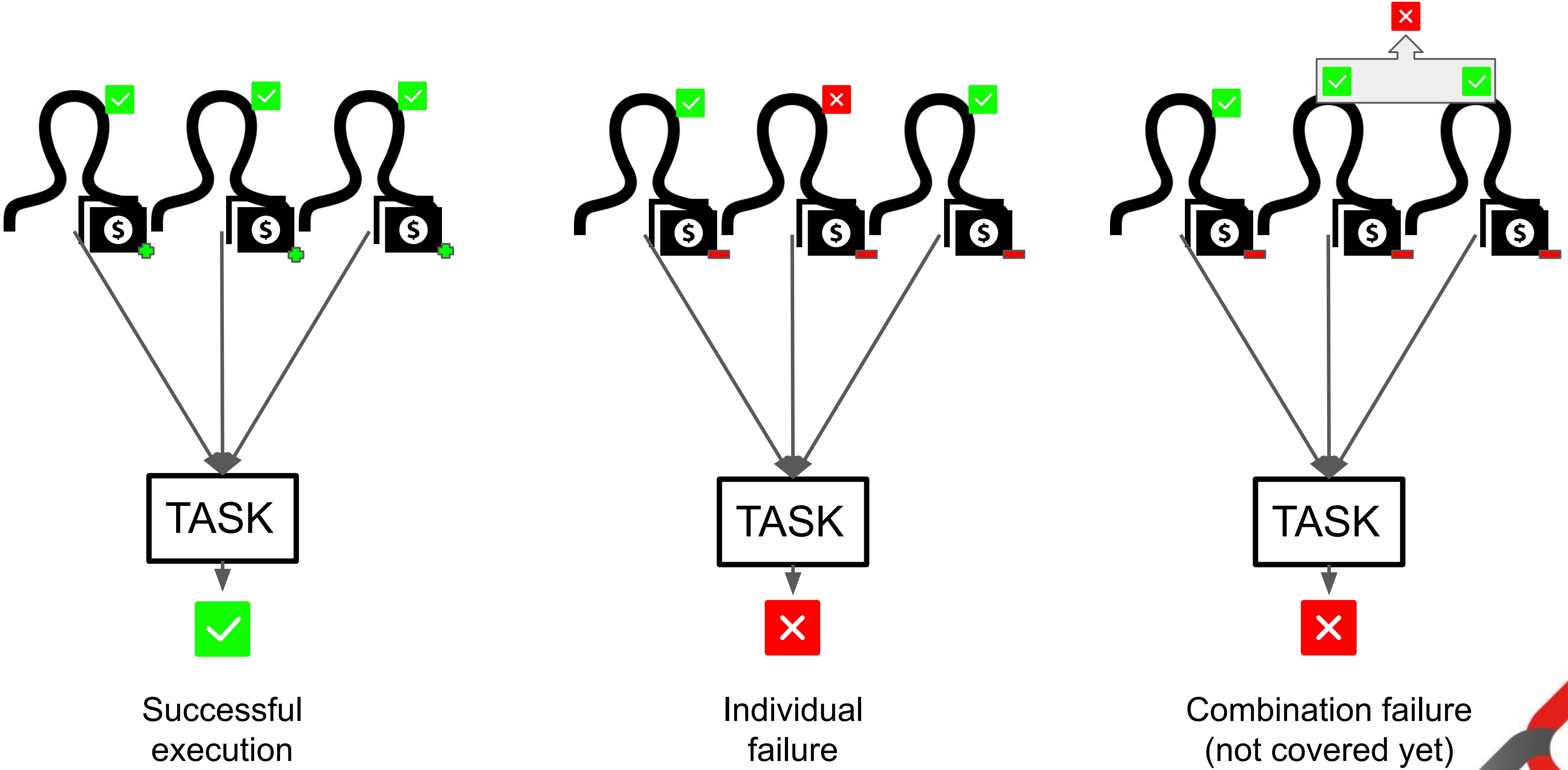
Blind Slashing Mechanism Simulation Strategy

26/08/2024

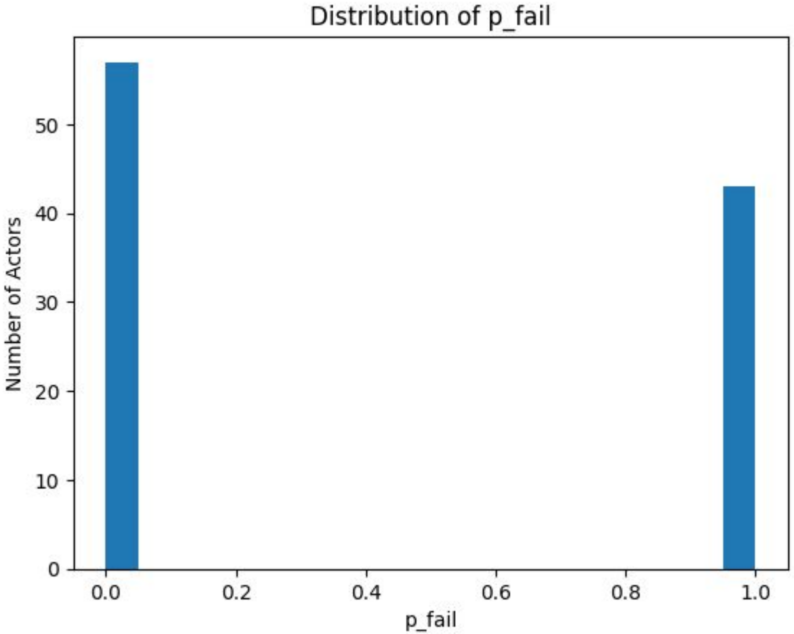
Henry MONT
Intern – DRIM

Blind Slashing: Context

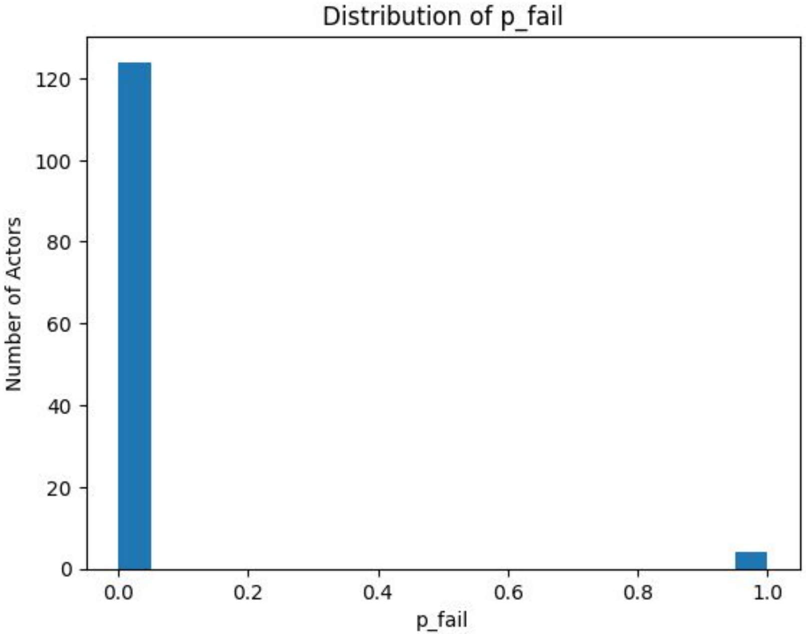


Scenarios to simulate:
The good

Binary behavior and single actor fault

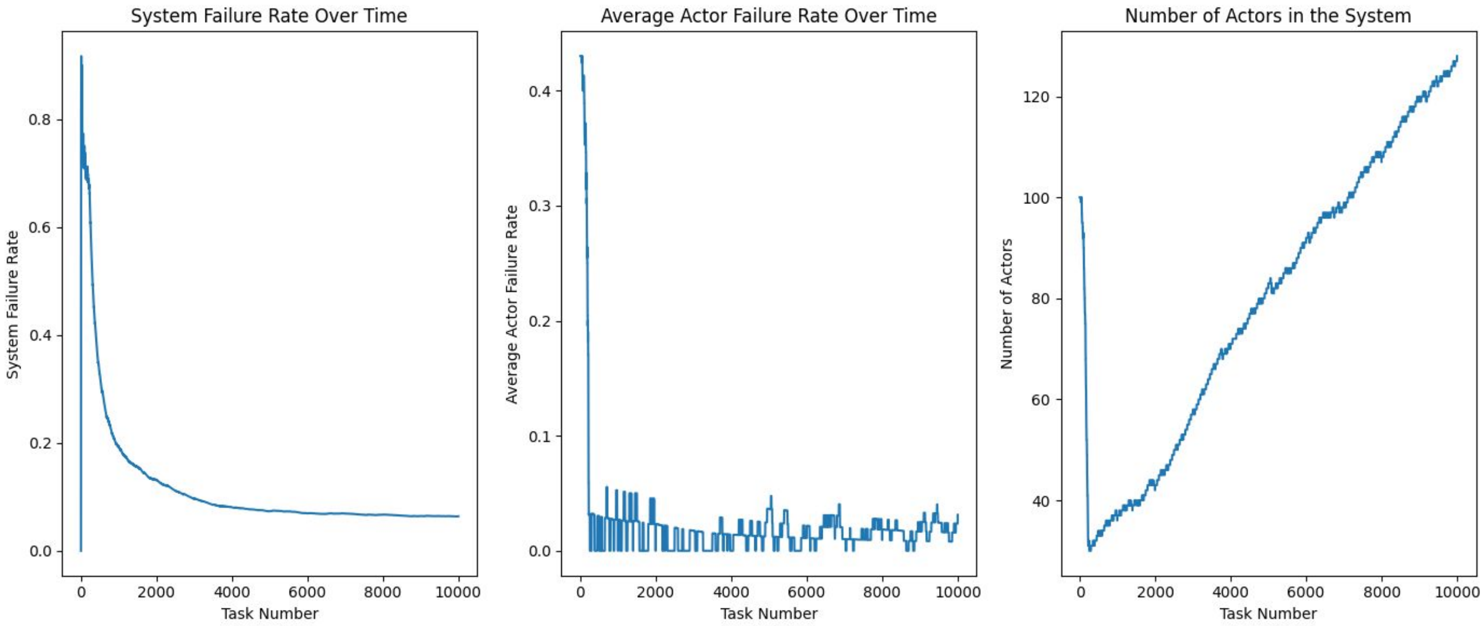


Before



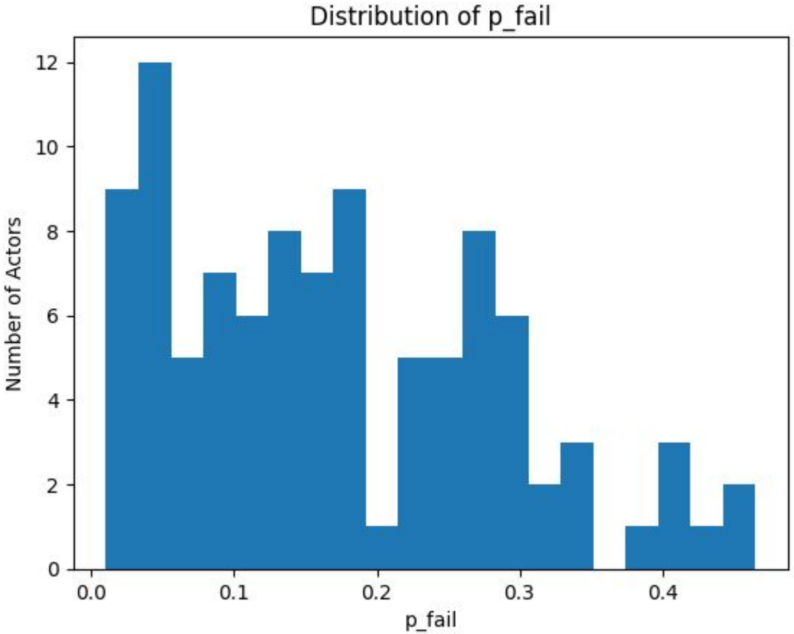
After

Binary behavior and single actor fault

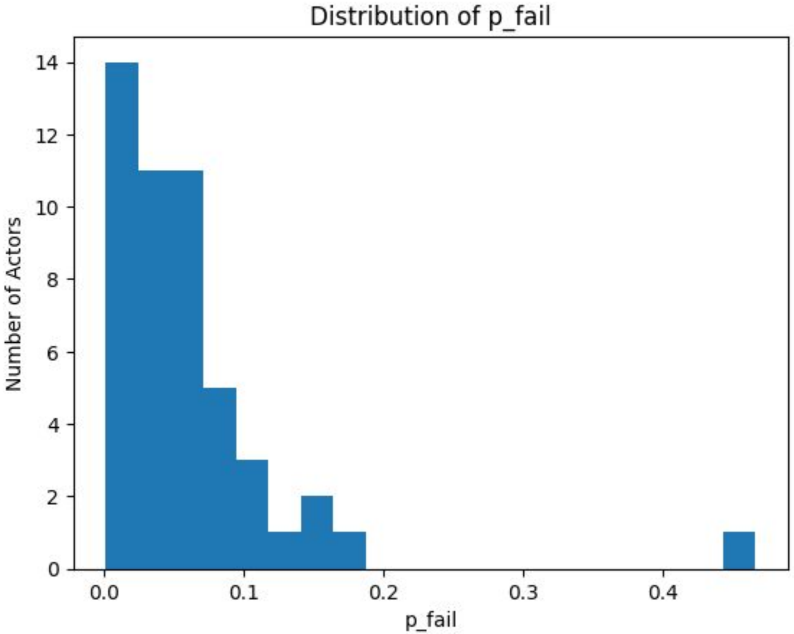


Total successful tasks: 9361	<u>Classification Metrics:</u>
Total failed tasks: 639	Accuracy: 0.8866666666666667
Total number of actors: 300	Precision: 0.8255813953488372
Final number of actors: 128	Recall: 0.9726027397260274

Continuous behavior and single actor fault

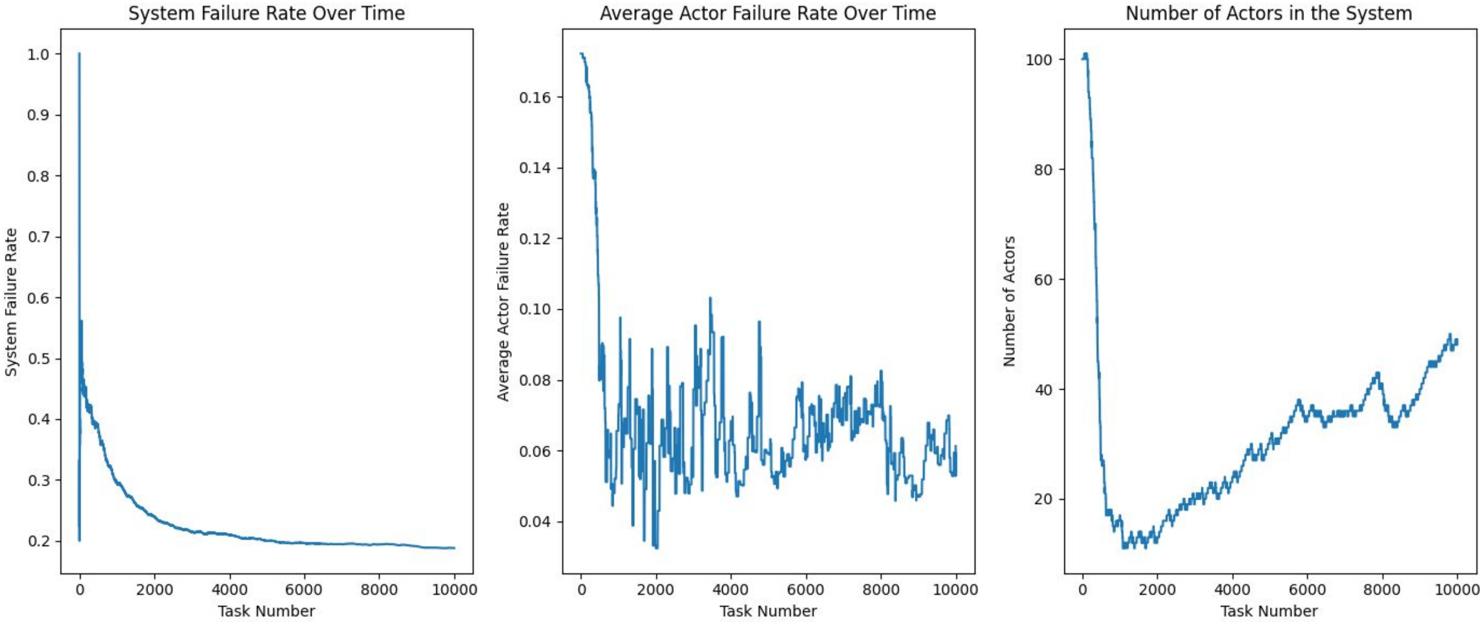


Before



After

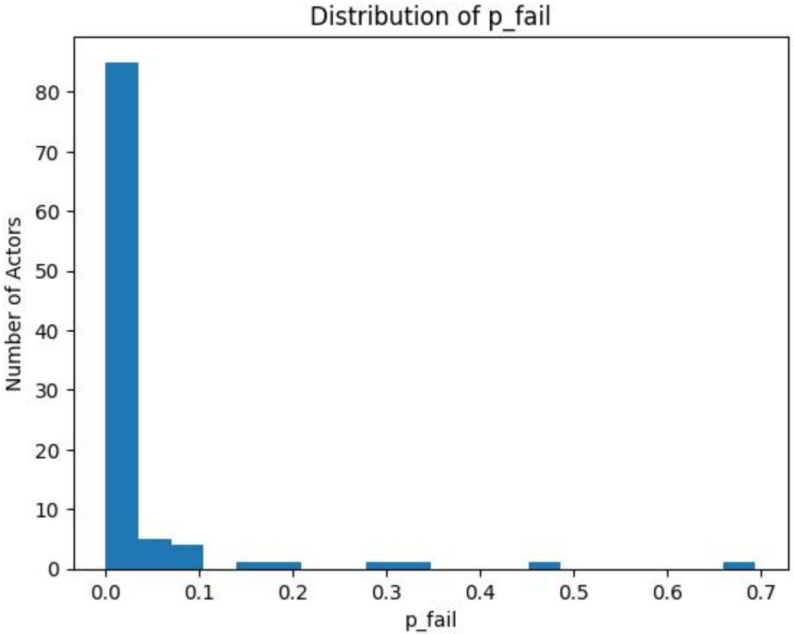
Continuous behavior and single actor fault



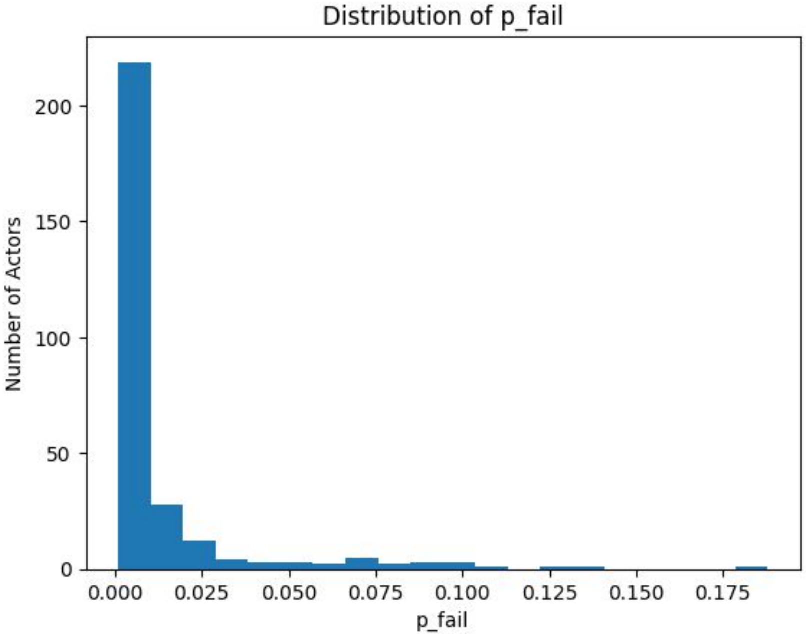
Total successful tasks: 8119	<u>Classification Metrics:</u>
Total failed tasks: 1881	Accuracy: 0.8733333333333333
Total number of actors: 300	Precision: 0.9282868525896414
Final number of actors: 49	Recall: 0.9209486166007905

Scenarios to simulate:
The bad

Mostly reliable system

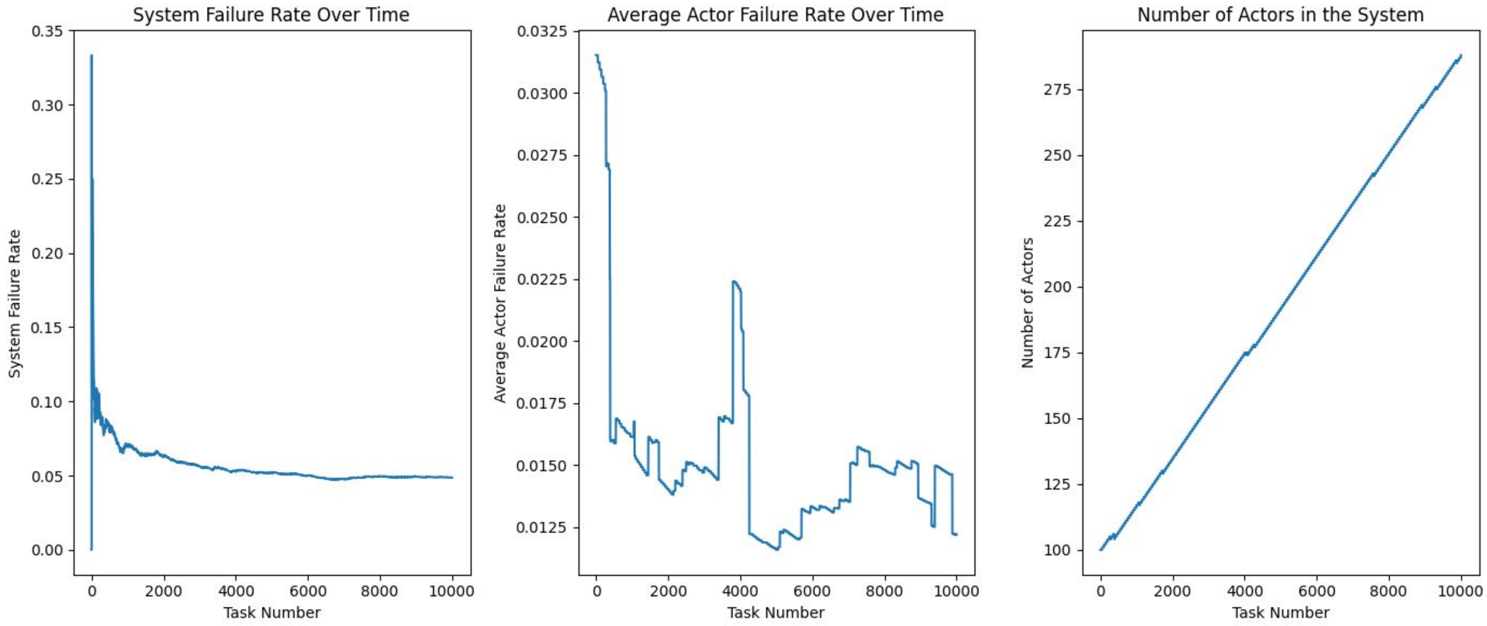


Before



After

Mostly reliable system

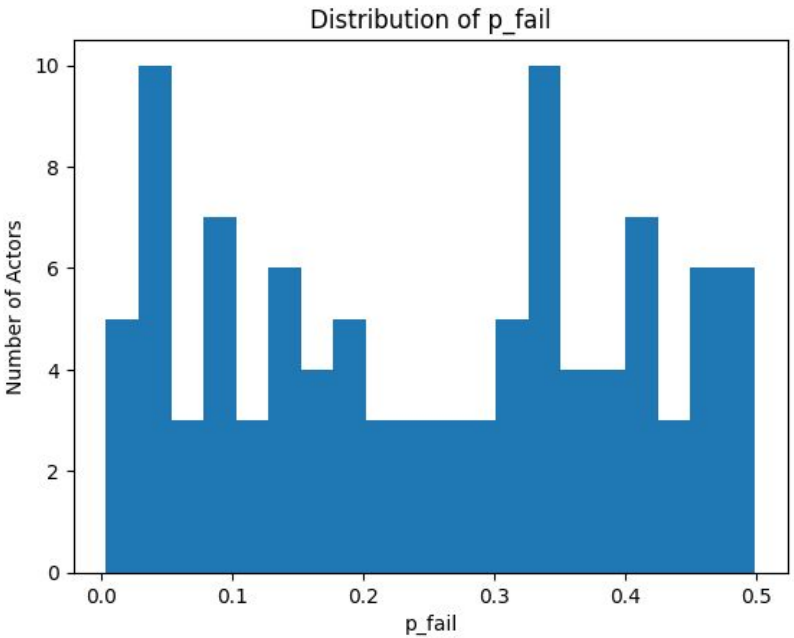


Total successful tasks: 9513
Total failed tasks: 487
Total number of actors: 300
Final number of actors: 288

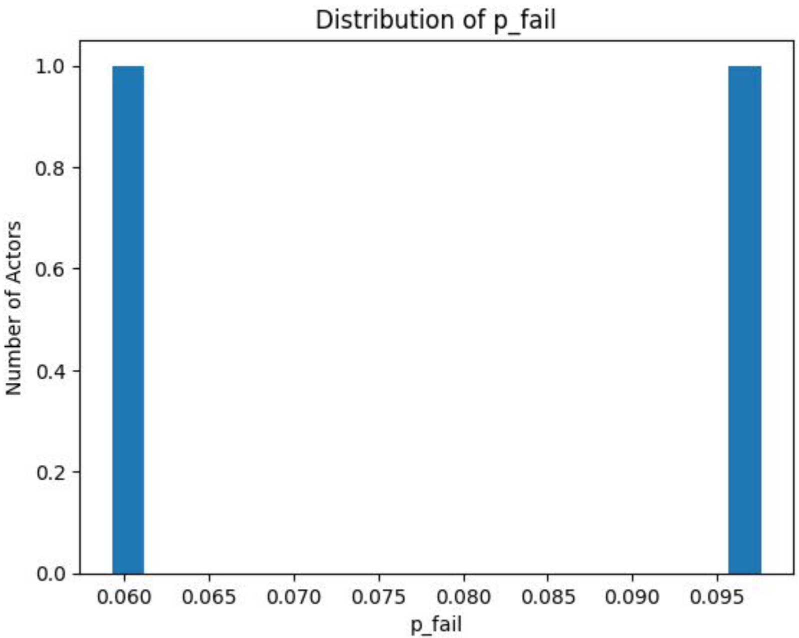
Classification Metrics:
Accuracy: 0.9366666666666666
Precision: 1.0
Recall: 0.3870967741935484

⇒ Struggling to eliminate unreliable actors when majority of reliable actors.

Overwhelmingly unreliable system

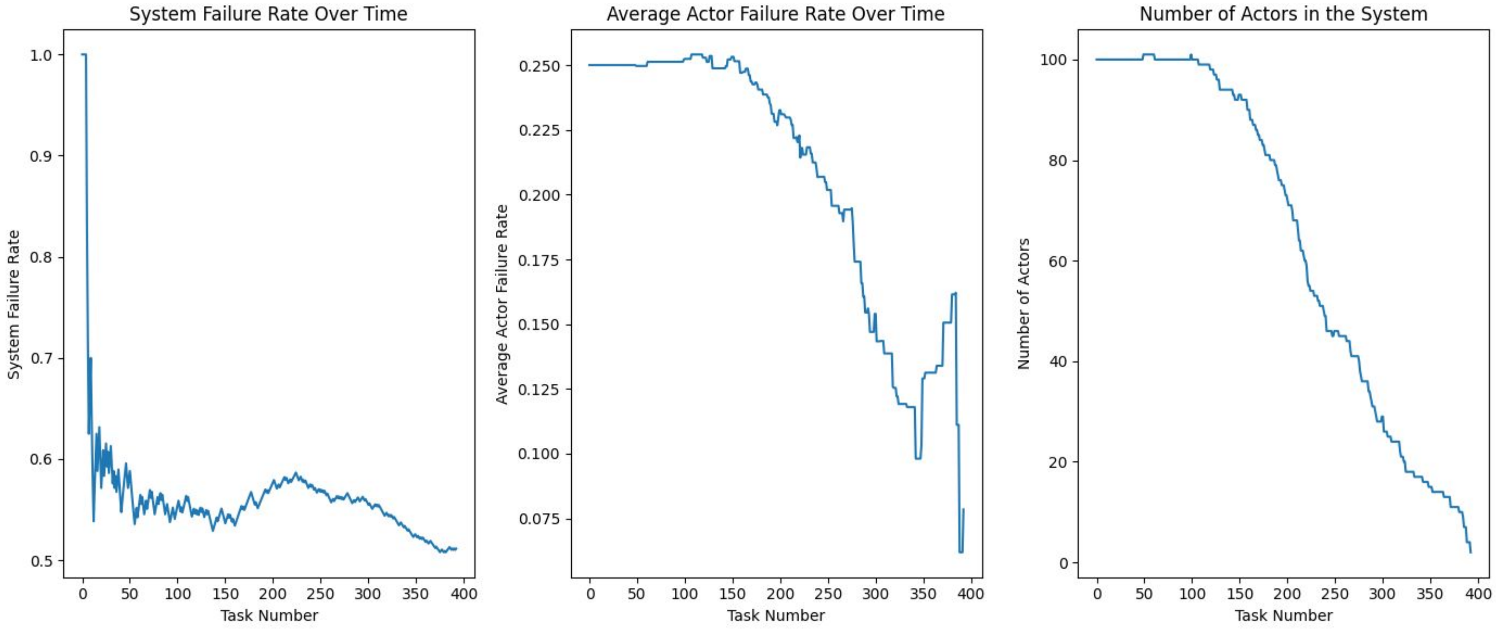


Before



After

Overwhelmingly unreliable system



Total successful tasks: 192	<u>Classification Metrics:</u>
Total failed tasks: 201	Accuracy: 0.3
Total number of actors: 300	Precision: 0.8571428571428571
Final number of actors: 2	Recall: 0.9782608695652174

⇒ In this situation, with reasonably higher proportion of unreliable actors, we over classified actors as unreliable and ruined almost everyone.

Early observations:

The good points:

- We are able to clean up a system containing a fair share of bad actors pretty efficiently in some situations. However, I need to find out what is the tipping point between a stable system and a ruin-all system.
- With some initial distributions of failure rates, even if we do not ruin everyone, we end up with terrible precision and accuracy. As if the system was blind firing and ruining random people. It might be due to a too high proportion of unreliable actors, this need to be investigated.

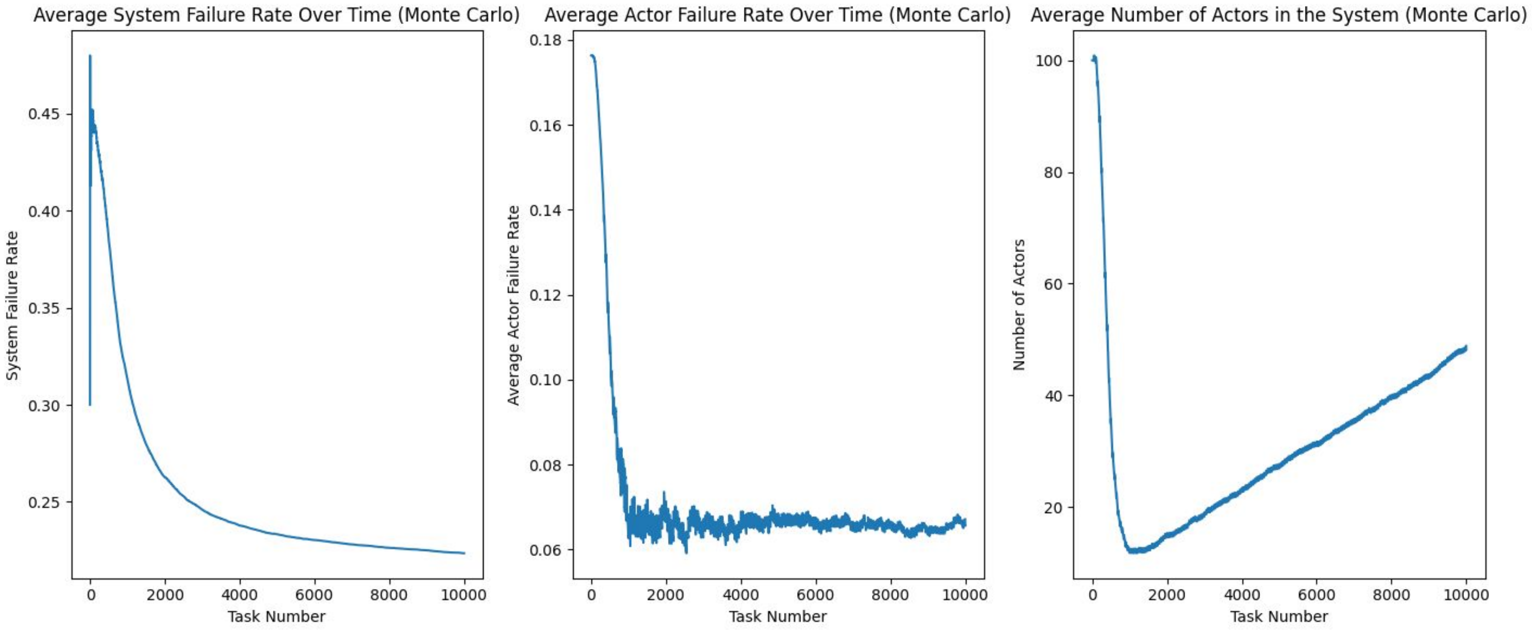
The bad points:

- A lot of actors are being ousted from the system at the beginning, sometimes it ends up with everyone's ruin, sometimes the number of actors build back up. I need to understand this dynamic.
- Everyone getting ruined might be due to the small initial pool of actors with no high enough reliability actor being represented from the start on some simulations. I will try simulations with bigger pools of actors.
- I am struggling to see the impact of slashing and rewarding on final system reliability. I need further simulations to find out whether it is behaving as expected.

⇒ Next step: use Monte Carlo simulation to achieve more consistent behaviors.

Monte-Carlo simulations

Monte Carlo experiment

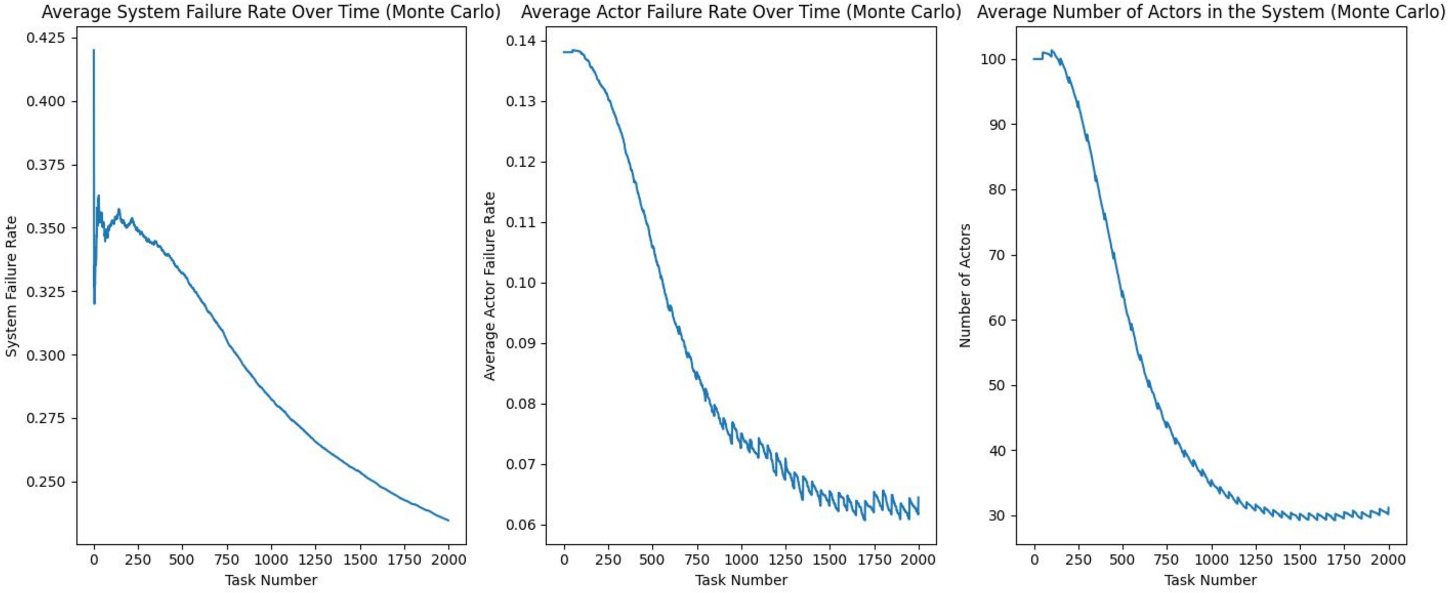


Total successful tasks: 6586.66
Total failed tasks: 1569.36
Total number of actors: 300
Final number of actors: 48.84
Ruin-all simulations: 10/50

<u>Classification Metrics:</u>
Precision: 0.9007557270558828
Accuracy: 0.7465333333333333
Recall: 0.9186016740379572

⇒ Better overview of the behaviour.

Monte Carlo experiment



Total successful tasks: 1530.68
Total failed tasks: 469.32
Total number of actors: 140
Final number of actors: 31.14
Uncompleted simulations: 0/50

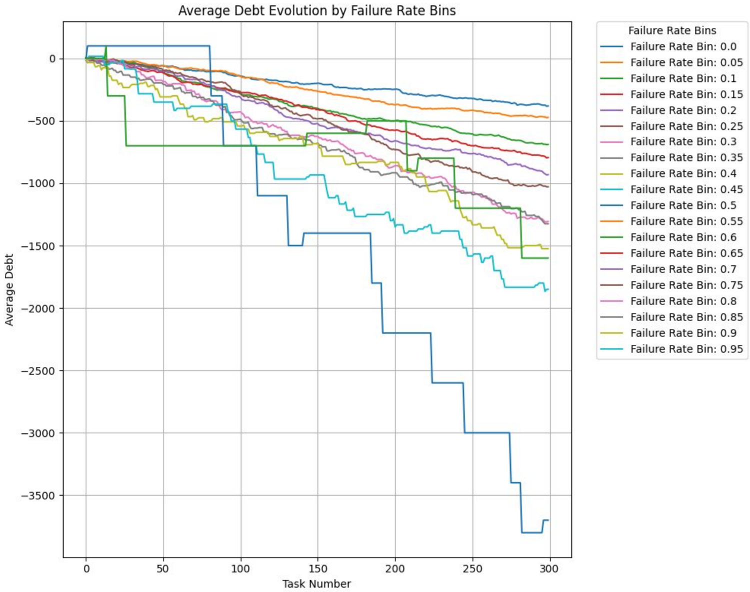
<u>Classification Metrics:</u>
Precision: 0.8735346115154263
Accuracy: 0.8014285714285714
Recall: 0.8701863589079584

⇒ Trying to find at which point there is not simulation that leads to everyone’s ruin. However still very different behaviour depending on initial distribution. I need to define several distribution metrics and compare the results depending on those metrics.

Average reward and loss per task

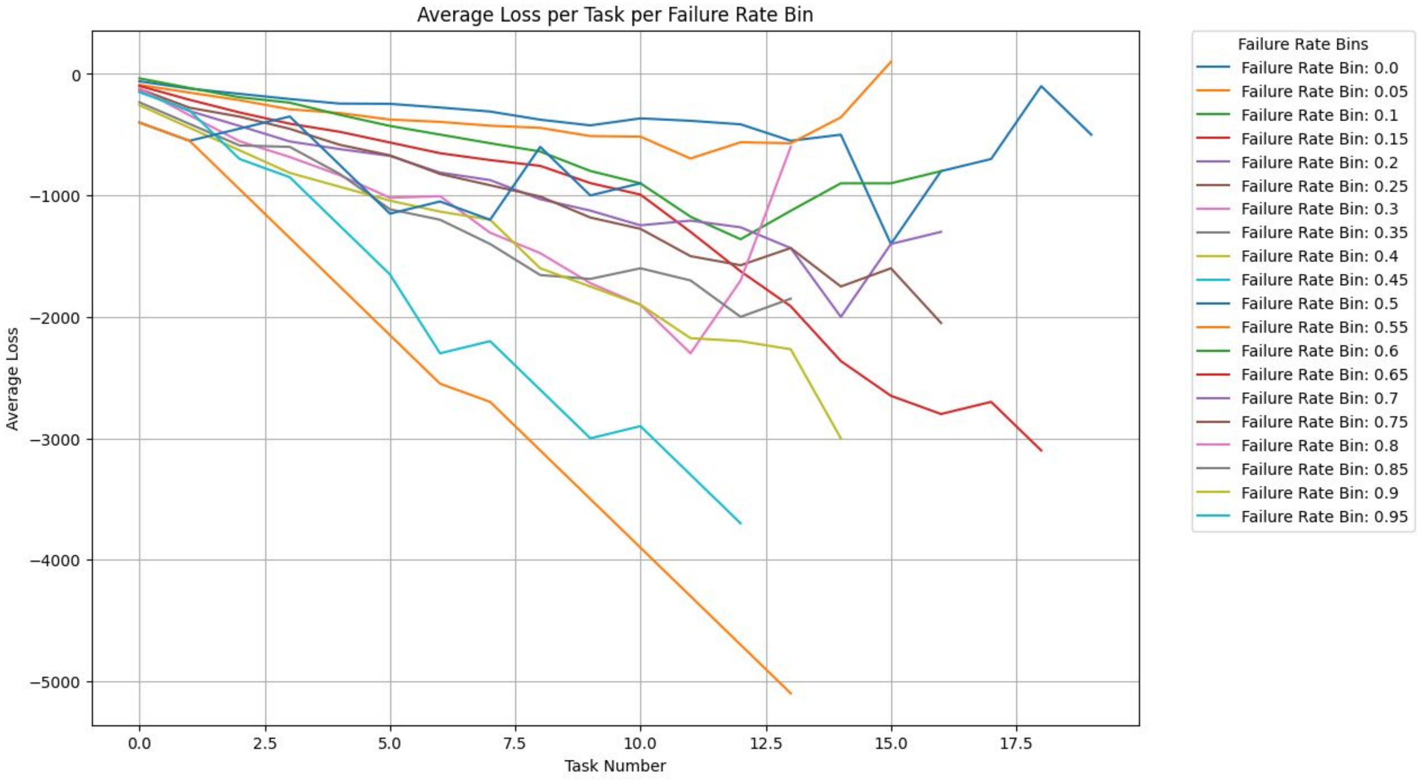
⇒ In this system, it appears that no one is profitable at first. However, I believe that reliable actors typically start becoming profitable after the initial removal of unreliable ones. During the first phase of the simulation, their average loss increases. The goal is to see the reward and loss curves converge.

Average debt evolution in a static system



⇒ In a system where we don’t remove or add actors, the slashing and rewarding would result in the following accumulation of debt. Over 300 tasks in total (no per actor), actors with the highest failure rate are losing more money. This correspond to the initial situation in the previous slide, where even reliable actors end up losing money.

Last work



Average Loss per Task per Failure Rate Bin
0
−1000
−2000
−3000
−4000
−5000
Average Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Task Number
Failure Rate Bins
Failure Rate Bin: 0.0
Failure Rate Bin: 0.05
Failure Rate Bin: 0.1
Failure Rate Bin: 0.15
Failure Rate Bin: 0.2
Failure Rate Bin: 0.25
Failure Rate Bin: 0.3
Failure Rate Bin: 0.35
Failure Rate Bin: 0.4
Failure Rate Bin: 0.45
Failure Rate Bin: 0.5
Failure Rate Bin: 0.55
Failure Rate Bin: 0.6
Failure Rate Bin: 0.65
Failure Rate Bin: 0.7
Failure Rate Bin: 0.75
Failure Rate Bin: 0.8
Failure Rate Bin: 0.85
Failure Rate Bin: 0.9
Failure Rate Bin: 0.95