

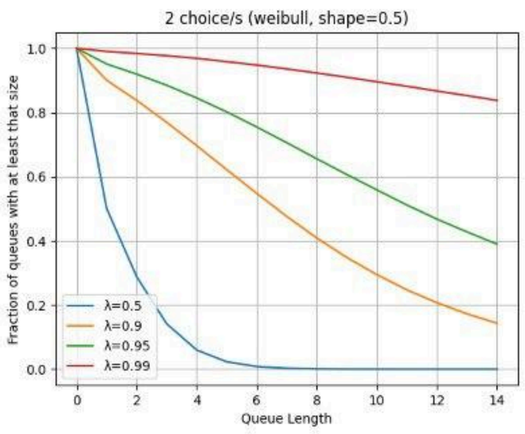

1 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

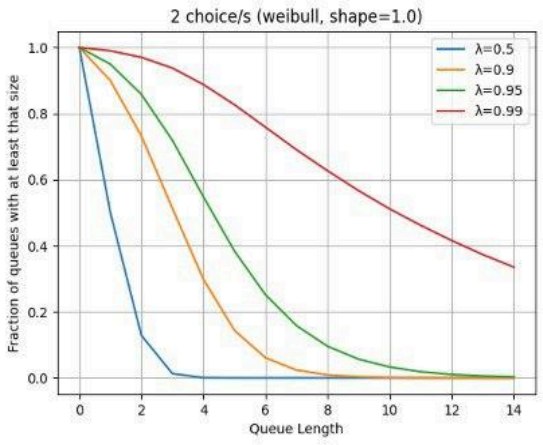

1 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

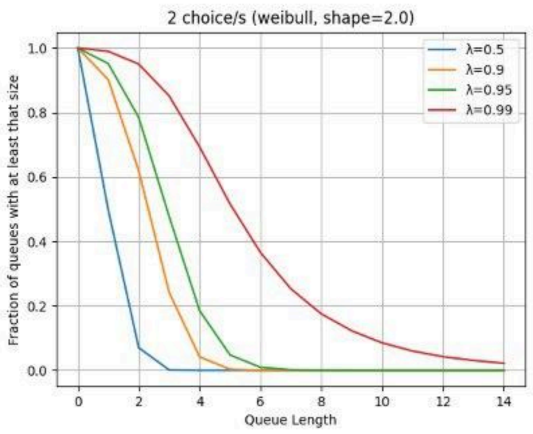

1 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

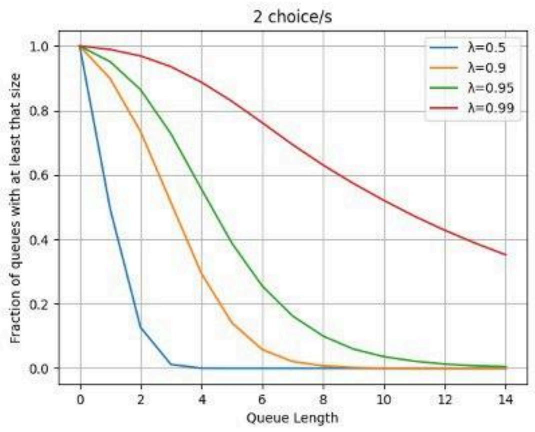

SRPT: 2 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

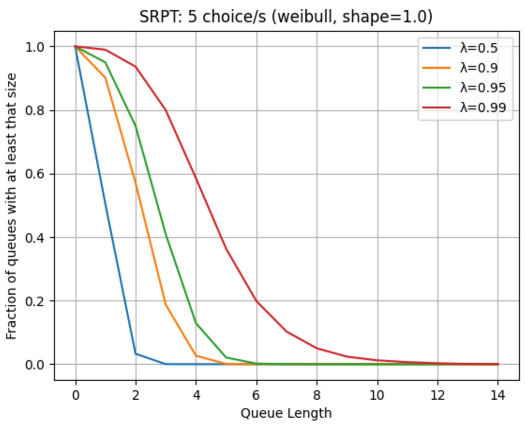

SRPT: 2 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

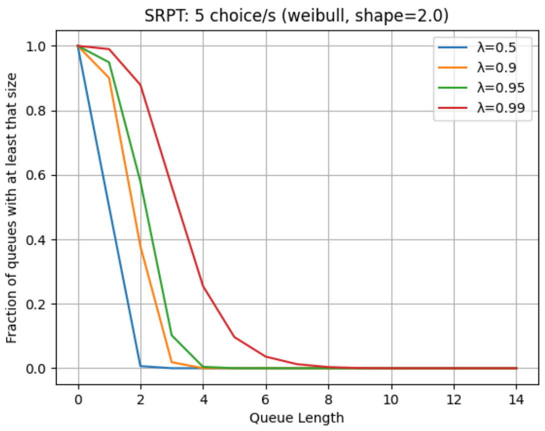

2 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

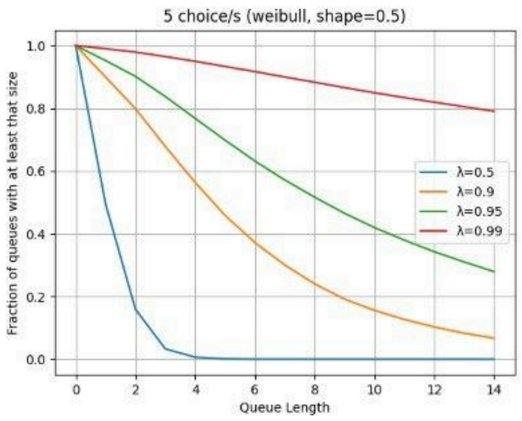

2 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

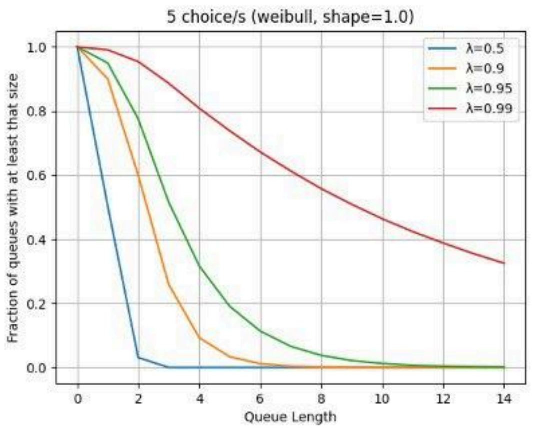

2 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

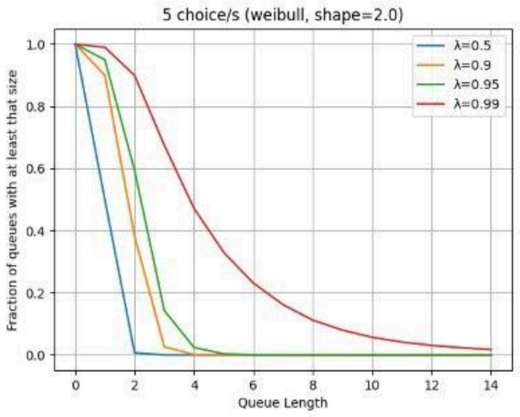

2 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

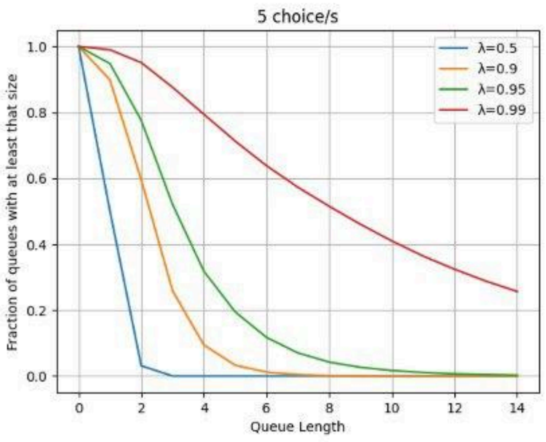

SRPT: 5 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

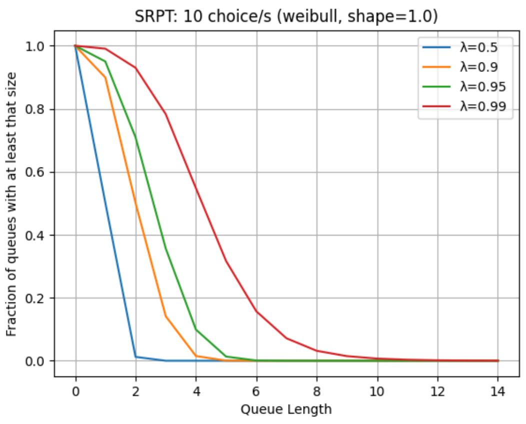

SRPT: 5 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

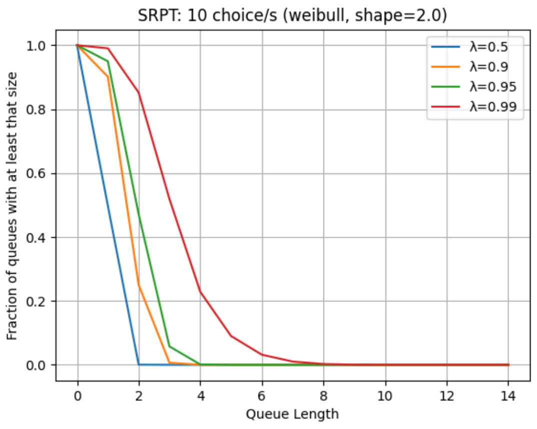

5 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

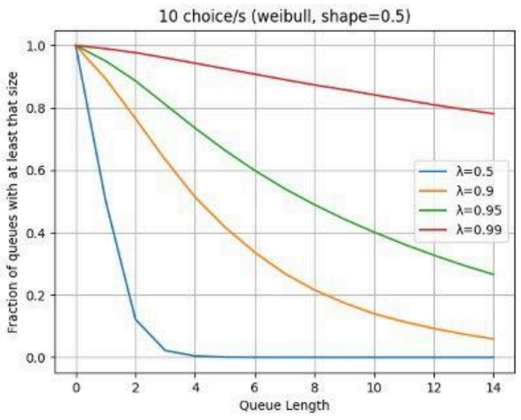

5 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

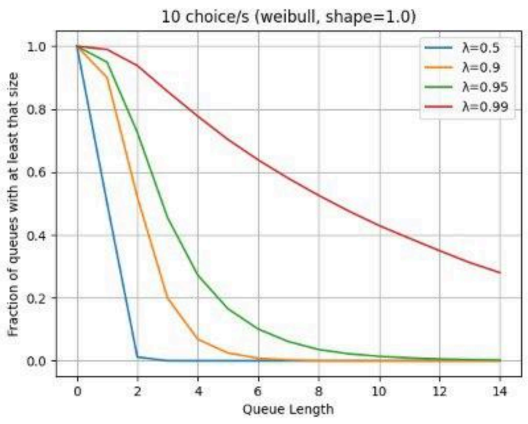

5 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

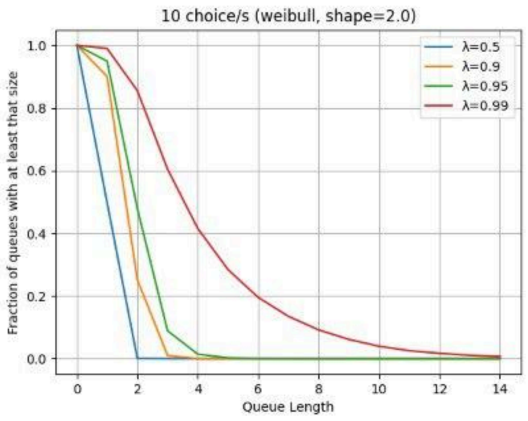

5 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

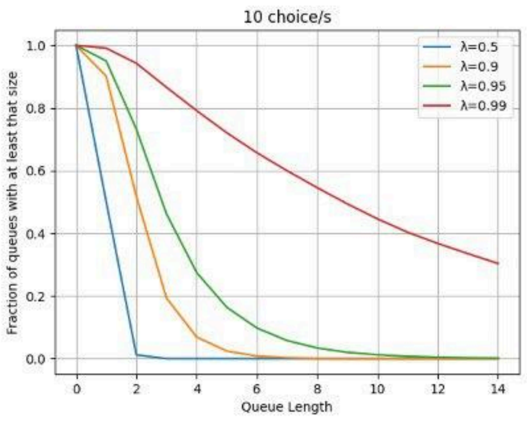

SRPT: 10 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


SRPT: 10 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


10 choice/s (weibull, shape=0.5)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


10 choice/s (weibull, shape=1.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


10 choice/s (weibull, shape=2.0)
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99


10 choice/s
Fraction of queues with at least that size
Queue Length
λ=0.5
λ=0.9
λ=0.95
λ=0.99

Comments and Conclusion

The analysis of the **Shortest Remaining Processing Time (SRPT)** algorithm shows significant performance improvements compared to both the memoryless queue (exponential distribution) and the Weibull distribution.

SRPT prioritizes jobs with the shortest remaining processing time, leading to reduced queue lengths and faster task completion.

We tested **SRPT** using two Weibull shapes (**k=1.0** and **k=2.0**) and four levels of choice sampling (**D=1,2,5,10**). The shape **k=1.0** was chosen as it mirrors the memoryless property of exponential distributions, while **k=2.0** was selected based on its prior strong performance in our simulation, where it reduced the overall job completion time.

The results demonstrate that SRPT performs better across all scenarios. The improvements are particularly pronounced for **D≥2**, where tasks are more effectively distributed among shorter queues, allowing smaller jobs to be prioritized and completed quickly. With **D=1**(random queue selection), the gains are less significant but still noticeable.

As the simulation progresses toward its maximum time, the advantages of **SRPT** become increasingly evident, as smaller tasks are consistently cleared from the system, reducing congestion and average queue lengths.

When comparing Weibull shapes, **k=2.0** exhibits stronger performance despite having a larger mean job size than **k=1.0**.

This is due to its lower variability in job size, which complements **SRPT**'s scheduling mechanism by reducing extreme outliers that could otherwise cause delays.

These findings highlight SRPT's robustness as a scheduling strategy, particularly in systems with limited resources (**N=10**).

Assignment 2

P2P simulator

The simulator utilized in this assignment is still based on **Discrete Event Simulation (DES)** principles. Hence, the **discrete_event_sim.py** is reused.

The system is designed to model a distributed backup network, focusing on how nodes interact, share data, and recover from failures. It incorporates mechanisms to manage redundancy and recover lost information, simulating the dynamic nature of a peer-to-peer environment where nodes frequently connect, disconnect, or fail entirely.

The workflow of the simulation is as follows:

- **Initialization Phase**:
  - All nodes are set up with their initial states.
  - Each node is scheduled to come online at a specific time.
  - At the same time, each node is scheduled to fail after a randomly determined period based on its expected lifetime.
- **Normal Operation Loop**:
  - When a node comes online:
    - It begins its routine operations, including uploading and downloading data.
    - A disconnection is scheduled after a randomly determined period based on how long the node typically stays online.
  - When a node goes offline:
    - It stops all ongoing data transfers and disconnects.
    - It is scheduled to come back online after a randomly determined downtime.
- **Failure Phase**:
  - When a node fails, it permanently stops functioning until it is recovered.
  - All of the node's local data is lost, and any backups of its data on other nodes are invalidated.
  - The node is scheduled to recover after a randomly determined period based on its recovery time.

- **Recovery Phase**:
    - When a node recovers, it returns to normal operation and begins its routine tasks again.
    - The next failure is scheduled after a new randomly determined lifetime.

Equal Peer Configuration

In this configuration, our script supports parameterization to enable various experiments. The key parameters are:

- **N** = The total number of peers (nodes) in the system.
- **K** = The redundancy factor or the number of blocks each peer should store for the data to be considered reliably backed up.
- **AL** = The data availability duration, representing the length of time for which data needs to be available in the system.
- **n** = The number of local data blocks each peer stores.
- **k** = The number of backup copies required for redundancy.
- **data_size** = The size of the data stored by each peer.
- **storage_size** = The total storage capacity of each peer.
- **upload_speed** = The maximum upload speed at which data can be transferred from the peer.
- **download_speed** = The maximum download speed at which data can be received by the peer.
- **average_uptime** = The average time that a peer remains online and operational before potentially going offline.
- **average_downtime** = The average time a peer is offline due to failures or other reasons before it comes back online.
- **average_recover_time** = The average time a peer needs to recover from a failure and come back online.
- **average_lifetime** = The expected lifetime of a peer in the system.
- **arrival_time** = The time when a peer first enters the system.

Our script `plot_p2p.py` is useful to visualize the data availability in the peer-to-peer storage system over time.

It plots two key lines:

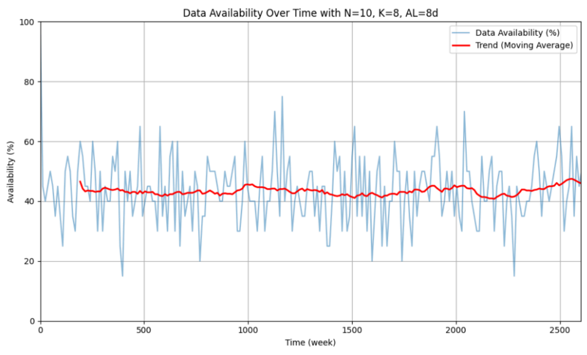
1. **Data Availability Line (%)**: This line represents the percentage of data availability over time. It directly shows how the availability changes as the simulation progresses. This is important for understanding how well the system maintains data across peers, highlighting periods of stability or potential issues.
2. **Trend Line (Moving Average)**: The trend line is a smoothed version of the data availability, calculated using a moving average. It helps to visualize the overall trend in the data availability, filtering out short-term fluctuations. This is useful for identifying long-term patterns in the system's performance, such as improving or declining availability over time.

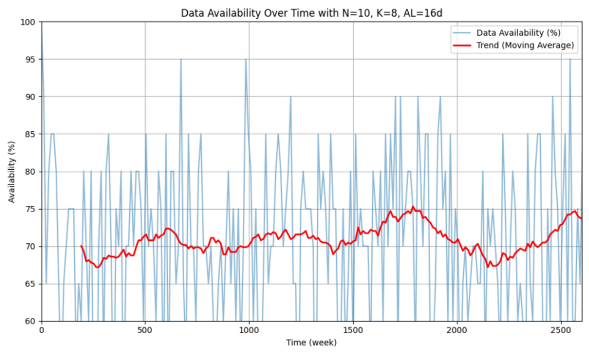
The data in the generated CSV files are derived from our **monitoring function**, which tracks and records the availability of the system over time. This function periodically checks the state of each node, updating whether they are online or offline, and calculates the overall data availability based on the redundancy parameters (N and K).

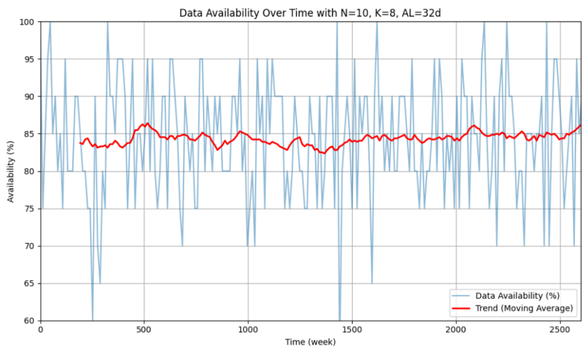
Specifically, it monitors the number of available nodes required for data recovery and ensures that sufficient data blocks are accessible for restoration. The availability percentage is then computed and logged at each time step (**12 weeks**), creating the dataset used for analysis and visualization in the CSV file.

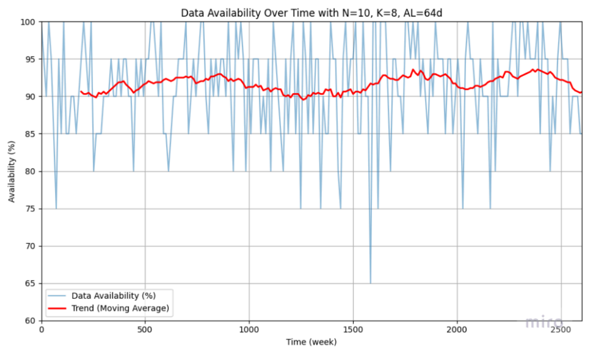
Comparison of Simulations

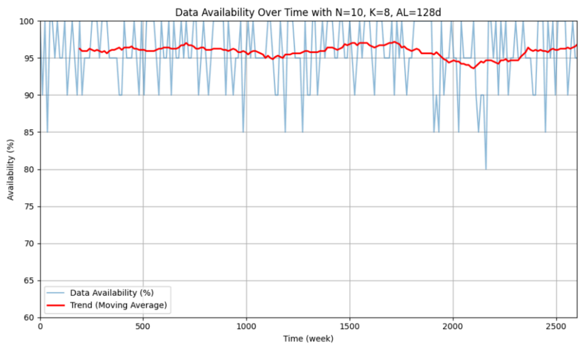
We conducted two types of analyses, keeping **K** fixed (K=8) and varying the **AL** parameter over a range of durations:, **AL**={8,16,32,64,128,256,364,512} days

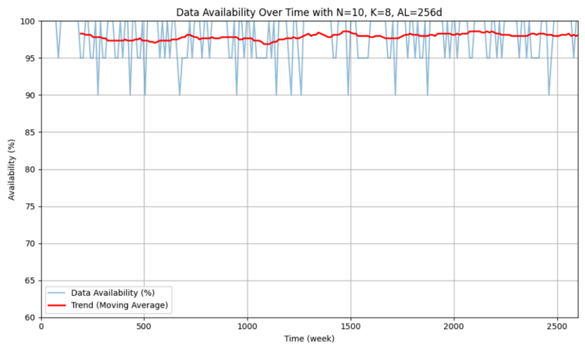


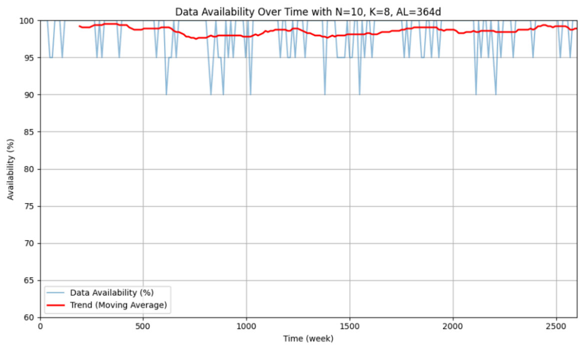


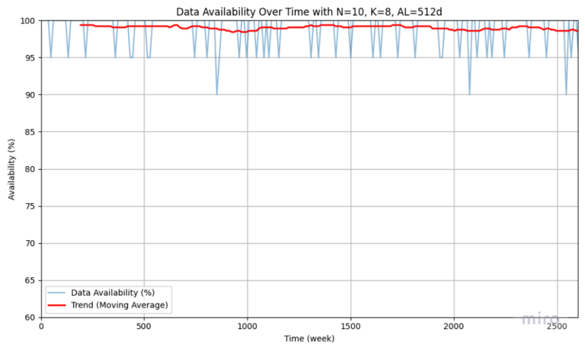












Short-Term Availability (AL=8 to 64 days):

- These configurations show significant variability in data availability, particularly in shorter durations (e.g., AL=8 and 32 days).
- The trend lines in these graphs stabilize at lower availability percentages (40%-90%), highlighting the challenges in maintaining consistent data access with limited redundancy and shorter availability durations.

Long-Term Availability (AL=128 to 512 days):

- These graphs demonstrate much greater stability, with availability consistently exceeding 90% and nearing 99% for higher AL values.
- The trend lines are smoother, indicating fewer disruptions in the system's overall performance.

Comments

Longer availability durations (AL) significantly reduce fluctuations in data availability. Thanks to increased redundancy and longer recovery windows, it ensures that nodes have sufficient time to restore or replicate data before permanent failures impact the system.

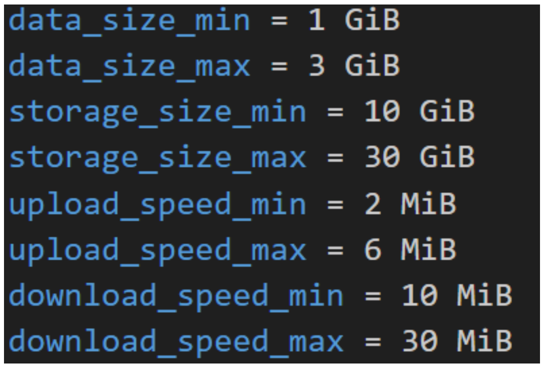
Diverse Peer Configuration (Extension)

In distributed systems, simulating a peer-to-peer (P2P) backup network helps analyze how nodes interact, share data, and recover from failures. This is particularly crucial in environments where nodes exhibit variability in their behavior, such as differences in storage capacity, network speeds, and uptime/downtime cycles.

To make the simulation more realistic, we introduced a randomized configuration model for node parameters, simulating a more diverse and dynamic environment. The revised simulation incorporates random sampling for data size, storage size, upload and download speeds.

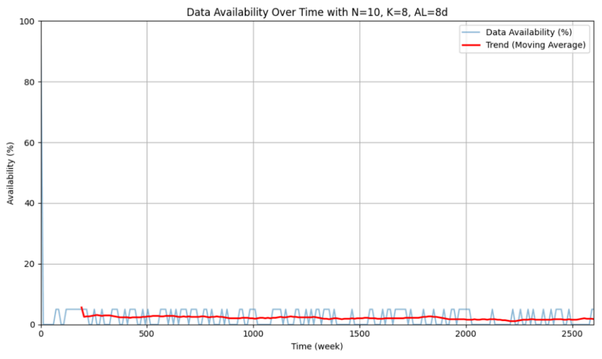
The primary goal of this enhancement is to assess how randomized configurations affect system stability, data availability, and the efficiency of backup and recovery mechanisms.

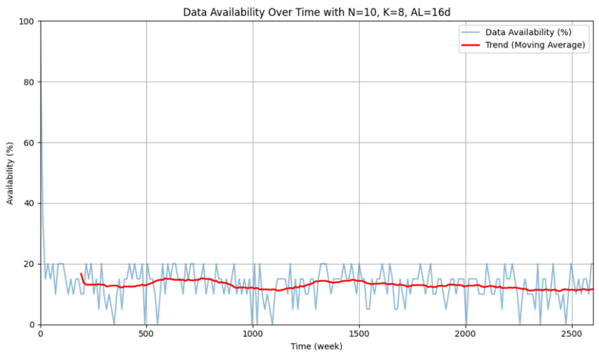
Here is an example of the configuration file that the script accepts as input for randomized peer configurations.
These values are randomly selected from ranges defined by the minimum and maximum limits specified in the configuration file.

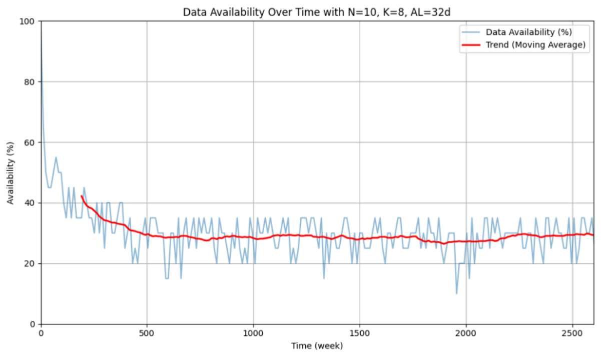

```
data_size_min = 1 GiB
data_size_max = 3 GiB
storage_size_min = 10 GiB
storage_size_max = 30 GiB
upload_speed_min = 2 MiB
upload_speed_max = 6 MiB
download_speed_min = 10 MiB
download_speed_max = 30 MiB
```

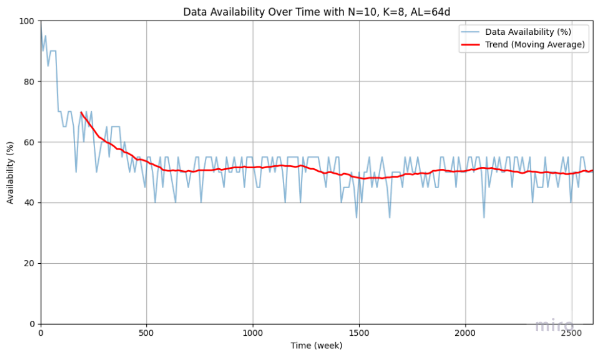
Comparison of simulations

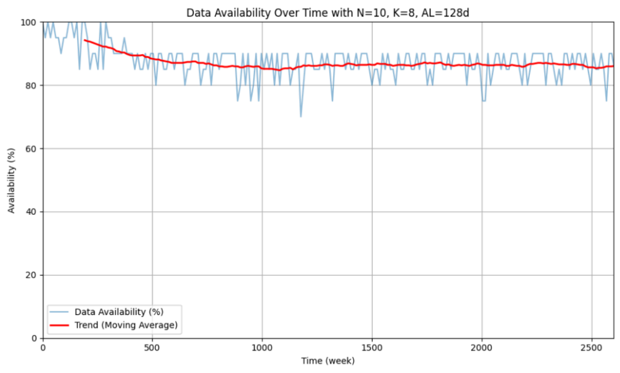
As we have done before, we keep **K** fixed (K=8), varying the **AL** parameter over a range of durations:, **AL**={8,16,32,64,128,256,364,512} days

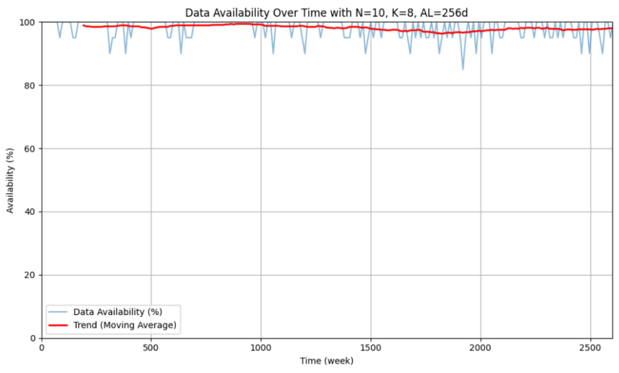


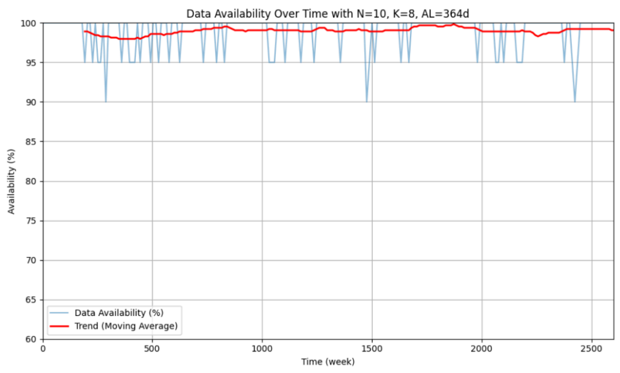


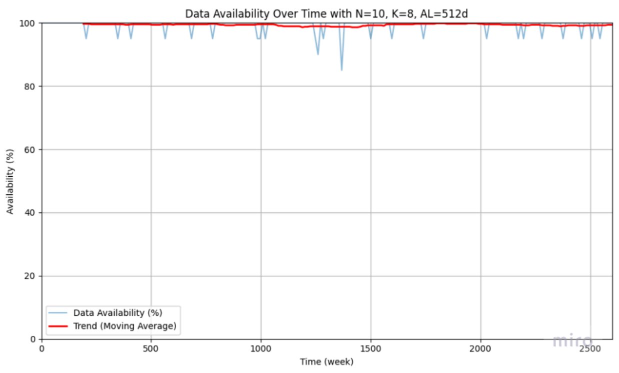












Short-Term Availability (AL=8 to 64 days):

- Data availability in this range is highly volatile, with frequent dips to near-zero availability, particularly at shorter AL values (e.g., AL=8 days).
- Trend lines converge to low percentages (10%-50%), indicating insufficient recovery time to counteract node failures effectively.
- The graphs for AL=32 and AL=64 days show a gradual stabilization compared to the highly unstable graphs for AL=8 and AL=16 days. However, even at AL=64 days, availability remains below 60%, signaling a lack of redundancy and recovery balance.

Long-Term Availability (AL=128 to 512 days):

- For AL≥128 days, the system demonstrates much greater stability, with availability consistently exceeding 80%. At AL=256 days, availability surpasses 90%, and at AL=512 days, it achieves near-perfect availability (99%).
- Trend lines in these graphs are smoother, reflecting the effectiveness of redundancy mechanisms and prolonged recovery windows.
- The transition from AL=128 to AL=256 days shows significant improvement in availability. However, the incremental gains decrease between AL=256, AL=364, and AL=512 days, as the system reaches its redundancy limit.

Conclusions

Equal Configuration

- Homogeneity and Predictability: All peers share identical configurations. This results in consistent system behavior but limits the ability to model real-world variability.
- System Behavior: Availability trends are smoother, and fluctuations are less pronounced since every peer contributes equally.
- Performance Limitations: The lack of diversity in peer capabilities constrains the system's flexibility to adapt to varying workloads or failure scenarios.

Diverse Configuration

- Heterogeneity and Realism: Randomized configurations introduce variability in peer metrics, closely mimicking real-world distributed systems where nodes differ in capability and reliability.
- System Behavior: Availability trends show more pronounced fluctuations in the short term but achieve better stability in the long term. High-capacity and high-bandwidth peers compensate for weaker nodes, enabling the system to recover more effectively under stress.
- Resilience: The diversity in peer capabilities allows the system to distribute tasks more dynamically. Stronger peers stabilize availability during recovery periods, while weaker peers have less impact on overall redundancy.

Peer review

Minor changes:

- The table of contents is added at the beginning of the report.
- Srpt simulation average time table is added in SRPT extension chapter.
- Explicit mention of edge cases.
- *Readme* readablity improved.