Assignments Report

COURSE: DISTRIBUTED COMPUTING

BUSHRA MUNEER (6180175)

University of Genoa

Instructor: Professor Matteo Dell 'Amico

ABSTRACT

The two assignments that were finished for the Distributed Computing course make up this report. The course covers subjects such as queueing and scheduling, as well as erasure coding techniques used in distributed systems.

The part 1 of the assignment consists of software simulating the behavior of:

Queuing and Scheduling

We have:

- M/M/1
- M/M/N
- M/M/N With supermarket
- M/M/N with RR (Round Robin)

Then we have another part where we simulate a peer-to-peer backup application using erasure coding:

- Single upload/Download in Distributed System
- A P2P System with multiple uploads/downloads

Contents

ABSTRACT 1
Part One: Queuing and Scheduling 3
  Introduction 3
  Approach 3
    Overview of the Approach 3
    Details 3
  Implementation 5
  Results for M/M/1 and M/M/N 5
  M/M/N (SuperMarkt) 6
  M/M/N (Round Robin) 6
  Conclusion 8
Part Two: Erasure Coding for Data Restoring 9
  Introduction 9
  Approach 9
    Overview 9
    Details 9
  Implementation 12
  Conclusion 13

Part One: Queuing and Scheduling

Introduction

Basically, this scenario aims to simulate and compare the actions of a queuing system composed of two components: waiting areas, and the queues where jobs wait to be executed, and service nodes.

Approach

Overview of the Approach

Random variables in the simulation have been given values that are used to produce arrival and completion delays.

- Simulation-generated data has been compiled and plotted.
- The generated data was contrasted with theoretic outcomes.

Details

Theoretical results related to the system were determined using Little's law.

$L = \lambda W$

where:

L - the average queue size

λ - the rate at which jobs arrive on the system

W - the average time expenditure that jobs take on the system.

Additionally, some assumptions were made that is

$$\lambda < \mu$$

where:

μ – the rate at which jobs leave the system

● Otherwise, the queue is not stable.

It will be out of queue memory.

● The probability of a job leaving/arriving the system does not change over time.

Arrival delays and completion delays are generated using an exponential distribution.

- Since the distribution is memoryless, this guarantees prior assumptions.

If there's only one server in a system, and we're following the assumptions mentioned earlier (referred to as an M/M/1 queue), then W can be figured out like this:

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

$$W = \frac{1}{1-\lambda}$$

Figure 1: Theoretical formula for W

When a system has multiple servers, it's known as an **M/M/N queue**. If we keep the queues independent (meaning we randomly pick a queue when a task comes in), it doesn't change how the system behaves. We can still use the same formula to calculate W, a measure of system performance.

To explore different ways of balancing the workload in the M/M/N system, **a supermarket model was created**. This model introduces a new factor called "d," which represents the sample size. When a task arrives, the system randomly picks a sample of queues. The task is then added to the least busy queue among those sampled.

According to theory, when we have a large number of servers, using the supermarket model for load balancing should result in specific patterns in graphs.

In real life, tasks don't always come at a steady, predictable pace. Sometimes there's a mix of small and large tasks. Because of this variability, it's important to consider scheduling methods beyond the basic First-In-First-Out (FIFO) approach.

To prevent smaller tasks from waiting too long and not getting done, we need to explore other options.

One such scheduling method that allows for interrupting and switching between tasks is called **Round-robin.**

Round-robin scheduling works by giving each task a turn to execute within a set time slot. If a task finishes in time, the next one starts; if not, the scheduler pauses it, puts it back in line, and moves on to the next task.

In theory, how well round robin performs depends on the randomness of task times, modeled by the Weibull distribution.

It tends to be better than FIFO (First-In-First-Out) when the randomness is below 1. But when it's above 1, FIFO works better.

The question is, can combining Round-robin with supermarket scheduling change this threshold?

To adapt to different situations, some settings can be adjusted using command line parameters:

"**–rr**": If set to 1 (which is the default), it uses Round-robin; if set to 0, it goes for the supermarket method.

"**–d**": This sets the sample size for the supermarket decision. If set to -1, the queue choice is random; otherwise, it samples and selects the least busy queue.

"**–weibull**": This determines the shape of the Weibull distribution for task delays, with a default value of 1 for an exponential distribution.

"**–plots-dir**": This is the directory where any generated plots will be saved.

Implementation

The simulation software is ready to simulate various queueing systems based on command line options. The simulation runs for a duration of 100,000-time units.

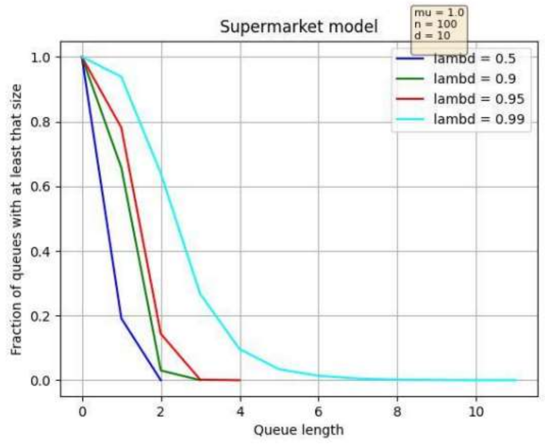
- For a single-server queue (M/M/1) with random choice and FIFO scheduling, use the flags: "-d -1 -n 1 -rr 0".
- For a multi-server queue (M/M/N) with random choice and FIFO scheduling, where N is the number of queues (set to 10 for this experiment), use the flags: "-d -1 -n N -rr 0".
- For a multi-server queue (M/M/N) with supermarket choice, use the flags: "-d <sample_dim> -n N -rr 0".
- For a multi-server queue (M/N/N) with Round-robin scheduling, use the flags: "-d <sample_dim> -n N".

Results for M/M/1 and M/M/N

	λ	mu	Max_time	w	d	N	**Weibull shape**
M/M/1	0.7	1	100000	3.3410975484953768	-1	1	1
M/M/N	0.7	1	100000	3.3273295876078643	-1	10	1

It basically shows that the results are quite similar, there is not a lot of difference from the theoretical expectations, and this is happening because the solution is complicated and complex and the operating system has extra work to do, not necessarily because the algorithm itself isn't working well (performance factor).

M/M/N (SuperMarkt)



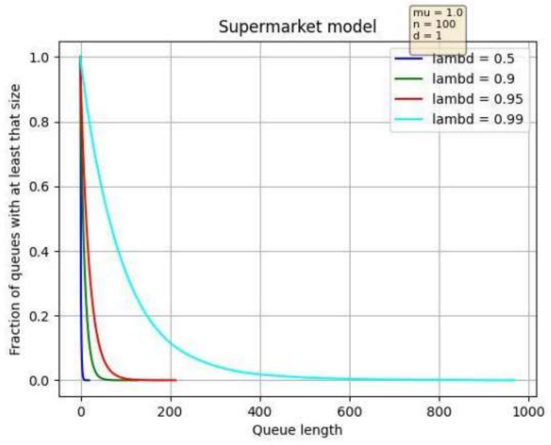


Figure where d=1

The obtained results for mu, number of servers and increase of lambda value with random choice.

Figure where d=10

The obtained results for mu, number of servers, sample dimension and increase of lambda value.

This basically shows how the queues size grows when having a sample dimension.

M/M/N (Round Robin)

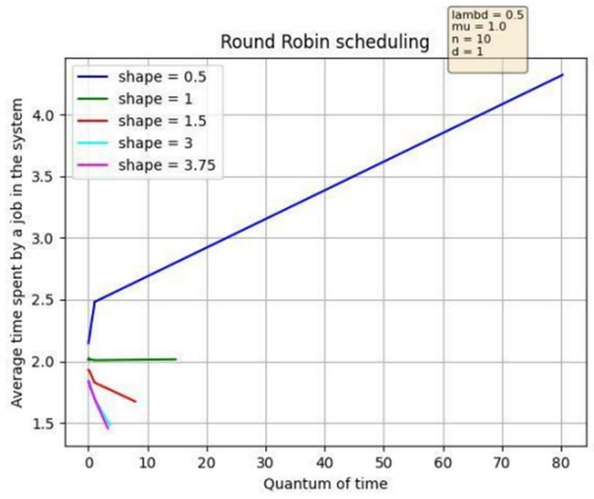


Figure: Experimental results for different Weibull forms with constant lambda, Mu, number of servers, and random selection.

The above figure shows how the combination of round-robin scheduling and supermarket selection can shift the threshold at which FIFO performs better than round-robin.

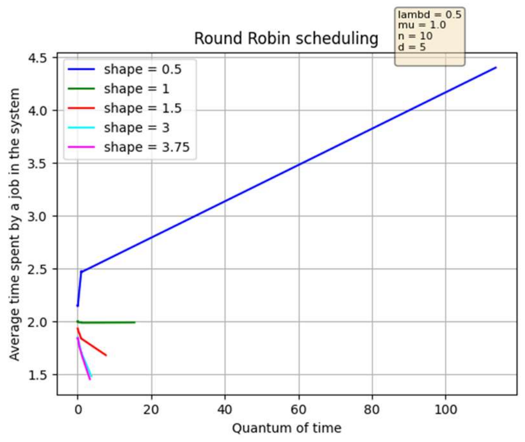


Figure: Experimental Results for Constant Lambda, μ, Number of Servers, and Different Weibull Forms According to Supermarket Decisions

The above figure relates to the theory:

In fact, for Weibull format where is less than 1 (continuously decreasing), round robin is better than his FIFO (this is a special case of round robin, the time quantum is larger) more than that (corresponds to the maximum job duration due to large differences between job dimensions).

From the data collected, round robin spends less time in the system than FIFO on average, and that time does not exceed the maximum job duration. FIFO performance improves when the Weibull shape is greater than 1.1 is the optimization threshold. Other cases such as exponential distribution (shape = 1), right skewed (1.5 <= shape <=2.6).

Conclusion

For M/M/1 and M/M/N queues:

As expected, when we let queues work independently with random choice and FIFO scheduling, both M/M/1 and M/M/N systems act similarly.

Their results are close to each other and align with the theoretical expectations shown earlier (see figure 1) for the average time a task spends in the system.

For the supermarket model:

Compared to completely random choice (when the sample dimension is 1), the supermarket model (with a sample dimension greater than 1) significantly reduces queue lengths while lessening the impact of the lambda parameter.

This also matches what we expected theoretically.

For Round-robin scheduling:

When compared to FIFO scheduling, Round-robin with random choice behaves as we theorized. Specifically, for tasks with widely varying sizes (like those with a steadily decreasing Weibull distribution with a shape less than or equal to 1) and high differences in size, Though Round-robin works better than FIFO in terms of the average time a task spends in the system.

However, when task sizes are similar, FIFO turns out to be the better option.

Moreover, if we combine Round-robin with the supermarket model it basically shifts the threshold where FIFO performs better.

In fact, Round-robin outperforms FIFO even with a Weibull distribution that is close to the exponential one (shape = 1), while FIFO becomes more efficient when the shape of the Weibull distribution is distinctly over 1.

Part Two: Erasure Coding for Data Restoring

Introduction

The purpose of this scenario is to simulate and analyze the behavior of a backup and recovery network that uses erasure coding for data recovery.

- This system consists of nodes, each with its own configuration and data.
- The simulation involves an analysis of the behavior of her system when certain events occur.
- System node configuration consists of upload/download speed, lifetime/uptime/downtime.
- The size and number of blocks are specified for the node.

Examples of events include node failure, connection loss, and recovery.

Approach

Overview

This experiment focuses on analyzing bandwidth waste in upload and download operations.

For this, a simulation of a node network with the specified attributes was created.

- Node configurations are saved in ".cfg" files.
- Node configuration files store values in a format that Python parses through an easy-to-use library.

  This library allows you to represent time and size values using common units, such as "days" for time and "MiB" (meaning MB) for size.

- The simulation was supplied with values from a configuration file used by random variables to generate event delays.
- The data generated were compared with theoretical results.
- The data generated by the simulation is collected and graphed.

Details

When single upload/single download mode is used, each node pursues a bandwidth loss equal to the difference between the download and upload speeds.

In a real scenario, the download speed of each node is greater than or equal to the upload speed, so only these cases are considered in this experiment.

As a result, the bandwidth waste calculator for each node follows the following formula:

$$W_i = D - U$$

Where we have:

U → the uploading speed

W → the waste of bandwidth for the node i

D → the downloading speed

In the case of transfer, bandwidth is wasted on the downloader node side.

Therefore, the expected waste of upload bandwidth is 0.

The opposite situation occurs when the upload speed is faster than the download speed.

However, in real-world scenarios, it is rare to force a machine to perform one download/upload on a one-to-one basis.

To use bandwidth more efficiently, the system has been enhanced to allow nodes to perform multiple uploads and multiple downloads in parallel.

The theoretical results of this extension depend on the ratio of upload speed to download speed.

In practice: –

- If the download speed is a multiple of the upload speed (e.g.5x), download bandwidth waste is reduced (this does not affect the upload speed results).
- If the download speed is equal to the upload speed, no bandwidth is wasted (works the same as the single upload/single download scenario).

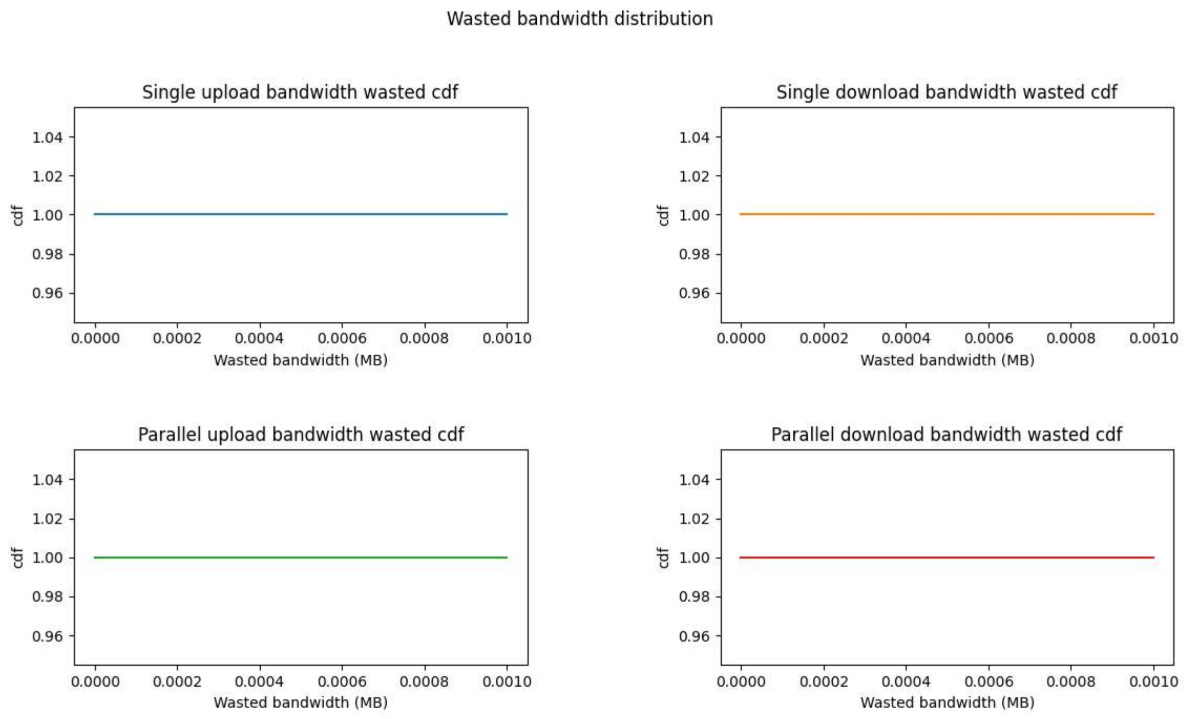


Figure: This figure shows the result for single/parallel download and uploading. Although we have download and upload speed same here.

Data for the above figure:

```
[peer]
    number = 10
    k = 8
    n = 10
    data_size = 1 MiB
    storage_size = 100 GiB
    average_uptime = 60 days
    upload_speed = 8 MiB  # per second
    download_speed = 8 MiB  # per second
    average_recover_time = 8 hours
    average_lifetime = 1 year
    average_downtime = 1 hour
    arrival_time = 0
```

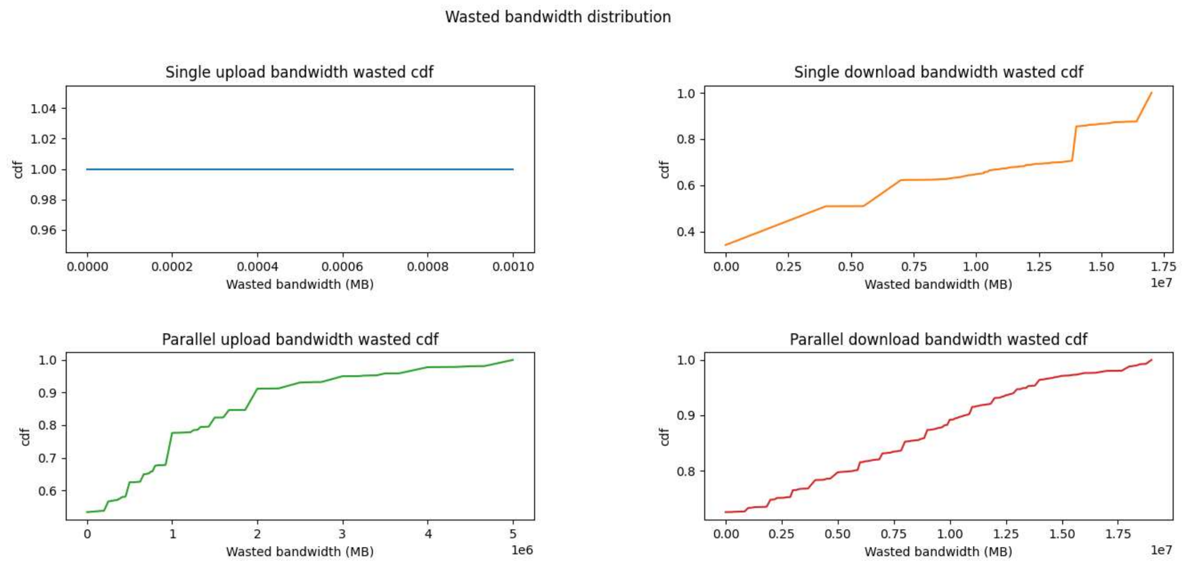


Figure: Obtaining results from experiments where we compare single and parallel download/upload actions for two types of nodes: one group with stronger network features, and another with weaker ones.

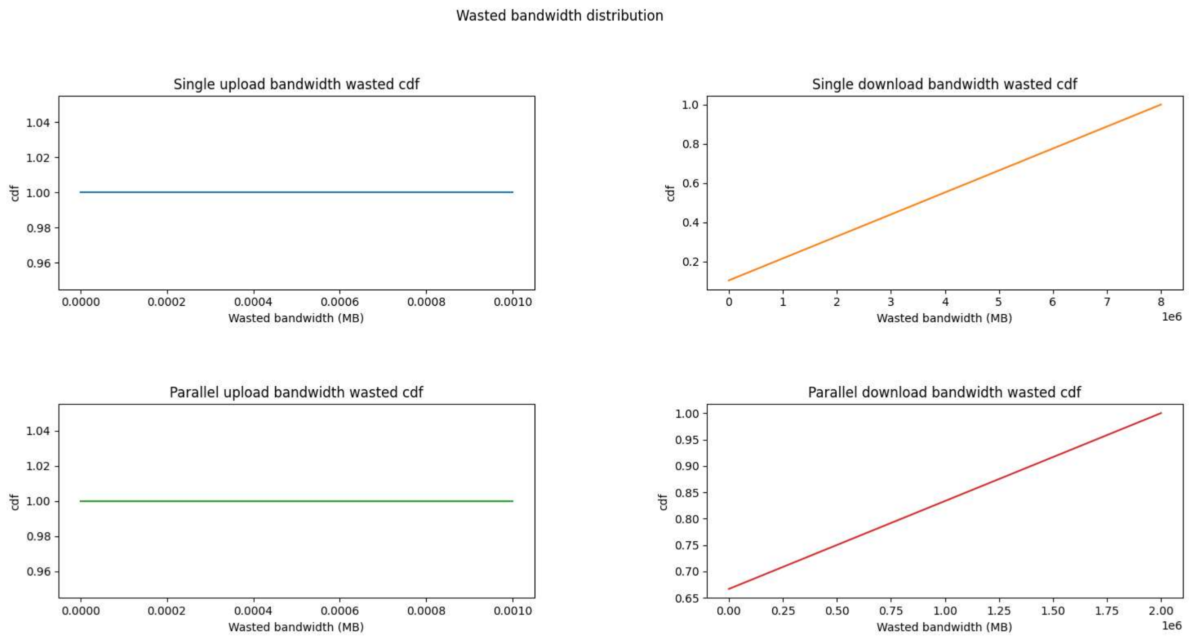


Figure: Obtaining results from experiments where we compare single and parallel download/upload actions where download speed is multiple of of the upload speed and is 5 times.

Implementation

Simulations were performed for below different cases using example configurations.

Single upload and single download:

Where:

- Download speed is a multiple of upload speed.
- Download speed is the same as upload speed.

Parallel upload and download:

Where:

- Download speed is a multiple of upload speed.
- Download speed is the same as upload speed.

If the network consists of two different node classes.

```
[peer]
    number = 10
    k = 8
    n = 10
    data_size = 1 MiB
    storage_size = 100 GiB
    average_uptime = 60 days
    upload_speed = 6 MiB  # per second
```

```
[weak_peer]
    n = 10
    k = 8
    number = 10
    average_uptime = 30 days
    data_size = 1 MiB
    storage_size = 100 GiB
    average_lifetime = 180 days
    download_speed = 20 MiB  # per second
    average_recover_time = 8 hours
    average_lifetime = 1 year
    average_downtime = 1 hour
    arrival_time = 0
```

```
    upload_speed = 3 MiB  # per second
    download_speed = 10 MiB  # per second
    average_downtime = 2 hours
    average_recover_time = 8 hours
    arrival_time = 0
```

Conclusion

The cumulative distribution function (CDF) was plotted to show how the average wasted bandwidth is distributed for the presented scenarios.

This represents the probability that the average wasted bandwidth is less than or equal to a certain value.

Analyzing the above graph, we get the following conclusions:

Since the download speed of each node is many times the upload speed (all nodes are equal in terms of network parameters), parallel increases in downloads are possible and more bandwidth is likely to be saved.

Whereas on the other hand, upload doesn't change anything (Even more efficient than the previous scenario). However, it costs and wastes upload bandwidth. The two systems appear to behave the same because the download speed of each node is equal to the upload speed (all nodes are equal in terms of network parameters).

In this case, each node is actually limited to only a single ongoing upload/download, so bandwidth conservation takes precedence over efficient usage.

For two classes of nodes (the first is more powerful in terms of network parameters and availability, the second is weaker), enabling parallel transmission may reduce the probability of saving download bandwidth (peaks higher and achieves waste).

This increases the likelihood of wasted upload bandwidth.

This may be due to differences between peer classes.