Discrete Simulations Project

SYSC 4005

By. Emmitt Luhning, Chase Scott
Github: https://github.com/Em-kale/discrete-simulations-project

Chase Scott - Emmitt Luhning

Problem Formulation

Manufacturing Facility Entities

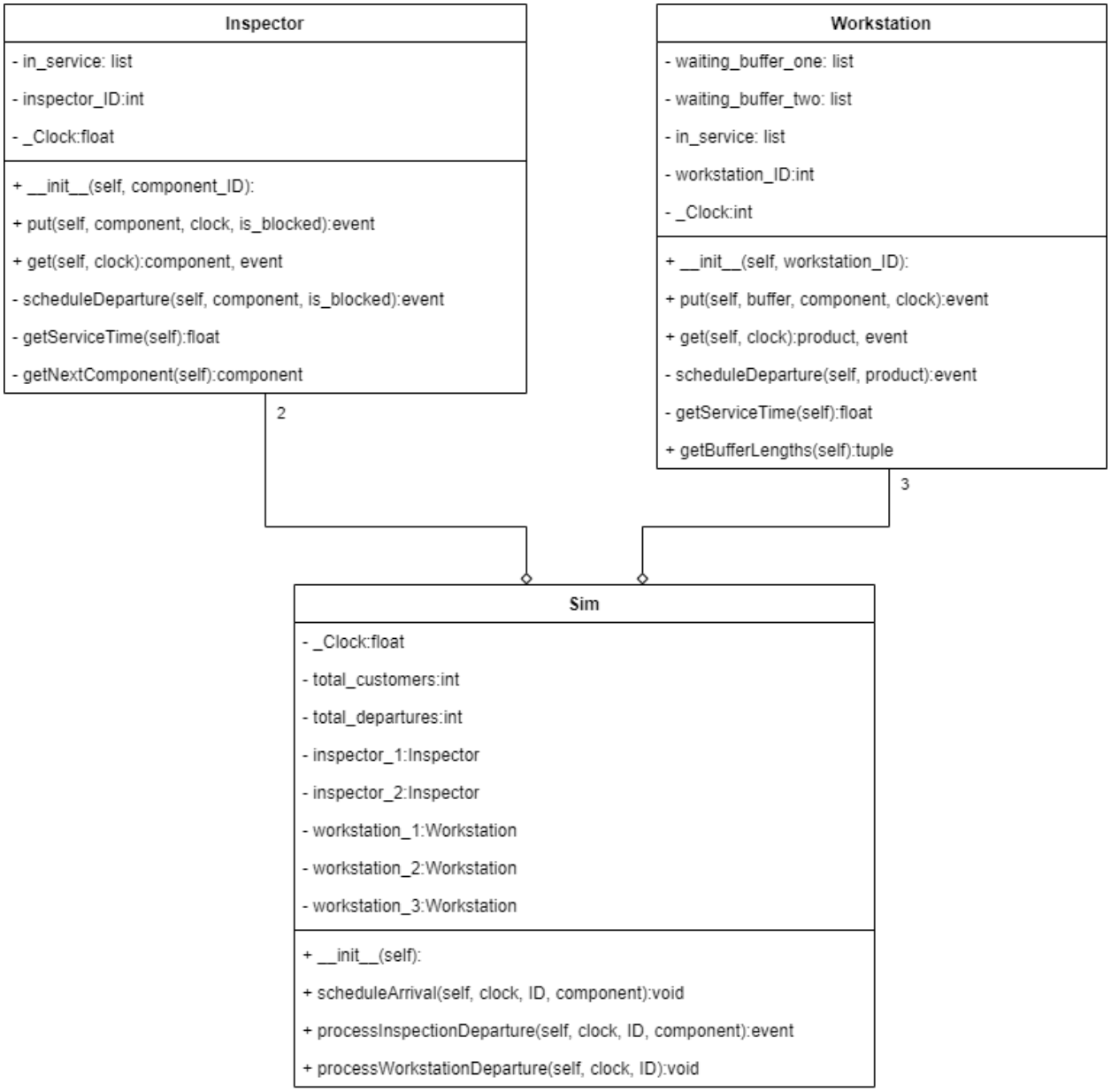
Within the manufacturing facility there exist two types of stations, inspectors and workstations, along with two types of work products, components and products. The inspectors inspect components, while the workstations assemble products from components that have already been inspected. The components come in three variations denoted as C1, C2, and C3, while the products come in three variations denoted as P1, P2, and P3. Products are constructed of components following the scheme:

- P1: 1 C1 component
- P2: 1 C1 component, and 1 C2 component
- P3: 1 C1 component, and 1 C3 component

Each workstation contains a buffer of size 2 for each type of product that it receives. For example, W3, which assembles products of type P3, has a buffer for C1 components, and a buffer for C3 components, each with a maximum length of two.

The purpose of this simulation is to determine the performance of the manufacturing facility, as well as to explore alternatives for delivering components to workstations from inspector 1.

Chase Scott - Emmitt Luhning



Setting of Objectives

Simulation Suitability

To determine whether a simulation is adequate to solve the specified problem, consider the metrics of complexity, historical data, verification/validation efficacy, cost, difficulty of direct experimentation, and time.

The specified problem is not too complex to model, which indicates that a simulation would be possible to implement. Additionally, historical data has been provided for which analysis can be

done to determine distributions for input domain modeling, and values can therefore be generated relatively easily. Furthermore, as the simulation would be of a manufacturing facility, there is no doubt that it would be cheaper, and more time effective to simulate the process, than to halt manufacturing and perform direct experimentation, where the time of manufacturing would come into play, as well as the costs associated with manufacturing the goods.

Metrics for Evaluation

Several metrics will be used to evaluate the performance of the system. The amount of products delivered by the system in a given unit of time (throughput) will be calculated along with the average capacity of the workstation buffers, the amount of time the workstations are busy, and the blocked time of the inspectors.

The goal of the manufacturing facility should be to increase the throughput, along with the average buffer capacity, and workstation busy time, while minimizing the buffer time. These goals are how the performance of the system will be evaluated based on the data collected from the simulation.

Resources, and Schedule

This simulation project will have three developers dedicated to the formulation of the simulation, and the subsequent analysis of the data. The timeframe for completion is two months, separated into several milestone's as follow:

Input modeling and Generation

The purpose of this milestone will be to find a suitable distribution to use for simulating service times for the inspectors, and workstations, as well as implementing that solution in the simulation.

Model Verification and Validation

The purpose of this milestone will be to validate the model using a variety of verification techniques, and to run the simulation to compute the values of interest, along with their confidence intervals and justification for the initialization length.

Alternative Operating Policies

This milestone will use the gathered information to propose alternative policies to increase the values discussed that should be maximized, and minimize those which should be minimized, as discussed in the Metrics for Evaluation section.

Chase Scott - Emmitt Luhning

Model Conceptualization

Stakeholders

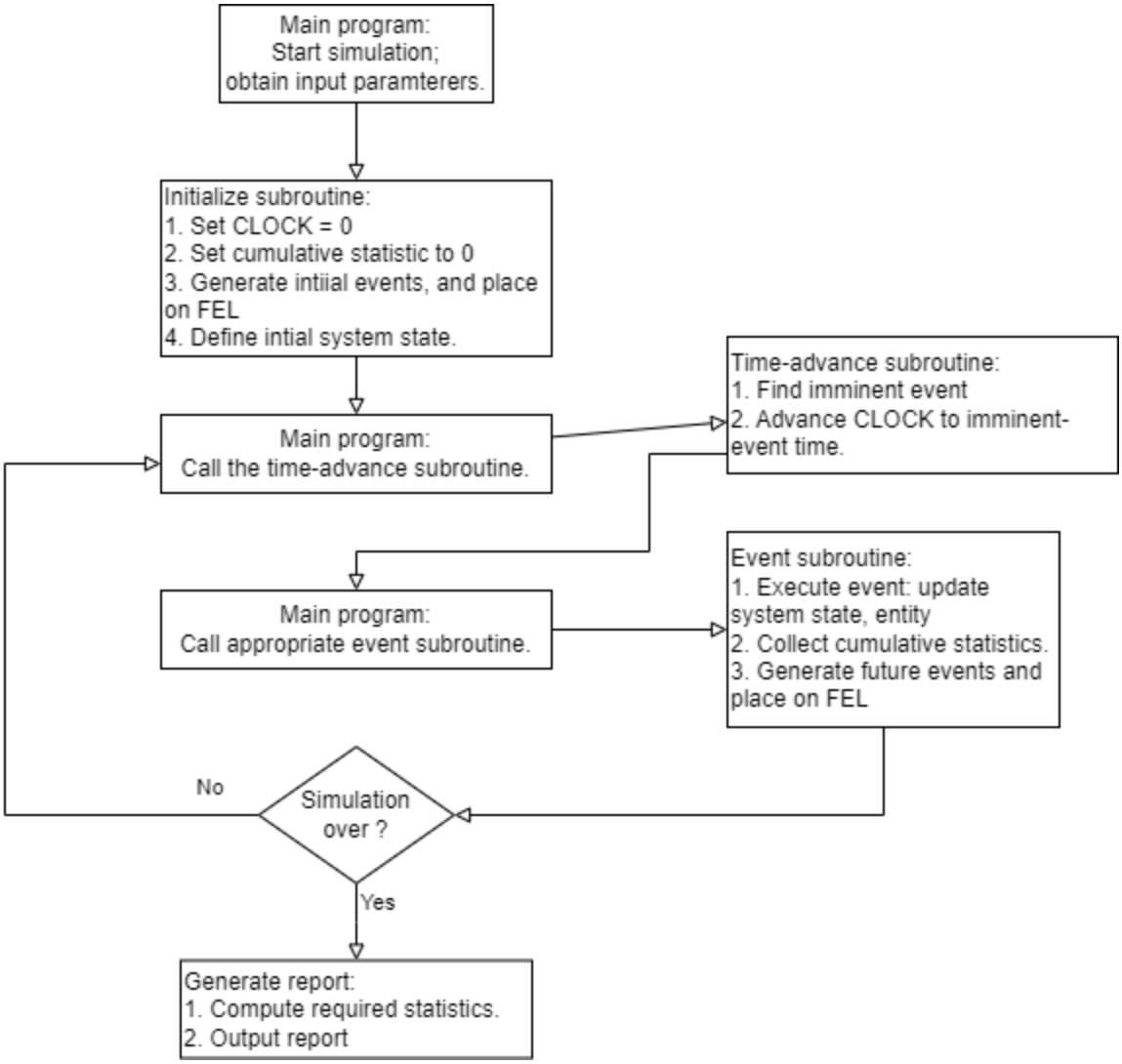
The identified stakeholders for this simulation project are the operators of the manufacturing facility being analyzed. In order for the simulation to be successful it must address their needs by supplying the performance information requested.

Entities and Interactions

The simulation will consist of three major components, the simulation controller itself, the inspectors, and the workstations. These will be represented in the implementation as three distinct classes.

Main Loop

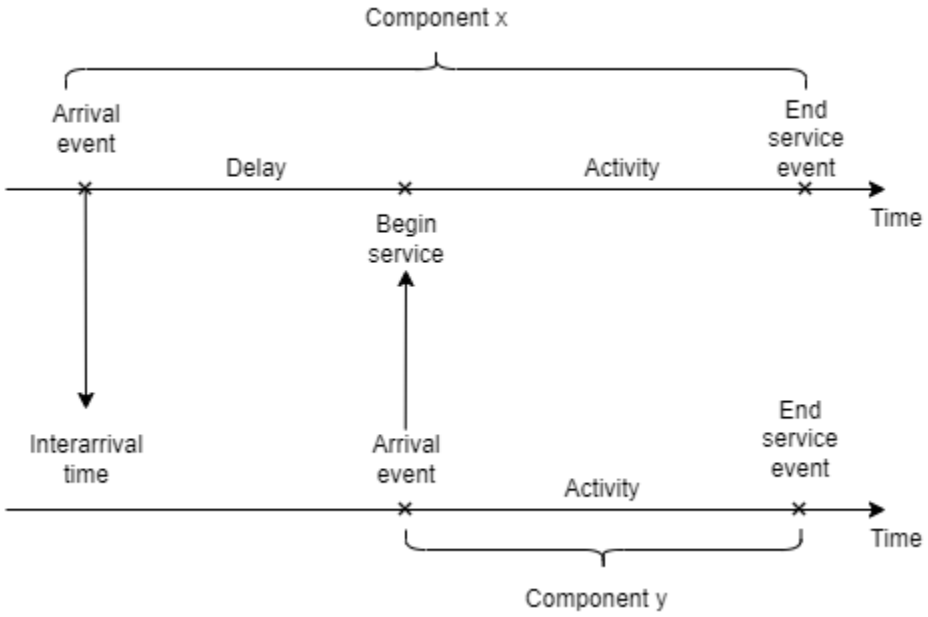
The driving force of the simulation will be the primary loop, which removes an event from the future event list (FEL), executes some subroutine based on the type of event it is, which may or may not place new events in the FEL, and then repeats until a set number of departures from the system have been met. Here, the logic for a basic event loop utilizing an FEL can be seen, this is the basis of the system we are going to implement.



Simulation Class

The simulation class represents the simulation as a whole, and will instantiate two inspectors, and three workstations from their respective classes. It then includes two functions, schedule arrival, and schedule departure, which will handle the two primary types of events. Depending on the type of event, and the current state of the system these functions will make calls to update the inspectors, and workstations to reflect the event that has occurred. This class also contains the logic which decides where to send the components to based on the status of the workstation buffers.

Here shows the interaction for a workstation, a component arrives at a certain time, then is delayed until the other component arrives. Then the workstation can assemble the product from these components:



Inspector Class

The inspector class represents the inspectors in the simulation. They include two primary functions that are accessed by the simulation class. The primary functions are put, and get, which are used to add new components to the inspector, and try to remove components from the inspector respectively. The put function is only called when initializing the system, as all consequent movement of components from waiting to in service will be done upon the previous component being removed, and this action is performed by the get method. The put method also includes a flag to indicate if the component is currently in service, or if the inspector is currently blocked, which will result in a new departure event being generated for a short period of time in the future.

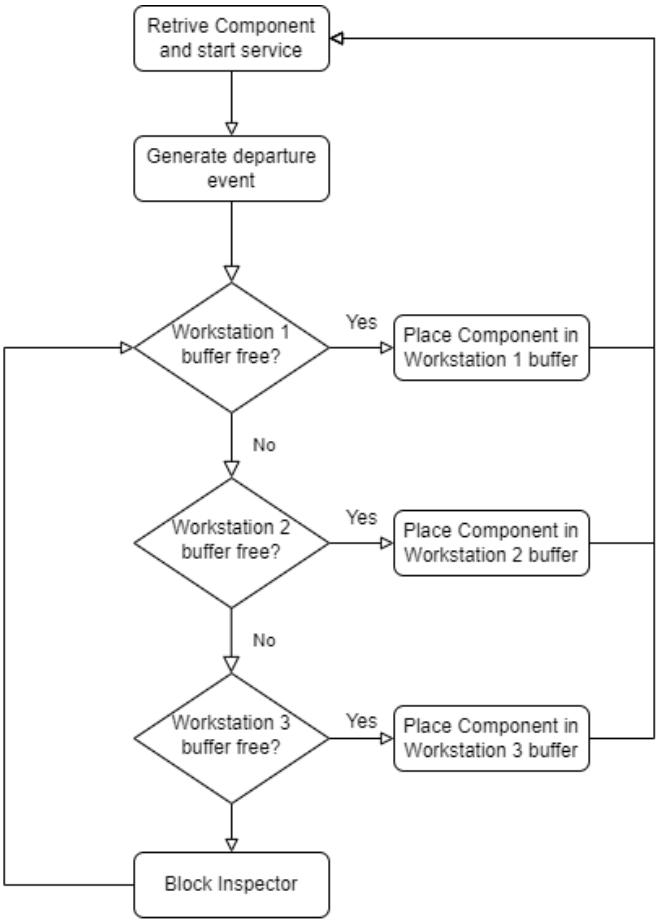
```
    def put(self, component, clock, is_blocked):
        """ update clock"""
        self._Clock = clock
        """ start service """
        self.in_service.append(component)
        depart = self.scheduleDeparture(component, is_blocked)
        return depart
```

Additionally, the inspector class includes a method for generating departure events which are returned to the simulation after a new component is added to the inspector and include the necessary information for the simulation to add these to the FEL.

```
    def get(self, clock):
        """ get component from the service queue"""
        component = self.in_service.pop(0)
        """ update clock"""
        self._Clock = clock
        """ get next component to be serviced"""
        next_component = self.getNextComponent()
        self.in_service.append(next_component)
        """ schedule departure for next component"""
        depart = self.scheduleDeparture(next_component, False)

        return component, depart
```

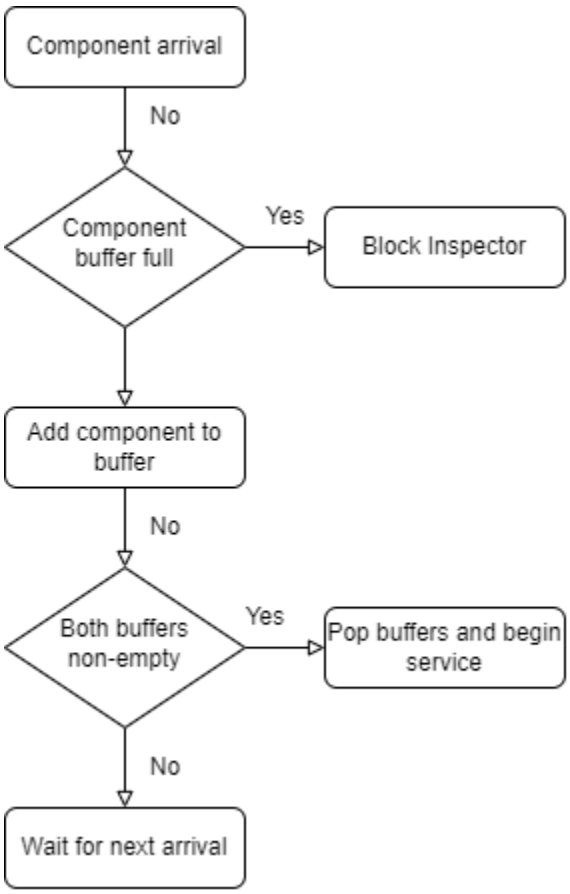
Here is a basic flowchart showing the execution loop for the Inspector class:



Chase Scott - Emmitt Luhning

Workstation Class

The workstation class is used to represent the workstations within the simulation, and like the Inspector class, includes two primary methods: put, and get. Workstation deviates from Inspector in the additional logic necessary for managing the buffers, and determining whether or not the workstation has the necessary components to begin service on a product, thus generating a departure event. If the necessary components are not yet in the buffers, then no event is generated. Otherwise, workstation works similarly to Inspector in that it will attempt to add a new product to service when one is removed if the buffers contain those components required. Here is a flow chart showing the logic for a component arrival to a workstation:



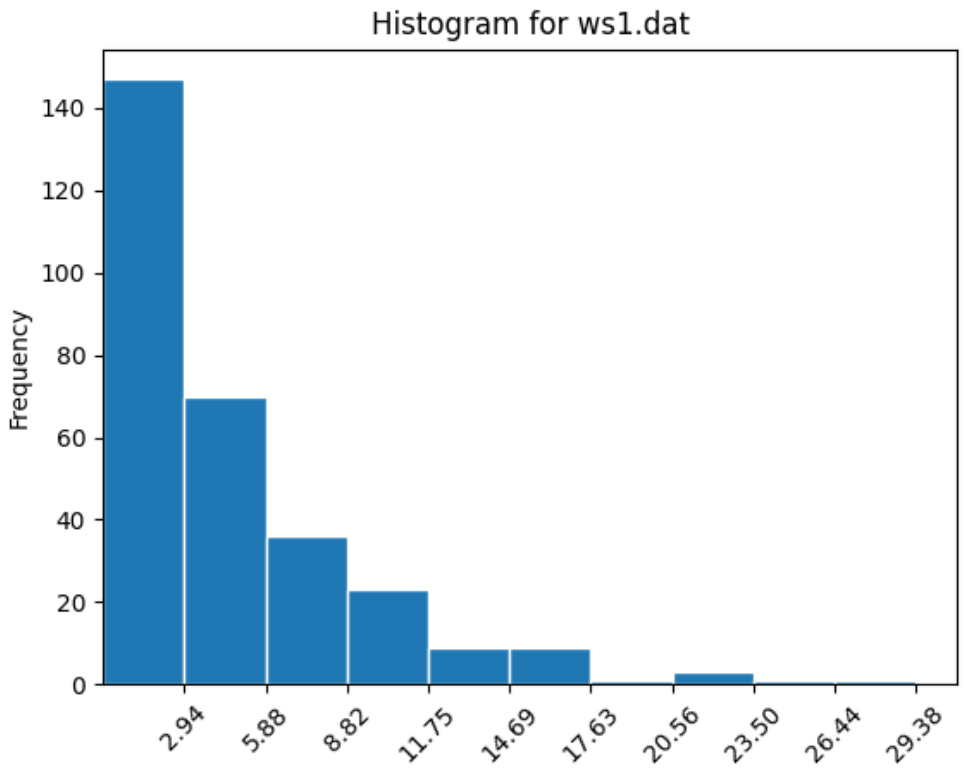
Chase Scott - Emmitt Luhning

Model Translation

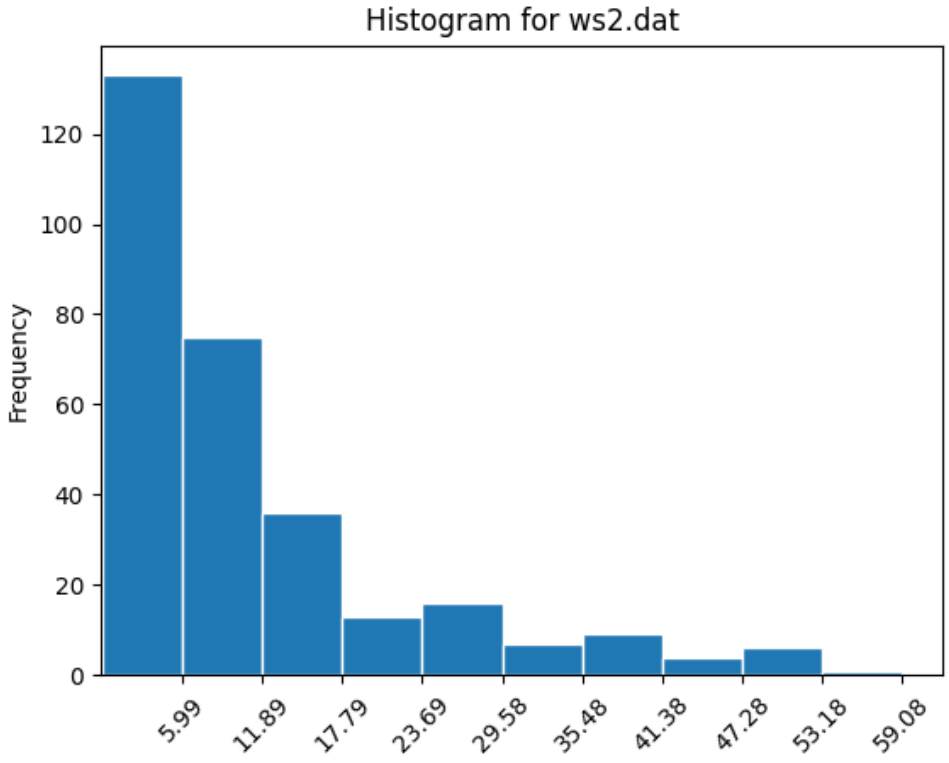
The implementation of the discussed model will be created using python. A general purpose, high level programming language was chosen due to the relative simplicity of the simulation, along with the low cost. Additionally, the team developing the simulation have experience with python and thus no time will be wasted learning a new tool. Python also allows for the simulation to be flexible, and alternative configurations to be easily tested and verified.

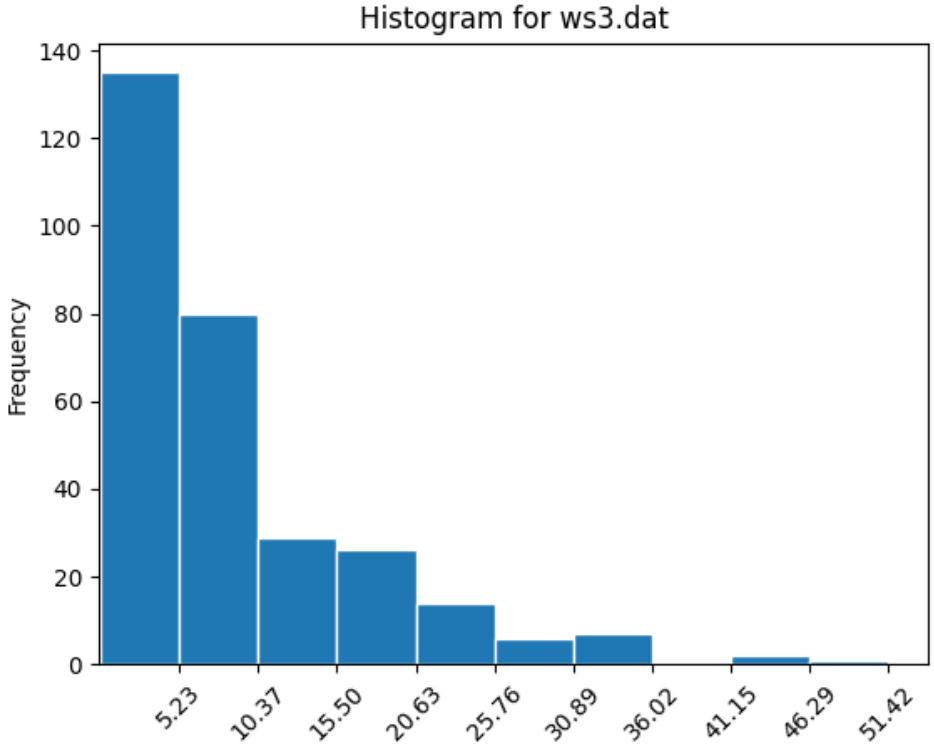
Data Collection Input Modelling

First we generated histograms for each file containing historical data for the 2 inspectors and 3 workstations. The bin size was chosen using Strurge's rule, which is a common method used to determine the appropriate number of bins for a histogram[1]. Here are the resulting histograms:

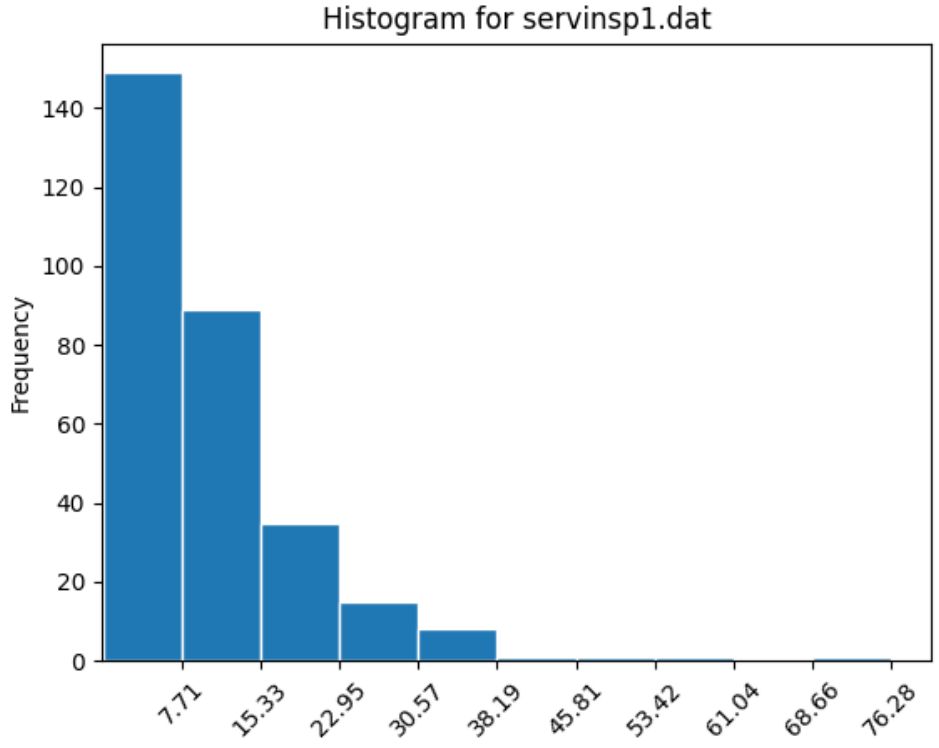


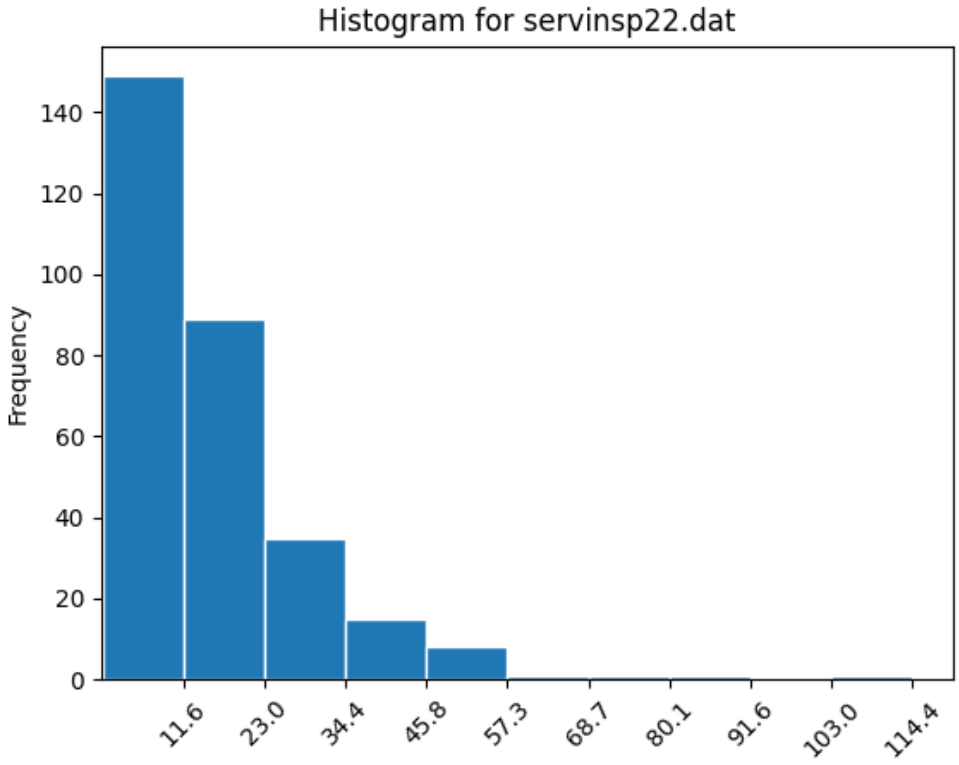
Chase Scott - Emmitt Luhning

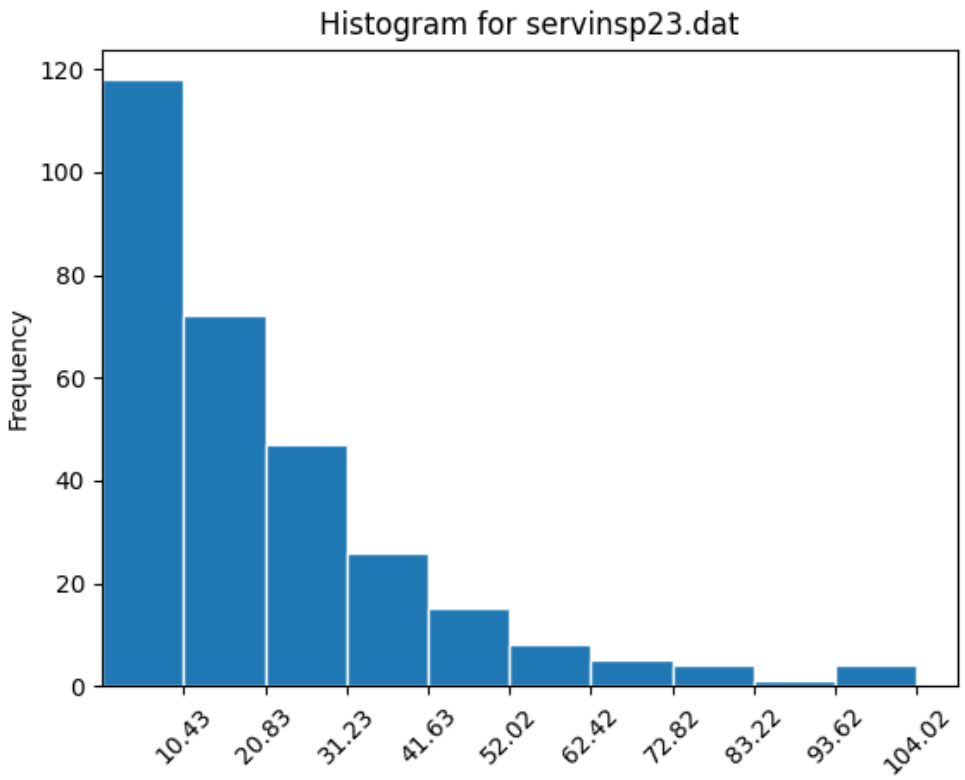




Chase Scott - Emmitt Luhning







These however are not fully adequate for input modelling, since a histogram just provides the general shape of the distribution of data. We need to employ more rigorous statistical techniques like QQ plots and chi-square tests to verify that they actually follow a theoretical statistical distribution that can be estimated from a histogram.

QQ plots

QQ plots are a tool used for comparing two distributions. In this section we are going to use these plots to compare our measured data against a distribution to determine if our data fits that distribution closely. Based on the nature of the data being the times between events (arrival and departure), and the shape of the histograms, we will assume an exponential distribution, and compare our values against it.

A QQ plot works by comparing the quantiles of our measured data versus the distribution. If the quantiles are close, then the plot should show a slope close to 1. To do this we need to first obtain the inverse cdf of the theoretical distribution, and calculate it for each point, where q is equal to the quantile of our measured data for that point. Then, by calculating the inverse cdf of the distribution with respect to the quantile, we should get a value close to the measured value.

Chase Scott - Emmitt Luhning

To get the slope expected, we need to calculate each quantile such that it is with respect to our values in ascending order.

Knowing the cdf of an exponential distribution is:

$$1 - e^{-\lambda q}$$

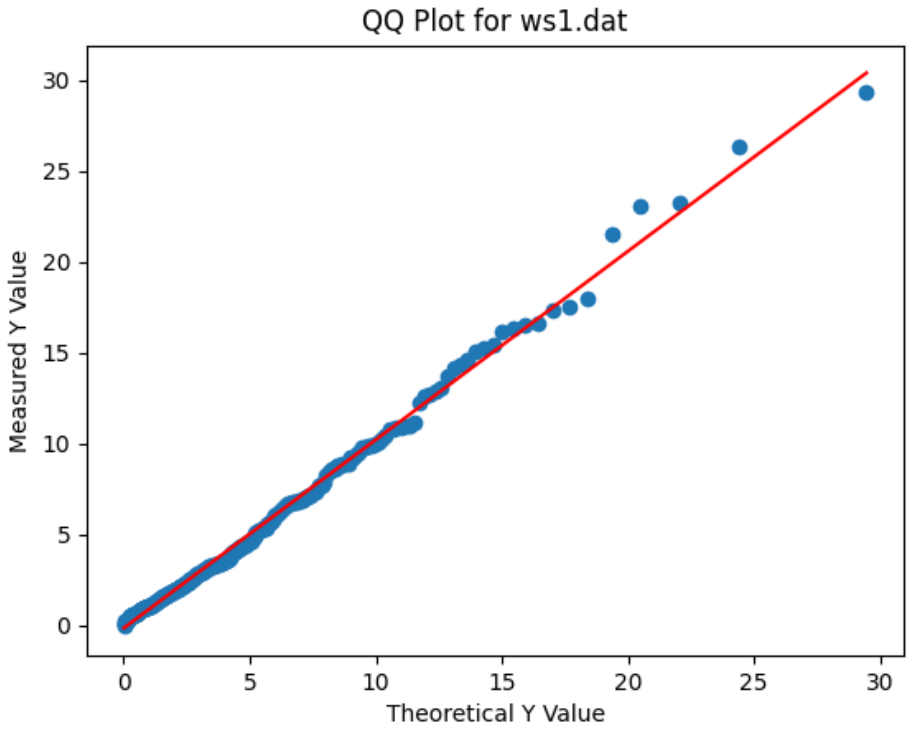
We can calculate the inverse cdf to be:

$$- ln(1 - q)/\lambda$$

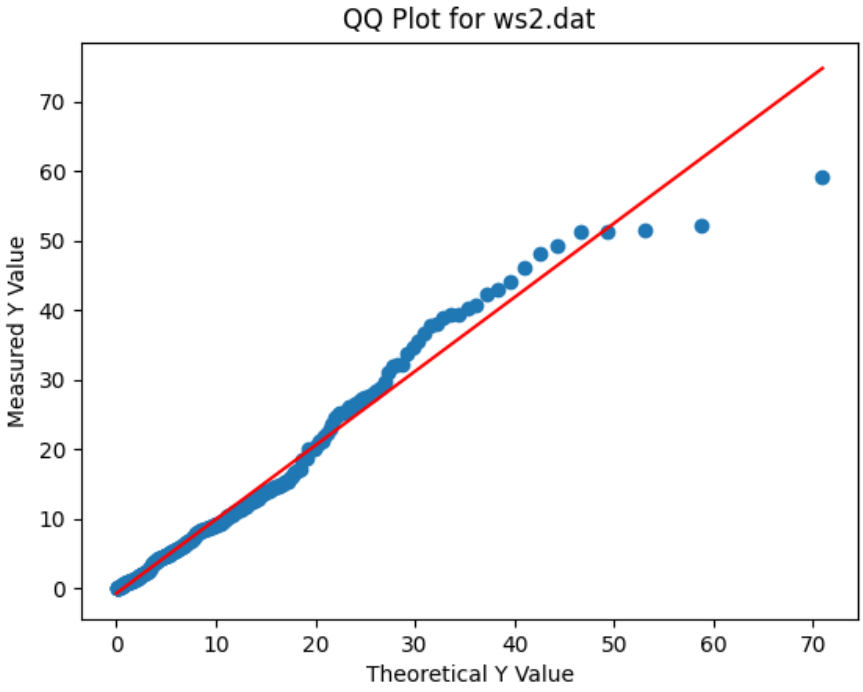
Where $q = j - \frac{0.5}{n}$, knowing that j is equal to the sequence number of the value in our ordered list of values, and n is the total number of values.

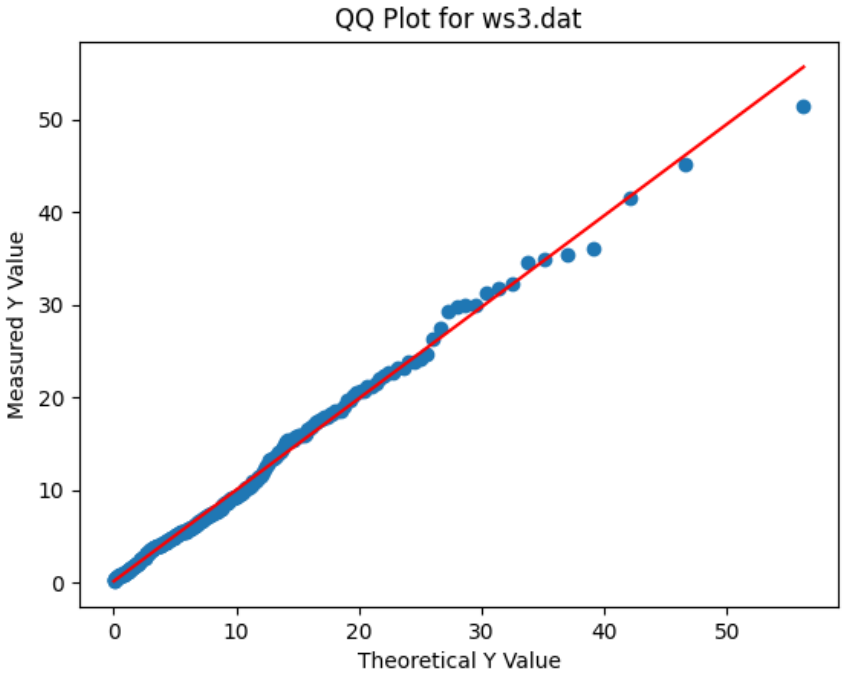
In order to calculate the values, we need to estimate what the rate parameter λ. To do this, we will use the maximum likelihood estimation, which leaves the recommended estimation for the rate parameter of an exponential distribution as 1 / X, where X is the mean of the data set. This means that for each QQ plot, the rate parameter will be unique. The resultant plots are as follow where x is the theoretical quantiles and y is the sample quantiles:

Plots:

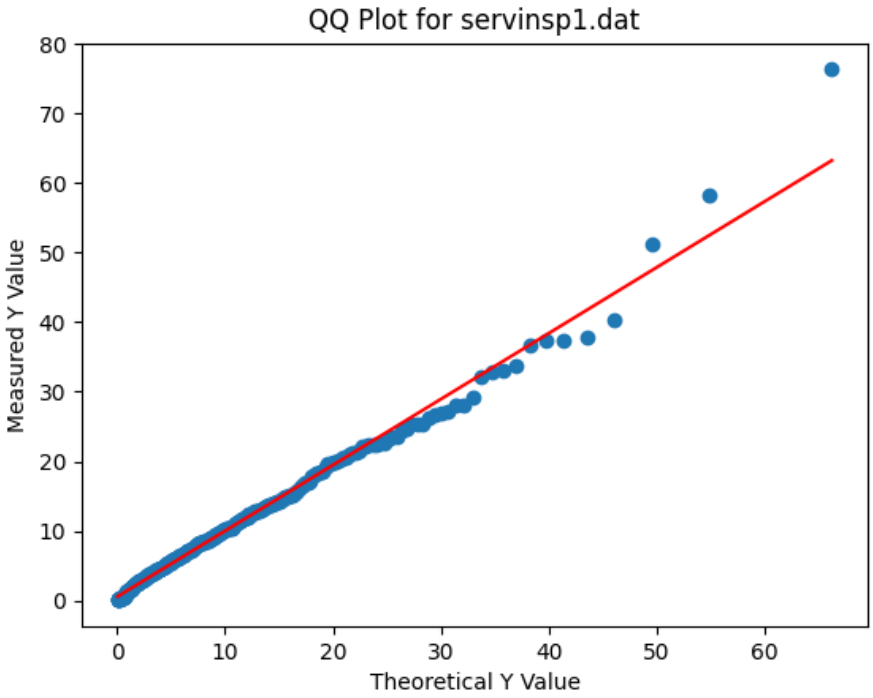


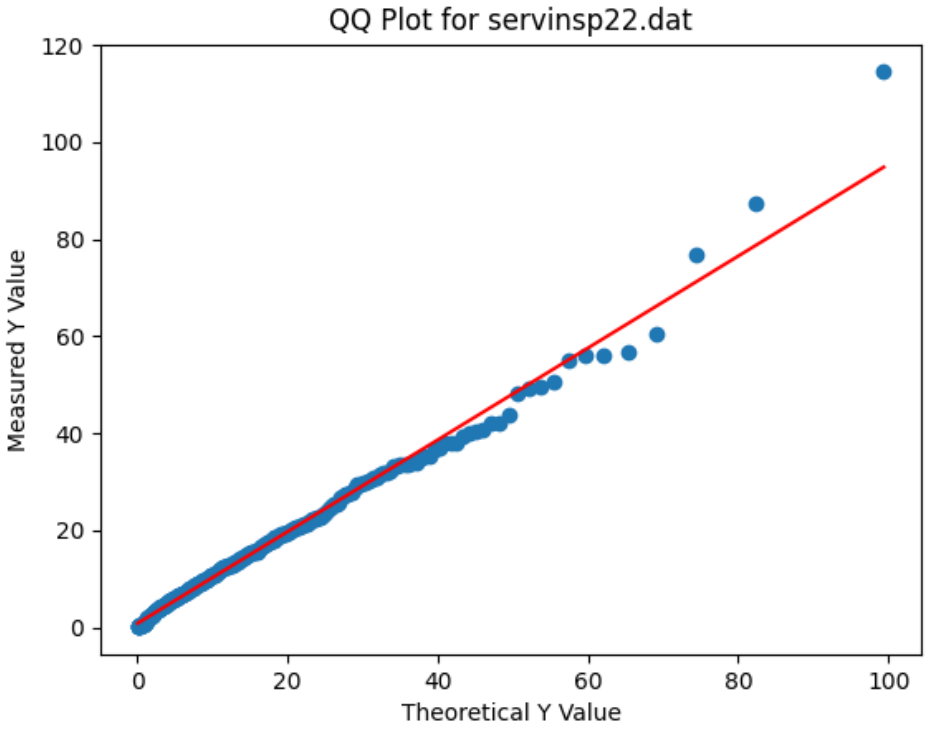
Chase Scott - Emmitt Luhning

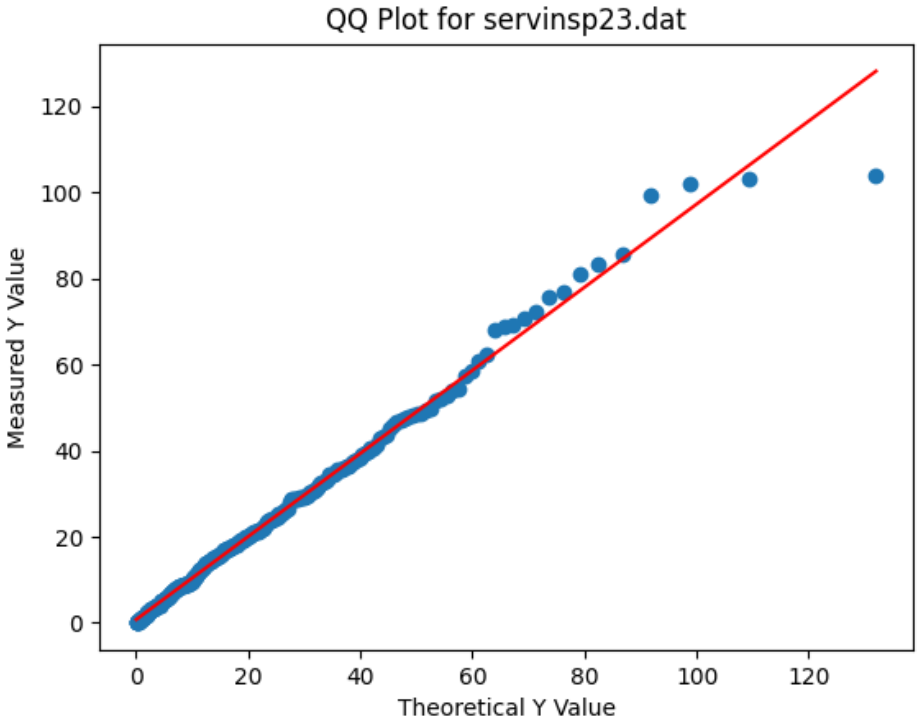




Chase Scott - Emmitt Luhning







QQ Plot Conclusion

It can be seen that the plots all follow a fairly straight line. QQ plots offer a visual indicator of a distribution's fit, but there is no metric for precise evaluation. Additionally, it can be noted that the plots appear to deviate from the straight line when approaching the tail, the variance for the higher values are higher, and as a consequence, the slope representing the middle of the data set is more useful for identifying the quality of the fit. Given this information, it can be seen that an exponential distribution results in a slope close to 1, and will consequently be used moving forward.

Chi Square Tests

Now that we have a robust reasoning for our hypothesis that the data follows an exponential distribution, we need to use a goodness of fit test to get a quantitative determination on whether it is adequate. To do this, a chi-square test will be used. The chi square test is used to analyse the difference between our expected values and observed ones, specifically using the number of values that we expect to fall in some intervals for each distribution. This test relies on the central limit theorem, which states that the mean of independent random variables tends towards a normal distribution, even if the original random variables follow a different one, like in our case, potentially an exponential one. A random variable with the chi square distribution is a variable representing the sum of some number of these independent normal random variables.

Therefore, knowing that the following equation produces an approximately normal random variable:

$$x = \frac{(Oi - Ei)^2}{Ei}$$

Where Oi is our observed number of values, and Ei is the theoretical number of values for a given interval. We can sum these values for each interval, and check the result against the chi-squared distribution.

Sample Calculation for Servinsp1:

Hypothesis Formulation

Let our hypotheses be:

$$H_0: \textit{Sample data follows exponential distribution}$$

$$H_1: \textit{Sample data does not follow exponential distribution}$$

Determination of k value

The k value for a data set represents the number of intervals to be used and is determined based on the number of samples. We have n = 300 samples for each data set, and it is recommended that k be between the root of n and n / 5 for n > 100. For this section we have selected k = 20, which falls in the recommended range.

Determination of Probability

Each interval should have an equal probability. Given k = 20:

$$p = 1/k$$
$$p = 1/20 = 0.05$$

Determination of class intervals

For a given k, and knowing the mean of servinsp. = X, the interval is determined by:

$$\alpha_i = -\frac{1}{\lambda} ln(1 - kp)$$
$$\alpha_i = -\mu\, ln(1 - kp)$$

For the first k = 0:

$$\alpha_0 = -\mu\, ln(1)$$
$$\alpha_0 = 0$$

For the second k = 1:

$$\alpha_1 = -\ \mu\, ln(\ 1\ -\ \ 1(0.05))$$
$$\alpha_1\ = 0.5313$$

Therefore, the first interval will be from 0 to 0.5313. This then needs to be repeated for k intervals.

Determination of observed value for interval 1

The observed value for the first interval is the number of values from the data set that fall within that interval. For servinsp1.dat, 16 values fall between 0 and 0.5313, meaning E1 = 16.

Determination of expected value for interval 1

The intervals represent equal probabilities, and as such we expect all the data values to be evenly divided among them. Therefore, the expected value for interval 1 is the same as the expected value for all other intervals, and is n / k, or 300 / 20 = 15, meaning O1 = 15.

Determination of Test Statistic

$$\chi_0^2\ = \sum_{i\ =\ 1}^{k} \frac{(Oi - Ei)^2}{Ei}$$

Sample Calculation for interval 1

$$\frac{(Oi - Ei)^2}{Ei}\ = \frac{(15 - 16)^2}{16}\ = 0.0625$$

This then needs to be done for each interval, and the results summed. The individual values for each calculated value, and their sums can be found in the appendix of this report. The sum for the results of the calculations for all intervals of servinsp1.dat has been determined to be 16.6667.

Comparison of calculated value versus expected statistic for servinsp1

$$\chi_0^2\ = 16.667$$
$$\chi_{0.5,18}^2\ = 28.869$$

Since $\chi_{0.5,18}^2 > \chi_0^2$ the null hypothesis is not rejected, and the exponential distribution is determined to be a good fit with the confidence interval of 0.05.

Chi-Squared tests for other data sets

The individual intervals and calculated values are provided for each data set in the appendix. Here we will simply list the resultant calculated statistic for each set.

$$\text{Servinsp22: } \chi_0^2 = 16.667$$

$$\text{Servinsp23: } \chi_0^2 = 16.933$$

$$\text{W1: } \chi_0^2 = 24.8$$

$$\text{W2: } \chi_0^2 = 20.8$$

$$\text{W3: } \chi_0^2 = 21.333$$

It can be seen that for all of these data sets, $\chi^2_{0.5,18} > \chi^2_0$, which means that the null hypothesis will be accepted for each.

Input Generation

Now that we have completed our input modelling, we can generate random numbers that adequately mirror the actual behaviour of the system being modelled by using the historical data that was collected. It is impossible for computers to generate truly random numbers, so we must create a way in order to generate random numbers that are uniform and independent. This means that every random number has an equal chance of being selected, and each random number is not affected by future or previous values in the sequence. In addition to uniformity and independence, two important properties of a good random number generator are maximum density and maximum period. This means that our generator should not produce clusters of similar values, which is important because if the RNG is not densely populated, it can lead to biased results and affect the accuracy of the simulation or other application. The generator should also have a large period, which is the number of unique values that can be produced before there is repetition. This is important because if the period is too short, the RNG may start to produce the same values again, leading to repeating patterns.

Implementation

We chose to use a CLCG generator, because LCG has a period which isn't not adequate for complex system simulation like our project. The implementation of CLCG in code involves setting up multiple LCGs with different parameters, and then combining their outputs to produce the final sequence of random numbers. Each LCG is typically defined by the following formula:

$$X_{i+1} = (aX_i + c) mod\, m,\ i = 0, 1, 2, \ldots$$

where X_i is the current value in the sequence, a is a multiplier, c is an increment, and m is the modulus. When then need to combine the outputs from each of our LCG's using the following formula:

$$X_i = \left(\sum_{j=1}^{k} (-1)^{j-1} X_{i,j} \right) mod\, m_1 - 1$$

To implement CLCG, we first choose several sets of values for the parameters a, c, and m, and then create a separate LCG for each set of parameters. We can then use the outputs of these LCGs to produce the final sequence of random numbers. This can be seen in our random function, which returns our random number after performing the preceding equations:

```
def random(self):
    # Generate two random integers using the two LCGs
    self.x1 = (self.a1 * self.x1) % self.m1
    self.x2 = (self.a2 * self.x2) % self.m2
    z = (self.x1 - self.x2) % (self.m1 - 1)
```

In terms of choices for our constants, L'Ecuyer suggests using two LCGs with prime moduli and with the values of the parameters chosen such that the LCGs have a long period and good statistical properties[2]. In particular, L'Ecuyer recommends using the following values:

a1 = 40,014,
m1 = 2,147,483,563,
a2 = 40,692,
m2 = 2,147,483,399.
c1 = c2 = 0

Uniformity Testing

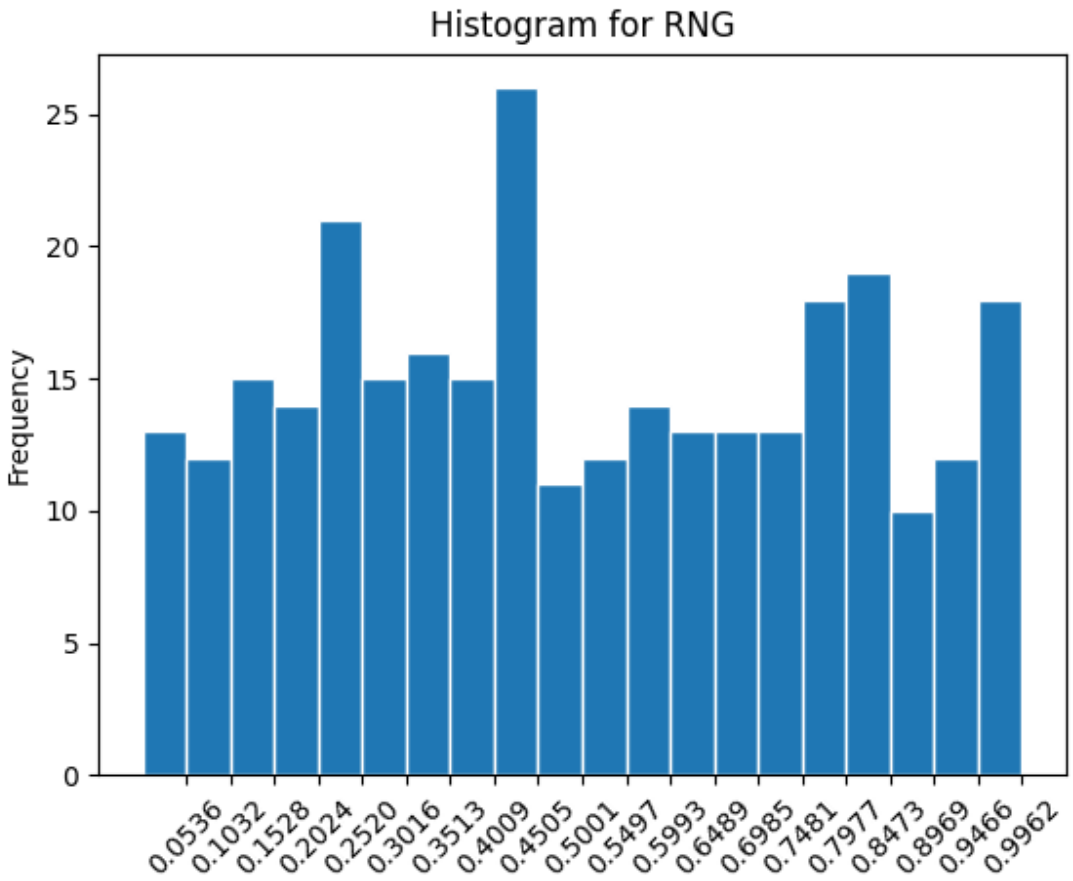
In order to confirm the uniformity of our random number generator, we must test our hypothesis of:

$$H_0: X\ conforms\ to\ a\ uniform\ distribution$$

$$H_1: X\ does\ not\ conform\ to\ a\ uniform\ distribution$$

To perform chi-square testing on our randomly generated numbers to determine how well they fit a uniform distribution, we must first divide our data into class intervals. Since our sample size

is 300, we can choose a number of class intervals in the range $n^{1/2}$ to n/5. The same as before, we will choose 20. Since the expected distribution is uniform, our expected frequency is 300/k which equals 15. The histogram representing this grouping is:



We can then calculate the chi-squared test statistic using this equation:

$$\chi_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

Doing this for our histogram, we get the chi-squared test statistic value of 18.53. We then need to compare this to $\chi^2_{0.5,18} = 28.869$.

Since $\chi^2_{0.5,18} > \chi^2_0$ the null hypothesis is not rejected, and the exponential distribution is determined to be a good fit for a uniform distribution with the confidence interval of 0.05.

Independence Testing

In order to verify for the independence of our random number generator, we must test our hypothesis of:

$$H_0: \rho_{il} = 0$$

$$H_1: \rho_{il} \neq 0$$

In order to calculate this, we must first determine $\hat{\rho}_{il}$ by using:

$$\hat{\rho}_{il} = \frac{1}{M+1}\left[\sum_{k=0}^{M} R_{i+kl}R_{i+(k+1)l}\right] - 0.25$$

We must also find $\hat{\sigma}_{\rho il}$ using:

$$\hat{\sigma}_{\rho il} = \frac{\sqrt{13M+7}}{12(M+1)}$$

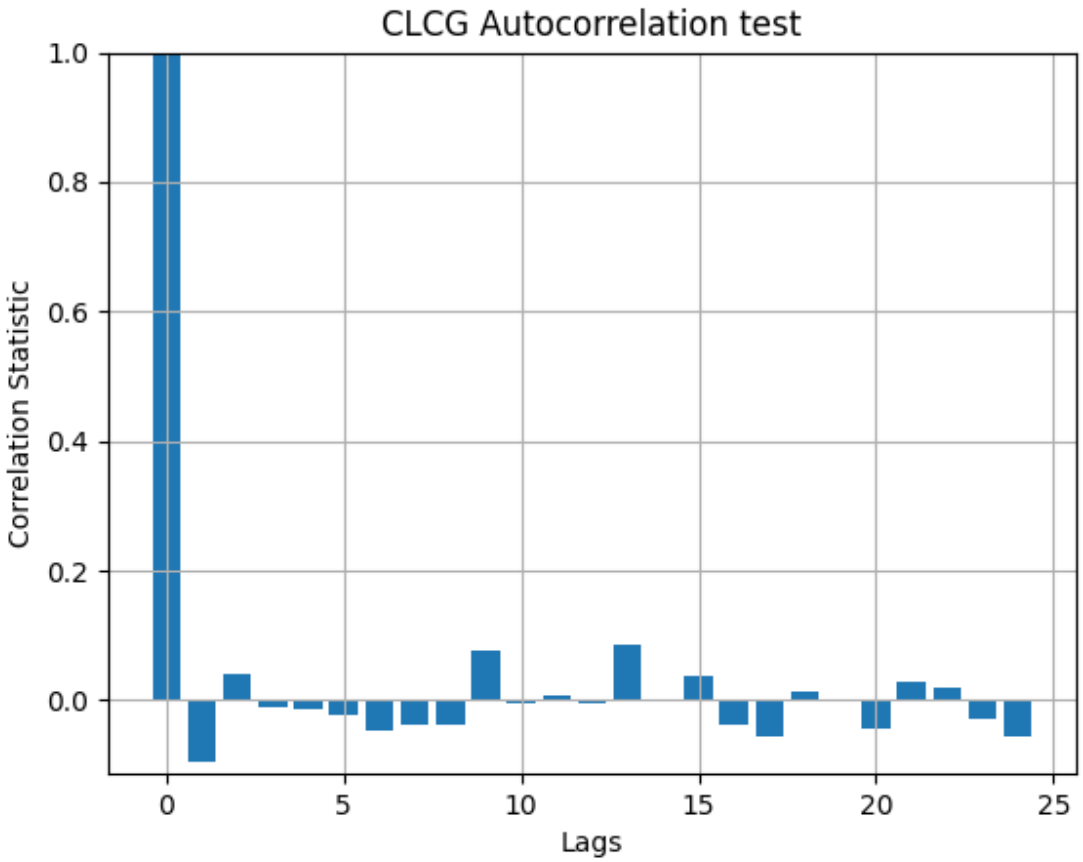
Doing this, we can calculate our test statistic by dividing these two quantities. From Table A.3, we can determine our z score for 0.025 confidence interval to be 1.96. If our z score from our autocorrelation test is less than this, we fail to reject the null hypothesis.

α = 0.05, i = 5, l, = 5 N = 300, and M = 58

$$\hat{\rho}_{5,5} = \frac{1}{58+1}\left[\sum_{k=0}^{58} R_{5+5k}R_{5+5(k+1)}\right] - 0.25 = \quad -0.16494$$

$$\hat{\sigma}_{\rho il} = \frac{\sqrt{13*58+7}}{12(58+1)} = 0.03896$$

Diving these two together, we get -0.16494 / 0.04316 = −4.2336, therefore we fail to reject the null hypothesis. I also had this test repeated for every lag value from 1 to 25 in order to find any evidence of correlation. As we can see, there isn't any significant deviation through the lag values, so we can conclude that the values are not correlated.



RVG Generation

In order to generate the appropriate RVG using the CDF technique, we must first use our random number generator to produce a value between 0 and 1. This value must be from a uniform distribution, which we confirmed in the previous step. Since our target distribution is exponential, we can calculate the inverse cumulative distribution function using:

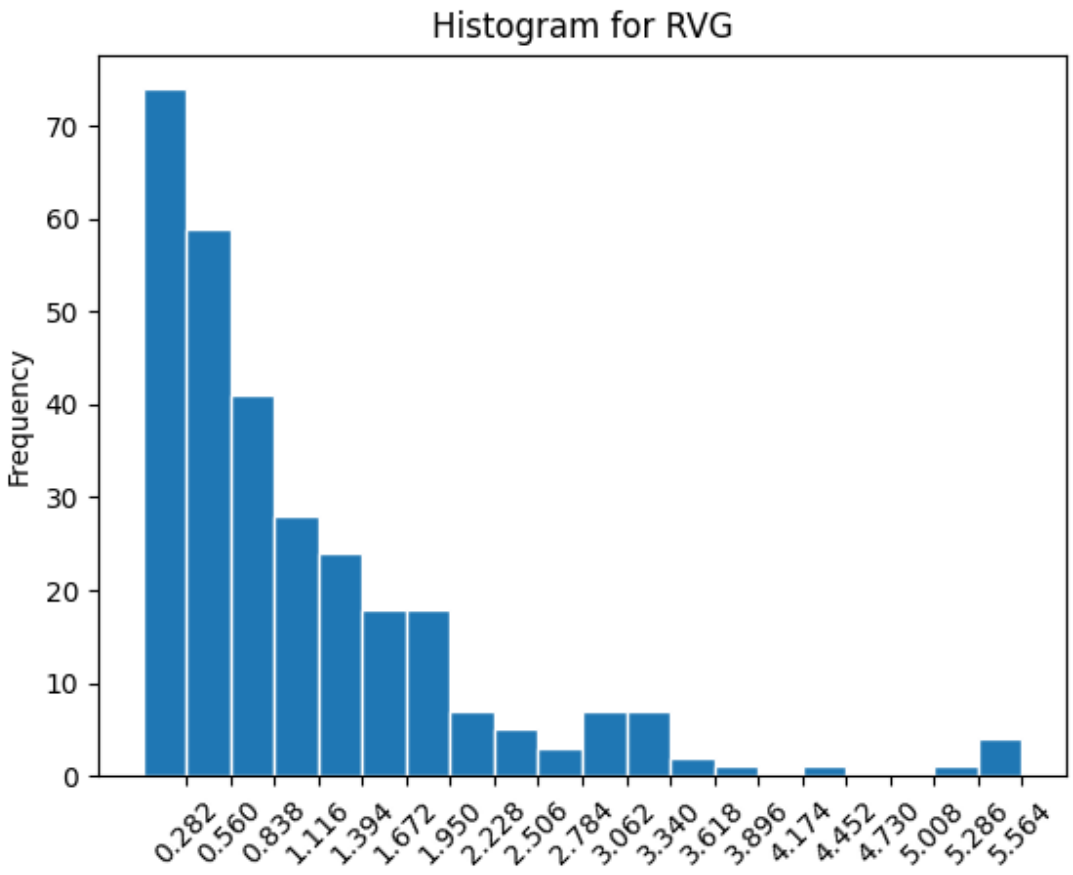
$$F^{-1}(U) = -\frac{log(1-U)}{\lambda}$$

Where lambda represents the rate parameter of the exponential distribution. This was estimated for each sample file earlier, so it can be used in our RVG for the main simulation. Implementing this in code is fairly simple where U is our uniform random number and y is lambda:

```
return -np.log(1 - U) / y
```

Running this function on our random uniform distribution we generated earlier, we can create the following histogram using a lambda value of 1 as an example which follows an approximately logarithmic distribution:

Chase Scott - Emmitt Luhning



Verifying a model is an important step in the simulation process. It ensures that the model accurately reflects the real-world system being studied and that it is reliable for making predictions and drawing conclusions. There are several validation techniques that can be used to assess the accuracy and reliability of the simulation model.

One method that can be used to validate the system is to check the output for reasonableness. This involves looking at specific variables to ensure that they fall within a reasonable range. For example, our average buffer occupancies for each workstation's buffers must be between 0 and 2. Similarly, the throughput cannot be negative, and the average components in the system cannot be less than 2 and more than 15. The values can be seen throughout the report as they are all used throughout the statistical analysis.

Furthermore, another validation technique that can we used is to perform a trace of the simulation run. This involves examining the current contents and total counts of variables throughout the simulation run. Analytical results can also be used to validate the model. This includes comparing the simulation results using statistical tests like Little's Law, and confidence interval testing which will be seen later in this report.

These validation techniques can be used to assess the accuracy and reliability of a simulation model. By using a combination of these techniques, it is possible to ensure that the model is robust and can be used to make informed decisions.

Code Verification by Little's Law

Little's Law relates the number of entities in the system (L) to the arrival rate and the amount of time each entity spends in the system. To verify the system is behaving as expected, we have evaluated little's law for two levels of granularity. First for the entire system, including the inspectors, workstations, and their buffers, and then for each individual workstation and its associated buffers.

If the system is behaving as expected, the average number of components in each of these systems should equal to the arrival rate multiplied by the amount of time each component spends in the system, barring some slight variation caused by rounding error. To get the values, the system was run for eight replications, and the values were only measured beyond the deletion point as determined by initialization analysis in the following sections. The arrival rate was calculated by measuring the total number of components that arrive at each inspector, or buffer divided by the time. However, it could also have been measured by analyzing the throughput of the components that left the system, as, for a steady state system throughput is equal to the arrival rate.

Chase Scott - Emmitt Luhning

Entire Facility

- Arrival Rate: 0.10765805429501399
- Amount of time component Spends in System: 51.6897127104437
- Average number of entities in system: 5.572567720300621

Little's Law: 0.107658*51.6897 = 5.572568

5.5648 = 5.5727

Workstation 1

- Arrival Rate 0.08525171777719182
- Amount of time component Spends in System 6.044015773077008
- Average number of entities in system 0.5152603555791313

Little's Law: 0.08525*6.0440 = 0.51526

0.51526 = 0.51526

Workstation 2

- Arrival Rate 0.011252449842275126
- Amount of time component Spends in System 96.44537860522436
- Average number of entities in system 1.0836005232596713

Little's Law: 0.011252* 96.445379 = 1.0836005232596713

1.08543 = 1.0836

Workstation 3

- Arrival Rate 0.011134323374336744
- Amount of time component Spends in System 177.7061792973173
- Average number of entities in system 1.973706841461812

Little's Law: 0.0111343 * 177.706 = 1.97371

1.9786 = 1.9737

It can therefore be seen that the values calculated through the use of little's law match very closely to the values obtained through measuring the results of the simulation.

Chase Scott - Emmitt Luhning

In order to show that the number of replications is enough, we can use an operating characteristic curve to find the minimum sample size needed to detect that effect size with a certain level of statistical power and significance level. Statistical power refers to the ability of a test to detect a true effect when it is present, while the significance level is the probability of rejecting the null hypothesis when it is true. By choosing an appropriate number of replications, we can increase the power of our analysis and reduce the likelihood of making a type II error (i.e., failing to detect a true effect).

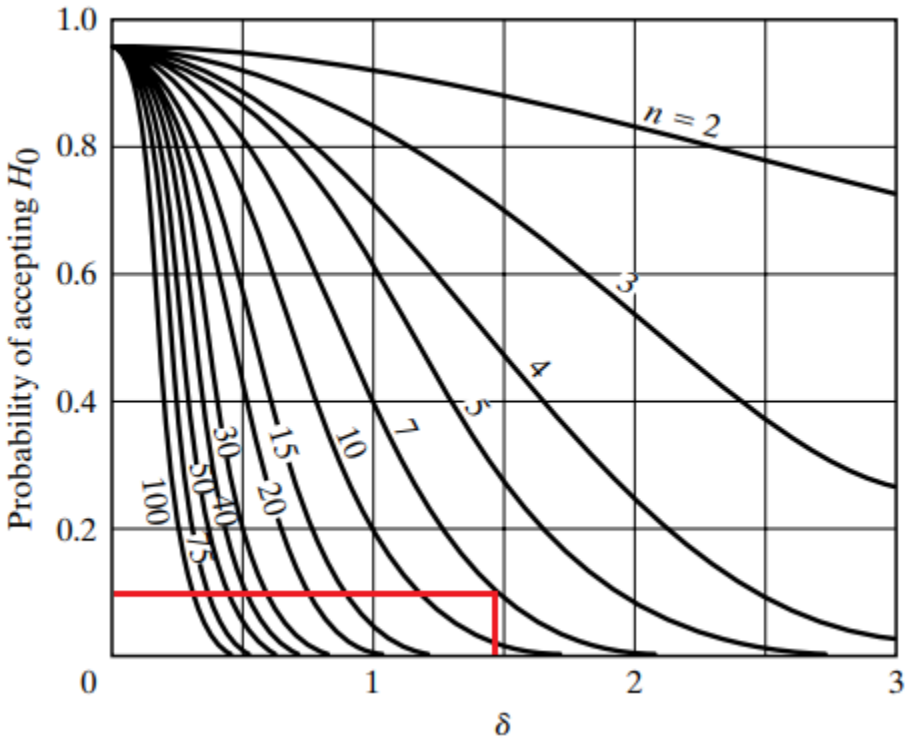
Estimating $\widehat{\sigma} = S = 0.0556209$. This was determined from our simulation.

Specify the critical difference $\left|E(Y_2) - \mu_0\right| = 0.082$

Calculate:

$$\widehat{\delta} = \frac{0.082}{0.0556209} = 1.47427$$

Find our sample size n by look up operating characteristic curve with α = 0.05 and β = 0.1. The OC curve plots the probability of correctly rejecting the null hypothesis (i.e., detecting a true difference) against the sample size for different effect sizes:



Therefore we can conclude that n ≅ 7 replications is a good starting point for our analysis.

We assume that there is a known output performance measure for the existing system. This was provided in the following table as the average buffer occupancy for each workstation:

C1 Buffer of Workstation 1	C1 Buffer of Workstation 2	C2 Buffer of Workstation 2	C1 Buffer of Workstation 3	C3 Buffer of Workstation 3
0.28	0.41	0.60	0.32	1.75

This value will be denoted by μ_0, and our unknown performance measure of the simulation will be denoted by μ. The confidence-interval formulation tries to bound the difference $|\mu - \mu_0|$ to see whether it is ≤ ε. The requirement that confidence intervals do not exceed 20% of the estimated value can only be controlled by the number of replications, so we'll do 20% * the average buffer occupancy for each buffer. For C1 buffer of workstation 2 this is 0.41
* 0.2 which is 0.082. The results for each buffer from the simulation for 6 replications are as follows:

W1:C1 Average Buffer Occupancy: 0.12205

W1:C1 Standard Deviation: 0.00571

W2:C1 Average Buffer Occupancy: 0.43097

W2:C1 Standard Deviation: 0.05562

W2:C2 Average Buffer Occupancy: 0.56418

W2:C2 Standard Deviation: 0.08180

W3:C1 Average Buffer Occupancy: 0.01823

W3:C1 Standard Deviation: 0.00695

W3:C3 Average Buffer Occupancy: 1.86116

W3:C3 Standard Deviation: 0.03838

Then, if Y is the simulation output and μ = E(Y), then we execute the simulation *n* number of replications and form a confidence interval for μ, such that $\overline{Y} \pm t_{\alpha/2,n-1} S/\sqrt{n}$. Here is the sample calculation for 6 replications on W2 C1 buffer:

$$\overline{Y} \pm t_{\alpha/2,n-1} S/\sqrt{n} \Rightarrow 0.4309748 \pm t_{0.05/2,6-1} 0.0556209/\sqrt{6}$$

Using a t-distribution table or a calculator, we can find that $t_{0.025,6-1}$ = 2.447 for a double sided t-test.

$$0.4309748 \pm 2.447 * 0.0556209/\sqrt{6} = (0.37541,\ 0.486539)$$

Therefore, with 95% confidence, the true population mean μ is expected to fall within the range of $0.37541 - 0.486539$. Our confidence interval contains μ_0, so we must find the best and worst case error next. The best case error is |0.41 - 0.37541| = 0.0346, and the worst case error is |0.4865 - 0.41| = 0.0765. Our worst case error of 0.0765 is less than our ε value of 0.082, so we can conclude that the simulation model is close enough to be considered valid at 6 replications.

This procedure was repeated for the other buffers. We got acceptable confidence range measurements for w2c2 and w3c3, but not for w1c1 and w3c1. We were told that only the procedure would be marked, not numbers because the data is sensitive and can be quite variable with minor implementation changes, we will stop at 6 replications. The confidence intervals for the other 4 buffers are as follows:

W1 C1: $0.12205 \pm 2.447 * 0.0057075/\sqrt{6} = (0.11635,\ 0.12775)$

W2 C2: $0.56418 \pm 2.447 * 0.081799/\sqrt{6} = (0.48246,\ 0.645896)$

W3 C1: $0.018231 \pm 2.447 * 0.0069501/\sqrt{6} = (0.011288,\ 0.025174)$

W3 C3: $1.8612 \pm 2.447 * 0.038379/\sqrt{6} = (1.82286,\ 1.89954)$

To determine the initialization phase, a Welch plot can be used. By calculating the average of a performance metric across all replications, and plotting it, one can see where the data starts to vary around a consistent mean. It is after this point that the data can start to be collected. For the Welch plot, the time for a component to pass through the system was considered as the performance metric, and measured against the total

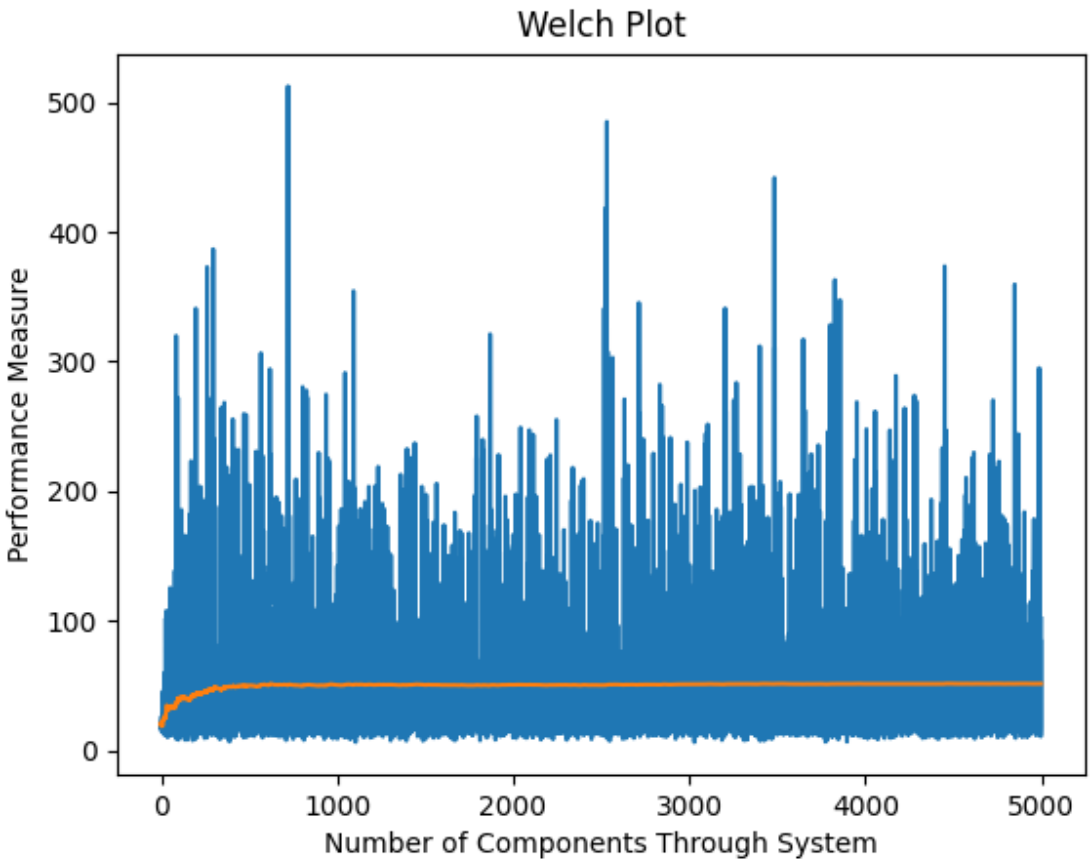
number of components that have passed through the system to represent the time that the system has been alive. Six replications were adequate to view the initialization period.

The averages plotted were determined using the following equation:

$$Y.j = \frac{1}{R}\sum_{r=1}^{R} Y_{rj}$$

Where $Y.j$ represents the average of the value across all replications.

The data can be noisy, so to smooth it, a cumulative average of the Welch averages can be calculated for each point and plotted alongside. This is represented by the orange line below, and shows the initialization period clearly.



From this data, we can determine that an initialization phase of about 1000 components having passed through the system is adequate, therefore we will set d = 1000. Knowing

that Te should be at least 10 times greater than Td, the phase for data collection will be set to be 10 000

Replication Method - Sample Calculation for Component Time in System

First step: get averages of all non-deleted data, for each replication.

$$Y_r(n,d) = \frac{1}{(n-d)} \sum_{j=d+1}^{n} Y_{rj}$$

For replication 1.

Replication	$Y_r(n,d)$
1	52.023904404742765
2	51.908259952716485
3	51.14631579340045
4	51.41186974742246
5	52.817739924031294
6	51.606831811349075

Calculate overall point estimator

$$Y_{..}(n,d) = \frac{1}{R} \sum_{r=1}^{R} Y_r(n,d)$$
$$= 51.81915361$$

Calculate standard variance

$$S^2 = \frac{1}{R-1} \left(\sum_{r=1}^{R} (Y_r - Y..)^2 \right)$$
$$S^2 = \frac{1}{6} \left(\sum_{r=1}^{6} (Y_r - 51.81915361)^2 \right)$$
$$= 0.2851181455$$

Calculate Standard Error

$s.e(Y..) = \frac{S}{\sqrt{R}} = 0.5339645546 / \sqrt{6} = 0.2179901166$

Using 95% confidence, alpha = 0.5, and R - 1 = 5

$t_{0.025,5} = 2.571$ leaving the confidence interval

$51.2595 < w < 52.3804$

Replications required to reduce the interval to range of 0.2

$R >= (\frac{z_{0.025} S_0}{0.2})^2 = (\frac{1.96 * 0.2851181455}{0.2})^2 = 7.8$

Therefore, 8 replications would be required

Alternative Operating Policy

The current system spends much of its time blocked, as W1 uses so many C1 components, W2 and W3 often have long wait-times to receive the components they need from Inspector 1. In order to solve that, we propose a policy wherein, instead of providing C1 components to the buffer with the least current components, the system simply provides components to the buffer's in order of w1, w2, and w3, before rolling back to w1. This ensures that all of the workstations get an equal number of C1 buffers, and no buffer is starved.

To compare the difference in the system for each alternative, let's consider the percent of time each of the inspectors spent blocked. The old operating time resulted in the following:

```
Inspector 1 blocked times 0.0
Inspector 2 blocked times 0.9088432956904341
```

Now, let's consider the new operating policy with respect to blocked times.

```
Inspector 1 blocked times 0.0
Inspector 2 blocked times 0.6395168591336154
```

To compare these results, we can take the confidence interval of the difference.

$$\bar{Y}_{.1} - \bar{Y}_{.2} \pm t_{\alpha/2,\upsilon} s.e.(\bar{Y}_{.1} - \bar{Y}_{.2})$$

In order to do this, we must calculate the standard error, and the degrees of freedom for the data. Where standard error is given by:

$$s.e.\left(\bar{Y}_{.1} - \bar{Y}_{.2}\right) = \sqrt{\frac{S_1^2}{R_1} + \frac{S_2^2}{R_2}}$$

And degrees of freedom are given by:

$$\upsilon = \frac{\left(S_1^2 / R_1 + S_2^2 / R_2\right)^2}{\left[\left(S_1^2 / R_1\right)^2 / (R_1 - 1)\right] + \left[\left(S_2^2 / R_2\right)^2 / (R_2 - 1)\right]},$$

For data collection, 8 replications were used, and the means of the blocked time for inspector two are 0.90844 and 0.6395 respectively. Using these values, this results in the following standard error and degrees of freedom:

This results in a standard error of 0.39278, and 81 degrees of freedom.
That leaves the t value to be: 1.990 for an alpha of 0.5.

These leaves the confidence interval to be

0.268941 +/- 0.10563487

Lower bound difference results in 0.163, which equates to a 17.97% reduction in blocked time for inspector 2.

Appendix

Chi-Square Calculations

Servinsp1

interval	actual	expected	calculation
0.0 - 0.5312913268697537	16	15	0.06666666667
0.5312913268697537 - 1.0913147387373547	7	15	4.266666667
1.0913147387373547 - 1.6833564450	11	15	1.066666667

342968			
1.683356445034296 8 - 2.311300821592964 5	10	15	1.666666667
2.311300821592964 5 - 2.979785015069024	15	15	0
2.979785015069024 - 3.694406968572433 7	14	15	0.0666666667
3.694406968572433 7 - 4.462010674423191	18	15	0.6
4.462010674423191 - 5.291085836661989	16	15	0.0666666667
5.291085836661989 - 6.192341848496644	19	15	1.066666667
6.192341848496644 - 7.179556112993658	17	15	0.2666666667
7.179556112993658 - 8.270870851731015	15	15	0
8.270870851731015 - 9.490856934586626	19	15	1.066666667
9.490856934586626 - 10.873963081	18	15	0.6

566092			
10.873963081566092 - 12.470641949655649	14	15	0.0666666667
12.470641949655649 - 14.359112225987316	21	15	2.4
14.359112225987316 - 16.670413047580283	13	15	0.2666666667
16.670413047580283 - 19.650198062649313	13	15	0.2666666667
19.650198062649313 - 23.84996916057394	20	15	1.666666667
23.84996916057394 - 31.02952527356761	12	15	0.6
> 31.02952527356761	12	15	0.6
	300	300	16.66666667

Servinsp22

interval	actual	expected	calculation
0.0 - 0.7969389565	16	15	0.0666666667

47582			
0.796938956547582 - 1.636976146925798 8	7	15	4.266666667
1.636976146925798 8 - 2.525040897443742 6	11	15	1.066666667
2.525040897443742 6 - 3.466959786225580 3	10	15	1.666666667
3.466959786225580 3 - 4.469688550416313	15	15	0
4.469688550416313 - 5.541624125398168	14	15	0.0666666667
5.541624125398168 - 6.693032524979904	18	15	0.6
6.693032524979904 - 7.936648336641895	16	15	0.0666666667
7.936648336641895 - 9.288535689829995	19	15	1.066666667
9.288535689829995 - 10.76936074013240 8	17	15	0.2666666667
10.76936074013240 8 - 12.406336887	15	15	0

05821			
12.4063368870 5821 - 14.2363205260 357993	19	15	1.066666667
14.2363205260 357993 - 16.3109848650 53058	18	15	0.6
16.3109848650 53058 - 18.7060090760 774304	14	15	0.0666666667
18.7060090760 774304 - 21.5387214800 264815	21	15	2.4
21.5387214800 264815 - 25.0056812660 4904	13	15	0.2666666667
25.0056812660 4904 - 29.4753698160 90672	13	15	0.2666666667
29.4753698160 90672 - 35.7750420060 62281	20	15	1.666666667
35.7750420060 62281 - 46.5444027460 75523	12	15	0.6
> 46.5444027460 75523	12	15	0.6
	300	300	16.66666667

Servinsp23

Chase Scott - Emmitt Luhning

interval	actual	expected	calculation
0.0 - 1.0583220617 3003	15	15	0
1.0583220617 3003 - 2.1738778818 42453	12	15	0.6
2.1738778818 42453 - 3.3532135260 547458	15	15	0
3.3532135260 547458 - 4.6040665960 01935	15	15	0
4.6040665960 01935 - 5.9356741982 599655	15	15	0
5.9356741982 599655 - 7.3591873273 84841	9	15	2.4
7.3591873273 84841 - 8.8882390838 92696	23	15	4.266666667
8.8882390838 92696 - 10.539740794 261903	14	15	0.0666666667
10.539740794 261903 - 12.335025362 92053	9	15	2.4
12.335025362 92053 -	14	15	0.0666666667

14.30153711067941 4			
14.30153711067941 4 - 16.47541499252187	18	15	0.6
16.47541499252187 - 18.90560370668135 8	18	15	0.6
18.90560370668135 8 - 21.66072443806425 7	21	15	2.4
21.66072443806425 7 - 24.84127790494131 8	14	15	0.06666666667
24.84127790494131 8 - 28.60307422135882 8	10	15	1.666666667
28.60307422135882 8 - 33.20714081736077	20	15	1.666666667
33.20714081736077 - 39.14281501562075	15	15	0
39.14281501562075 - 47.50867792804018 6	14	15	0.06666666667
47.50867792804018 6 - 61.81021503871962	14	15	0.06666666667
> 61.81021503 8	15	15	0

71962			
	300	300	16.93333333

W1

interval	actual	expected	calculation
0.0 - 0.23617569956627757	6	15	5.4
0.23617569956627757 - 0.4851237143034894	12	15	0.6
0.4851237143034894 - 0.7483048676283796	18	15	0.6
0.7483048676283796 - 1.0274458867303353	21	15	2.4
1.0274458867303353 - 1.3246081290981868	18	15	0.6
1.3246081290981868 - 1.642280056453898	17	15	0.2666666667
1.642280056453898 - 1.9835040385713643	19	15	1.066666667
1.9835040385713643 - 2.352054015828523	16	15	0.0666666667

2.352054015828523 - 2.752690650229 19	13	15	0.2666666667
2.752690650229 19 - 3.191538430623 22	15	15	0
3.191538430623 22 - 3.676662144926 71	23	15	4.266666667
3.676662144926 71 - 4.218984317353 557	11	15	1.066666667
4.218984317353 557 - 4.833818487077 118	13	15	0.2666666667
4.833818487077 118 - 5.543592446451 743	11	15	1.066666667
5.543592446451 743 - 6.383076861246 44	10	15	1.666666667
6.383076861246 44 - 7.410522747976 776	20	15	1.666666667
7.410522747976 776 - 8.735130877074 965	9	15	2.4
8.735130877074 965 - 10.60206117859 9996	17	15	0.2666666667

10.602061178599996 - 13.793599609223222	13	15	0.2666666667
> 13.793599609223222	18	15	0.6
	300	300	24.8

W2

interval	actual	expected	calculation
0.0 - 0.568976339278639 3	19	15	1.066666667
0.568976339278639 3 - 1.168722758389441 3	20	15	1.666666667
1.168722758389441 3 - 1.802758560806548	17	15	0.2666666667
1.802758560806548 - 2.475243644931677 7	17	15	0.2666666667
2.475243644931677 7 - 3.191144074759107 3	8	15	3.266666667
3.191144074759107 3 - 3.956454860967742 3	10	15	1.666666667
3.956454860967742 3 - 4.778505446	18	15	0.6

33266			
4.778505446933266 - 5.666387719690786	17	15	0.2666666667
5.666387719690786 - 6.631570700161799	15	15	0
6.631570700161799 - 7.688809036060 4515	13	15	0.2666666667
7.688809036060 4515 - 8.857531794449894	19	15	1.066666667
8.857531794449894 - 10.164052680992134	16	15	0.0666666667
10.164052680992134 - 11.645263897028196	15	15	0
11.645263897028196 - 13.355196755751239	14	15	0.0666666667
13.355196755751239 - 15.377618072120903	17	15	0.2666666667
15.377618072120903 - 17.852861717052583	9	15	2.4
17.852861717052583 - 21.044005791	8	15	3.266666667

811696			
21.044005791811696 - 25.541670753113035	11	15	1.066666667
25.541670753113035 - 33.230479789 1735	15	15	0
> 33.230479789 1735	22	15	3.266666667
	300	300	20.8

W3

interval	actual	expected	calculation
0.0 - 0.4511542742 4925236	7	15	4.266666667
0.4511542742 4925236 - 0.9267068443 096642	22	15	3.266666667
0.9267068443 096642 - 1.4294482459 120401	13	15	0.2666666667
1.4294482459 120401 - 1.9626769570 682376	16	15	0.0666666667
1.9626769570 682376 - 2.5303306828 154364	11	15	1.066666667
2.5303306828 154364 - 3.1371630034	11	15	1.066666667

08638			
3.137163003408638 - 3.788985601124472	16	15	0.06666666667
3.788985601124472 - 4.493007639883675	21	15	2.4
4.493007639883675 - 5.258323167106123	18	15	0.6
5.258323167106123 - 6.096631478389447	22	15	3.266666667
6.096631478389447 - 7.023338322699113	15	15	0
7.023338322699113 - 8.059308435457687	18	15	0.6
8.059308435457687 - 9.233794481798085	13	15	0.2666666667
9.233794481798085 - 10.589639118273123	13	15	0.2666666667
10.589639118273123 - 12.193262956778893	9	15	2.4
12.193262956778893 - 14.155939913	11	15	1.066666667

847133			
14.155939913847133 - 16.686270596662574	16	15	0.06666666667
16.686270596662574 - 20.252571392236582	16	15	0.06666666667
20.252571392236582 - 26.349202870626037	17	15	0.2666666667
> 26.349202870626037	15	15	0
	300	300	21.33333333

X2,0.05 = 28.869

References

[1] Sturges, H. A. (1926). The choice of a class interval. Journal of the American Statistical Association, 21(153), 65-66.
[2] P. L'Ecuyer, "Efficient and portable combined random number generators," Communications of the ACM, vol. 31, no. 6, pp. 742-749, June 1988
[3] "Discrete time simulation inspector handling" YouTube, uploaded by Tadhg McDonald-Jensen, Feb 21, 2023, 40:48, https://www.youtube.com/watch?v=KaG1uRvW2_A.