

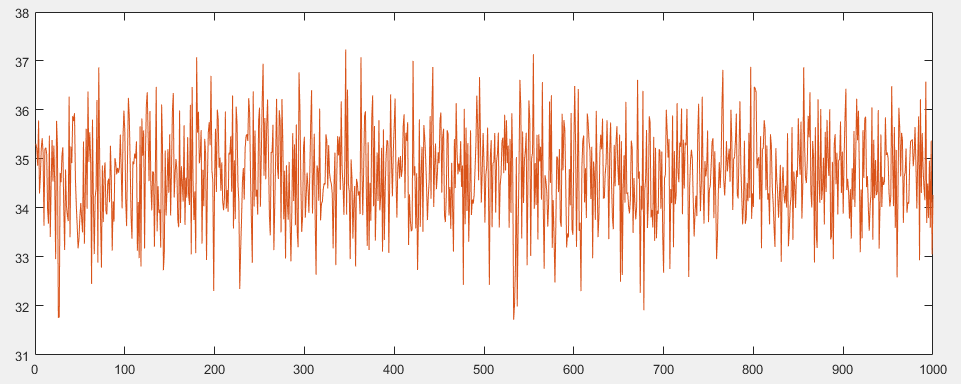
In RL, before the first time frame is processed, a random configuration is selected. Now, we start processing. In the first request of the first time frame, a request is generated with Exponential Inter-arrival rate. Now, for the basic service time we generate a random number according to Uniform Distribution (between 80,100), and calculate the latency time for this request. For each of the next 4 requests (we have a total of 5 requests in a time frame), we randomly generate them with exponential inter-arrival rate and assume random service time (uniformly distributed between 80,100). Because all the basic service time are randomly distributed, and the configuration selected initially has some impact on the average latency for this current time frame, we expect the average latency to be representative of the configuration.

For the next time frame, some configuration is randomly selected and again requests are generated using exponential inter-arrival rates and are processed with basic service time equaling some random number belonging to same Uniform distribution (80,100). The selected configuration affects the parameter , ulThreadParaIndex, which brings about difference in different configurations selected. The cost of this time frame is obtained, and corresponding values of reward function and probability function are updated for all configurations.

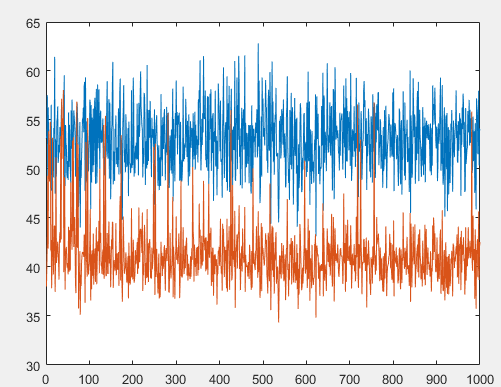
As we process more time frames, we tend to select among a few configuration that produces less average latency.

For SC, we select a certain configuration based on profiling and then run all time frames on that configuration. Since each requests have different random basic service time, the average service time for each time frame seems to be highly fluctuating.

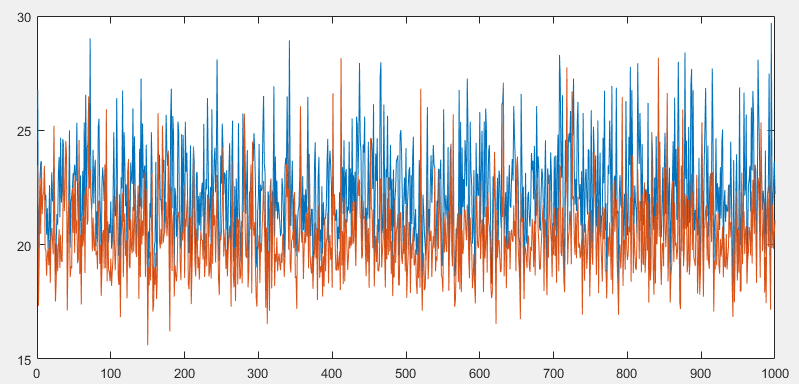
As we can see, RL performs better than SC in this case.



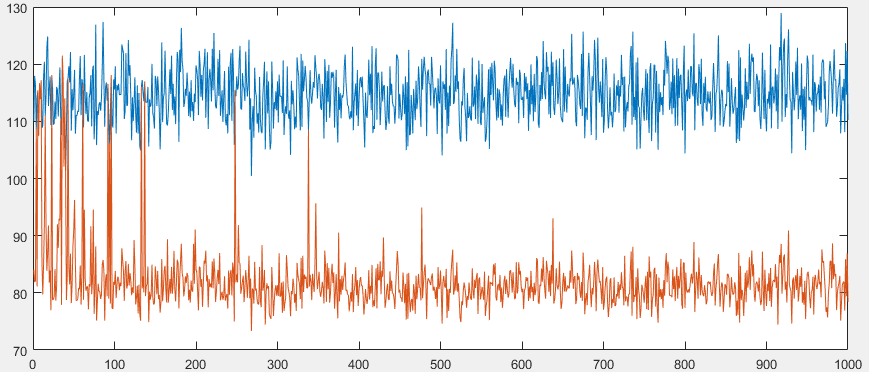
This is when the inter-arrival rates is Erlang distributed (arrival,2). It shows that both RL and SC performs same in this setup.



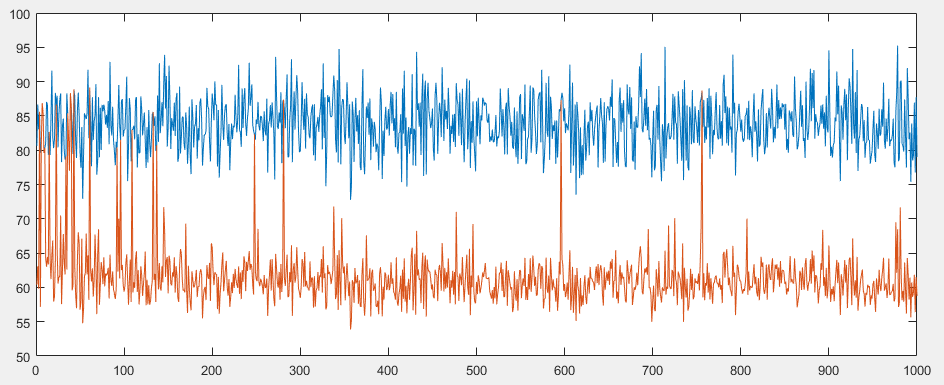
Setting: when ServiceTime=Erlang(100,1).



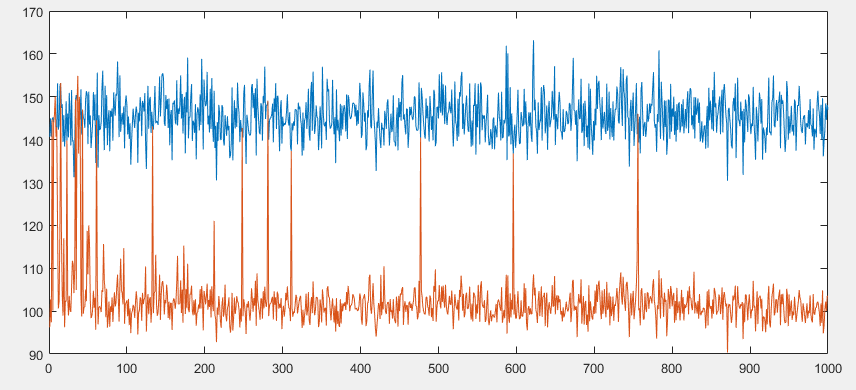
Setting: When serviceTime=Erlang(50,1).



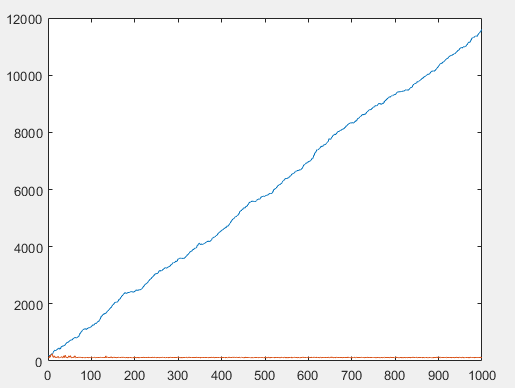
Setting: When serviceTime=Erlang(200,1)



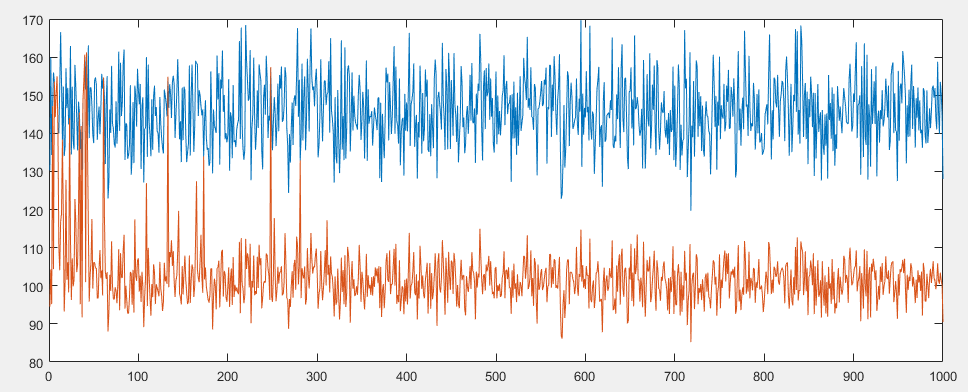
Setting: When serviceTime=Erlang(150,1)



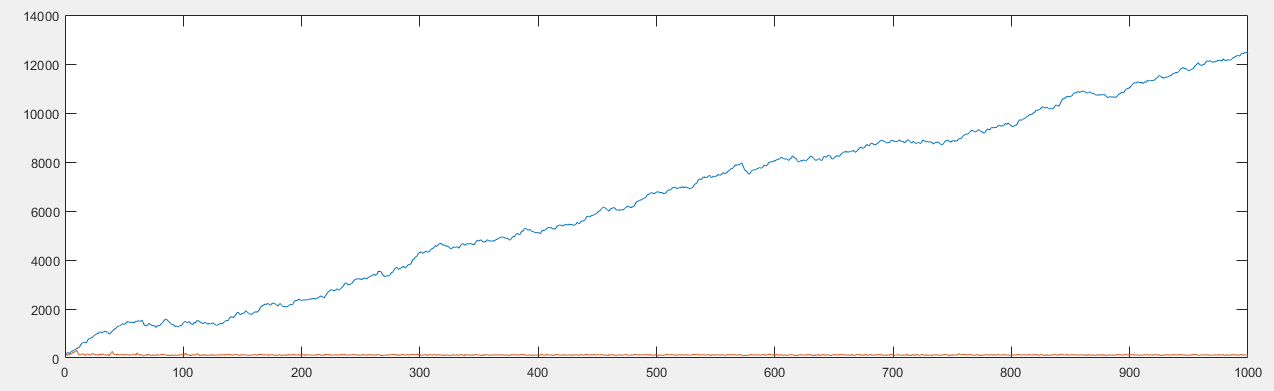
Setting: When serviceTime=Erlang(250,1)



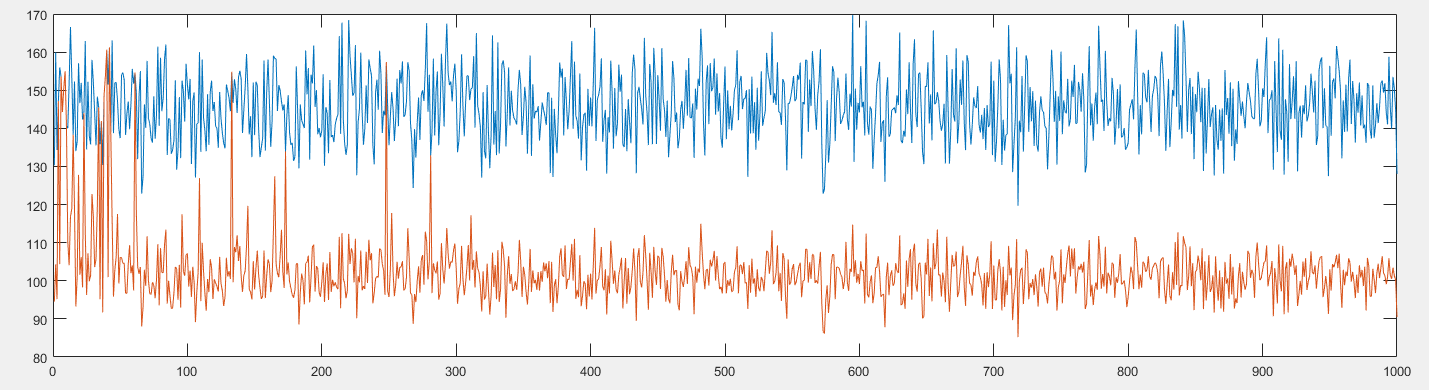
Setting: When serviceTime=Erlang(300,1)



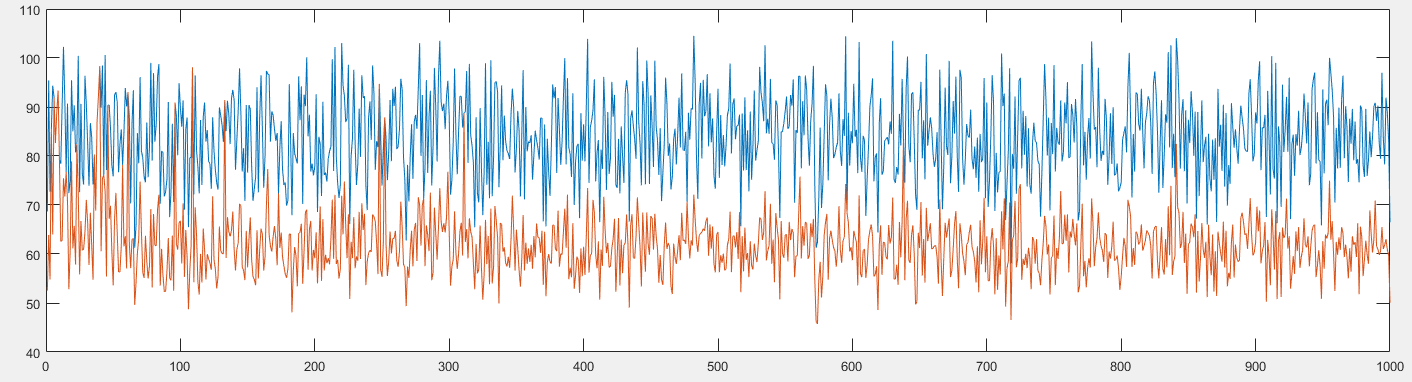
Setting: When serviceTime=Uniform(200,300)



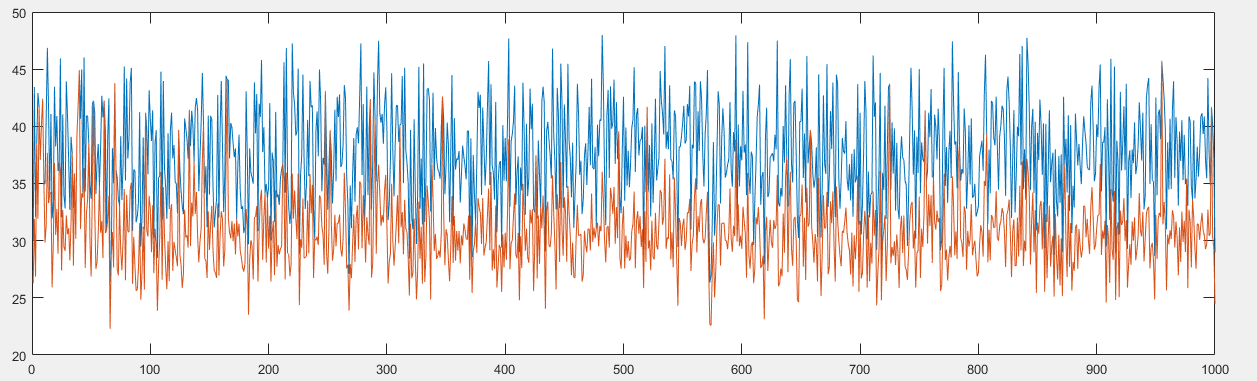
Setting: When serviceTime=Uniform(200,400)



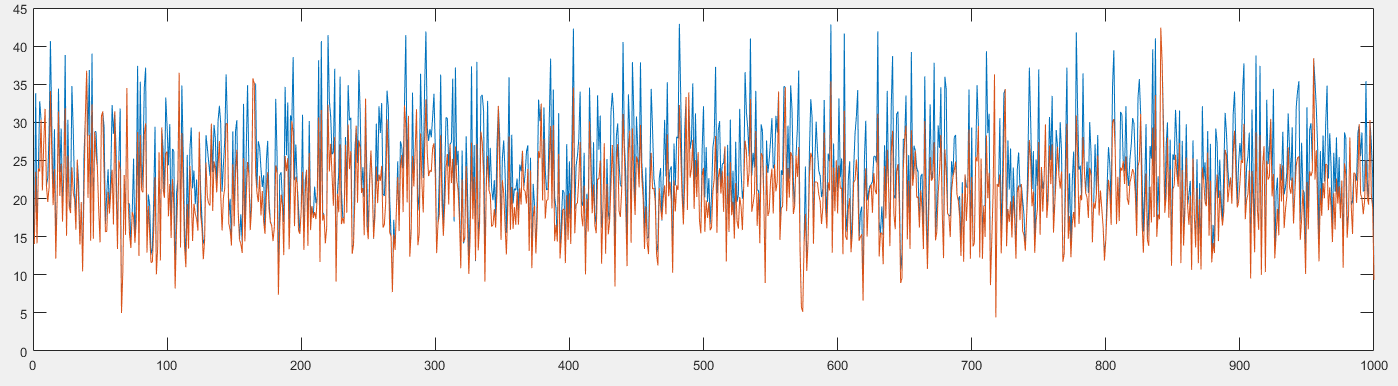
Setting: When serviceTime=Uniform(200,300)



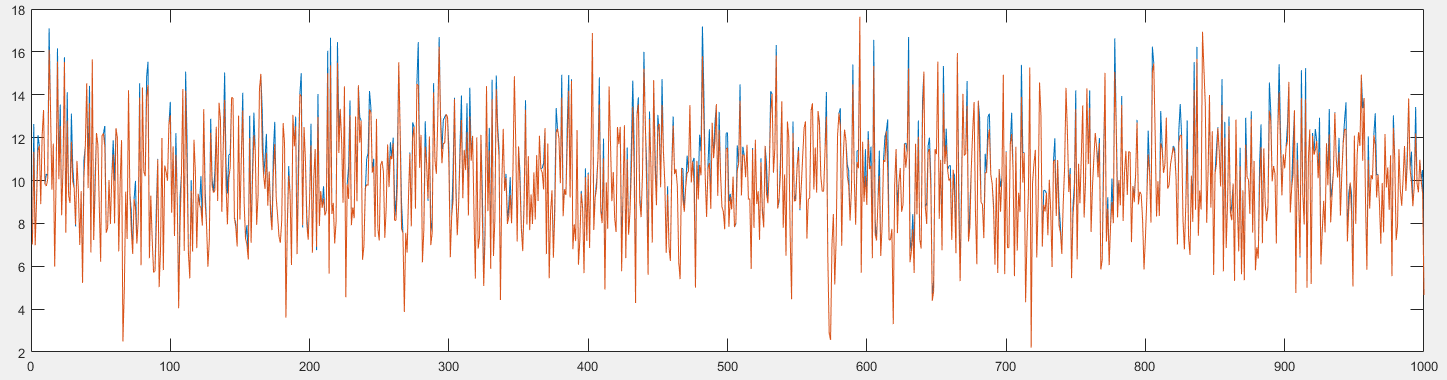
Setting: When serviceTime=Uniform(100,200)



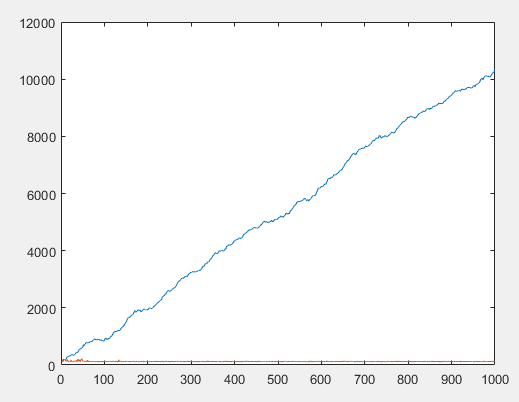
Setting: When serviceTime=Uniform(50,100)



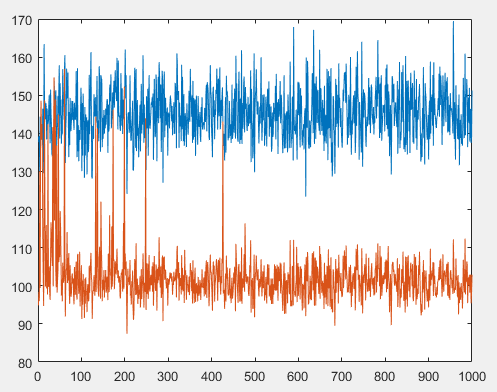
Setting: When serviceTime= Uniform(0,100)



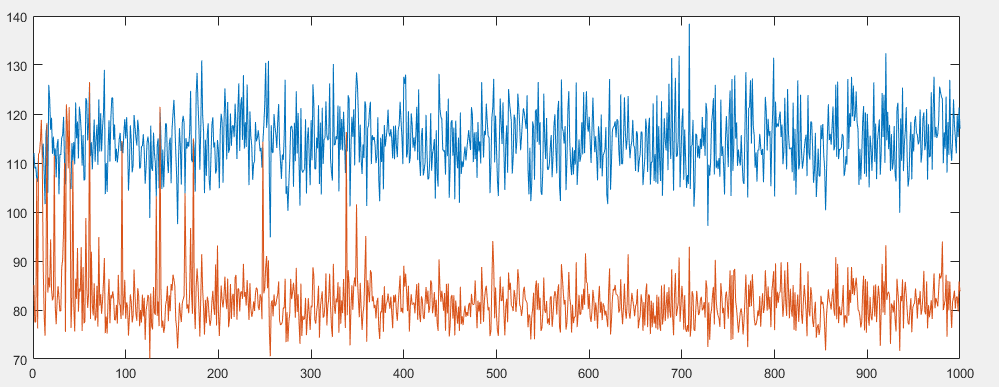
Setting: When serviceTime=Uniform(0,50)



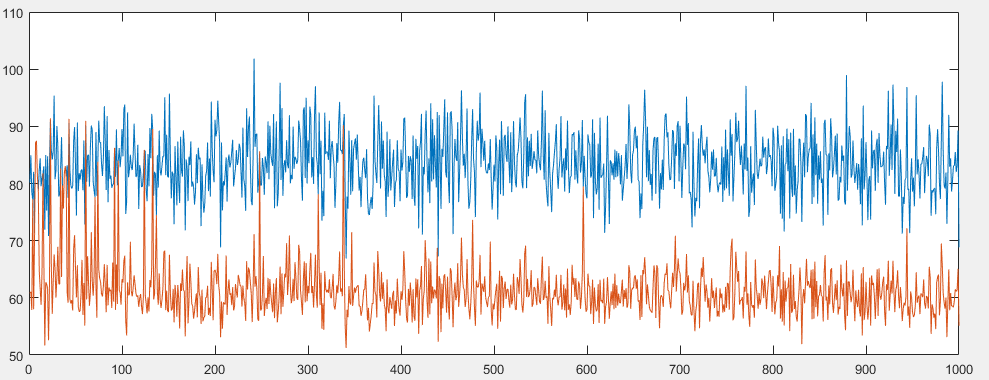
Setting: ServiceTime=Chisquare(300)



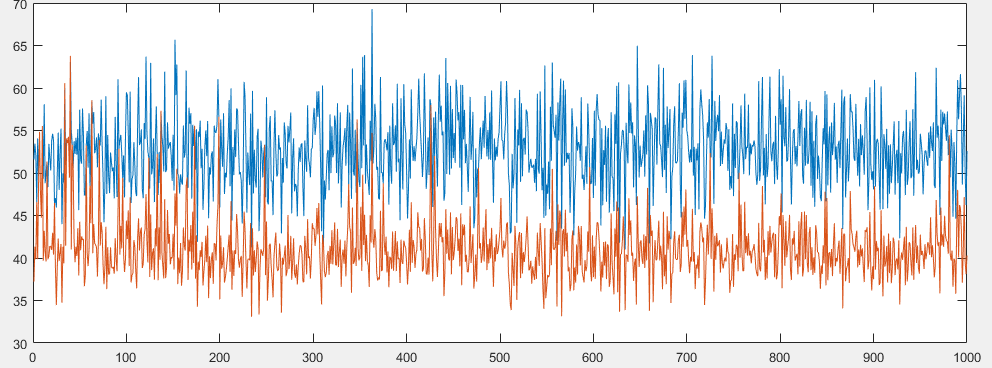
Setting: ServiceTime= chisquare(250)



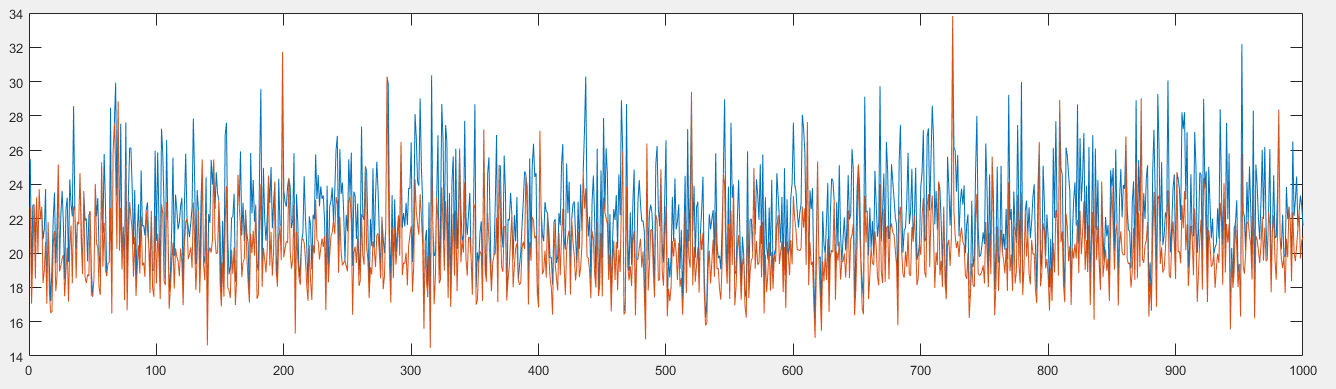
Setting: ServiceTime=chisquare(200)



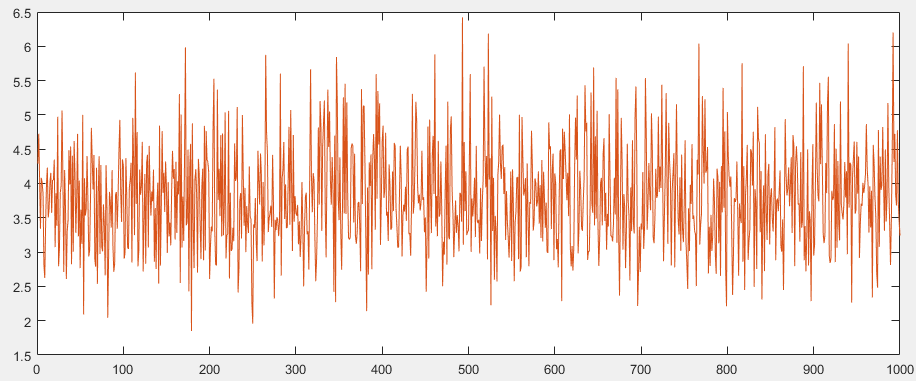
ServiceTime=chisquare(150)



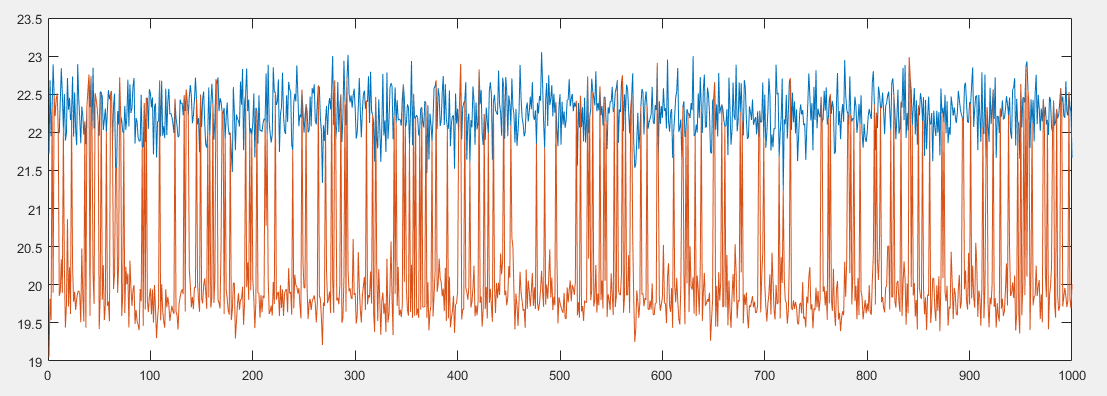
ServiceTime=chisquare(100)



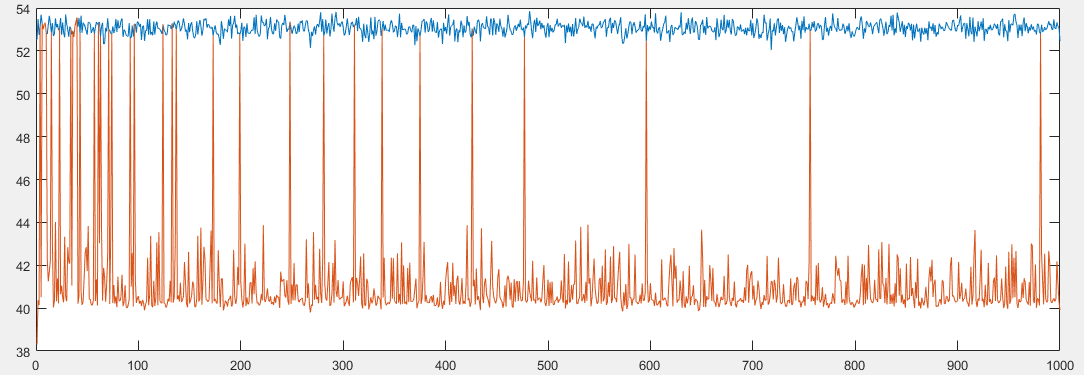
Service Time=chisquare(50)



Setting: ServiceTime=chisquare(10) : same performance of Rl and SC



Setting: ServiceTime=Uniform(50,1)



Setting: ServiceTIme=Uniform(100,1)

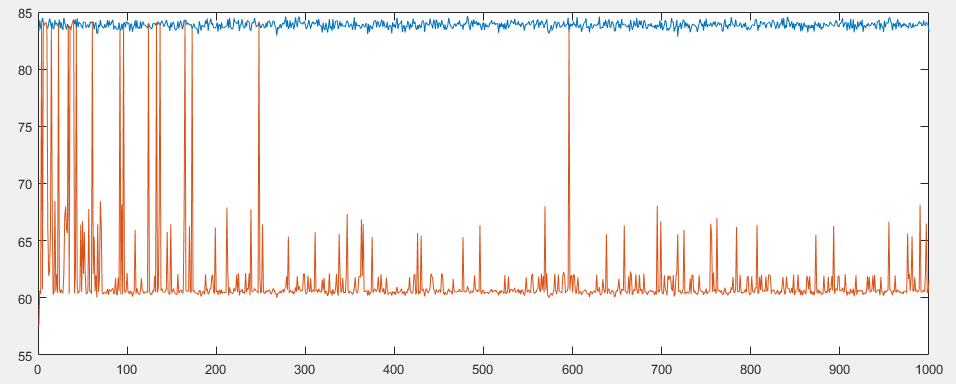


Fig. ServiceTime=Uniform(150,1)

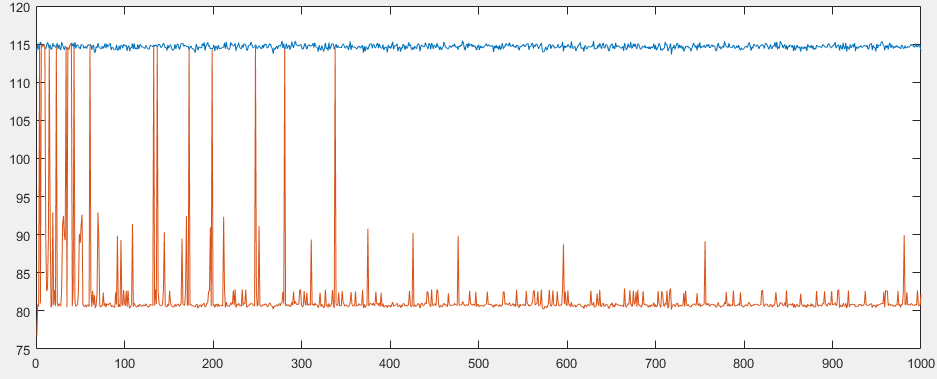


Fig. ServiceTime=Uniform(200,1)

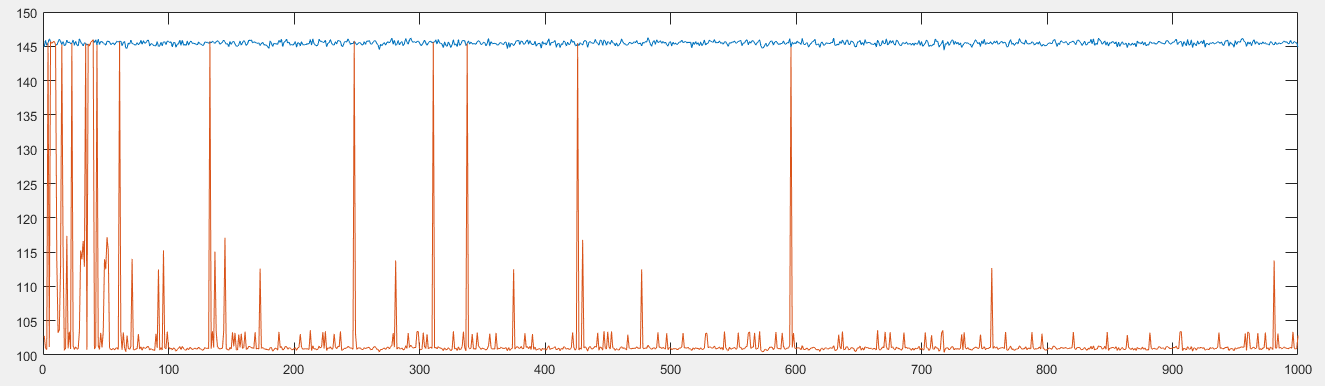


Fig ServiceTime=Uniform(250,1)

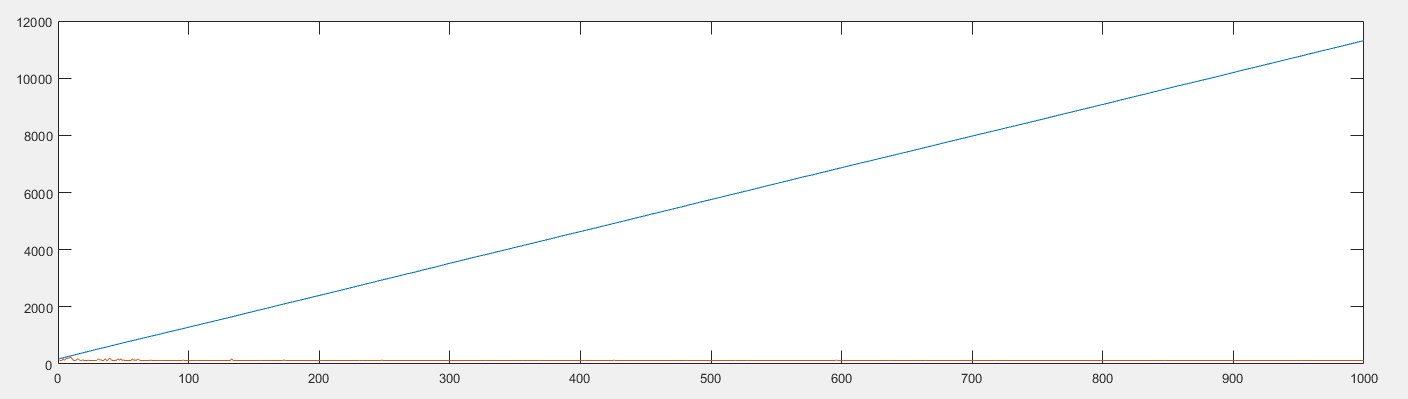


Fig. ServiceTime=Uniform(300,1)

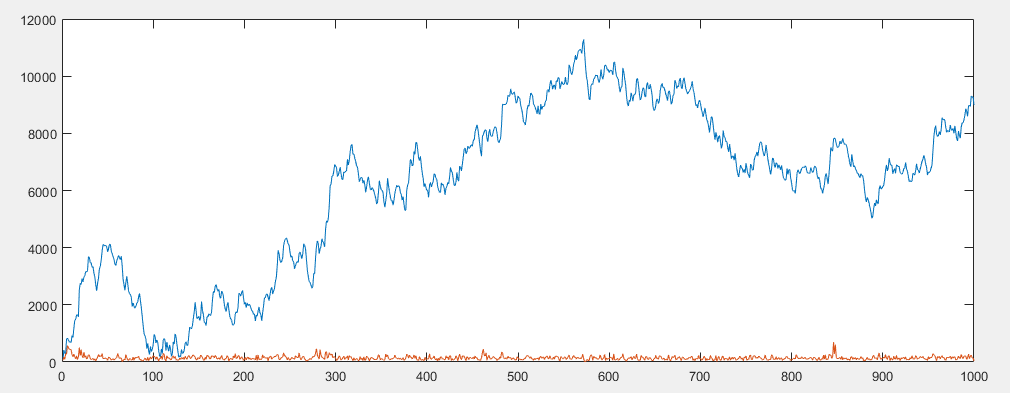


Fig. ServiceTIme=Exponential(300)

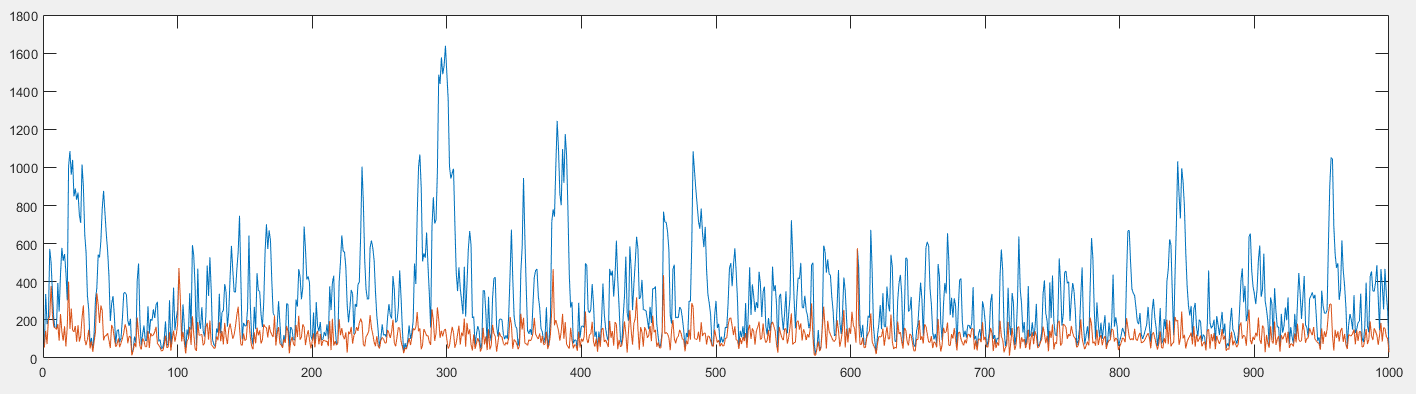


Fig. ServiceTime=Exponential (250)

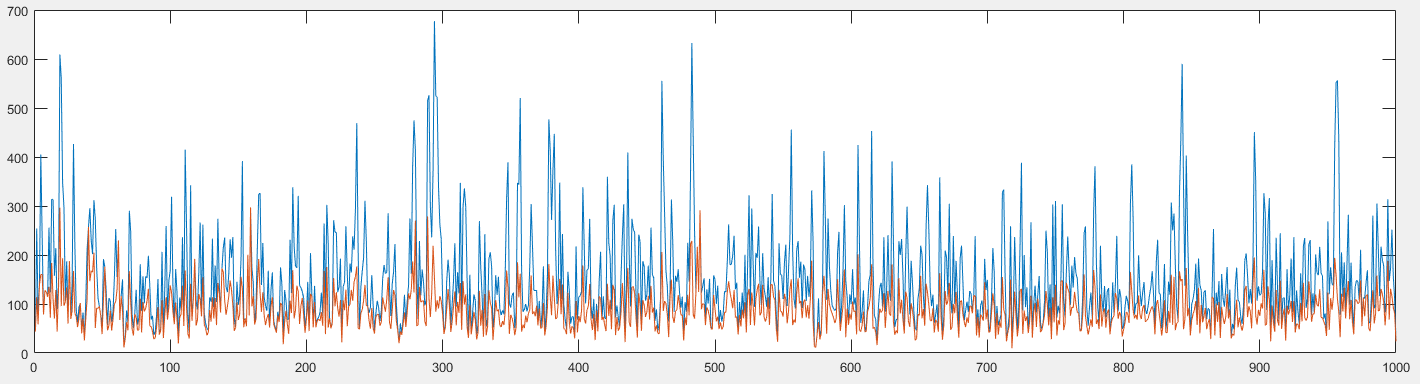


Fig. ServiceTime=Exponential(200)

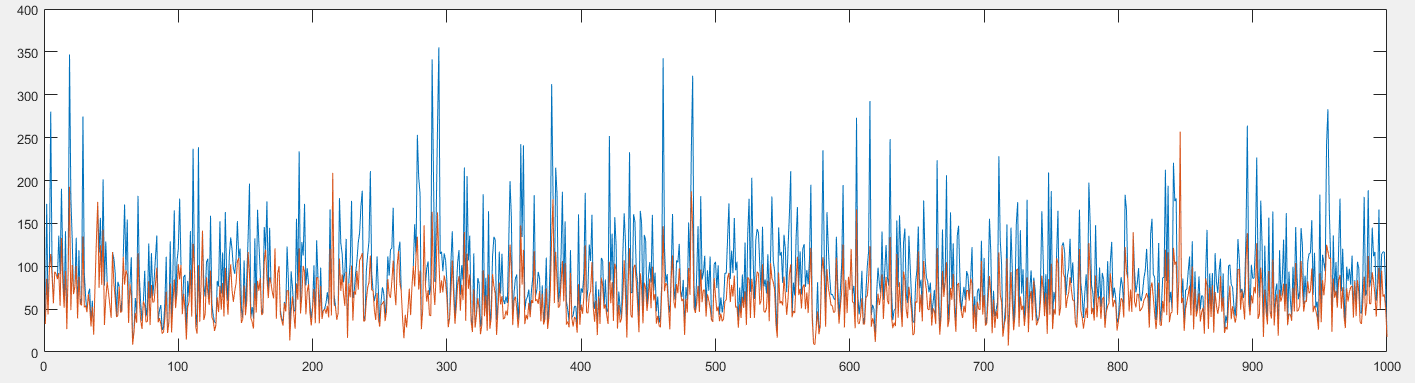


Fig. ServiceTIme=Exponential(150)

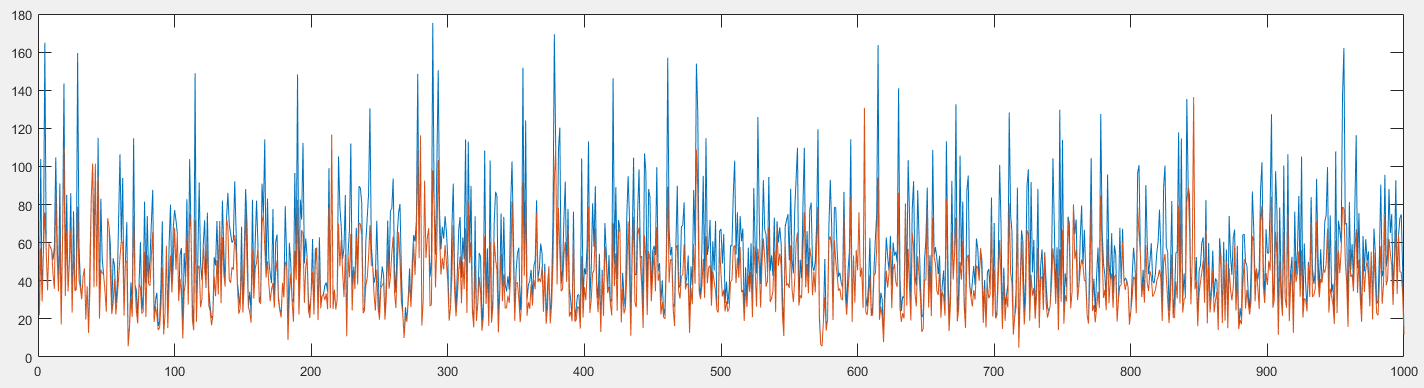


Fig. ServiceTime=Exponential(100)

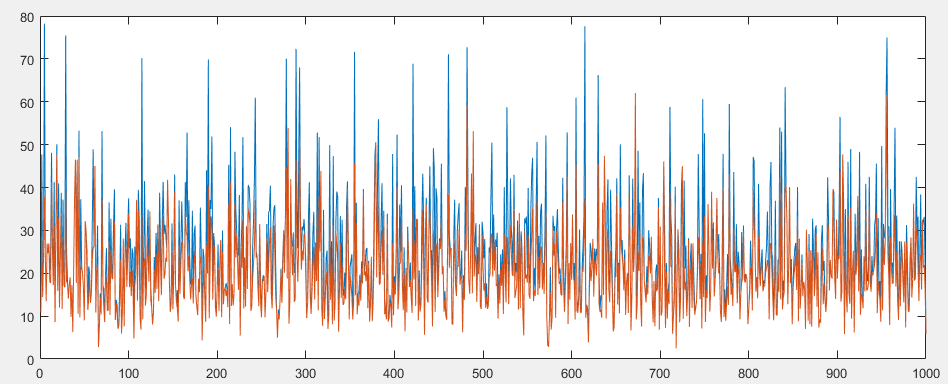


Fig ServiceTIme=Exponential(50)

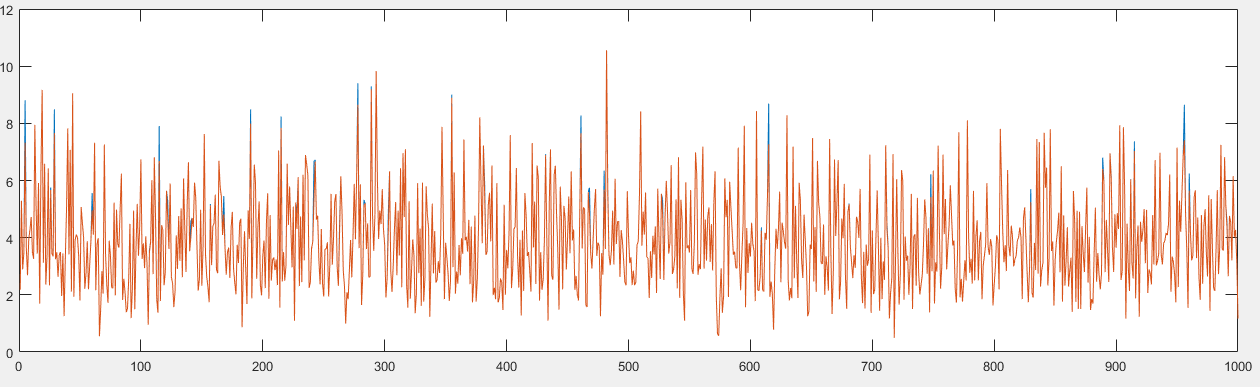


Fig. ServiceTime=Exponential(10)