## Abstract

Monte Carlo simulation is an extremely useful method to analyze complex mathematical relationships. When used to investigate a stochastic process, Monte Carlo simulation allows a researcher to analyze thousands of independent random outcomes given underlying distributional assumptions. In this paper, we model computer job queueing as a Poisson process. We use a Poisson distribution to estimate the number of jobs and an exponential distribution to estimate the lifetime of each respective job. We investigate the number of jobs created given these assumptions by simulating outcomes using different values of the Poisson lambda and exponential rate parameters. We find that the ratio of the two distribution parameters is more important than two discrete values in determining the number of jobs created. Thus, simulation parameters can be simplified to use only the ratio of the two parameters instead of iterating over hundreds of combinations.

## Introduction

In the context of software engineering, it is good practice to anticipate the likelihood of a release date, the cost of a project, or the impact of implementing new code base. These types of exercises usually fall into the category of risk analysis. However, it is often extremely difficult to successfully identify or anticipate outcomes if those outcomes are considered mostly random. One such software engineering example producing random outcomes is computer job queueing.

Even with parallel processing, a queue of interdependent tasks can form, often slowing down processing time (Nolan, et. al 1). The number of these tasks can be completely random, specifically when considering the amount and type of requests the computer must consider. Typically, a parent job generates children which then generate additional children recursively. Each one of these jobs have a distinct run time that begins after the completion of their respective parent job. Obviously, this queueing process can result in significant slowdown times, therefore it is often useful to simulate the outcomes of this branching process to optimize code and job processing logic.

In order to model the queueing process, we must utilize probability distributions to estimate the number of jobs created and their associated lifetimes. Given the randomness of the queueing process, we explore many different outcomes utilizing these probability distributions via Monte Carlo simulation. These simulations allow us to explore the distributions of job counts our queue model produces.

Of particular importance are the associated distributional parameters that must be identified as part of our simulation analysis. Our objective in this paper is to determine if these parameters and their values are related to the longevity of the queueing process. Specifically, we investigate whether discrete parameter values or the ratio of the two parameters is more important to the longevity of the queueing process.

In the following section, we review academic literature on stochastic processes and Monte Carlo simulation. In the methods section, we provide a description of the probability distribution assumptions used to model the CPU queueing process. We also identify the parameter selections and logic used to test the sensitivity of our model given the distributional parameters. Results of Monte Carlo simulation for the queueing process and associated parameters are then presented and compared. We conclude this paper by summarizing results and considering future work.

## Background

Gabbiani and Cox (2) define a stochastic process as a collection of random variables indexed by a variable t, usually representing time. These processes are grouped into two categories: discrete-time and continuous-time stochastic processes. They are studied intensely as part of mathematical models to describe systems that occur in a random manner. The applications of these models can be found in computer science, cryptography, telecommunication, finance and many other academic disciplines and industries.

One of the easiest stochastic processes to understand is the Bernoulli process (3) as it only has two possible outcomes: success or failure, given probabilities of {p, 1-p}. For example, tossing a fair coin ten times can be modeled as a Bernoulli process with a sequence of ten coin-flip results and probabilities of 0.5 for head and 0.5 for tail. Each coin-flip is a binary-valued random variable.

Another example of a stochastic process is the Poisson process which we use in this paper to estimate the number and run time of jobs in a queueing process. It is used in instances when we need to count the number of occurrences of an event that happens at random. In practice, (Pishro-Nik 4) the Poisson process is used to model the number of car accidents at a site, the location of users in a wireless network, the request for individual documents on a web server and the outbreak of wars.

The Poisson process with rate is defined as:

Let > 0 be fixed.

The counting process { N(t), t [ 0, ) } is called a Poisson process with rate if all the following conditions hold:

1. *N*(0) = 0;

2. *N*(t) has independent increments;

3. the number of arrivals in any interval of length >0 has *Poisson*() distribution.

Modeling a random process given only a few samples of that respective process can lead to inconclusive or erroneous assumptions. In order to empirically analyze these random processes, Monte Carlo simulations are used frequently. This method is used to generate random variables for modeling risk or uncertainty of a system (5). Monte Carlo simulations have some key advantages over more deterministic analyses (6) including simulated outcomes and probabiities for each outcome. It is easy to generate graphical representations based on the results, and one can easily spot which output has the largest effect on the bottom line results. Monte Carlo simulations are used extensively to simulate outcomes, especially for electoral processes and in sports betting.

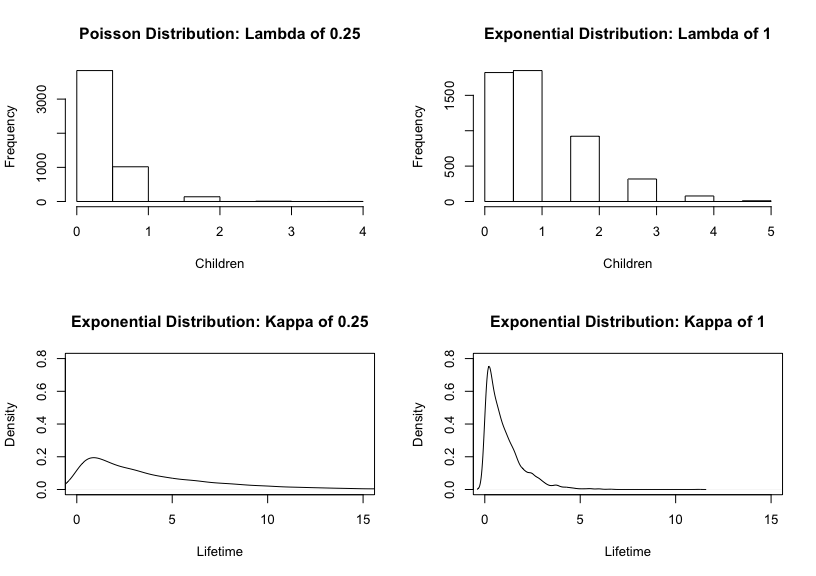
## Methods

We must identify a method to model the previously described queue forming process in order to analyze many random outcomes given a set of distribution parameters. Our model assumptions are as follows:

1. The queueing process begins with a single parent job
2. This parent (and all subsequent parents) have a randomly generated run time or “lifetime”
3. Each parent can randomly generate n amount of children jobs
4. All child jobs are “protected” and do not run until their parent job completes or “dies”
5. Each child can spawn n amount of child jobs
6. The process runs until no additional children are generated or infinitely

This model represents a branching process, the outputs of which are the number of generations and the number of offspring in a family tree of jobs. Starting with the parent, each job has a fixed lifetime that is determined randomly. Within this fixed interval of time, a parent can generate *n* amount of child jobs. Thus, we can use a Poisson process to model the counts of jobs over each fixed time interval. Specifically, we use the Poisson distribution to randomly generate the number of children for each parent given a rate parameter. Finally, each job runs for a certain amount of time, defining the time boundaries between parent end and child start times. We can randomly generate these run times using a random variable from the exponential distribution to determine the lifetime of each job. What results is a family tree with distinct generations containing *n* number of children. We can subsequently count these generations and the number of jobs to determine the longevity of our queueing process.

The Poisson and exponential distributions allow us to parameterize our queueing process. Each distribution contains a rate parameter, which represent the number of events per unit time. Specifically, we use lambda and kappa for our rate designations for the Poisson and exponential distributions, respectively. These rate parameters can elongate or shorten the queueing process dramatically. Obviously, if a parent generates many child jobs, the probability of a process continuing is greater. Further, if a job has a longer run time, the probability of a process continuing is also greater. It is the interaction of these two parameters we are interested in analyzing. As an example, we randomly generate 5000 samples from a Poisson and exponential distribution with rate parameters of 0.25 and 1 below.

Figure 1: Rate Parameter Comparisons from 5000 Randomly Generated Variables 

As can be seen in Figure 1, each distribution’s rate parameter can significantly change the outcome for the number of children and the lifetime for each job in our model. We seek to understand if the ratio of these parameters are of importance, or if one parameter dominates when considering the longevity of our queueing process.

In order to investigate the relationship between lambda and kappa in our queueing model, we carry out a Monte Carlo simulation study by fixing kappa, which represents the parameter for the lifetime of each job, to be one. We run 500 simulations to determine the number of child jobs created using each of ten increasing values of lambda, which represents the rate parameter for the number of offspring for each parent. We then run another set of simulations where kappa is not equal to one, however, the ratio of lambda to kappa is equal to one of the lambda values from the earlier simulation where kappa was equal to one. This allows us to analyze whether the relationship between these two parameters are important or if one or more parameters dominates the longevity of our queueing process.

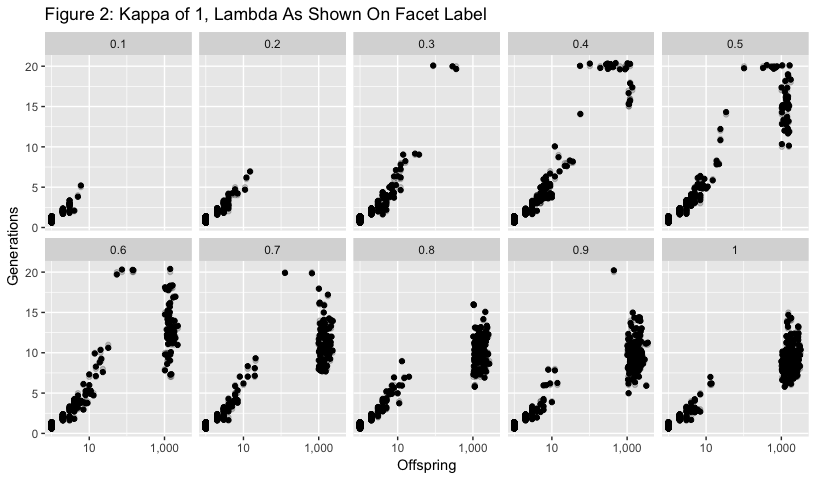
## Results

In order to test the relationship between the kappa and lambda parameters, we identified two test cases. In the first case, we hold kappa constant at one and test a range of lambda values from zero to one. We then double the lambda and kappa rates in the second case and examine comparisons between the two tests cases using Monte Carlo simulation. In both cases, kappa, or the rate parameter for the exponential distribution, is held as a constant *c*. This allows us to test the ratio of the two parameters directly. Additionally, it also ensures the ratio value of *k*/*c* is equal to one of the values of lambda from the first test case.

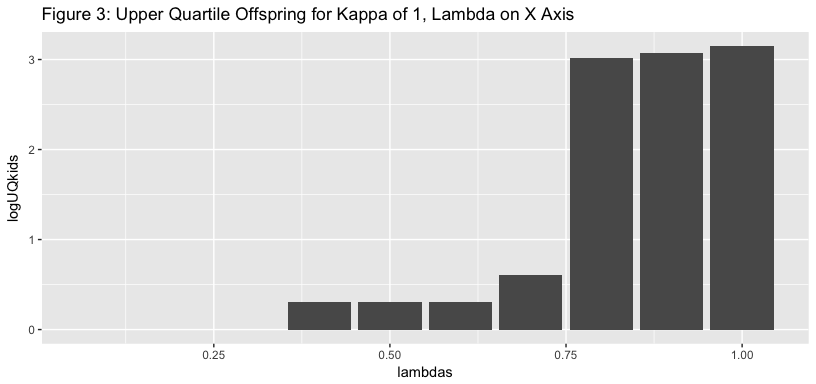
### Test Case One Results

Utilizing methods described in the previous section, we simulate our queueing process using Poisson lambda values between zero and one and a constant exponential rate parameter kappa of one.

Figure 2 clearly shows a low value of lambda results in smaller quantities of offspring when kappa, which controls the run time of each job, is held constant at one. This makes logical sense, as the lambda rate as part of the Poisson distribution controls the amount of offpsring each parent produces in our queueing process. As lambda rises, holding kappa constant, we notice that our queueing process runs up against the generational (20) and offspring boundaries (1000), indicating a longer queueing process.



The effect tapers off for generations above lambda = 1.0, with fewer generations, but more offspring produced for values of lambda above 1.2. We can aggregate this effect by examining the log upper quartile value for each lambda and kappa parameter value in Figure 3.



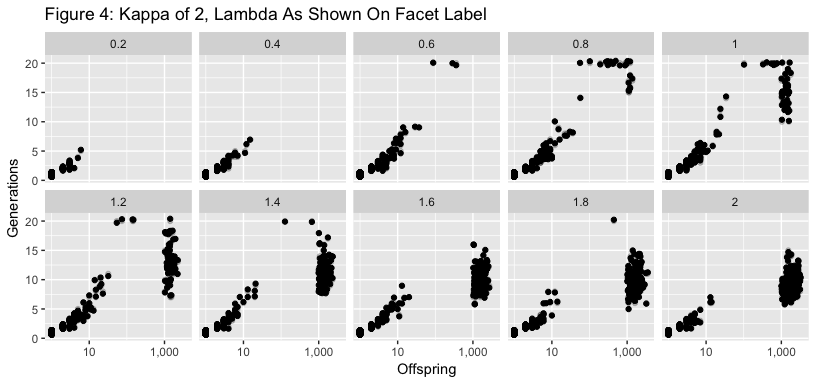
A large jump in offspring is noticeable when lambda crosses the 0.75 threshold. Given the Poisson rate parameter lambda has a significant influence on the number of offspring, we could hypothesize that raising lambda would ultimately result in a longer queueing process in most scenarios. However, we do not consider kappa, or the rate parameter for the lifetime of each job, in our simulation.

### Test Case Two Results

Previously, we noticed that a smaller value of kappa results in longer lifetimes for each job. However, we have just seen that increasing the lambda parameter controlling the number of offspring for each parent resulted in dramatic increases in processing time in our queueing simulations. We directly test whether one parameter has an effect on our queueing process, or if the ratio between kappa and lambda is more important.

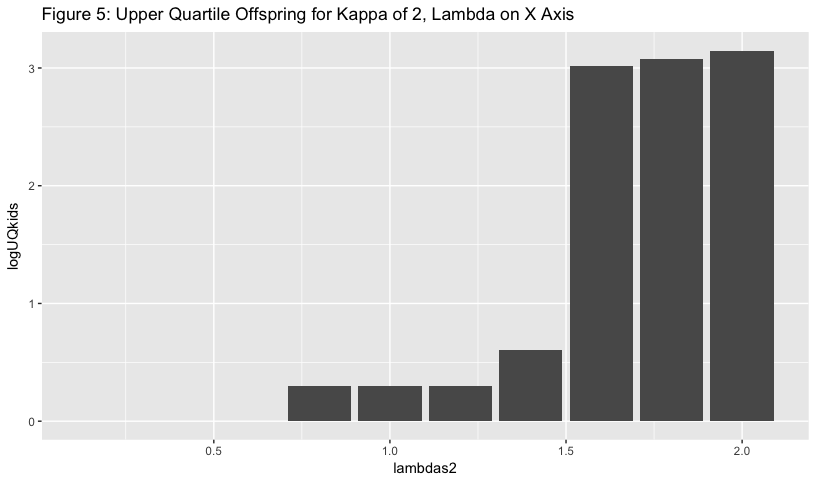
We double the lambda and kappa rate parameters for our second test case. This allows us to maintain the ratio between lambda and kappa from the first scenario, but also test whether or not lambda has a more dramatic effect on our queueing process longevity. Additionally, each ratio is equal to a lambda value from the previous test, to ensure we address the parameter ratio impact on our queueing process appropriately.

We test kappa using a constant of two and let lambda range from 0.2 to two. We iterate over each parameter combination and simulate offspring and generations from our queueing process 500 times. The offspring and generations output can be seen in Figure 4.



A comparison of the visualization of the two tests show nearly identical results for generations and offspring. This would lead us to conclude that the ratio of the two values, lambda and kappa, are the determinants of the number of generations for a given number of offspring. One variable does not dominate the results of our queueing process given our simulation exercise. The process shows similar clusters produced - a number of examples of the product of the Monte Carlo process producing up to ten offspring, and a number of iterations on larger values of lambda producing more than 1,000 offspring.

Additionally, the bar chart for the upper quartile of offspring produced for each simulated parameter combination result is nearly identical to test case one.



## Conclusions and Future Work

After investigating two tests cases, we have empirically confirmed that the ratio of parameters for the queueing process drives the longevity of the process. This result is in line with expectations. The exponential distribution shows longer lifetimes for jobs when its rate parameter is small. The Poisson distribution produces more children for each parent when its parameter is larger. Thus, there is a balance between these two parameters that must be attained in order to lengthen the queueing process.

For a software engineer, these results could help optimize parallel processing and cpu load. For instance, code could be written to optimize application performance by offloading processes to underutilized cores if the probability of branching is high. We know a process is more likely to extend if a parent job produces more children or if each job tends to have a longer run time. These characteristics can be analyzed dynamically and sent to the cpu in a structured and efficient manner to optimize available processing cycles. Software engineers could easily apply sensitivity analysis using Monte Carlo simulation while analyzing parameter variables for assumed distributions in order to determine worst and best case scenarios for application performance.

The use of Monte Carlo simulation has become an increasing part of the public lexicon in recent years. Many models that are visible to the public include at some level a Monte Carlo simulation. These include election results modeling, as well as the outcomes of major sports leagues. Fivethirtyeight.com’s models for both the election, as well as for results of sports leagues, include Monte Carlo simulations (Boice 7). The method has also been mentioned in the context of quantitative finance, genetics and many other fields.

The pitfalls of Monte Carlo simulation may be somewhat obvious. It relies on an assumption that the future follows some probabilistic distribution, and that the future distribution is consistent with the past values of the variable. Like most statistically-based forecast methods, there’s some expectation that major shifts in the underlying causes of change in the behavior of the variable will remain the same. In the example of a model for a Financial Market, while the major shifts and catastrophic market events are possible in the simulation, the actual probability of the event might differ from the *n*-standard deviations from the mean that would be expected.

In this case, we simulated a process of birth and death of jobs for a central processing unit, with dependencies between the start and finish. We attempted to understand the total time required for a queueing process. In the context of the case, these events were referenced as jobs or processes that are broken up for parallel computing, and the process by which the later jobs must wait for the completion of earlier jobs. This same method could be employed for the analysis of a manufacturing line. In manufacturing, the Monte Carlo process can be used to simulate a process with interdependencies, and variability. These studies allow for a greater understanding of risks specific to a process or even a business model. These simulations can help to enhance enterprise resource planning, supply chain continuity and minimize financial losses (Heffernan 8).

## 

## References

1. Nolan, D., Temple Lang, D. DATA SCIENCE IN R: a Case Studies Approach to Computational Reasoning and Problem Solving. CRC PRESS, 2017.
2. F. Gabbiani, S. Cox. “Mathematics for Neuroscientists”" (Second Edition). 2017.
3. “Bernoulli Distribution: Definition and Examples.” Statistics How To, 29 July 2016. www.statisticshowto.com/bernoulli-distribution/.
4. H. Pishro-Nik. “Basic Concepts of the Poisson Process.” Probability Course. <https://www.probabilitycourse.com/chapter11/11_1_2_basic_concepts_of_the_poisson_process.php>
5. “Definition of Monte Carlo Simumation.” The Economic Times. <https://economictimes.indiatimes.com/definition/monte-carlo-simulation>
6. “Monte Carlo Simulation.” Palisade, 2018. <http://www.palisade.com/risk/monte_carlo_simulation.asp>
7. Boice, J. “How Our Soccer Projections Work.” FiveThirtyEight, 19 Jan 2017. <https://fivethirtyeight.com/features/how-our-club-soccer-projections-work/>
8. Heffernan, R. “Monte Carlo and Manufacturing.” Quality Magazine, 1 April 2014. <https://www.qualitymag.com/articles/91769-monte-carlo-and-manufacturing>