Report on dc assignment

23/02/2023

—

Members of the Group
Ali Haider - S4811831
Giacomo Pedemonte - S4861715

A quick view of this report

We report in this relation, our consideration related to the DC assignment.

In those exercises we use Simulations to obtain simulated results about topics viewed during the course.

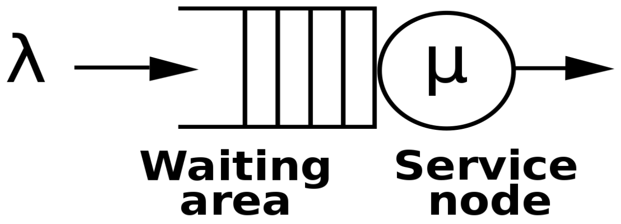
Objectives

1. From M/M/1 to a multi-server version with M/M/N
   a. Implementing the Supermarket Theorem to check theoretical expectations and the real average time spent in the system.
   b. Obtain different graphs depending on lambda and the supermarket choices
2. Implementation of a hundred year simulation about storing in client server config and also in peer to peer configuration.
   a. Plot views referring to lost blocks at the end of more tests on different simulations for configuration sets.
3. Extensions for the storage part of the assignment.

M/M/1 queue description:

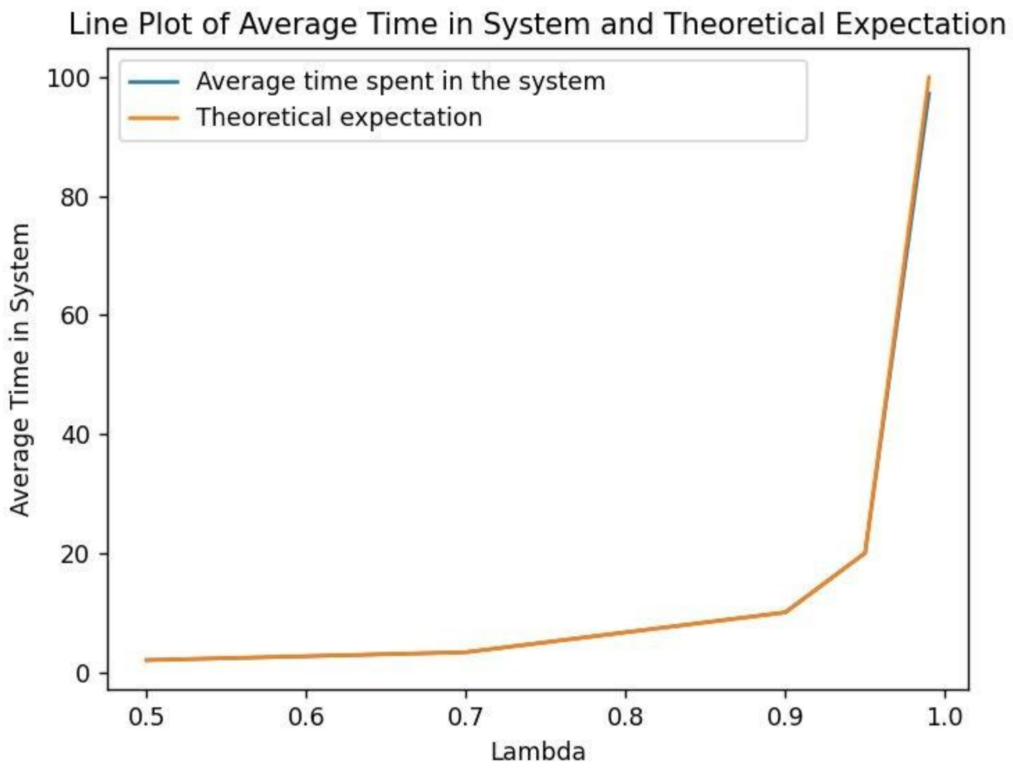
Starting with M/M/1: we have a single server and a queue of jobs it has to serve, jobs arrive and are served in a memoryless fashion. When a job arrives the process phase checks if no job is running on the only queue that we have, if there's no job then it runs the new job otherwise puts it in that queue and waits for the completion.

λ and μ are the average rates at which jobs respectively join the queue and leave the system when they are complete.



We simulated an M/M/1 FIFO queue, obtaining the average time spent in the system. The theoretical model results derive from an equilibrium situation.

In order to compare our results to the theoretical ones, we plotted the average time spent in the system in multiple tests, in particular 100, on different lambdas. We noticed that they were very similar.



So we choose a line plot to simply understand that the average time spent in the system is related to lambda in a way that if lambda increases, the average time in the system increases as well.

We can see that the two lines are almost one over the other and this represent that our result of the Average time spent in the system is almost the same of the Theoretical expectation for random server choice that in the case of M/M/1 represent that we obtain the same results of the Theoretical ones.

Now we can move considering if we increase the number of queues what we need to do?

M/M/N queue description:

In the M/M/n version we added n queues related to their running queue.

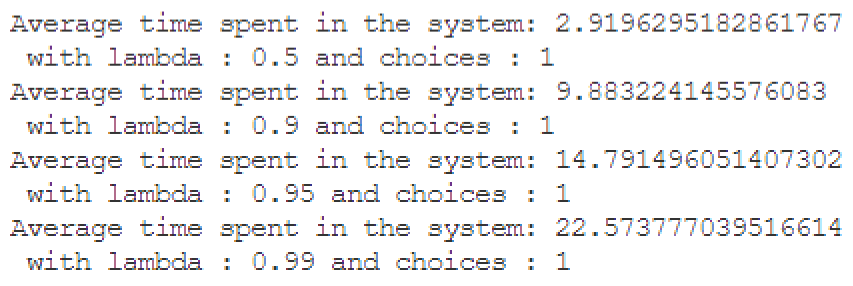
We need a way to decrease the average time spent in the system, so if we need to consider n queues that are two ways of getting it:

- randomly selecting a queue and assigning the job to it.
- making a "supermarket model choice" of the queue.

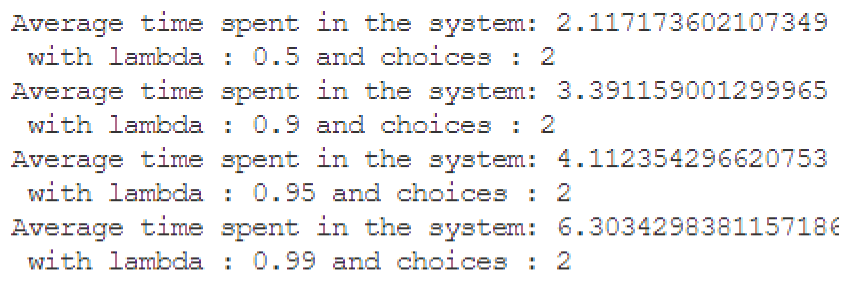
We implemented the supermarket model that is a strategy utilized to perform better than choosing a random queue. But what is the supermarket model?
With the supermarket model, we choose a limited set of queues and find it in the queue with the minimum length. The job is assigned in this queue.

We start thinking about the difference in terms of Average Time Spent in the system of the jobs with the usage of random choice or starting with having 2 choices with the supermarket model.

So, we reported in the following the results that we obtain with different lambda and supermarket choice of queues = 1 and the number of total queues is 100(choose randomly one of those queues) :


```
Average time spent in the system: 2.9196295182861767
 with lambda : 0.5 and choices : 1
Average time spent in the system: 9.883224145576083
 with lambda : 0.9 and choices : 1
Average time spent in the system: 14.791496051407302
 with lambda : 0.95 and choices : 1
Average time spent in the system: 22.573777039516614
 with lambda : 0.99 and choices : 1
```

Now, the following results represent the same code with more choices for the supermarket phase and we obtain better results each time we increase that number of choices.

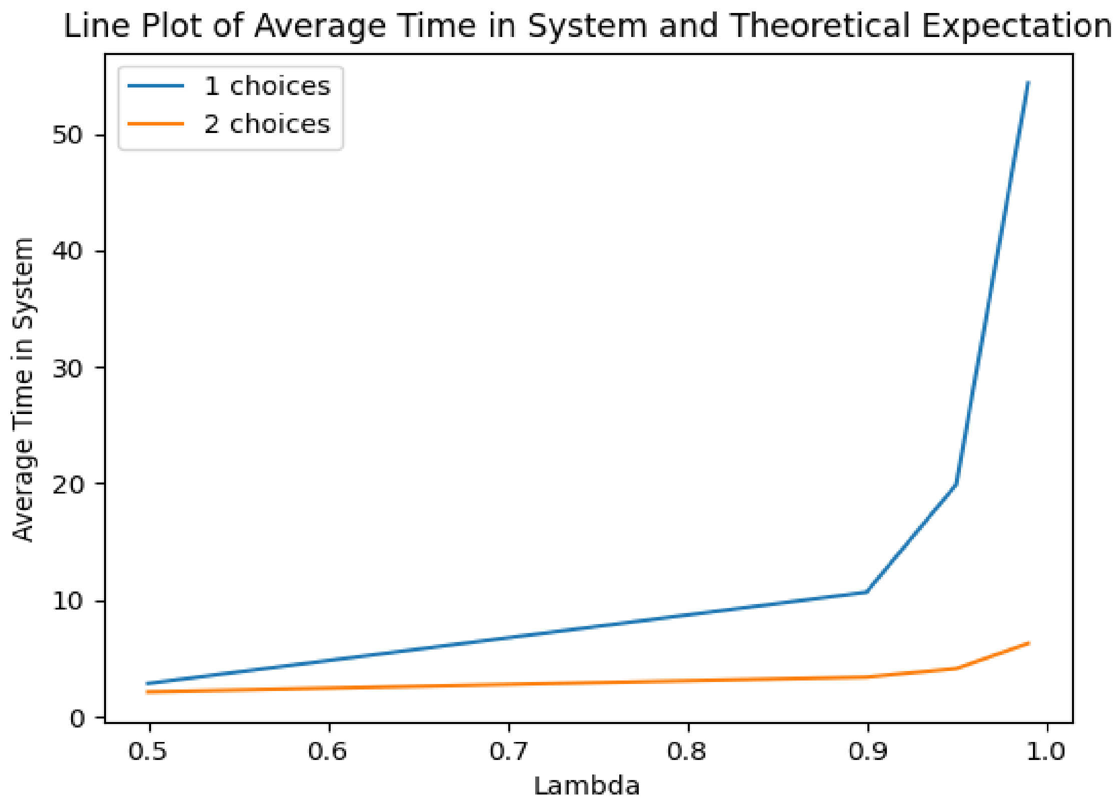


```
Average time spent in the system: 2.117173602107349
 with lambda : 0.5 and choices : 2
Average time spent in the system: 3.391159001299965
 with lambda : 0.9 and choices : 2
Average time spent in the system: 4.112354296620753
 with lambda : 0.95 and choices : 2
Average time spent in the system: 6.3034298381157186
 with lambda : 0.99 and choices : 2
```

So those results represent the different average time spent in the system when the choices in the supermarket model are 1 or 2.

If this number is 1 is like having a random choice and we obtain the result presented in the first block of execution.

We can easily see that if we increase the number of choices at 2 we obtain a gain in terms of the average time spent in the system that is decreased obtaining that the system gains performance and fastness of completing a job.



To better understand this difference we can see here that the average time in the system for 2 choices is much more efficient than 1 random choice. We did this graph running 100 on 100 queues with different lambda.

Michael Mitzenmacher in his paper "The Power of Two Choices in Randomized Load Balancing", published in 2001, analyzed the performance of the supermarket model, which is a load balancing model to distribute incoming requests among servers.

Observation:

He found that in this model, the fraction of queues with at least “i" jobs drops from

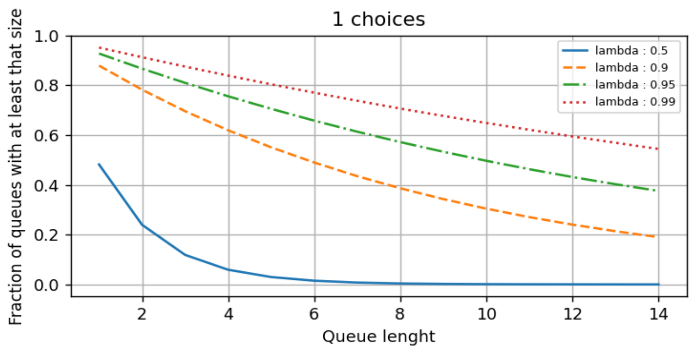
$$\lambda^{i} \text{ to } \lambda^{\frac{d^{i}-1}{d-i}}$$

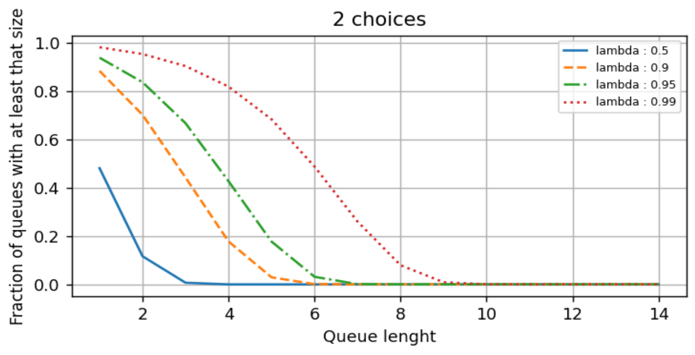
as “i” increases, where λ is the average arrival rate.

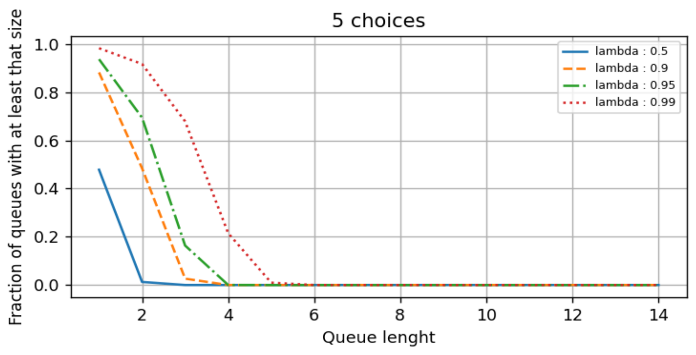
We can see this in a better way with graphs that figure out the fractions of queues with at least that size on y axis and different queue length on the x axis.

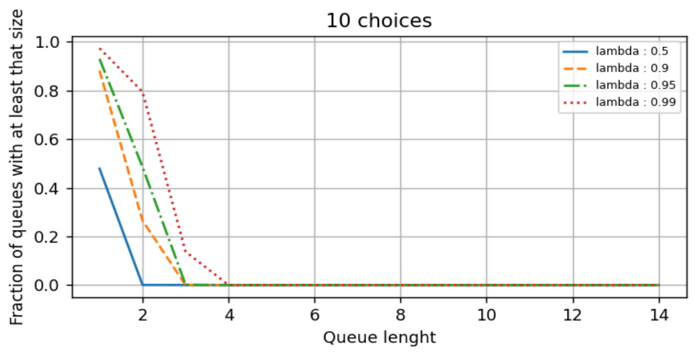
Queue Length vs Theoretical Queue Length

In the following we want to consider and be able to take a look of Our results that we obtain by running simulations take in consideration the theoretical expectation that we see in the slide during the course. So, we report here the graph that will be obtained with the execution of the “mmn.graphics.py” file in the assessment.









In this graph are reported different views of the supermarket model used in our simulation in the M/M/N context (where in our case N is 1000).

We obtain this plot using a “special” event that monitorizes our queue in different intervals of time that can be setted as we like. To obtain this we set this interval to 1 so we obtain those queue length values for each unit of time of the simulation.

In this way, for each lambda we obtain all the length of the queues during the simulation in order to figure out the Fraction of those queues with at least ‘x’ number of jobs.

The first graph on top left of this image represents the random choice of one of the queues available (in this case we run this with 1000 queues).

The second graph, on the same line, it’s the graph representing the supermarket model applied with 2 queues chosen at time.

This gave us the view of what we had reported when talking about the average time spent in the system for a job, so we obtained an improvement in terms of performance and as you can see also in terms of fastness.

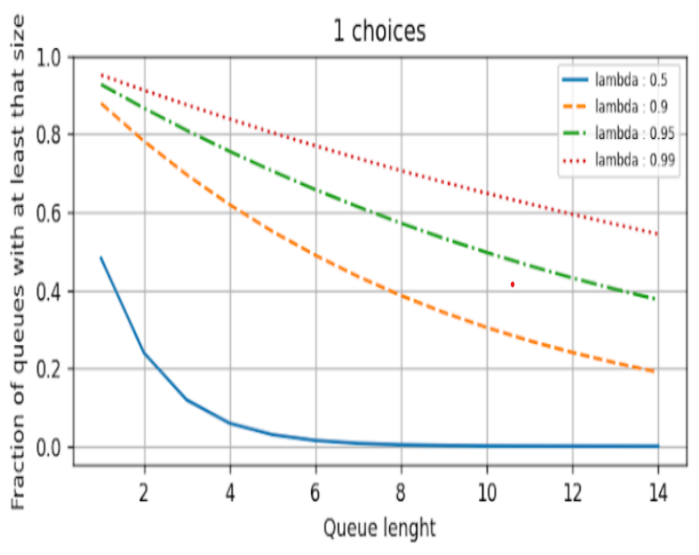
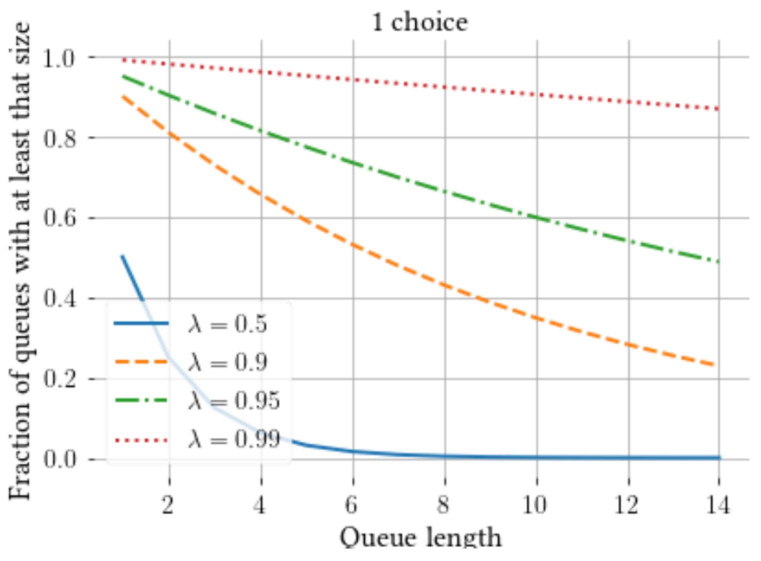
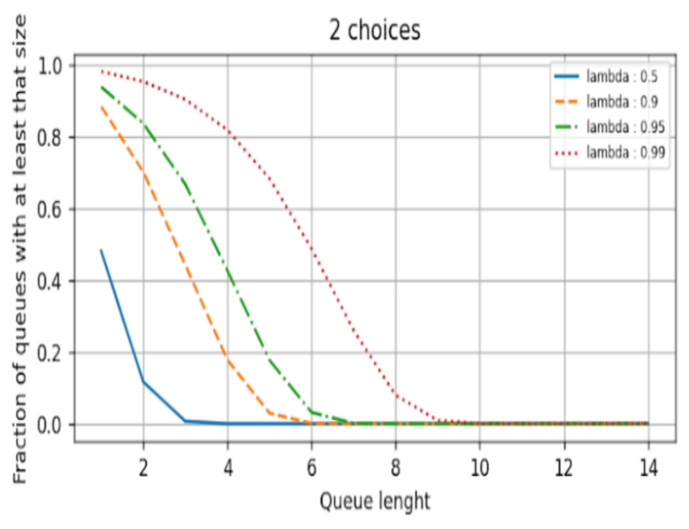
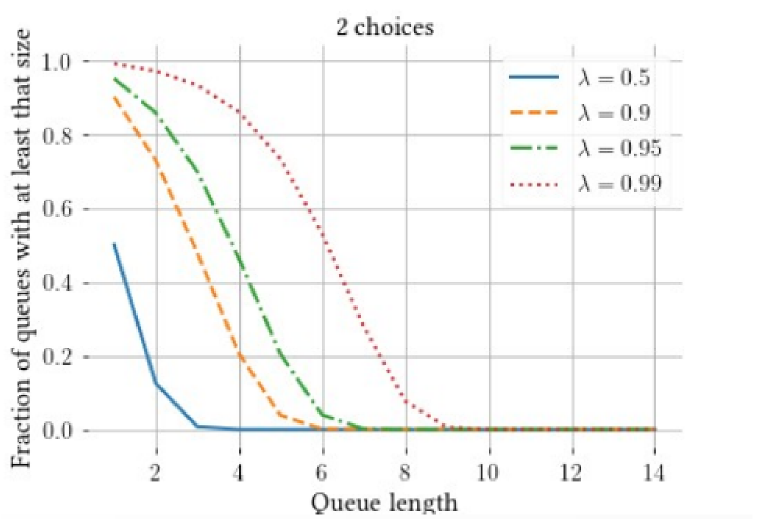
The graph in the bottom left, on the second line, represents that by increasing the choices of the queues we obtain a more efficient redistribution of the queue length during the simulations.

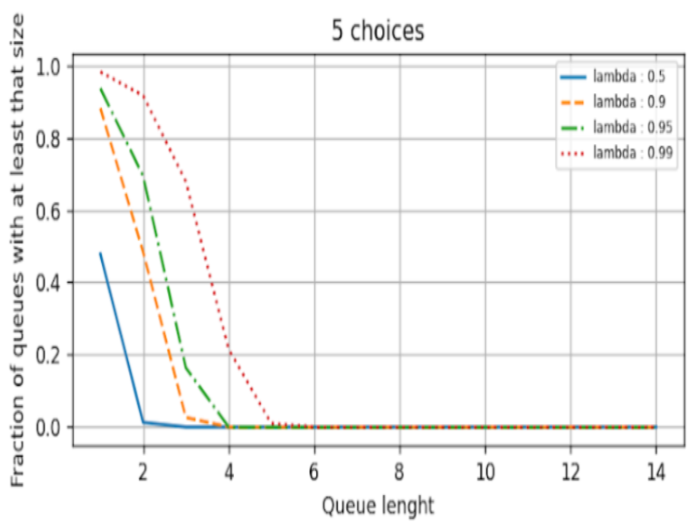
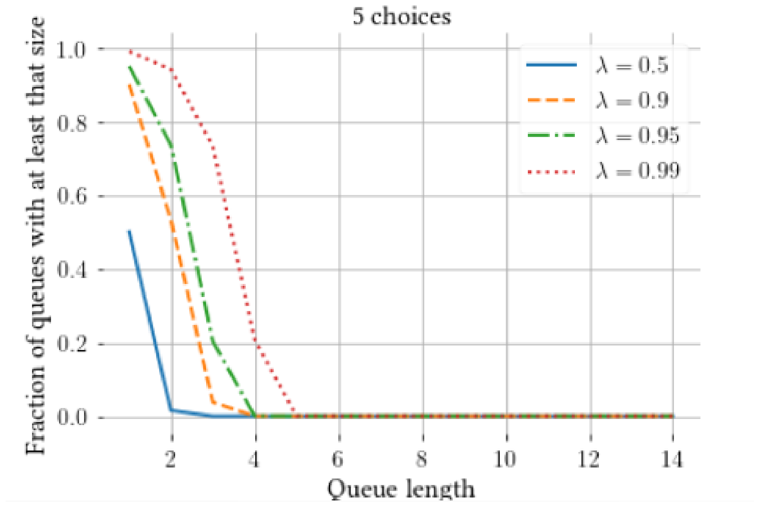
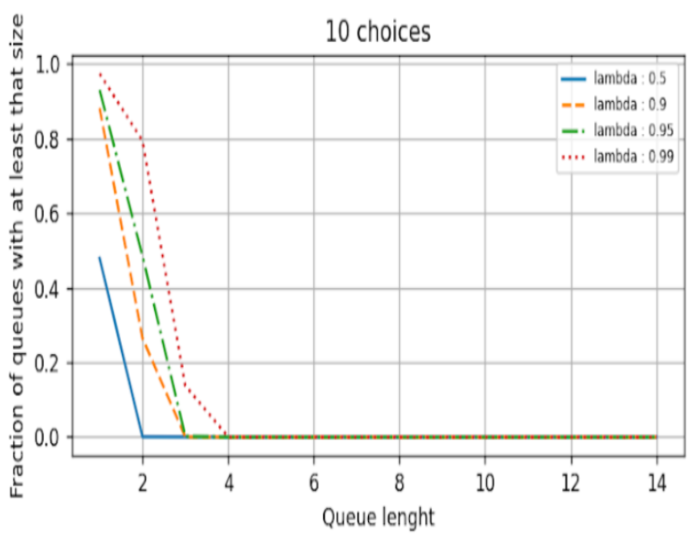
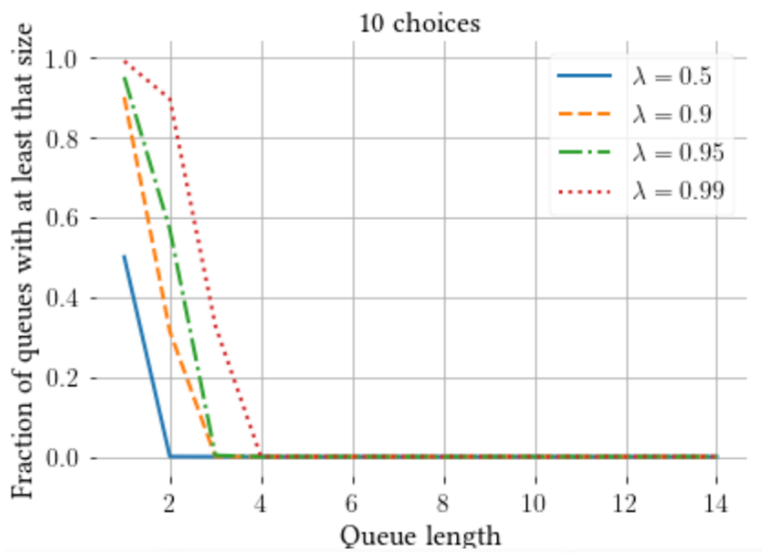
It’s possible to see this written above in a better way on the fourth graph where, with a higher lambda we obtain a more decreasing line for all the lambda plotted on that specific graph.

In the following page we compare those graphs above with those that we can obtain applying the Mitzenmacher formula and so the theoretical view of the fraction of queues with at least that size.

In order to obtain a comfortable view of the two graphs we present a table containing all the graphs one by one.

We start reporting the main difference that Mitzenmacher wants to figure out, so we can see the difference between the fraction of queues with at least that size when using the random choice(1 choice) or when we use 2 choices in the supermarket model.

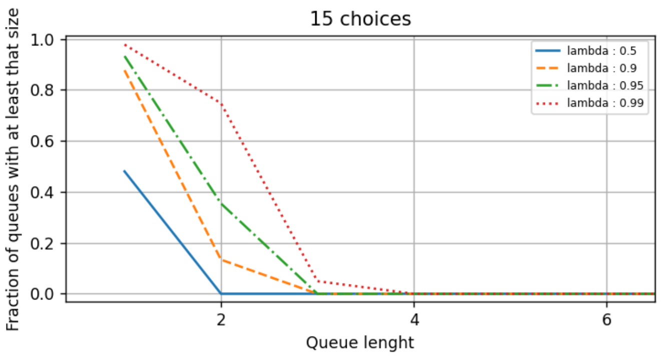
Queue Length	Theoretical Queue Length
	
	

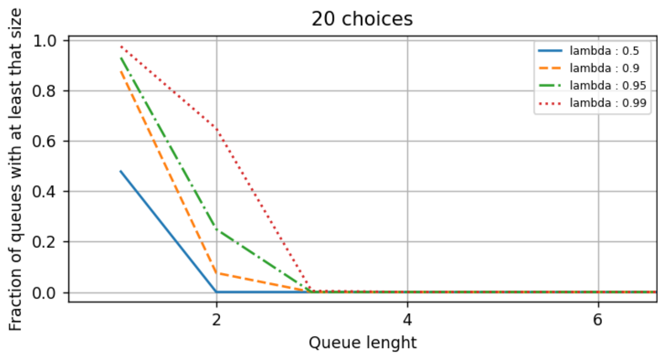
Queue Length	Theoretical Queue Length
	
	

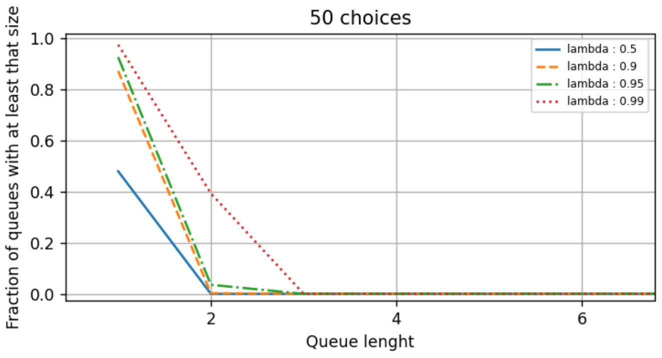
In each graph is represented the difference between the same run with different types of lambda reported in the relative graph legend.

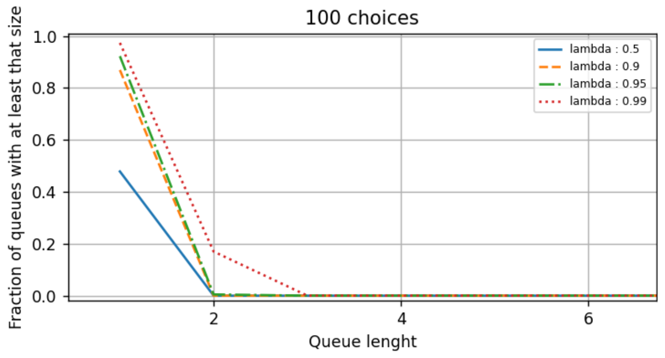
With these views we can see that the results that we obtained for those runs are just like the theoretical ones that we see during the lessons.

In the following we present also if we increase more the number of choices considering always that we have an overall of 1000 queues to address.









We reported those graphs(obtained with the same sampling as before), zoomed in with respect to the previous one to observe better the minimal difference between the different lambda runs when the choices increase to a higher value than the theoretical ones.

Moreover, we can observe with those different runs that the more choices we make, the more gain we obtain.

Obviously the real impact of this supermarket model is when we go from 1 choice(random) to 2 choice, in this graph we can observe that there are no more big differences when choices increase to higher values.

Storage description:

In the Storage part we have data splitted in N blocks, and uploaded to peers. We simulate recovering all N blocks as soon as K are available.

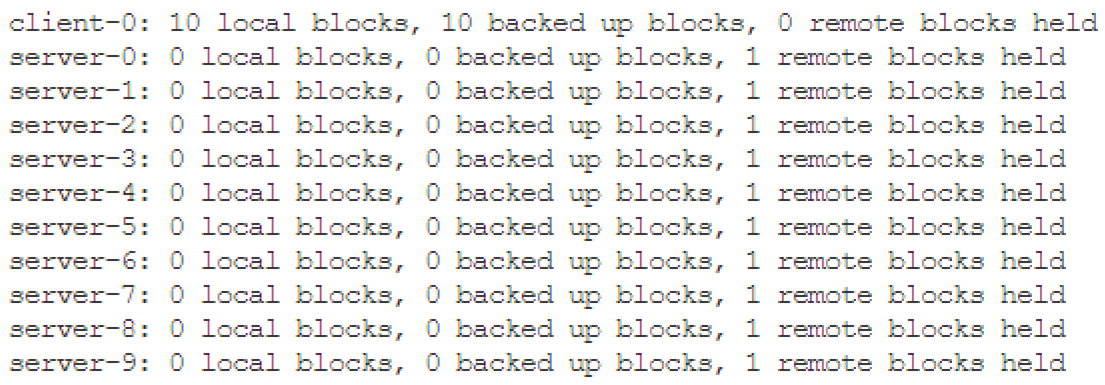
Each node has configurable disk space, bandwidth, amount of time between crashes, etc.

Client-Server simulations:

In the client server simulation we simulate a client server architecture where client nodes need to store(backup) blocks of data in the server nodes.

First of all we consider the case of one only client and the same number of servers as the number of blocks that we are using to store "different pieces" of our original data.

We obtain that all the servers have a block of the client that want to store his data like reported in the following:



```
client-0: 10 local blocks, 10 backed up blocks, 0 remote blocks held
server-0: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-1: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-2: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-3: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-4: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-5: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-6: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-7: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-8: 0 local blocks, 0 backed up blocks, 1 remote blocks held
server-9: 0 local blocks, 0 backed up blocks, 1 remote blocks held
```

But what happens if we consider having more clients that want to store data on the available server?

At the beginning of our implementation we obtain an hybrid variant of this where clients can also store data of other clients if needed.

So in that case was a hybrid variant of the client server architecture that we were thinking about.

```
client-0: 10 local blocks, 10 backed up blocks, 1 remote blocks held
client-1: 10 local blocks, 10 backed up blocks, 1 remote blocks held
server-0: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-1: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-2: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-3: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-4: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-5: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-6: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-7: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-8: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-9: 0 local blocks, 0 backed up blocks, 0 remote blocks held
lost blocks = 0 of 20
```

So in this way we can obtain a situation like this above where a server doesn't store any blocks of the clients and make clients lose a bit of their space to maintain blocks for the other client. In this way we lose the separations of behaviors of clients and servers.

Now we implement this in a way that clients can't hold blocks of other clients in order to use only the servers to store blocks.

In this way we improve the hybrid system to obtain the following results:

```
client-0: 10 local blocks, 10 backed up blocks, 0 remote blocks held
client-1: 10 local blocks, 10 backed up blocks, 0 remote blocks held
server-0: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-1: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-2: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-3: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-4: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-5: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-6: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-7: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-8: 0 local blocks, 0 backed up blocks, 2 remote blocks held
server-9: 0 local blocks, 0 backed up blocks, 2 remote blocks held
lost blocks = 0 of 20
```

Like we can easily see, no more clients store data blocks for the other one in order to fill the servers and make the servers store data instead occupying clients data space for other clients blocks of data.

Peer-to-peer simulations:

In .cfg files we have the configurations of the nodes:

- name: the node's name (client, server, peer)
- number: number of nodes of “name” type

- n: number of blocks in which the data is encoded
- k: number of blocks sufficient to recover the whole node's data

- data_size: amount of data to back up (in bytes)
- storage_size: storage space devoted to storing remote data (in bytes)

- upload_speed: node's upload speed, in bytes per second
- download_speed: download speed

- average_uptime: average time spent online
- average_downtime: average time spent offline

- average_lifetime: average time before a crash and data loss
- average_recover_time: average time after a data loss

- arrival_time: time at which the node will come online

- evil : evil is a boolean attribute that all the configurations set to False.
    - Spoiler: we use this in the extension of this p2p system in order to consider the impact of selfish nodes(that we call for simplicity evil) in our system

TESTS:

We organize the code to obtain multiple **tests** of the simulations with different configuration files.

We take in the point of view of a project manager or a systemist that wants to evaluate/simulate the behavior of different kinds of p2p systems depending on different needs that can be considered when organizing the system architecture.

In order to discover this we see with a more detailed view those parameters of the configurations trying to find out the perfect configuration depending on our needs where perfect is measured by the loss of blocks at the end of the simulations.

Choosing the optimal parameters for our simulations to minimize data loss:

Increasing the **number of nodes** in the system can help to reduce the data loss rate, but it also increases the storage space and the network resources required.

The **number of blocks in which the data is encoded**, **n,** should be greater than or equal to k, the number of blocks sufficient to recover the whole node's data.

The **number of blocks sufficient to recover** the whole node's data, **k**, should be greater than or equal to the number of blocks that can be lost.

The **data_size** to back up should be proportional to the storage space devoted to storing remote data.

The **storage_size** devoted to storing remote data should be greater than or equal to the amount of data to back up.

The **upload and download speeds** should be sufficient to transfer the data in a reasonable amount of time.

The **average uptime and downtime** should be set to achieve the desired availability of the system.

The **average lifetime** should be set to a value that is long enough to ensure that data is not lost due to node failure.

The **average recovery time** should be set to a value that is short enough to minimize the amount of data lost due to node failure.

The **arrival time** should be set to a value that ensures that the nodes are online and available when the data needs to be backed up.

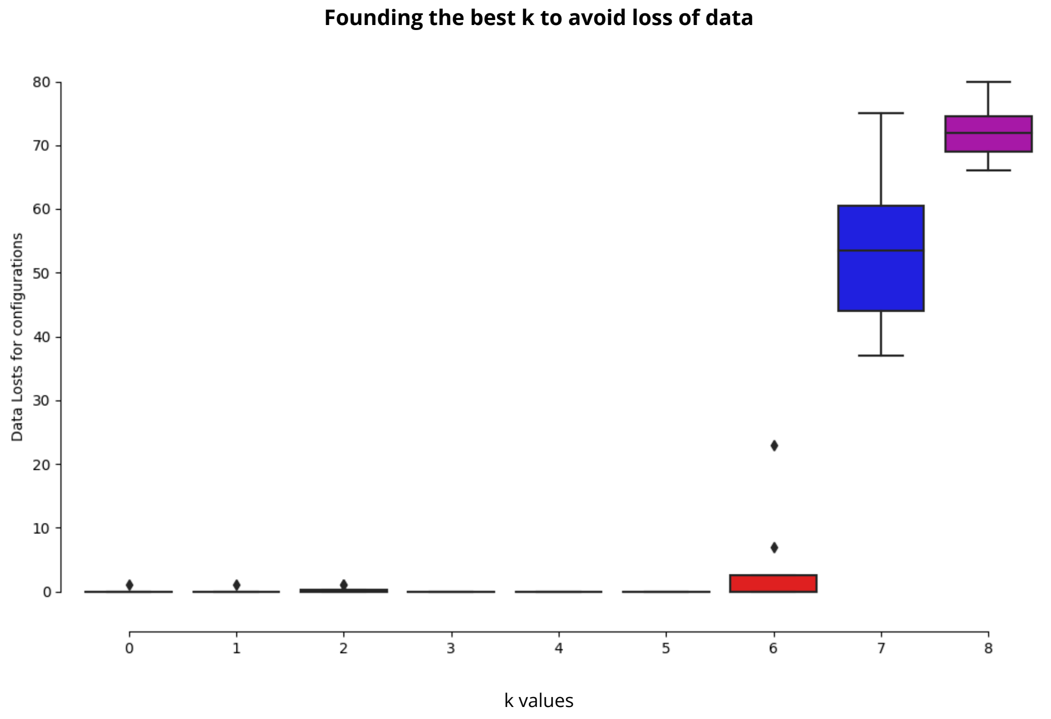
Founding the best k to avoid loss of data

We start modifying the configuration thinking about how much impact the k value has on the loss of blocks during the simulation.

So we start to analyze the impact of k where ***k* = *n-m***, where n is the number of blocks in which the data is encoded and m is the number of failures that we want to avoid.

K is also important because the block size depends on it because the block size is ***1/K^th*** of the original data. In the following we try to summarize which k is the best in order to obtain that we don't lose data in the following configuration.

```
number = 10

n = 10

k = x

data_size = 1 GiB

storage_size = 10 GiB

upload_speed = 2 MiB

download_speed = 10 MiB

average_uptime = 8 hours

average_downtime = 16 hours

average_recover_time = 3 days

average_lifetime = 1 year

arrival_time = 0
```



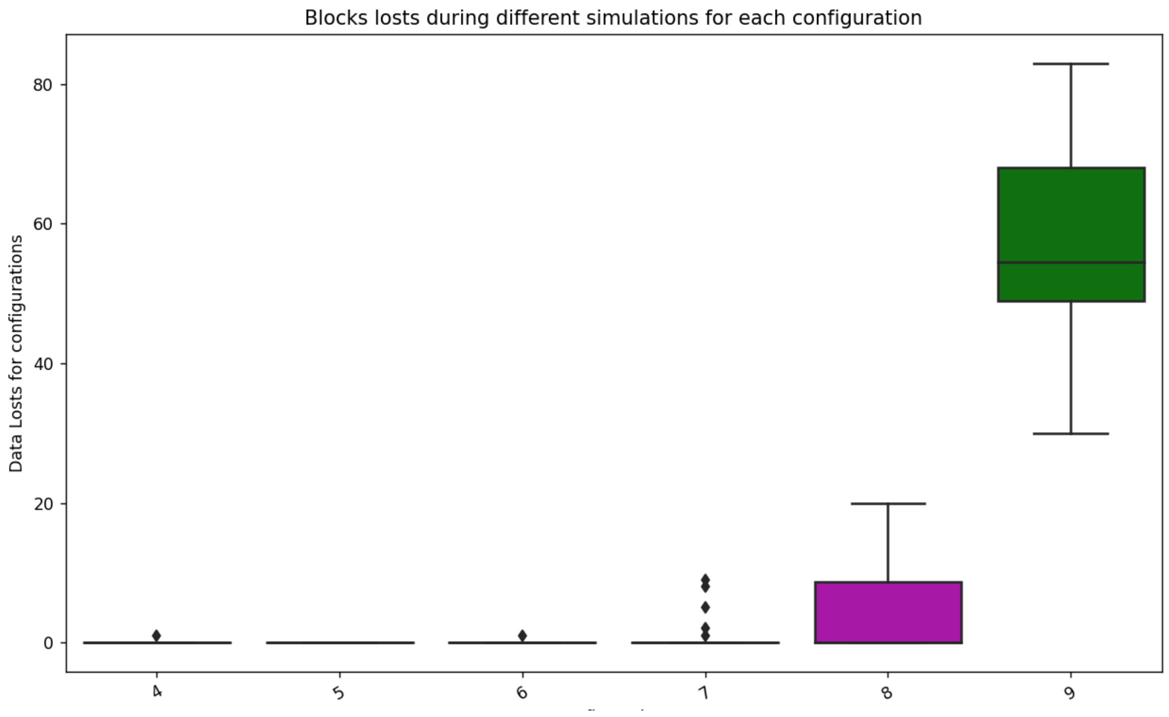
In the previous graph is reported a view of the lost blocks obtained at the end of each simulation obtaining the value running the code 100 times for each k value.

The best value to make sure in this configuration we don't lose blocks of data are those configurations with a k value less or equal than 5. As we said before we need always to consider the size of blocks but we need less blocks to recover the original data.

Obviously in this configuration, a higher value of k than 6 makes loss of blocks because we distribute the blocks round nodes that can fail and then we can't figure out how to recover the k number of blocks to recover the n data blocks.

Increasing number of nodes

So what's the problem in this case? Maybe the number of nodes? We want to prove that obviously increasing the number of nodes this system will perform better even if we increase the number of k because there are more nodes in the system that can be available when others have failed. So, we put in the following plots representing the impact of increasing the number of nodes.



We obtain this graph with 50 simulations for each k value.In this case we can see that we have an improvement in terms of k so improving the number of nodes makes us lose less blocks of data because there are more nodes that can be used when other nodes fail in order to increase the number of k that we can take in consideration decreasing the block size.

This happens also because before we had a number of peers equal to n blocks that we wanted to store in the p2p network so obviously increasing the number of nodes gave us a better way to store those data(we have more space in which to store blocks that needs to be backed up).

Using Simulation to obtain different system characteristics

In the following we present the table of different configurations that try to achieve the less loss of blocks following the previous tests searching the best configuration depending on system needs:

Configuration Name	Description	Configuration file

Cost-Efficient Configuration	This configuration is optimized for cost efficiency, meaning that it minimizes the storage space and network resources required while still maintaining a low data loss rate.
	
	To achieve this, the system has a moderate number of nodes, a moderate storage space, and moderate upload and download speeds.
	
	The average uptime and downtime are setted up to obtain a cost-efficient configuration with half uptime and half downtime in a day.
	
	We also increase the recovery time because we think that this can impact in terms of cost-efficiency.
		[peer]
		number = 15
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 15 GiB
		upload_speed = 3 MiB
		download_speed = 15 MiB
		average_uptime = 12 hours
		average_downtime = 12 hours
		average_lifetime = 3 years
		average_recover_time = 5 days
		arrival_time = 0

High Availability Configuration	This configuration is optimized for high availability, meaning that the data is less likely to be lost due to node failure.
	
	To achieve this, the system has a high number of nodes, a large storage space, and a high upload and download speed.
	
	In order to obtain a high availability configuration we think that we can increase the uptime and downtime values of the nodes in order to see what happens to the blocks and if this has an impact.
		[peer]
		number = 100
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 20 GiB
		upload_speed = 2 MiB
		download_speed = 10 MiB
		average_uptime = 23 hours
		average_downtime = 1 hour
		average_lifetime = 5 years
		average_recover_time = 1 day

Configuration Name	Description	Configuration file

High Availability Configuration n°2	In this configuration, like the previous one, we want to obtain a high availability.
	
	In this case we decrease the number of nodes increasing the upload and download speed to see what can change in a system like this with a high availability in terms of uptime and downtime of the nodes.
		[peer]
		number = 20
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 20 GiB
		upload_speed = 4 MiB
		download_speed = 20 MiB
		average_uptime = 23 hours
		average_downtime = 1 hour
		average_lifetime = 5 years
		average_recover_time = 1 day
		arrival_time = 0

High storage capacity configuration	This configuration allows for more data to be stored on the nodes.
	The number of nodes is lower compared to the previous configuration, but the storage capacity makes up for it.
	The upload and download speeds are lower, but the storage capacity ensures that the data can be stored even if the nodes are offline for a long period of time.
	Obviously we think that by increasing the storage size we can only reduce the loss of blocks because we guarantee enough space for each node.
		[peer]
		number = 50
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 100 GiB
		upload_speed = 1 MiB
		download_speed = 5 MiB
		average_uptime = 8 hours
		average_downtime = 16 hours
		average_lifetime = 1 year
		average_recover_time = 3 days
		arrival_time = 0

Configuration Name	Description	Configuration file

High data encoding configuration	This configuration has a high number of data encoding blocks, which increases the chances of having a sufficient number of blocks to reconstruct the data.
	
	The storage capacity is lower than the previous configurations, but the high number of data encoding blocks compensates for it.
	
	The upload and download speeds are also lower, but the high number of data encoding blocks ensures that the data can be reconstructed even if the nodes are offline for a long period of time.
		[peer]
		number = 50
		n = 20
		k = 10
		data_size = 1 GiB
		storage_size = 50 GiB
		upload_speed = 1 MiB
		download_speed = 5 MiB
		average_uptime = 8 hours
		average_downtime = 16 hours
		average_lifetime = 1 year
		average_recover_time = 20 days
		arrival_time = 0

Low Latency Configuration	This configuration is optimized for low latency, meaning that the data transfer speed is high.
	
	We want to figure out this to find if it results in faster backups and recoveries and how can this impact in terms of loss of blocks.
		[peer]
		number = 20
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 20 GiB
		upload_speed = 20 MiB
		download_speed = 20 MiB
		average_uptime = 23 hours
		average_downtime = 1 hour
		average_lifetime = 5 years
		average_recover_time = 1 day
		arrival_time = 0

Configuration Name	Description	Configuration file

Low Cost Configuration	This configuration is optimized for low cost, meaning that it uses the minimum amount of resources to achieve the desired level of data loss.
	
	To achieve this, the system has a low number of nodes(this is not suggested), a small storage space, and a low upload and download speed.
	
	The average recovery time is set to a long value to try to represent a low cost set up even in the recovery phase.
		[peer]
		number = 5
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 5 GiB
		upload_speed = 1 MiB
		download_speed = 5 MiB
		average_uptime = 8 hours
		average_downtime = 16 hours
		average_lifetime = 1 year
		average_recover_time = 30 days
		arrival_time = 0

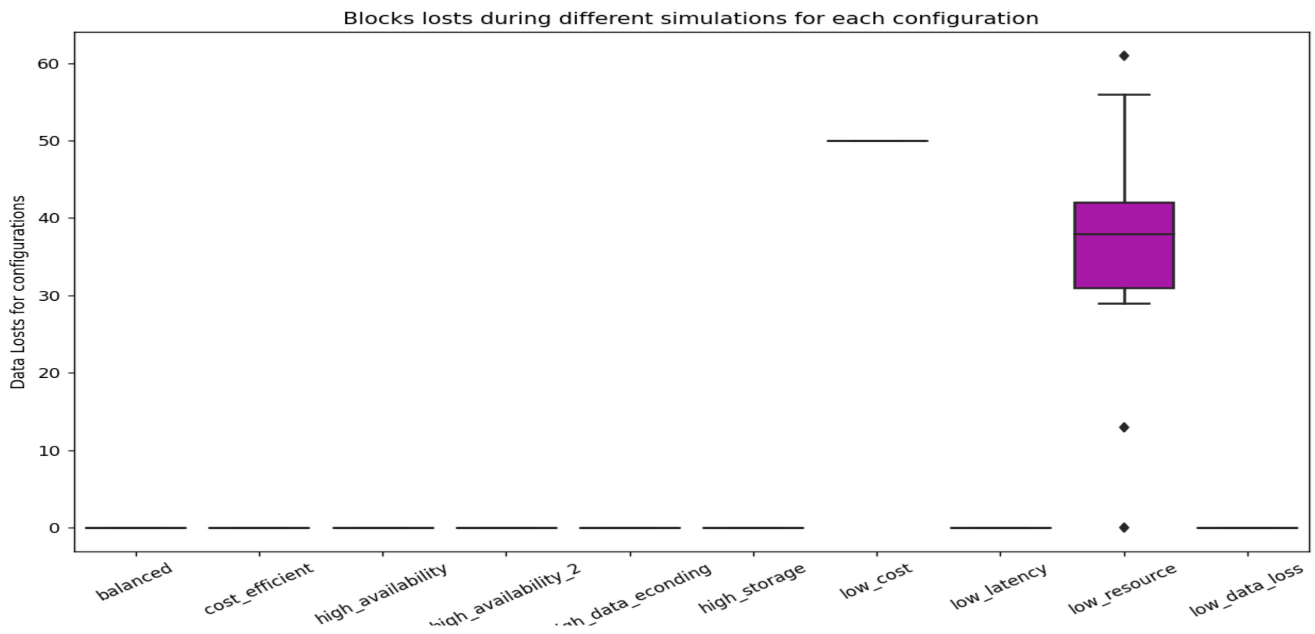
Balanced Configuration	This configuration is optimized for balance between data loss and resource usage but searching always to find out if this configuration will lose 0 blocks in different configurations utilizing a balanced set of parameters.	[peer]
		number = 15
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 15 GiB
		upload_speed = 3 MiB
		download_speed = 15 MiB
		average_uptime = 12 hours
		average_downtime = 12 hours
		average_lifetime = 3 years
		average_recover_time = 5 days
		arrival_time = 0

Configuration Name	Description	Configuration file

Low Resource Configuration	This configuration is optimized for low resource usage, meaning that it uses fewer resources than the other configurations.
	
	The system has a low number of nodes, a small storage space, and a low upload and download speed to represent the low resource configuration.
	
	Moreover, the average lifetime is set to a moderate value to understand what is the impact of this value that we think in a configuration with low resources could be high.
		[peer]
		number = 10
		n = 10
		k = 5
		data_size = 1 GiB
		storage_size = 10 GiB
		upload_speed = 2 MiB
		download_speed = 10 MiB
		average_uptime = 8 hours
		average_downtime = 16 hours
		average_lifetime = 1 year
		average_recover_time = 20 days
		arrival_time = 0

Low Data Loss Configuration	This configuration is optimized for low data loss, meaning that the system is designed to minimize the amount of data that is lost due to node failure or other factors.
	
	To achieve this, the system has a high number of nodes, a large storage space, and a high upload and download speed.
	
	The average lifetime is set to a high value to ensure that data is not lost due to node failure.
	
	The average recovery time is set to a short value to minimize the amount of data lost due to node failure.
	
	We prove if this system works well also with an higher value of k than the previous ones in order to obtain that the size of the blocks that are in the network is less then before
		[peer]
		number = 20
		n = 10
		k = 8
		data_size = 1 GiB
		storage_size = 20 GiB
		upload_speed = 4 MiB
		download_speed = 20 MiB
		average_uptime = 23 hours
		average_downtime = 1 hour
		average_lifetime = 5 years
		average_recover_time = 1 day
		arrival_time = 0

BOX PLOT of those CONFIGURATIONS taken "One by One"



This box plot is obtained by simulating 50 times for each configuration (more or less with 4 hours of execution).

The only two configurations that have been involved in losing blocks are the "low resource" and the "low cost" configuration.

If we analyze all the configuration parameters we can figure out that what causes us to lose blocks of data in the low cost configuration is the number of nodes less than the number of blocks that confirms our previous observation that is obviously a better choice to choose a number of n less than the number of nodes.

What makes lost data in the low resource configuration is the elevated recovery time that we put at a higher level because we thought that can impact the system and this confirms our suggestion.

All other configurations have only the difference from the default configuration the other parameters like upload/download speed increased or more storage size that are less relevant and has a limited impact in the system.

So we always obtain 0 loss of data in those configuration because we put the better k value to make those configuration to not lose data caused by a too elevated k value like we find out before in order to understand if the other parameters changes depending on needs to obtain system characteristics would impact in the data loss during several simulations.

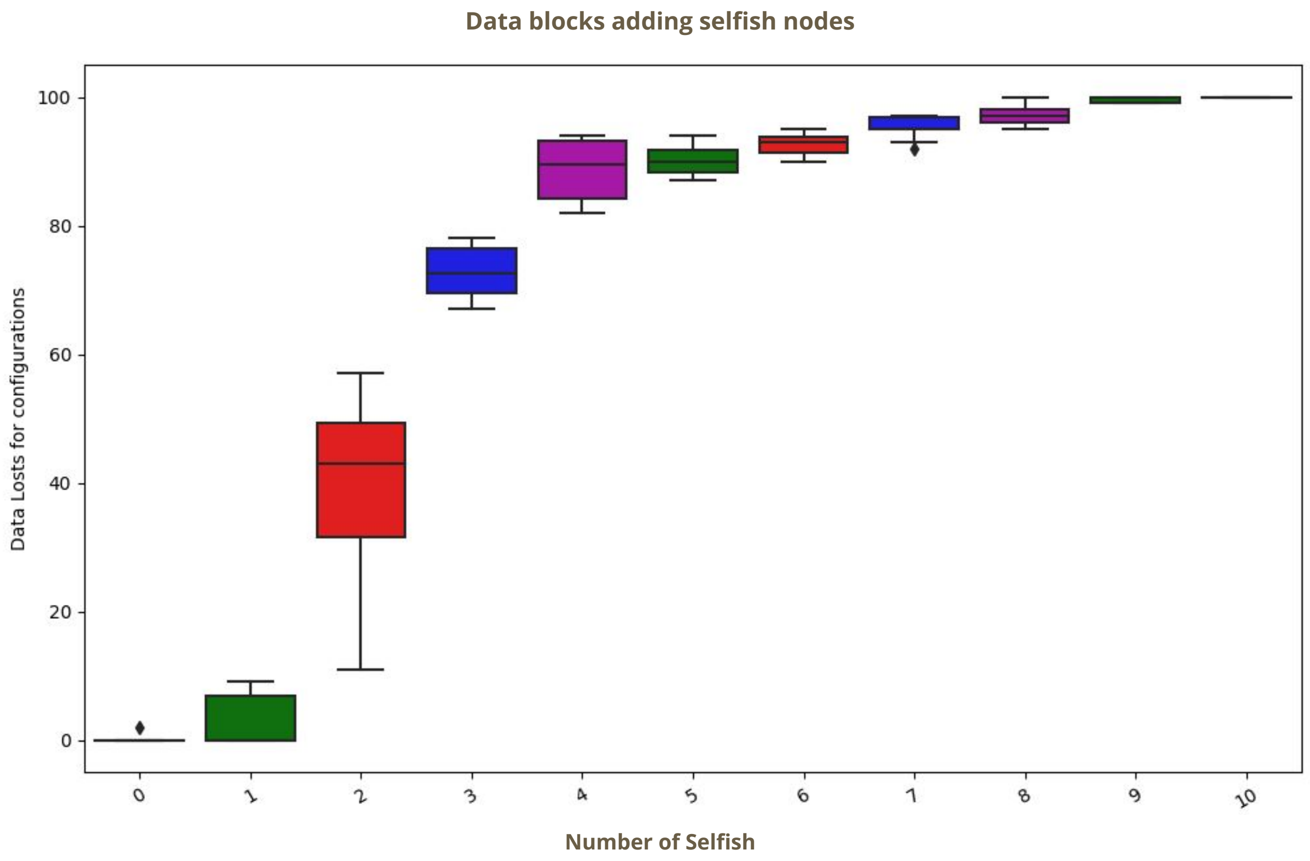
Extensions:

Introducing Selfish Nodes in the P2P system

The idea is to introduce some nodes that behave in a selfish way so keep their storage for only their blocks and do not backup the blocks of other nodes.

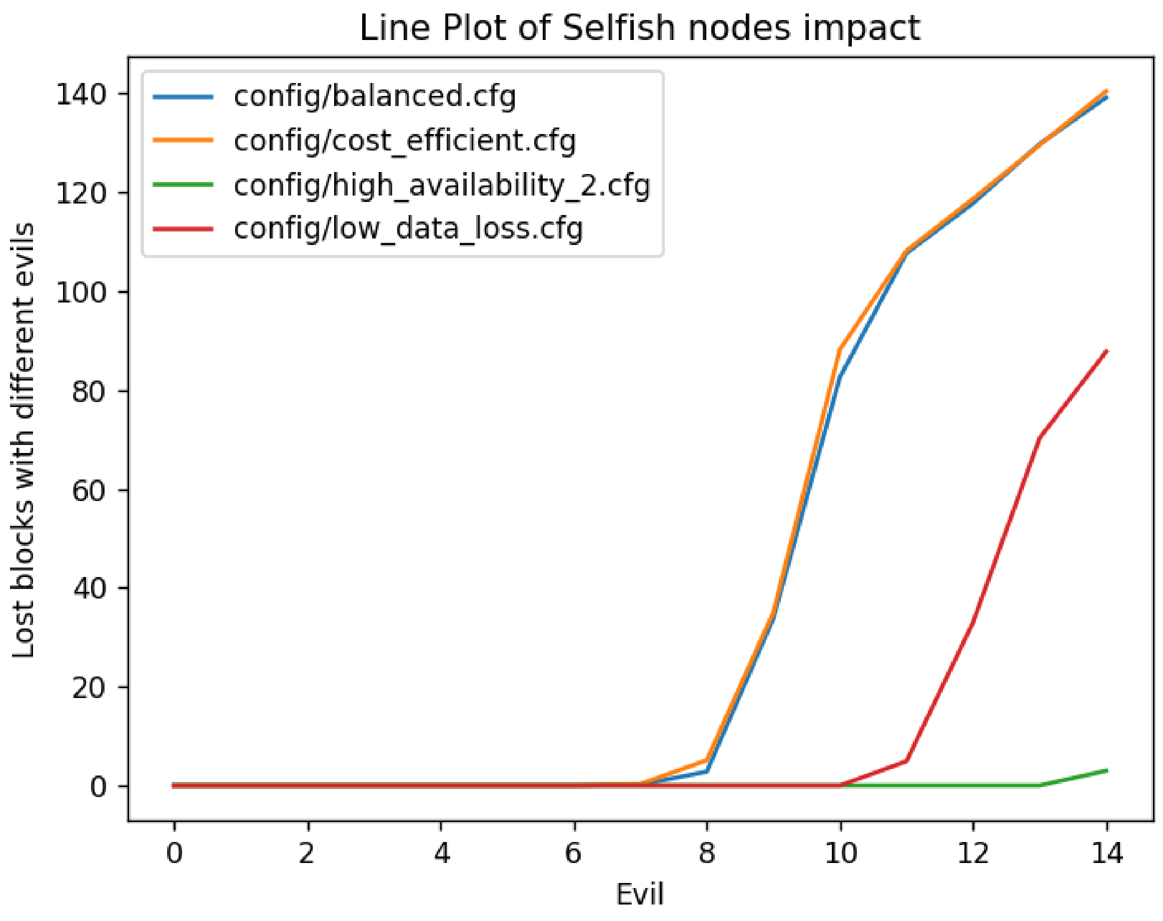
We want to see what happens when this type of node increases, In particular we want to see how the system behaves on optimal configurations. Before this we need to start verifying how impact the selfish nodes have in the basic configuration of the peer to peer system that we have reported when we find out the best values of k. In this case the value of k for that configuration is setted to 5.

We did 50 runs on each number of selfish nodes increasing this number to see how they corrupt the system. Let's see a boxplot that shows it better.



As expected we can see that as we increase selfish nodes we lose more blocks, as more nodes don’t store other’s data.

Now we try to run some of the optimal configurations that we have seen before, increasing selfish nodes.



We obtain this with the average of 50 runs for each evil number for each configuration (The execution lasted only 8 hours).

In this plot we need to observe that those configurations work well until half of the number of the nodes in the network are selfish so after this number the overall networks begin to lose blocks of data.

Note that those are configurations that we thought were the best depending on our needs so we can analyze the impact of selfish nodes also in other configurations.

In the specific case we can figure out that the high_availability_2 configuration is the best configuration that is better for selfish evil and does not lose blocks of data.

This is interesting since in the previous analysis we can't figure out this behavior of the low data loss configuration that in this case loses blocks of data before the high availability one.

So, in the following we introduce the tit for tat concept in order to think about how this concept can help us in a situation like this where selfish nodes "appear" in the simulations.

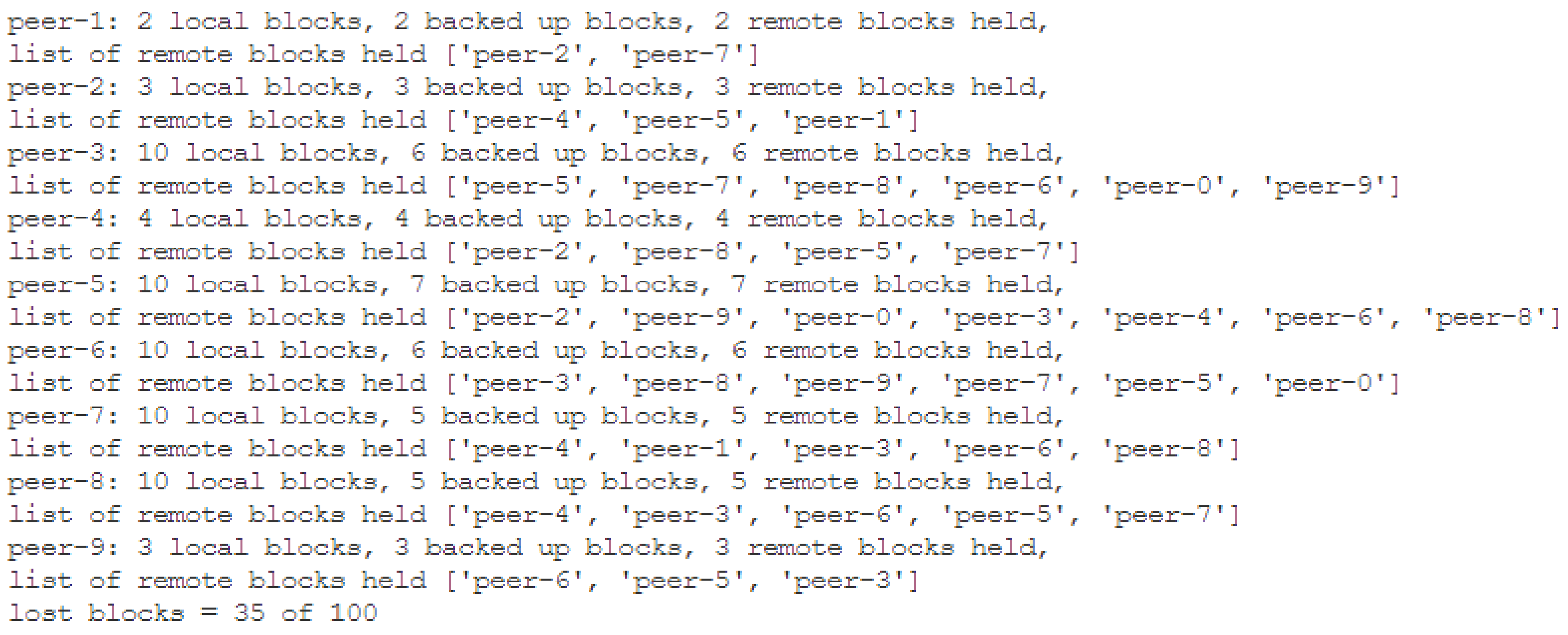
Implementing the Tit-for-Tat concept

Since we notice the problem that some selfish nodes can corrupt the system, we now want to incentivize cooperation using the tit for tat approach, such that to save their blocks, nodes will have to be honest.
Node x will only save node y's block if y has saved node x's block, so to do this first we run the simulation for a sufficiently long time to ensure that all peers have had enough time to interact with each other and exchange blocks, then, the tit for tat strategy come into play, where peers prioritize saving blocks of other peers who have saved them blocks in the past.
This helps to maintain a balanced distribution of blocks among the peers and ensures that no one peer has too many or too few blocks compared to the others.

We also notice that we are not doing the real tit for tat but we want to implement the “pan per focaccia” concept in our system and we discover that considering the optimistic unchoke is a way to obtain a sort of full simulation of the tit for tat behavior on the nodes in the system.

We started thinking from the bottom to grow up with some conclusion.

Tit For Tat without optimistic unchoke



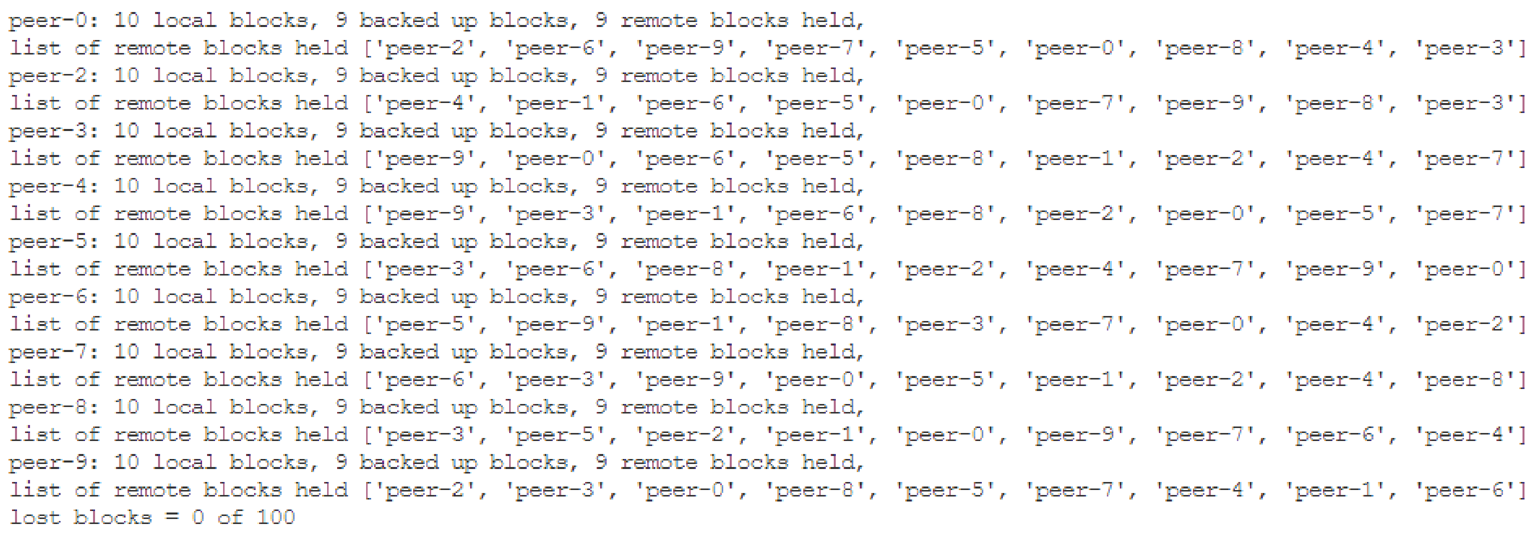
```
peer-1: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-2', 'peer-7']
peer-2: 3 local blocks, 3 backed up blocks, 3 remote blocks held,
list of remote blocks held ['peer-4', 'peer-5', 'peer-1']
peer-3: 10 local blocks, 6 backed up blocks, 6 remote blocks held,
list of remote blocks held ['peer-5', 'peer-7', 'peer-8', 'peer-6', 'peer-0', 'peer-9']
peer-4: 4 local blocks, 4 backed up blocks, 4 remote blocks held,
list of remote blocks held ['peer-2', 'peer-8', 'peer-5', 'peer-7']
peer-5: 10 local blocks, 7 backed up blocks, 7 remote blocks held,
list of remote blocks held ['peer-2', 'peer-9', 'peer-0', 'peer-3', 'peer-4', 'peer-6', 'peer-8']
peer-6: 10 local blocks, 6 backed up blocks, 6 remote blocks held,
list of remote blocks held ['peer-3', 'peer-8', 'peer-9', 'peer-7', 'peer-5', 'peer-0']
peer-7: 10 local blocks, 5 backed up blocks, 5 remote blocks held,
list of remote blocks held ['peer-4', 'peer-1', 'peer-3', 'peer-6', 'peer-8']
peer-8: 10 local blocks, 5 backed up blocks, 5 remote blocks held,
list of remote blocks held ['peer-4', 'peer-3', 'peer-6', 'peer-5', 'peer-7']
peer-9: 3 local blocks, 3 backed up blocks, 3 remote blocks held,
list of remote blocks held ['peer-6', 'peer-5', 'peer-3']
lost blocks = 35 of 100
```

So in this case we have started to make the nodes behave with the tit for tat behavior after 50 years of simulations so at half of the simulation.

This allows us to obtain like we reported above that nodes store only blocks for those nodes that are storing blocks for them and vice versa.
For instance if we see the first peer(peer-1) only stores 2 blocks that are owned respectively by the peer-2 and the peer-7; with the implementation of Tit for Tat in our system we obtain that peer-2 stores a block of peer-1 and peer-7 stores a block of peer-1.
This has a complication in our simulation that makes us lose blocks of data: since we have 50 years of simulation we can occur in node failures that make us lose all the data that we are storing on it.
In this way when this node returns online again any other node refuses any kind of “connection”/operation with that node because it doesn't behave like before because it has lost all the data so the other nodes can’t figure out its behavior like before.

Tit For Tat with optimistic unchoke

So, since we need a lot of time to make sure that the peers have had enough time to interact with each other, we thought about implementing Optimistic unchoke as well, so that if a peer doesn't find any node that has done backups for it, it chooses someone randomly.
In this way we are removing the concept of waiting a lot of years of simulation before starting behavioring using the tit for tat concept and making since the first second of the simulation that nodes behave like that.
In short, we add a check we do an upload only with nodes that makes upload to us (so for those nodes that I’m storing at least one block), if this is not possible, we select randomly a peer from the list of nodes in order to obtain a node that maybe can behave like me in the future(when that node make the optimistic unchoke on me).



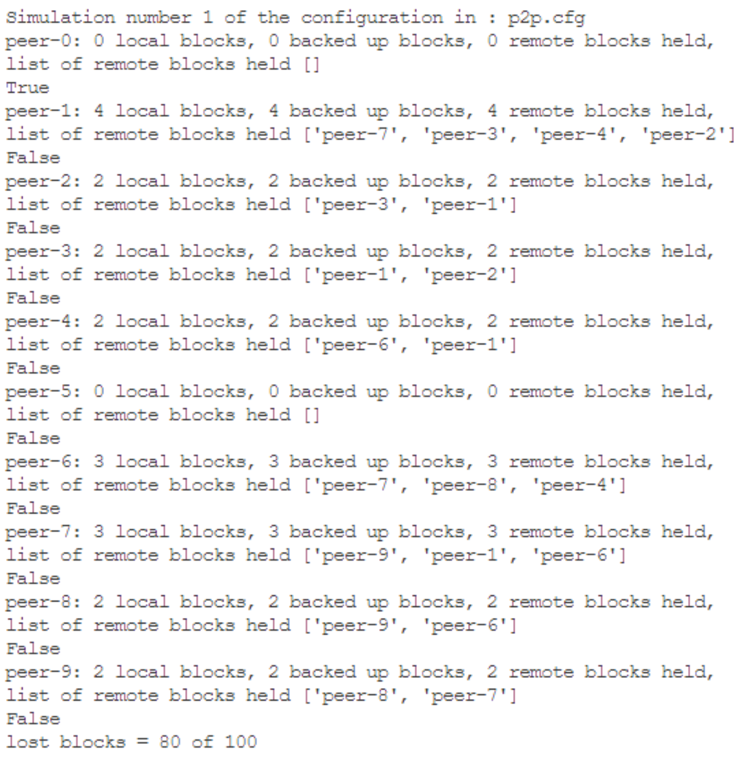
```
peer-0: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-2', 'peer-6', 'peer-9', 'peer-7', 'peer-5', 'peer-0', 'peer-8', 'peer-4', 'peer-3']
peer-2: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-4', 'peer-1', 'peer-6', 'peer-5', 'peer-0', 'peer-7', 'peer-9', 'peer-8', 'peer-3']
peer-3: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-9', 'peer-0', 'peer-6', 'peer-5', 'peer-8', 'peer-1', 'peer-2', 'peer-4', 'peer-7']
peer-4: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-9', 'peer-3', 'peer-1', 'peer-6', 'peer-8', 'peer-2', 'peer-0', 'peer-5', 'peer-7']
peer-5: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-3', 'peer-6', 'peer-8', 'peer-1', 'peer-2', 'peer-4', 'peer-7', 'peer-9', 'peer-0']
peer-6: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-5', 'peer-9', 'peer-1', 'peer-8', 'peer-3', 'peer-7', 'peer-0', 'peer-4', 'peer-2']
peer-7: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-6', 'peer-3', 'peer-9', 'peer-0', 'peer-5', 'peer-1', 'peer-2', 'peer-4', 'peer-8']
peer-8: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-3', 'peer-5', 'peer-2', 'peer-1', 'peer-0', 'peer-9', 'peer-7', 'peer-6', 'peer-4']
peer-9: 10 local blocks, 9 backed up blocks, 9 remote blocks held,
list of remote blocks held ['peer-2', 'peer-3', 'peer-0', 'peer-8', 'peer-5', 'peer-7', 'peer-4', 'peer-1', 'peer-6']
lost blocks = 0 of 100
```

Now, as we can see, we are back in the case that every node works correctly and makes note lose data. Obtaining that when the nodes can behave as the other nodes do with him, otherwise choose a node that optimistically may behave like you.

Tit For Tat without optimistic unchoke considering selfish nodes

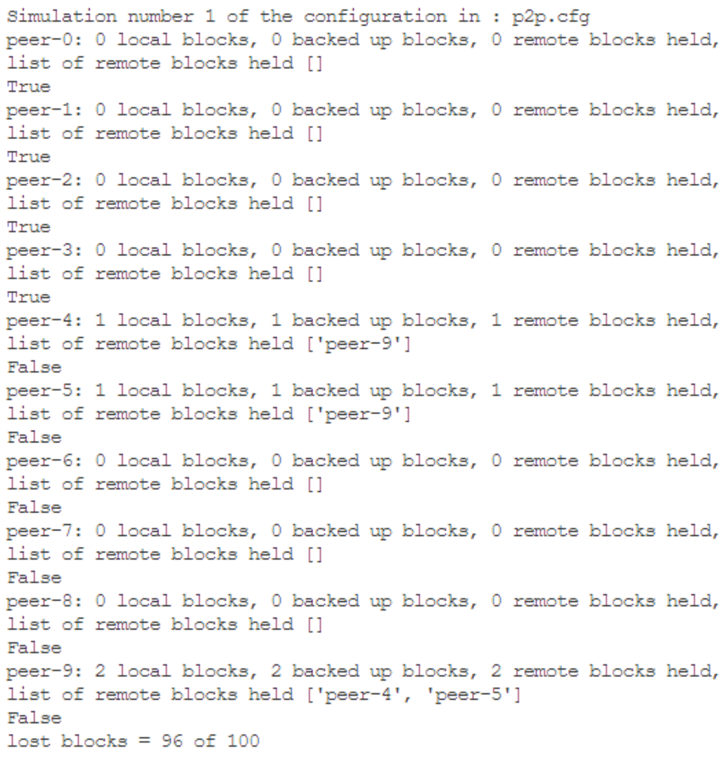
Considering now to introduce the selfish nodes without the optimistic unchoke to obtain what happens with this introduction in the system:

1 only selfish nodes about 10 nodes:



```
Simulation number 1 of the configuration in : p2p.cfg
peer-0: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
True
peer-1: 4 local blocks, 4 backed up blocks, 4 remote blocks held,
list of remote blocks held ['peer-7', 'peer-3', 'peer-4', 'peer-2']
False
peer-2: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-3', 'peer-1']
False
peer-3: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-1', 'peer-2']
False
peer-4: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-6', 'peer-1']
False
peer-5: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
False
peer-6: 3 local blocks, 3 backed up blocks, 3 remote blocks held,
list of remote blocks held ['peer-7', 'peer-8', 'peer-4']
False
peer-7: 3 local blocks, 3 backed up blocks, 3 remote blocks held,
list of remote blocks held ['peer-9', 'peer-1', 'peer-6']
False
peer-8: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-9', 'peer-6']
False
peer-9: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-8', 'peer-7']
False
lost blocks = 80 of 100
```

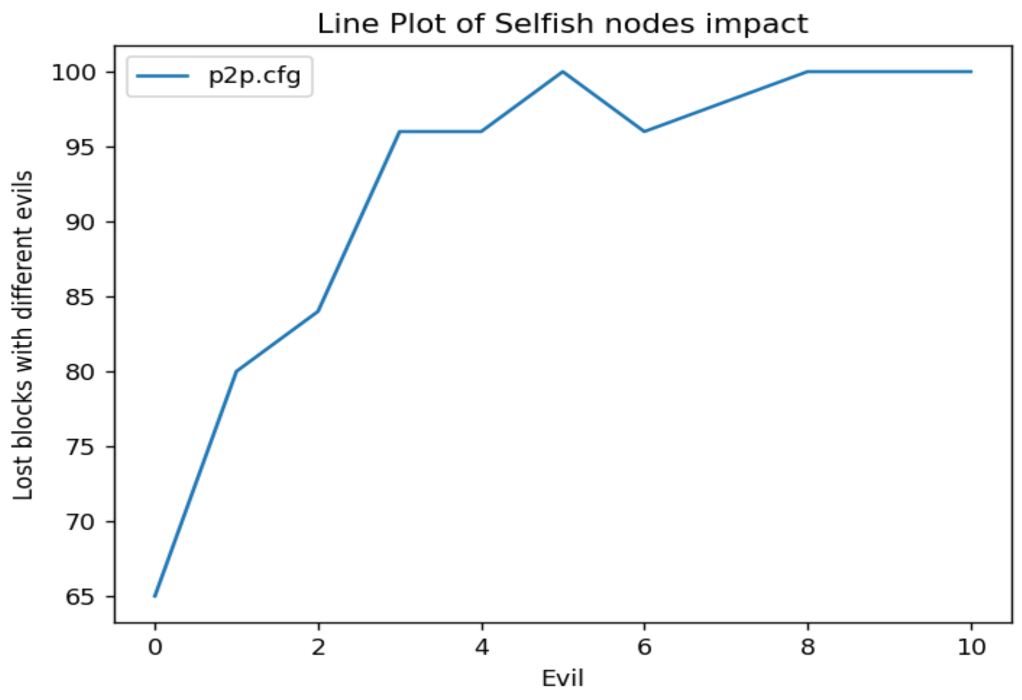
3 selfish nodes about 10 nodes:



```
Simulation number 1 of the configuration in : p2p.cfg
peer-0: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
True
peer-1: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
True
peer-2: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
True
peer-3: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
True
peer-4: 1 local blocks, 1 backed up blocks, 1 remote blocks held,
list of remote blocks held ['peer-9']
False
peer-5: 1 local blocks, 1 backed up blocks, 1 remote blocks held,
list of remote blocks held ['peer-9']
False
peer-6: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
False
peer-7: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
False
peer-8: 0 local blocks, 0 backed up blocks, 0 remote blocks held,
list of remote blocks held []
False
peer-9: 2 local blocks, 2 backed up blocks, 2 remote blocks held,
list of remote blocks held ['peer-4', 'peer-5']
False
lost blocks = 96 of 100
```

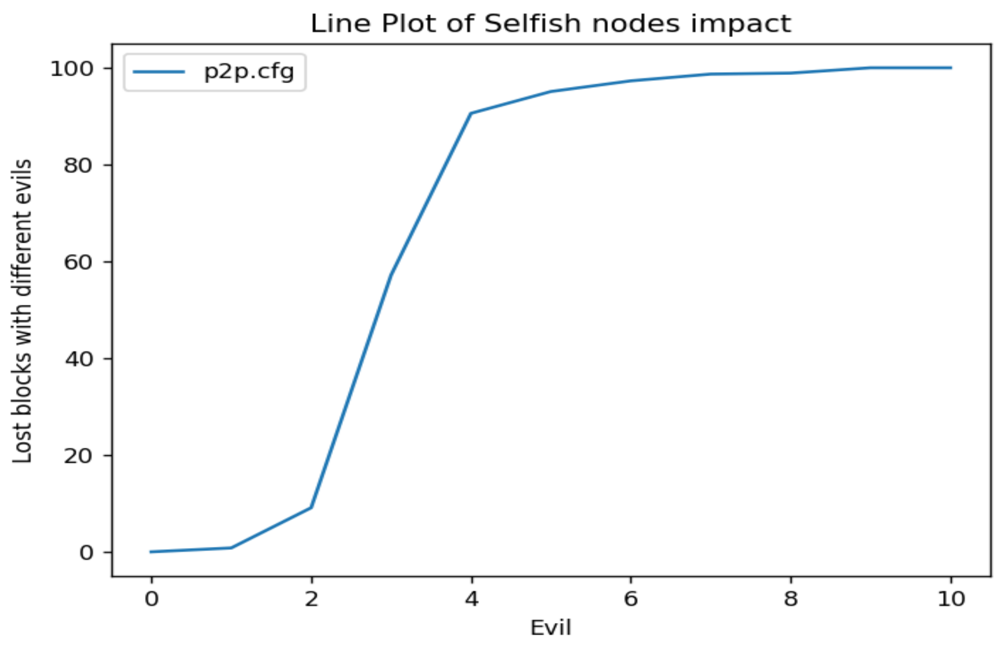
So with the increasing of the selfish nodes obviously we can figure out that we are increasing also the number of blocks that can be lost during the simulation because we are back in the case that we are not considering the optimistic unchoke so the non selfish nodes doesn't do anything with the selfish nodes because doesn't behaviors like them and this makes us to lose more data than before.

This is also reported in this following plot which represents the increasing loss of blocks of data when increasing the number of selfish nodes in the default peer to peer configuration.



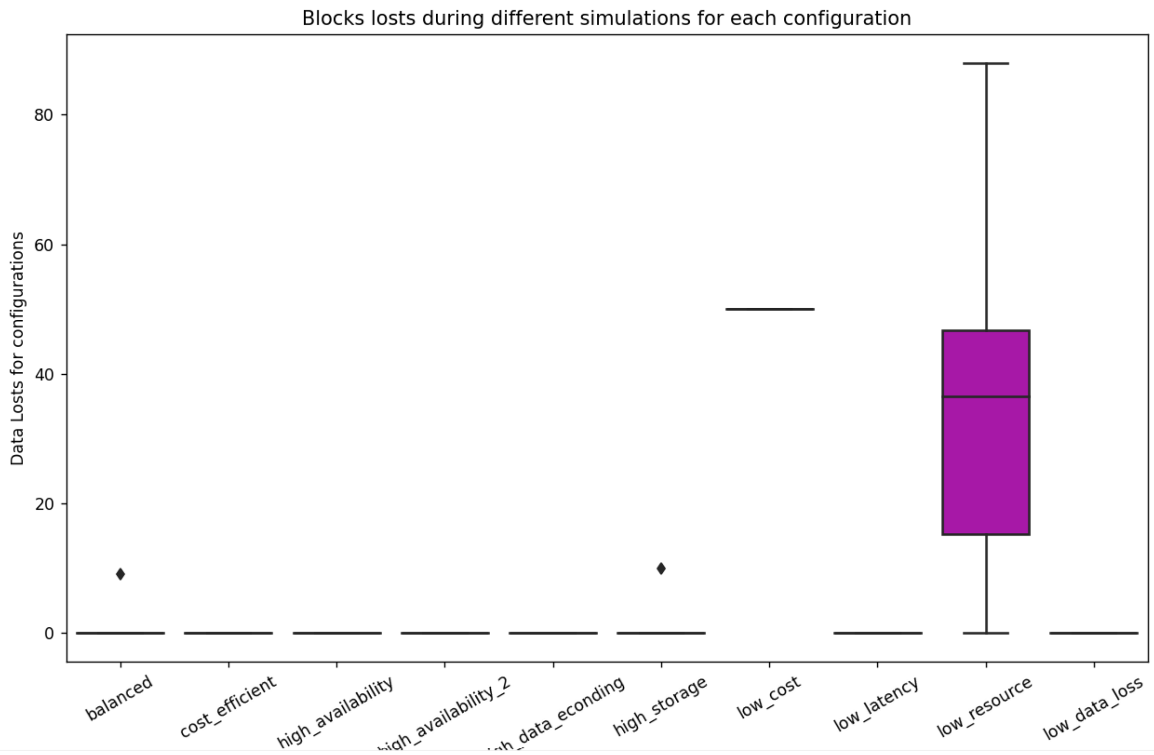
Tit For Tat with optimistic unchoke considering selfish nodes

We can now obviously see that with the tit for tat implementation using the optimistic unchoke on this configuration we go back to the situation without the implementation of tit for tat and optimistic unchoke with less loss of data but with the same behavior as before.



In this plot it is easy to view the improvement of the previous plot because until we put 4 selfish nodes in the total of 10 nodes we lose less data then before.
For instance if we have 2 selfish nodes we lose only 10 blocks instead of the 50 blocks that we had lost before. Moreover this represents an improvement instead of not using the optimistic choke like reported in the first box plot of the extensions part of the report.

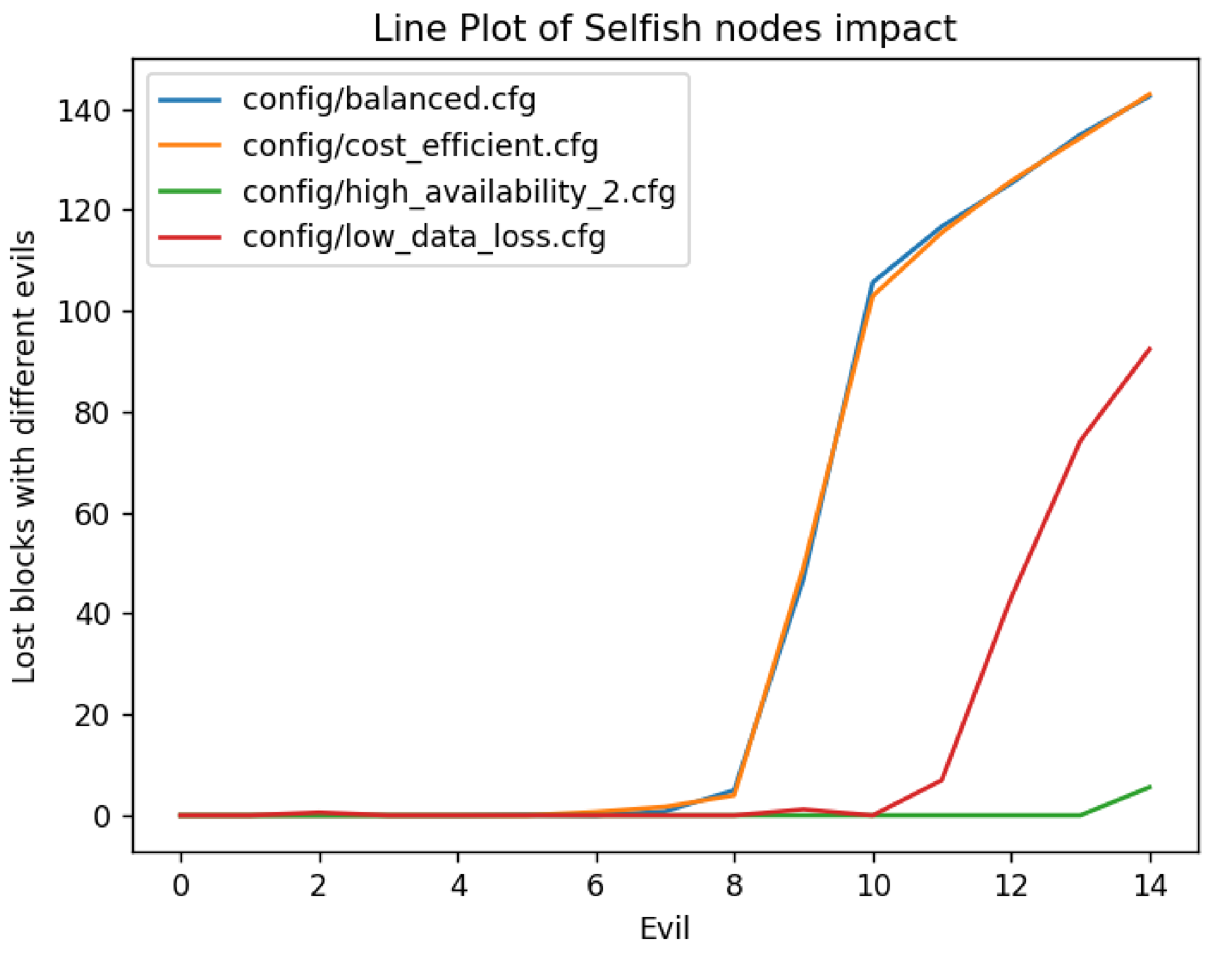
Tit For Tat with optimistic unchoke on "best configuration"



In this case we came back to the behavior that we notice when the tit for tat was not implemented in order to demonstrate that this implementation of tit for tat works like the normal one.

Tit For Tat with optimistic unchoke considering selfish nodes on "best configuration"

So, after presenting all these concepts we can merge it all and apply them in order to obtain if we apply this system behavior at the best configuration that we discover previously in this report we obtain an improvement of performance in terms of losing blocks.



In this plot we can figure out that this configuration resists in a good way at the selfish nodes in the system but this is not related to the behavior that the nodes have because we obtain a similar plot than the second one reported in this extension session in this report. This is because we only provide a way to improve uploads, not to make those nodes no more selfish during the simulation.

In conclusion we found it really interesting to improve this and view all the changes of those behaviors.