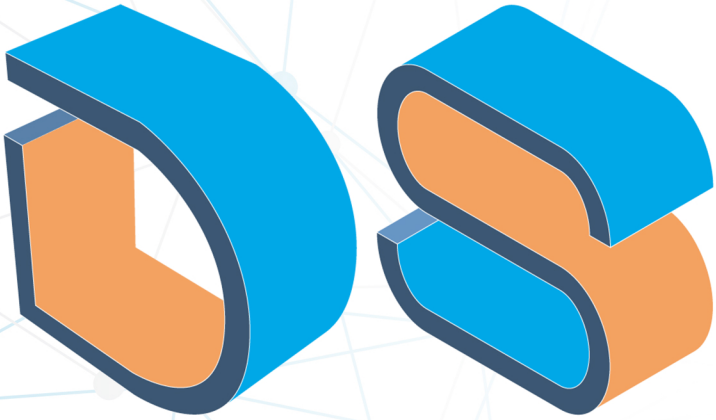


POLIMI
DATA SCIENTISTS

Performance Evaluation and Applications

Course Notes

Edited by:
Simone Drago

This course notes have been developed by **Polimi Data Scientist** staff.

Are you interested in **Data Science** initiatives at PoliMi?

follow
Polimi Data Scientist
on Facebook!

Credits

The following notes have been written by the Polimi Data Scientists student association by combining the slides and lectures of Prof. Marco Gribaudo and the notes by Simone Drago.
They refer to the Academic Year 2021/2022.

They are meant as a support for the students following the course and they should not be considered as a replacement for the professor's lectures.

For what concerns the use of the tools quoted inside the notes, please refer to the slides and the lectures from the professor.

Contents

1 Introduction to Performance Modelling

Performance Evaluation is the *quantitative* and *qualitative* study of systems, to evaluate, measure, predict and ensure target behaviors and performances. It is usually carried on using models of a system.
A **model** is an abstraction of a system: "*an attempt to distill, from the details of the system, exactly those aspects that are essential to the system behavior.*"
We abstract a system as a set of **Service Stations** that perform some task. The model defines:

- which tasks are carried out,
- when they are executed,
- how they are selected to be run,
- how long they last

and many other details to match the real system that is being modelled.

Performance indices measure the ability of the system to perform its tasks. **Workload** accounts for the difficulty, length and number of tasks that have to be performed.

From a service station, several performance indices and workload characterization can be computed. The most important are:

- Utilization
- Response time
- Average queue length
- Throughput
- Arrival rate
- Average service time

1.1 Performance indices

The *utilization* is the fraction of time a server is busy.
The *response time* is the average time spent by a job at a service center.
The *average queue length* accounts for the mean number of jobs in a service station, considering both the jobs being served and the ones in the queue.
The *throughput* describes the rate at which jobs are served and depart from the station.

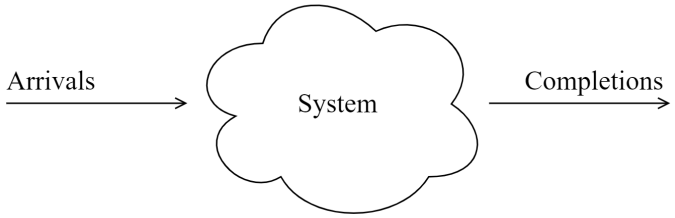


Figure 1: Example of a simple system.

Performance indices can be measured both on the real system and its model.
Model validation is checking if the performance indices of the real system and the model are similar to each other.

1.2 Workload characterization

The *arrival rate* is the frequency at which jobs arrive at a given station.
The *service time* is the average time required by a job to complete its service.

The workload is measured on the real system.
Once the model has been validated with the considered workload, it is studied varying both arrival rates and service times, to see their effects on the performance indices.
The use of models then allows us to address what improvements can be performed, how to plan them and how to implement them to achieve specific goals.

1.3 Basic relations

Consider now a simple system (Figure 1) that performs some jobs, arriving from outside.
By simply counting the jobs that enter and leave the system in a considered time frame T, we can determine its main parameters and performance indices.

Looking at Figure 2, we can count the number of jobs that enter the system up to time T with $A(T)$ and the number of jobs that exit the system as $C(T)$.
With these two basic measures and the assumption that *the system is initially empty*, all the performance indices defined above can be derived.

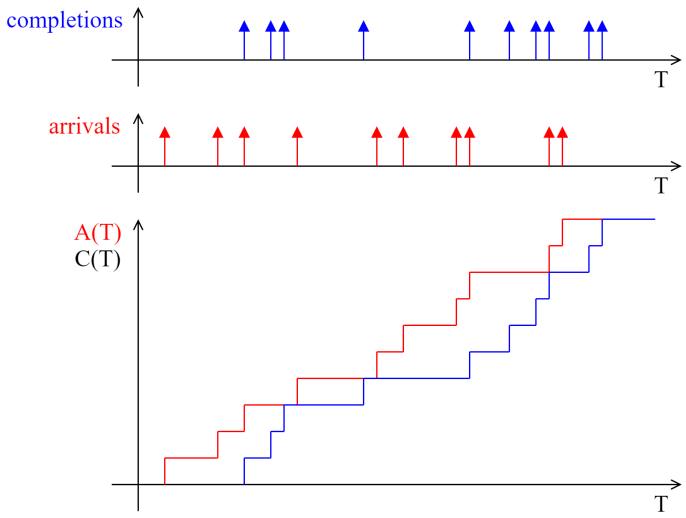


Figure 2: Plot of arrivals and completions of the system.

From $A(T)$ and $C(T)$, we can measure the *busy time* $B(T)$ as the time the system has **not** been idle during interval T, as seen in Figure 3.a).

In some cases, we can count the time each job spent in service. We can call s_i the *service time* for job i, as shown in Figure 3.b). In many practical situations, measuring the service time is not trivial since the execution of a job can be paused and resumed multiple times. However, we can work under the assumption that once a job is started it is never paused.

Similarly to service times, we can measure the time that passes from the moment a job enters the system to the moment it leaves, which is called *response time*. Sometimes, it can happen that a job i arrives before a job j but leaves after it. However, if we know that jobs are served one at a time in the order of arrival without being interrupted, we can estimate the response time from $A(T)$ and $C(T)$ as

$$r_i = C^{-1}(i) - A^{-1}(i)$$

as shown in Figure 3.c).

Another set of values we can measure from arrivals and completions are *interarrivals times*, which measure the difference in time between the arrivals of two jobs i and $i+1$, as shown in Figure 3.d). From $A(T)$, we can easily derive the interarrivals as

$$a_i = A^{-1}(i+1) - A^{-1}(i).$$

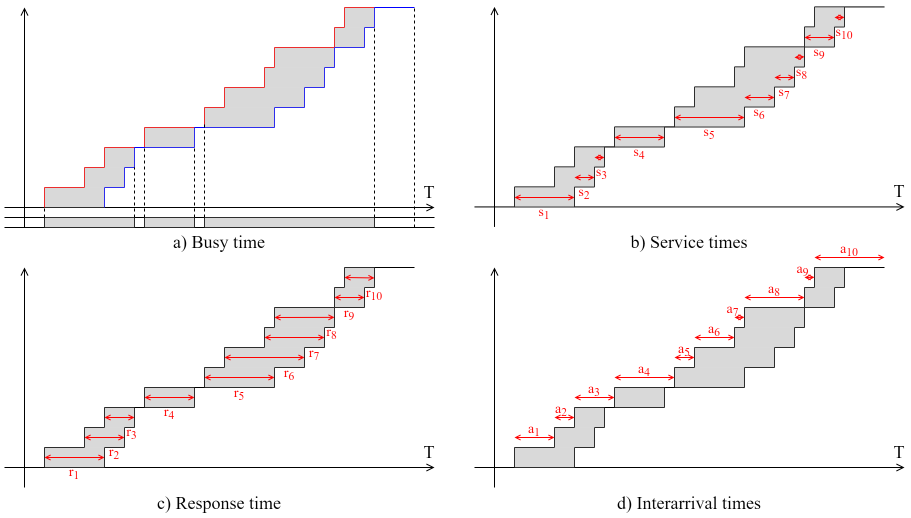


Figure 3: Plot of various measurements derived from arrivals and completions.

1.3.1 Estimating $A(T)$

Conversely, if we have interarrivals times a_i, we can easily derive $A(T)$. If we define the indicator function $I(X)$ which returns 1 if the proposition X is true and 0 otherwise, and assuming that a_0 accounts for the arrival time of the first job, then

$$A(T) = \sum_{K=1} I(\sum_{i=0}^{K-1} a_i \leq T)$$

and

$$A^{-1}(i) = \sum_{k=0}^{i-1} a_k$$

1.3.2 Estimating $C(T)$

With a_i and r_i we can determine $C(T)$ as

$$C(T) = \sum_{K} I(A^{-1}(K) + r_K \leq T) = \sum_{K} I(\sum_{i=0}^{K-1} a_i + r_K \leq T).$$

Notice that the infinite summation allows to account for jobs that "overtake" other jobs when being executed.
Having a_i and s_i, if we can suppose that jobs are served one at a time, in

the order in which they arrived, without being interrupted, we can iteratively determine $C^{-1}(i)$ from $C^{-1}(i-1)$ as

$$C^{-1}(i) = \max(A^{-1}(i), C^{-1}(i-1)) + s_i.$$

Notice that, if the time T is set starting and ending before new arrivals at an empty system, then the following relations hold:

$$A(T) = C(T) \qquad \sum_{i=1}^{A(T)} a_i = T \qquad \sum_{i=1}^{C(T)} s_i = B(T).$$

1.3.3 Main parameters and performance indices

From the measurements seen before, we can define both the **arrival rate** λ and the **throughput** X:

$$\lambda = \lim_{T\to\infty} \frac{A(T)}{T} \qquad X = \lim_{T\to\infty} \frac{C(T)}{T}.$$

If the system is stable (i.e., it is able to server all its jobs) and there are no losses, the two quantities are equal $\lambda = X$.

We can define the **utilization** as the ratio between the busy time and the total time:

$$U = \lim_{T\to\infty} \frac{B(T)}{T}.$$

We can also define the **average service time** (i.e., the average time a job spends being served) as the ratio between the busy time and the total number of completions:

$$S = \lim_{T\to\infty} \frac{B(T)}{C(T)}.$$

Note that, if the service time s_i is available for each job, we can compute the average service time as the average of those measures:

$$S = \frac{\sum_{i=1}^{C} s_i}{C}.$$

From these quantities we can express the **Utilization Law** as

$$U = X \cdot S = \frac{C(T)}{T} \cdot \frac{B(T)}{C(T)} = \frac{B(T)}{T}$$

This relation can be also used in the two alternative forms:

$$S = \frac{U}{X} \qquad X = \frac{U}{S}.$$

By construction, since $B(T) \leq T$, then the utilization should be less than 1. This allows us to define some limiting relations between the *arrival rate* and the *average service time* for a system to be stable:

$$X \cdot S = \lambda \cdot S \leq 1$$

$$\lambda \leq \frac{1}{S} \quad S \leq \frac{1}{\lambda}$$

If we define as

$$W(T) = \int_0^T (A(t) - C(T)) \cdot dt$$

the area of the difference between arrivals and departures, as shown in gray in Figure 3.a), we can use it to compute the **average number of jobs** in the system N as the ratio between $W(T)$ and the time T:

$$N = \lim_{T \to \infty} \frac{W(T)}{T}.$$

We can also compute the *average response time* R as the ratio between $W(T)$ and $C(T)$:

$$R = \lim_{T \to \infty} \frac{W(T)}{C(T)}.$$

Notice that this formula works even if the completion times are not related to the corresponding arrival times. This is because if a job is stalled, then another one is being executed so the total time remains the same.
Similar to the average service time, if we have collected all the response times r_i of each job, we can compute the average response time as:

$$R = \frac{\sum_{i=1}^{C} r_i}{C}.$$

This relation gives us an alternative to formula (1.3.3) to estimate W from the measurements of the real system:

$$W = \sum_{i=1}^{C} r_i.$$

From these quantities we can express **Little's Law**:

$$N = X \cdot R$$

which is always valid. Again, this formula can be used in its alternative forms:

$$R = \frac{N}{X} \qquad X = \frac{N}{R}.$$

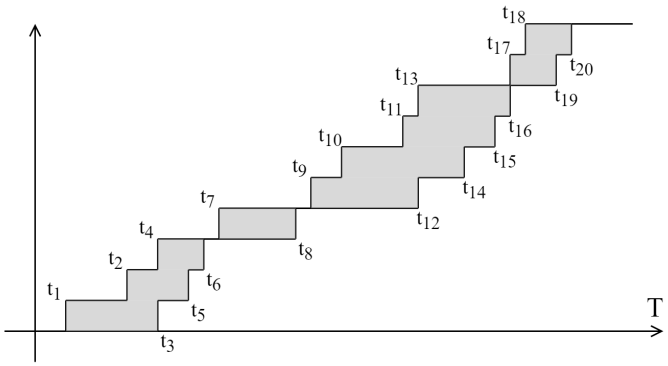


Figure 4: Arrival and completion time instants.

1.3.4 Response time distribution

If we have measured the response times for the single jobs r_i, we can approximate their distribution, estimating for example the probability that the response time is less than a threshold τ:

$$p(R < \tau) = \frac{\sum_{i=1}^{C} I(r_i < \tau}{C}$$

1.3.5 Queue length distribution

If we use a slightly more complex procedure, we can determine the probability of having n jobs in the system starting from $A(t)$ and $C(t)$.
First of all, we need to define all the time instant in which a job either arrives or departs as t_i, as shown in Figure 4.

Between two time points t_i and t_{i+1} the number of jobs in the system $n(t)$ under the assumption that the system starts empty is:

$$n(t) = A(t) - C(t)$$

otherwise we add the number of jobs inside the system at the start n_0.
Notice that the behavior between t_3 and t_4 is strange and we do not like it, but in a real environment it is extremely rare so can leave it be.

We can then compute Y_m as the fraction of time the system has been with m jobs:

$$Y_m = \int_0^T I(n(t) = m)dt.$$

We can then approximate the probability of having n jobs in the system as:

$$p(N = m) = \frac{Y_m}{T}.$$

We can use there relations to estimate both B, W and N as follows:

$$B = \sum_{m=1} Y_m = T - Y_0,$$

$$W = \sum_{m=1} m \cdot Y_m,$$

$$N = \sum_{m=1} m \cdot p(N = m).$$

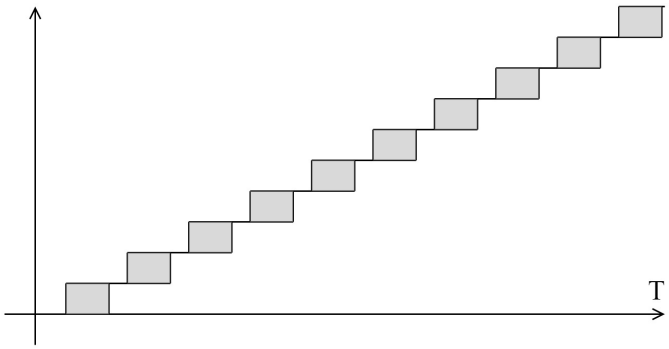


Figure 5: Arrival and completion time instants with deterministic interarrival times.

2 The Importance of Workload Characterization

In open models, the *arrival rate* λ influences all the basic performance metrics: utilization, throughput, average number of jobs in the system and average response time.
The arrival rate, however, is not the only parameter that influences the performance of a queue.

2.1 Interarrival Times Distribution

Suppose to have a system with an arrival rate of 2 $jobs/s$. If the service time S is of 0.35 s, then according to the *Utilization Law*, the utilization will be $U = S \cdot \lambda = 0.35 \cdot 2 = 0.70$.
Let's assume that the interarrival times are deterministic (*i.e., a new jobs arrives exactly* 0.5 s *after the previous job*), then we can plot arrivals and completions as shown in Figure 5.

In this case, the jobs never queue (since $\lambda < S$) and the performance indices are easily computed.

Let's assume now that the interarrival times are exponentially distributed and so they are non-deterministic. In this case, the non-determinism causes queueing, as we can see from the graph shown in Figure 6.

If the interarrival times are exponentially distributed, then the input is called a "*Poisson Process*".
In this case, both λ and S are the same as before, and the utilization U and

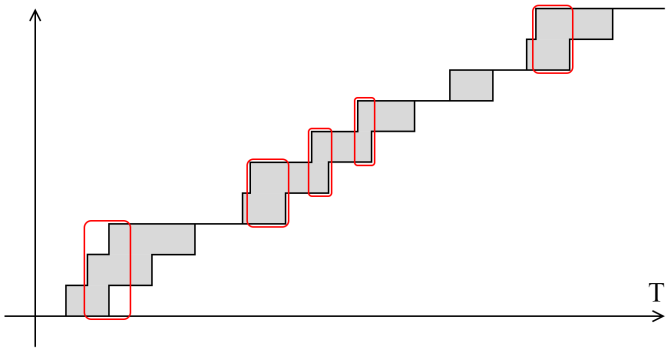


Figure 6: Arrival and completion time instants with exponential interarrival times.

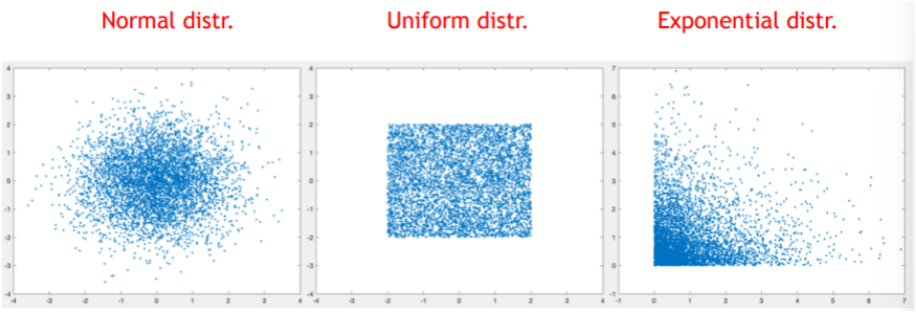


Figure 7: Point cloud for distribution with no correlation.

throughput X remain the same as they depend only on those two parameters. However, the *Average Response Time* R and the "*average number of jobs in the system*" N are greater than in the previous example because they also depend on the distribution on interarrival times.

Notice that, although the arrival rate remains the same, the interarrival distribution greatly influences the Average Response Time. In particular, **a higher variability of the distribution determines a worse performance**.

2.2 Correlation

The variability of the interarrival time distribution is not the only feature that characterizes the arrivals to a queue.

Figure 8: Point cloud for distribution with positive correlation.

Figure 9: Point cloud for distribution with negative correlation.

Correlation between interarrival times makes them no longer independent and creates bursts of arrivals, which in turn cause an even higher Average Response Time.
A simple technique to visually determine if there is a correlation between a set of samples is to plot the set as a *point cloud*, where two successive values are used respectively as the x and y coordinates of a point.
If the data is not correlated, the point cloud will have the same pattern in its density on the x and the y axes for the same value, as shown in Figure 7.

If data is positively correlated, the points will align towards the positive 45$\circ$ line, as shown in Figure 8.

Negative correlation, on the other hand, will align the points along the other 45$\circ$ line direction, as shown in Figure 9. Negative correlation is not as bad as positive correlation for the system, but it still changes how the system behaves.

Similar to the arrivals, also the service time (*i.e., the time it takes for the server to complete a job*) is subject to a distribution and performance indices

are affected by it.

3 Modelling Workloads

We have seen in the previous section how important the workload is in studying the performance metrics of a system.

But how can we quantify the workload starting from a set of collected measurements?
How do we check the amount of variability and the presence of correlation?

Workloads are in general unpredictable, but there are some patterns that we can use to characterize them.

3.1 Probability Distributions

Probability distribution are important in performance modelling, since they allow to *characterize most of the behaviors that cannot be deterministically predicted*.
Defining the proper probabilistic characterization of arrivals and service times is of great importance in accurately modelling a system.
Generally, starting from a set of measurements collected from the real system, it is possible to conduct a probabilistic characterization.
It is then used to build a model, which in turn allows to predict performances.

A probability distribution can use random variables of three types, based on they domain over which they are defined:

- Discrete
- Continuous
- Mixed

Probability distributions are used in performance evaluation to model *random choices*.

Discrete random variables associate a set of probabilities to a discrete number of possible outcomes.
Each element of the set will have a probability associated (called *probability value* $p(x)$) to it, and the sum of the probabilities of all the elements will always be equal to 1.
If the outcomes of the discrete random variable are numbers, then it is possible to define a *Cumulative Distribution Function* (CDF) $F(x)$, which is defined as "the probability of getting an outcome less than or equal to x".

$$F(N) = p(x \leq N)$$

In practice, the CDF of a certain value is the sum of the probability values of all the outcomes that are lower than or equal to that value.

On the other hand, in case of a *continuous domain*, outcome values are generated over a continuous space and the probability of having a specific value is always ~ 0, and so the probability of intervals are used, which are > 0.

Continuous variables are defined by a *probability density function* (PDF) $f(x)$ and a CDF $F(x)$, such that:

$$p(x_A \leq X \leq x_B) = \int_{x_A}^{x_B} f(x)dx = F(x_B) - F(x_A)$$

The PDF should be such that its integral over the support of the distribution is equal to 1.

A *mixed random variable* can have both discrete outcomes and values that are continuously distributed.
The Probability Density Function of a mixed random variable has both a Dirac's delta and continuous segments and is defined as:

$$f(x) = \sum_i p(x_i)\delta(x - x_i) + (1 - \sum_i p(x_i)) \sim f(x)$$

where $\delta(\cdot)$ represents the Dirac's delta centered in 0.

The Cumulative Distribution Function contains jumps in correspondence of the outcomes that have non-zero probability and is defined as:

$$F(x) = \sum_i p(x_i)I(x \geq x_i) + (1 - \sum_i p(x_i)) \sim F(x)$$

where $I(\phi)$ is the indicator function that returns 1 if ϕ is true, and 0 otherwise.

In performance evaluation, probability distributions are usually adopted to characterize the *interarrival times* (time between two subsequent events) and the *service times* (time needed to perform an action) of the components of a model. For this reason, they are typically defined only for positive values of the random variable.

3.2 Moments

For a probability distribution, the **expected value** according to a function $g(x)$ is defined as:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx$$

In the case that $g(x) = x^n$, the corresponding expected value is called the n^{th} *moment* of the distribution. In such cases, the expected value formula takes the form of:

$$E[X^n] = \int_{-\infty}^{\infty} x^n f(x)dx$$

Let's define some moments that are of particular relevance.
From now on, we will call $\mu = E[X]$ the *first moment of the distribution*. It is called the **mean** and corresponds to the average value of the distribution.
We can define the n^{th} *central moment* of the distribution as:

$$E[(X-\mu)^n]$$

We can define the **variance** of the distribution as the second central moment $Var[X] = E[(X-\mu)^2 = E[X^2] - \mu^2$.
We will call $\sigma = \sqrt{E[(X-\mu)^2]} = \sqrt{Var[X]}$ the **standard deviation**.
We can define the n^{th} *standardized moment* of the distribution as:

$$E\left[\left(\frac{X-\mu}{\sigma}\right)^n\right]$$

Central moments are meaningful for $n > 1$ and are insensitive to the *position* of the distribution.
Standardized moment are meaningful for $n > 2$ and are *insensitive* to both the *scale* and the *position* of the distribution.

The following properties apply to constant values and multiple random variables:

$$E[c] = c$$

$$E[X+Y] = E[X] + E[Y]$$

$$E[aX] = aE[X]$$

$$Y = \begin{cases} X_1 & p_1 \\ \cdots & \cdots \\ X_n & p_n \end{cases} \Rightarrow E[Y] = \sum_{i=1}^{n} p_i E[X_i]$$

Starting from the standard deviation and the mean, we can define the **coefficient of variation**:

$$c_v = \frac{\sigma}{\mu}$$

which tells how different the variance is with respect to the mean.

Notice that variance, standard deviation and coefficient of variation express the same concept: how the outcome spreads from the average.

An important moment to define is the third standardized moment, the **skewness**. It is defined as

$$\gamma = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{E[X^3] - 2\mu\sigma^2 - \mu^3}{\sigma^3}$$

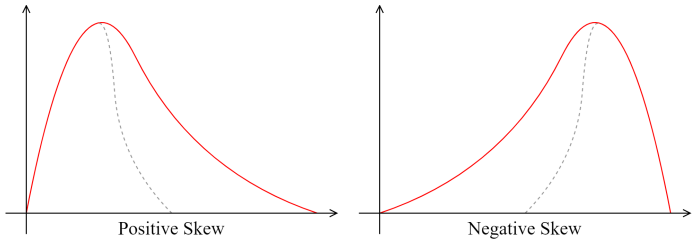


Figure 10: Distributions with a positive (left) and negative (right) skewness.

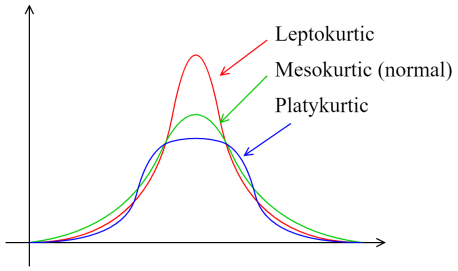


Figure 11: General forms of kurtosis.

and represents whether the distribution is symmetric, or if it has more probability mass to the left of to the right of its mean, as shown in Figure 10. A skewness of zero means that the distribution is perfectly symmetric.

Another important moment is the fourth standardized moment, the **excess kurtosis**. It is defined as

$$\beta = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] - 3 = \frac{E[(X-\mu)^4]}{\sigma^4} - 3$$

and it determines how flat the distribution is, by describing the shape of its peak, as shown in Figure 11.
The -3 is introduced in order to have the kurtosis of the *Standard Normal Distribution* equal to 0.

To summarize, the first four moments represent:

- First moment - measure of location
- Second moment - measure of spread
- Third moment - measure of symmetry

- Fourth moment - measure of peakedness

Moments play an important role in performance evaluation for several reasons:

- There are many results that are sensitive to the first or second moments of the distributions,
- Beside being computed analytically on distributions, they can be easily derived from measures, log files and data sets,
- Moments of collected data can guide the modeler to choose and parametrize the most appropriate distributions.

In the case of a distribution with *discrete support*, the moments and other properties can be computed as *finite sums*:

$$\mu = E[X] = \sum_i x_i p(x_i)$$

$$E[X^k] = \sum_i x_i^k p(x_i)$$

$$Var[X] = E[X^2] - \mu^2 = \sum_i x_i^2 p(x_i) - \mu^2$$

3.3 Percentiles

The k^{th} *percentile* of a distribution is the value for which its CDF is equal to $k/100$ and is defined as:

$$\Theta(k) = F^{-1}(k/100)$$

In performance evaluation, percentiles are used to *assess properties*.
For example, if we want to ensure that the response time R of a station exceeds a given threshold H only for a limited percentage of the k jobs, we can assess the following formula:

$$Prob(R \geq H) \leq \frac{k}{100} \Leftrightarrow \theta(100 - k) \leq H$$

This allows us to perform better analyses, since it is possible that most of the jobs are executed fast but a small amount takes a very long time, and so the average response time is less useful in an assertion of this kind with respect to a percentile.

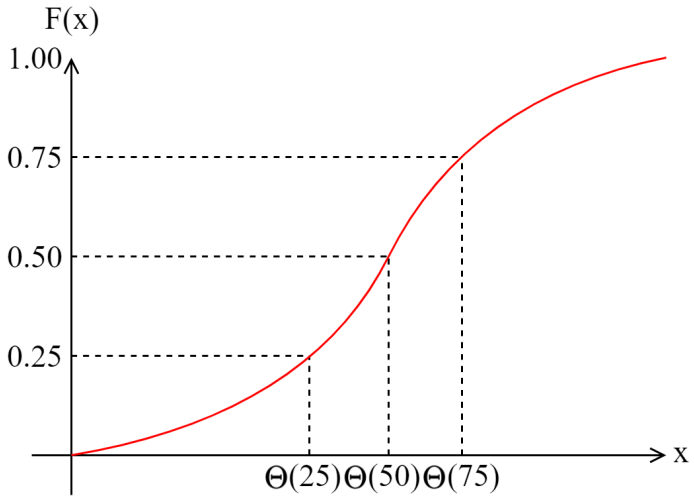


Figure 12: Example of a CDF and its related percentiles.

3.4 Correlation

To analytically verify the presence of correlation, we can compute the *cross-covariance* between successive samples of the distribution, comparing each one with the value m jobs later. This is called the *lag-m cross-covariance* $\sigma(m)$.
If X_n is the n^{th} instance of a given distribution, $\sigma(m)$ is defined as:

$$\sigma(m) = E\left[(X_n - \mu_X)(X_{n+m} - \mu_X)\right]$$

For a set of N samples, the *lag-1 cross-covariance* can be approximated as:

$$\sigma(1) = \frac{1}{N-1} \sum_{n=1}^{N-1} (x_n - \mu_X)(x_{n+1} - \mu_X)$$

If the cross-covariance is different from 0, then it means that the arrivals are correlated. This typically results in a higher average response time.
In general, the cross-covariance tends to 0 as m tends to infinity. However, the slower m tends to 0, the more correlated the samples are.

3.5 Characterizing a distribution from a trace

As said before, it is possible to use the measurements of a real system to help characterize the distribution that governs it.

Starting from a trace of measurements, it is not easy to estimate the PDF of the distribution that generated it. However, it is much easier to approximate its CDF.

Suppose to have N samples $x_1, \dots\ x_N$ taken from a unknown distribution X.
The first thing to do is to obtain the sorted version of the trace:

$$|y_1, \dots, y_N| = sort(x_1, \dots, x_N)$$

where $y_1 \leq \cdots \leq y_N$.
Then, the approximated CDF of X can be defined as:

$$F_X(i) = \frac{i}{N}$$

where i is the index of the sorted trace.
If instead we want to use the indicator function, we can write:

$$F_X(x) = \frac{1}{N} \sum_{i=1}^{N} I(y_i \leq x)$$

Assuming that each sample of the trace is equally probable, the moments can be approximated using discrete sums:

$$E[X^k] = \frac{1}{N} \sum_{i=1}^{N} x_i^k$$

where the $\frac{1}{N}$ signifies that each sample has the same probability.

Cross-covariance can also be easily computed, since such measure is already defined on the samples of a distribution:

$$\sigma(m) \cong \frac{1}{N-m} \sum_{n=1}^{N-m} (x_n - \mu_X)(x_{n+m} - \mu_X)$$

Percentiles can be derived from the samples, but several approaches exist.
A simple approach is to use *linear interpolation*:

$$h = (N-1)\frac{k}{100} + 1$$

$$\Theta(k) = \begin{cases} x_N & h = N \\ x_{\lfloor h \rfloor} + (h - \lfloor h \rfloor) \cdot (x_{\lfloor h \rfloor + 1} - x_{\lfloor h \rfloor}) & h < N \end{cases}$$

For a more exhaustive list, refer to https://axibase.com/use-cases/workshop/percentiles.html.

4 Basic Probability Distributions

Suppose to have the trace of a real system. Which is the distribution that is more appropriate to describe the trace? What are the parameters of the distribution that will reproduce the trace in the best way?

4.1 Discrete Distributions

The most important discrete distributions in performance evaluation are *Finite discrete*, *Geometric* and *Poission*.

4.1.1 Finite Discrete Distribution

Finite discrete distributions are encoded by tables that associate to each outcome the corresponding probability.

4.1.2 Geometric distribution

The geometric distribution is a discrete random variable with infinite support. It is characterized by a probability parameter q:

$$P(X_{Geom<q>} = k) = (1-q)^k q$$
$$F_{Geom<q>}(k) = 1 - (1-q)^{k+1}$$

The geometric distribution has the following average and variance:

$$\mu_{Geom<q>} = \frac{1-q}{q}$$
$$\sigma^2_{Geom<q>} = \frac{1-q}{q^2}$$

The geometric distribution represents the probability that an event, with probability q, will occur after a given number of failed trials.

4.1.3 Poisson Distribution

The Poisson distribution is characterized by a parameter $\lambda > 0$:

$$P(X_{Pois<\lambda>=k}) = \frac{\lambda^k e^{-\lambda}}{k!}$$
$$F_{Pois<\lambda>}(k) = e^{-\lambda} \sum_{i=0}^{\lfloor k \rfloor} \frac{\lfloor k \rfloor^k}{\lfloor k \rfloor !}$$

The Poisson distribution is concentrated around the parameter λ, in fact its mean and variance are:

$$\mu_{Pois<\lambda>} = \lambda$$
$$\sigma^2_{Pois<\lambda>} = \lambda$$

In performance evaluation, finite discrete distributions are used to encode the probability of making a specific choice from a set of alternatives.
Geometric can be used as the distribution of the number of jobs waiting in the queue.
Poisson can be used to count the number of arrivals at a given point in time.

4.2 Continuous Distributions

The most meaningful continuous distributions in performance evaluation are:

- Deterministic
- Exponential
- Extensions of exponential
- Uniform
- Normal
- Pearson
- Weibull
- Pareto

4.2.1 Deterministic Distribution

A *Deterministic random variable* represents actions that always take the same time to be completed (*i.e., actions that are not actually random*). Its parameter is the constant value τ:

$$f_{Det<\tau>}(t) = \delta(t - \tau)$$
$$F_{Det<\tau>}(t) = I(t \geq \tau)$$

Its average corresponds to its only value ($\mu_{Det<\tau>} = \tau$) and its variance is zero ($\sigma^2_{Det<\tau>} = 0$).
Notice that most of the common analysis techniques do not work well with deterministic random variables.

A deterministic random variable is typically used to model arrivals that have a

fixed time interval (*e.g., clock signals, scheduled jobs, rotation times*).

4.2.2 Exponential Distribution

The *Exponential distribution* is the most used distribution in performance evaluation.
It is characterized by a parameter λ that defines its rate:

$$f_{Exp<\lambda>}(t) = \lambda e^{-\lambda t}$$
$$F_{Exp<\lambda>}(t) = 1 - e^{-\lambda t}$$

Its main properties are:

$$\mu_{Exp<\lambda>} = \frac{1}{\lambda}$$
$$\sigma^2_{Exp<\lambda>} = \frac{1}{\lambda^2}$$
$$c_{v_{Exp<\lambda>}} = 1$$

The moments of the exponential distribution are defined as:

$$E[X^n] = \frac{n!}{\lambda^n}$$

The rate λ determines the speed at which the probability density of the distribution tends to zero.
An important property of the Exponential distribution is that it is *memory-less*:

$$Prob(X_{Exp<\lambda>} > t + s | X_{Exp<\lambda>} > s) = Prob(X_{Exp<\lambda>} > t)$$

This property allows us to employ the Exponential distribution to describe events that occur in a random and independent way from one another.
In most of the cases, a model that is based on an Exponential distribution can be analyzed with simpler computations.

Poisson Process
If the interarrival time of events in a process is exponentially distributed, then the corresponding process is called a *Poisson process*.
This is because, if we fix a time interval T, then the number of events with an exponential interarrival time of parameter λ that occurr in that interval is distributed according to a Poisson distribution of parameter λT:

$$\text{Prob}(n \text{ events } X_{Exp<\lambda>} \text{ in } T) = P(X_{Pois<\lambda T>} = n)$$

The minimum of a set of exponential random variables, is itself a random variable with a rate that is equal to the sum of the rates:

$$f_{min\{Exp<\lambda_1>,\ldots,Exp<\lambda_n>\}}(t) = f_{Exp<\lambda_1+\cdots+\lambda_n>}(t)$$

Because of this property, if we combine n Poisson processes with different rates, we obtain a new Poisson process that runs at the sum of the components rates.

If we split a Poisson process into n new processes with probabilities $q_1, \ldots, q_n$, the result is a set of independent Poisson processes with rates $\lambda q_1, \ldots, \lambda q_n$.

4.2.3 Hyper-exponential Distribution

The *Hyper-exponential distribution* corresponds to a set of Exponential distributions, selected according to a discrete random distribution, and is represented as:

$$X_{Hyper-exp<\lambda_i,p_i>} = \begin{cases} X_{Exp<\lambda_1>} & p_1 \\ \cdots & \cdots \\ X_{Exp<\lambda_n>} & p_n \end{cases}$$

Its Probability Density Function (PDF) and Cumulative Distribution Function (CDF) are defined as:

$$f_{Hyper-exp<\lambda_i,p_i>}(t) = \sum_i p_i \lambda_i e^{-\lambda_i t}$$

$$F_{Hyper-exp<\lambda_i,p_i>}(t) = 1 - \sum_i p_i e^{-\lambda_i t}$$

An Hyper-exponential is characterized by the number n of stages and by n couples of rates λ_i and probabilities p_i, which represent the probabilities of selecting each exponential.

$$X_{Hyper-exp<\lambda_i,p_i>} = \begin{cases} X_{Exp<\lambda_1>} & p_1 \\ X_{Exp<\lambda_2>} & p_2 \\ \cdots & \cdots \\ X_{Exp<\lambda_n>} & p_n \end{cases}$$

Depending on the values of the distributions, the Hyper-exponentials can have very different shapes and can generate samples with very different scales.

Its main properties are:

$$\mu_{Hyper-exp<\lambda_i,p_i>} = \sum_i \frac{p_i}{\lambda_i}$$

$$\sigma^2_{Hyper-exp<\lambda_i,p_i>} = \sum_i \frac{2p_i}{\lambda_i^2} - \left(\sum_i \frac{p_i}{\lambda_i}\right)^2$$

$$c_{v_{Hyper-exp<\lambda_i,p_i>}} \geq 1$$

The coefficient of variation being greater than one is the reason why this distribution is called *Hyper*.
The moments of a Hyper-exponential are defined as follows:

$$E[X^n] = \sum_i \frac{n! p_i}{\lambda_i^n}$$

In the simple case of a two stage Hyper-exponential, since $p_2 = 1 - p_1$, the first three moments are:

$$E[X] = \frac{p_1}{\lambda_1} + \frac{1 - p_1}{\lambda_2}$$
$$E[X^2] = 2\left(\frac{p_1}{\lambda_1^2} + \frac{(1 - p_1)}{\lambda_2^2}\right)$$
$$E[X^3] = 6\left(\frac{p_1}{\lambda_1^3} + \frac{(1 - p_1)}{\lambda_2^3}\right)$$

Hyper-exponential can be used as an interarrival distribution in the case we need to model actions caused by different types of event.
In service time distributions, it can instead be used to model services that span over different time-scales.

4.2.4 Hypo-exponential Distribution

The sum of n exponential distributions is called a *Hypo-exponential distribution*. It is represented as:

$$X_{Hypo-exp<\lambda_1,\ldots,\lambda_n>} = X_{Exp<\lambda_1>} + \cdots + X_{Exp<\lambda_n>}$$

and is characterized by the parameters $\lambda_1, \ldots, \lambda_n$ of its components.
Its PDF and CDF (*shown for the case of $n = 2$ for ease of description*) are as follows:

$$f_{Hypo-exp<\lambda_1,\lambda_2>}(t) = \frac{\lambda_1\lambda_2}{\lambda_1 - \lambda_2}\left(e^{-\lambda_2 t} - e^{-\lambda_1 t}\right)$$
$$F_{Hypo-exp<\lambda_1,\lambda_2>}(t) = 1 - \frac{\lambda_2 e^{-\lambda_1 t}}{\lambda_2 - \lambda_1} + \frac{\lambda_1 e^{-\lambda_2 t}}{\lambda_2 - \lambda_1}$$

The PDF of an Hypo-exponential is shaped with an increase followed by a decrease.

Its main properties are:

$$\mu_{Hypo-exp<\lambda_1,\lambda_2>} = \frac{1}{\lambda_1} + \frac{1}{\lambda_2}$$

$$\sigma^2_{Hypo-exp<\lambda_1,\lambda_2} = \frac{1}{\lambda_1^2} + \frac{1}{\lambda_2^2}$$

$$c_{v_{Hypo-exp<\lambda_1,\lambda_2>}} = \frac{\sqrt{\lambda_1^2 + \lambda_2^2}}{\lambda_1 + \lambda_2} < 1$$

Notice that the coefficient of variation is always less than one (that is the reason why it is called *Hypo*) and because of this, a Hypo-exponential is always better in terms of response time with respect to the Hyper-exponential.

The moments of a Hypo-exponential have no simple explicit expression. In the case of a two-stage distribution, they can be defined as:

$$E[X^n] = \frac{n!}{\lambda_1 - \lambda_2}\left(\frac{\lambda_1}{\lambda_2^n} - \frac{\lambda_2}{\lambda_1^n}\right)$$

A Hypo-exponential can be used to model processes that are composed by a set of subsequent exponential stages.

4.2.5 Uniform Distribution

A *Uniform distribution* is a random variable that returns values equally distributed between two values a and b.
Its PDF and CDF are defined as:

$$f_{Unif<a,b>}(t) = \begin{cases} \frac{1}{b-a} & a \leq t \leq b \\ 0 & t < a \vee t > b \end{cases}$$

$$F_{Unif<a,b>}(t) = \begin{cases} 0 & a < t \\ \frac{t-a}{b-a} & a \leq t \leq b \\ 1 & t > b \end{cases}$$

and have the shapes shown in Figure 13.

Its main properties are:

$$\mu_{Unif<a,b>} = \frac{a+b}{2}$$

$$\sigma^2_{Unif<a,b>} = \frac{(b-a)^2}{12}$$

The moments of a Uniform distribution are defined as:

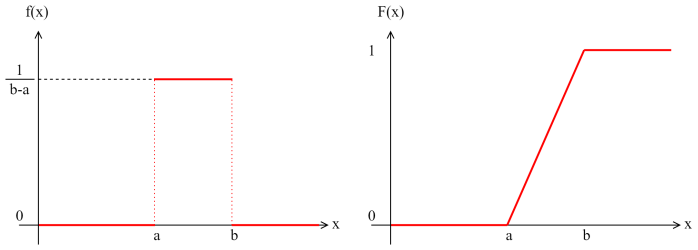


Figure 13: PDF and CDF of a Uniform distribution.

$$E[X^n] = \frac{b^{n+1} - a^{n+1}}{(n+1)(b-a)}$$

A typical example of Uniform distribution is the emulation of the rotational latency in a disk drive.

4.2.6 Erlang Distribution

An *Erlang distribution* is an Hypo-exponential distribution where all the exponential stages have the same rate. It can be written as:

$$X_{Erlang<\lambda,l>} = \sum_{i=1}^{k} X_{Exp<\lambda>}$$

The PDF of the Erlang distribution is defined as:

$$f_{Erlang<\lambda,k>}(t) = \frac{\lambda^k t^{k-1} e^{-\lambda t}}{(k-1)!}$$

Its main properties are:

$$\mu_{Erlang<\lambda,k>} = \frac{k}{\lambda}$$

$$\sigma^2_{Erlang<\lambda,k>} = \frac{k}{\lambda^2}$$

$$c_{v_{Erlang<\lambda,k>}} = \frac{1}{\sqrt{k}} \leq 1$$

Notice that the coefficient of variation is always less than one, and depends only on the number of stages.

The shape of an Erlang distribution depends both on k and λ.
A Erlang distribution can be used to approximate the Deterministic or the Normal distribution with exponential steps.

4.2.7 Hyper-Erlang Distribution

The *Hyper-Erlang distribution* is a mixture of Erlang distributions that are characterized by different rates and number of phases, each with its own probability:

$$X_{Hyper-Erlang<\lambda_i,k_i,p_i>} = \begin{cases} X_{Erlang<\lambda_1,k_1>} & p_1 \\ \cdots & \cdots \\ X_{Erlang<\lambda_n,k_n>} & p_n \end{cases}$$

The coefficient of variation of a Hyper-Erlang can be arbitrary as it greatly depends on its components.

The Hyper-Erlang distribution is used to approximate complex distributions. This is because, using special fitting techniques, it is possible to fit any arbitrary distribution to a Hyper-Erlang.

4.2.8 Weibull Distribution

The *Weibull distribution* is characterized by two parameters: the *shape* $k \in \mathbb{R}$ and the *scale* λ.
It resembles an exponential distribution raised to the power of k.
Its PDF and CDF can be defined as:

$$f_{Weibull<\lambda,k>}(t) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-\left(\frac{t}{\lambda}\right)^k}$$

$$F_{Weibull<\lambda,k>}(t) = 1 - e^{-\left(\frac{t}{\lambda}\right)^k}$$

Its main properties are:

$$\mu_{Weibull<\lambda,k>} = \lambda\Gamma\left(1+\frac{1}{k}\right)$$

$$\sigma^2_{Weibull<\lambda,k>} = \lambda^2\left[\Gamma\left(1+\frac{2}{k}\right) + \left(\Gamma\left(1+\frac{1}{k}\right)\right)^2\right]$$

It is typically used for modelling the lifetime of components subject to aging.

4.2.9 Pareto Distribution

The *Pareto distribution* has a simple monomial expression and is characterized by a *shape* parameter α and a *scale* parameter m.
Its PDF and CDF can be defined as:

$$f_{Pareto<\alpha,m>}(t) = \begin{cases} \frac{\alpha m^\alpha}{t^{\alpha+1}} & t \geq m \\ 0 & t < m \end{cases}$$

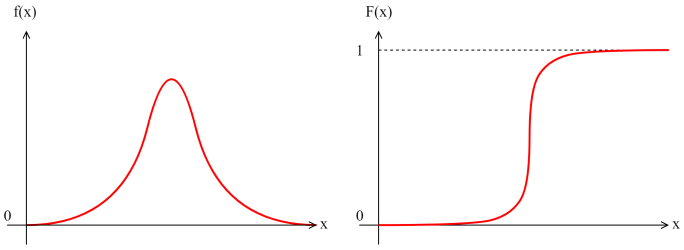


Figure 14: PDF and CDF of a Normal distribution.

$$F_{Pareto<\alpha,m>}(t) = \begin{cases} 1 - \left(\frac{m}{t}\right)^{\alpha} & t \geq m \\ 0 & t < m \end{cases}$$

Because of the fact that the integral of a PDF must be equal to 1, the Pareto distribution is valido only for $\alpha > 0$, and generates values in the range $[m, \infty]$.

The Pareto distribution has a finite mean μ only for $\alpha > 1$. This is because for $0 < \alpha \leq 1$, the values generated by the distribution are too large to be computed.

$$\mu_{Pareto<\alpha,m>} = \begin{cases} \infty & \alpha \leq 1 \\ \frac{\alpha m}{\alpha - 1} & \alpha > 1 \end{cases}$$

The Pareto distribution has a finite variance only for values $\alpha > 2$. For values $1 < \alpha \leq 2$, the variance is infinite.

$$\sigma^2_{Pareto<\alpha,m>} = \begin{cases} \infty & \alpha \leq 2 \\ \frac{\alpha m^2}{(\alpha-1)^2(\alpha-2)} & \alpha > 2 \end{cases}$$

The Pareto distribution is used to model systems where very large values can have a non-negligible probability of occurring.
This phenomenon is called "heavy tail", and it can be shown by plotting $1-F(x)$ in a log-log-scale.
It is a typical behavior of Internet traffic, where sometimes very large files are transferred.
Heavy tail distributions requires particular care in estimating its measures.

4.2.10 Normal Distribution

The *Normal* (or *Gaussian*) distribution is defined by its mean and its variance. Its PDF and CDF are defined as:

$$f_{N<\mu,\sigma^2>}(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}$$

$$F_{N<\mu,\sigma^2>}(t) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{t-\mu}{\sigma\sqrt{2}}\right)\right]$$

where erf is called the *error function* and it is defined as the integral of a normal distribution. It is used because the PDF cannot be analytically integrated. The PDF and CDF of a Normal distribution have the shape shown in Figure 14.
Notice that the PDF has is bell-shaped, with the peak of the bell corresponding to the mean of the distribution and the width of the bell representing the variance.

Its main properties are:

$$\mu_{N<\mu,\sigma^2>} = \mu$$
$$\sigma^2_{N<\mu,\sigma^2>} = \sigma^2$$

and they correspond to the parameters of the distribution.

The Normal distribution plays a very important role since, according to the *central limit theorem*, the sum of a large number of independent random variables tends to a Normal distribution:

$$\sum_i X_i \cong X_{N<\sum_i \mu_i, \sum_i \sigma_i^2>}$$

In the particular case in which $\mu = 0$ and $\sigma^2 = 1$, the distribution is called **Standard Normal** $N_{<0,1>}$. Every Normal distribution can be derived from the Standard Normal as:

$$X_{N<\mu,\sigma^2>} = \mu + \sigma \cdot X_{N<0,1>}$$

4.2.11 Truncated Normal Distribution

In performance evaluation, Gaussian distributions can be used to model the time required to perform some actions and the interarrival times of jobs. However, those values are intrinsically positive, and so the Normal distribution comes with a problem: it has an infinite support and it can also produce negative numbers.
If the modellization requires us to use a Normal distribution with a small variance, this issue might be mitigated. But if the variability is high, the probability of generating negative samples is not negligible.
One way to avoid this issue is to truncate the normal distribution to discard all negative samples. This is called the *Trucated Normal distribution* and is defined as:

$$f_{TruncN<\mu,\sigma^2>} = \frac{f_{N<\mu,\sigma^2>}(t)}{1 - F_{N<\mu,\sigma^2>}(0)}$$

However, the Truncated Normal distribution comes with its own problems since

the final mean and variance will not be the same of the starting Normal distribution. Still, if the c_v is small enough, the Truncated Normal can be considered as a good approximation since the difference with the Normal distribution can be negligible.

To be precise, it could be possible to obtain parameters for a Truncated Normal to match the ones of a Normal distribution by solving a non-linear system of equations, however, we will not consider this.

4.2.12 Log Normal Distribution

The *Log Normal Distribution* is computed starting from a Normal distribution simply as the exponent of an exponential function. It is defined as:

$$X_{LogNormal<\mu,\sigma^2>} = e^{X_{N<\mu,\sigma^2>}}$$

Notice that, due to the properties of the exponential function, this distribution only produces positive samples.
Its PDF and CDF are defined as:

$$f_{LogNormal<\mu,\sigma^2>}(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\ln(t)-\mu)^2}{2\sigma^2}}$$

$$F_{LogNormal<\mu,\sigma^2>}(t) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{\ln(t)-\mu}{\sigma\sqrt{2}}\right)\right]$$

Its mean and variance are defined as:

$$\mu_{LogN<\mu,\sigma^2>} = e^{\mu+\frac{\sigma^2}{2}} \qquad \sigma^2_{LogN<\mu,\sigma^2>} = \left(e^{\sigma^2}-1\right)e^{2\mu+\sigma^2}$$

4.2.13 Gamma Distribution

The *Gamma distribution* is an extension if the Erlang distribution in which the number of stages can be any positive real number.
The name of the distribution is due to the fact that the gamma function is used in its PDF:

$$f_{Gamma<\theta,k>}(t) = \frac{t^{k-1}e^{-t/\theta}}{\theta^k \Gamma(k)}$$

The Gamma distribution has the following mean and variance:

$$\mu_{Gamma<\theta,k>} = k\theta \qquad \sigma^2_{Gamma<\theta,k>} = k\theta^2$$

Notice that the Gamma distribution can be completely characterized by its parameters.

4.2.14 Pearson Family of Distributions

The *Pearson family of distributions* is defined using the first four moments of the distribution.
It is a family of distributions that includes many other cases (*e.g., Exponential, Gamma and Uniform distributions*), which are selected depending on their parameters.
The distributions of the Pearson family have a PDF that satisfies the following differential equation:

$$\frac{df_{Pearson<a,b_0,b_1,b_2>}(t)}{dt} = f_{Pearson<a,b_0,b_1,b_2>}(t)\frac{t-a}{b_2t^2+b_1t+b_0}$$

This differential equation can be solved, obtaining a closed-form expression of the PDF:

$$f_{Pearson<a,b_0,b_1,b_2>}(t) = e^{\int \frac{t-a}{b_2t^2+b_1t+b_0}dt}$$

Notice that the integral inside the exponential can assume many different expressions, depending on the roots r_1 and r_2 of $b_2t^2 + b_1t + b_0$.

4.3 Parameter Estimation and Fitting

Once we have identified a distribution, we then need to estimate its parameters from the measurements acquired from the corresponding real system.
The process of determining the best parameters that allow the distribution to generate samples similar to the ones of the trace is called *parameter estimation* or *fitting*.
These distributions build with parameter estimation have their own use, as we may be interested in studying the time required to complete a service or in the number of jobs that might arrive to a station.

The two main fitting methods are:

- Method of moments
- Maximum likelihood method

4.3.1 Method of Moments

The method of moments is used when we can express the moments of the distribution with a closed formula.
In order to use this method, we need to express as many moments as the parameters:

$$E[X] = \phi_1(p_1, \ldots, p_n)$$
$$\ldots$$

$$E[X^n] = \phi_n(p_1, \ldots, p_n)$$

Notice that the method of moments can be used only if an expression to define the moments starting from the parameters is available. The standard formula that employs the integral, although always applicable, is not useful as it is difficult to compute.

The empirical values of the moments are computed starting from the sample data of the traces:

$$E[X^k] = \frac{1}{N} \sum_{i=1}^{N} (x_i)^k$$

Then we can solve the system of n equations to compute the parameters that best approximate the trace.
This system can be typically solved analytically, however there are some complex scenarios that require us to use some non-linear equations.

4.3.2 Maximum Likelihood

The maximum likelihood method is based on a function that is called *likelihood* and is defined as:

$$\mathcal{L}(p_1, \ldots, p_n) = \prod_{i=1}^{N} f_{X<p_1,\ldots,p_n>}(x_i)$$

which represents how likely a set of parameters is to define a distribution that generates the set of values in the trace.
The parameters of the distribution can be estimated as the set of values that maximize the likelihood function:

$$|p_1, \ldots, p_n| = \underset{|p_1,\ldots,p_n|}{\operatorname{argmax}} \mathcal{L}(p_1, \ldots, p_n)$$

Usually, the logarithm of the likelihood is used as the maximization is between sums instead of products:

$$\ln\left(\mathcal{L}(p_1, \ldots, p_n)\right) = \ln\left(\prod_{i=1}^{N} f_{X<p_1,\ldots,p_n>}(x_i)\right) = \sum_{i=1}^{N} \ln\left(f_{X<p_1,\ldots,p_n>}(x_i)\right)$$

Thanks to this mathematical definition, the values of the parameters that maximize the likelihood function can typically estimated analytically.
If not possible, we need to perform numerical maximization.

Since those two approaches have opposite goals, which one to use must be selected carefully. If a very large number of samples is available, both methods produce very good results.

Differences between the two arise when the sample population is smaller or if the samples come from different distributions.
The method of moments gives a distribution that is closer to the sample points but with a more difficult matching of indices (*e.g., percentiles*).
Maximum likelihood produces a distribution that is more similar to the real distribution, but that may be further from the trace, causing problems in performance models.

For the maximum likelihood, it is important to correctly define the range of the parameters, otherwise the solver might find mathematically correct solution that however are unfeasible and do not represent a valid distribution.
For the method of moments, instead, it is better to write the equations in the form of:

$$\frac{\phi_i(p_1, \ldots, p_n)}{E[X^k]} - 1 = 0$$

The normalization of the moments allows to not directly search for the absolute values and improves numerical stability.

For both the numerical maximization and the solution of non-linear systems of equations, it is important to give a meaningful starting point for the parameters.
This typically depends on the distribution but in general it could be done by considering the moments and the meaning of the parameters.

4.3.3 The Coefficient of Variation

The *Coefficient of variation* c_v computed empirically from a trace can guide the choice of the most appropriate distribution.

If the c_v is greater than 1, a two-stage Hyper-exponential distribution can be used to model the system.
If the c_v is less than 1, Hypo-exponential distributions can be used to model the process, more so if we know that the composing stages are executed one after the other and so can be measured separately.
However, Erlang and Hyper-Erlang distributions are typically preferred to Hypo-exponential because they are easier to manage.

5 Generating a Trace

Probability distributions allow us to describe the workload of a system in terms of *interarrival* and *service* times.
If we are able to generate samples of the distribution, we can create meaningful synthetic traces that allow us to study the evolution of the performance of the system.

Most mathematical packages and programming languages have built-in primitives for generating uniformly distributed numbers in the $[0, 1]$ interval.
For the other distributions, we need to be able to generate random numbers distributed according to the given distribution.

A problem of generating pseudo-random numbers is that, although machines try to emulate a random pattern, the computation is deterministic and so there will for sure be a point after which the sequence of random numbers repeats itself.
To mitigate this problem, a value called **seed** is used to determine the initial value of the sequence. Notice that two sequences with the same seed will produce the same numbers.
In order to produce random sequences, the seed is set to a perpetually changing value (typically the system time) at the beginning of the experiment.
The use of the seed allows us to test different sequences in each experiment but also to repeat a sequence perfectly by using the same seed should we need to do so.

5.1 Uniform Distribution Generation

The simplest algorithm that generates random numbers is the *Linear Congruential Generator*.

Let X_n be the n^{th} number of a sequence X, and X_0 be the seed of the sequence.
X_n is computed as follows:

$$X_n = mod\,(a \cdot X_{n-1} + c, m)$$

where a, c and m are constant values identified by the sequence. In particular, m is the *maximum period*: the maximum number of samples after which the sequence repeats.

This algorithm is capable of generating number in the range $[0, m - 1]$.
In order to approximate the uniform distribution in the range $[0, 1)$, the values of the sequence are divided by either $m-1$ or m, depending on whether 1 should

be included in the distribution or not.
The most common implementation excludes the upper bound from the number generation and defines the values u_n of the sequence as:

$$X_n = \mod(a \cdot X_{n-1} + c, m)$$
$$u_n = \frac{X_n}{m}$$

The choice of the parameters is crucial for the algorithm and common values for the main libraries can be found on the Internet.

5.2 Random Variables Generation

The *Inverse Transform Sampling Algorithm* allows the generation of samples of any distribution, starting from a set of random number uniformly distributed in the range $[0, 1]$.
This algorithm is based on the fact that for every CDF of every distribution, the values on the y-axis are uniformly distributed in the $[0, 1]$ range.

If $F_X(x)$ is the CDF of the distribution X, then the values of the distribution can be computed starting from numbers u uniformly distributed between $[0, 1]$ and computing the values of X that would correspond to a CDF value of u:

$$u_i \sim X_{Unif<0,1>}$$
$$x_i = F_X^{-1}(u_i) \sim X$$

Notice that, if $v_i \sim Unif < 0, 1 >$, then also $u_i = 1 - v_i \sim Unif < 0, 1 >$. So, we can define all distributions starting from the uniform one, as we can see in the following examples:

$$u_i = F_{Unif<a,b>}(t) = \frac{t-a}{b-a} \qquad a + (b-a)u_i \sim X_{Unif<a,b>}$$
$$v_i = F_{Exp<\lambda>}(t) = 1 - e^{-\lambda t} \qquad -\frac{\ln(u_i)}{\lambda} \sim X_{Exp<\lambda>}$$
$$v_i = F_{Weibull<\lambda,k>}(t) = 1 - e^{-\left(\frac{t}{\lambda}\right)^k} \qquad \lambda^k \sqrt{-\ln(u_i)} \sim X_{Weibull<\lambda,k>}$$
$$v_i = F_{Pareto<\alpha,m>}(t) = 1 - \left(\frac{m}{t}\right)^{\alpha} \qquad \frac{m}{\sqrt[\alpha]{u_i}} \sim X_{Pareto<\alpha,m>}$$

In the case of discrete random variables, the uniform distribution can be used in a simple algorithm to select the sampled value, as shown below:

for $k = 1 : N$ **do**
  if $\sum_{j=1}^{k-1} p(a_j) < u_i \leq \sum_{j=1}^{k} p(a_j)$ **then**
    return a_k
  end if
end for

In the particular case in which we need to generate an integer uniform discrete distribution between two extremes a and b, we can use the expression:

$$a_i = \min\left(a + \lfloor u_i \cdot (b - a + 1) \rfloor, b\right)$$

The minimum is necessary only if the random variable can also generate its upper bound 1, otherwise it can be removed, leaving only the first part of the formula.

For continuous distributions, although the Inverse Transform Sampling Algorithm is theoretically always applicable, it might not be the best solution, as the function $F_X^{-1}(u)$ does not have an analytical closed form expression.
However, if we take into consideration the definitions of the distributions, some of the previous algorithms can be combined to generate samples in a more efficient way.

For example, an $Erlang < \lambda, k >$ distribution (section 4.2.6) can be computed as the sum of k samples generated by $Exp < \lambda >$ distributions:

$$X_{Erlang<\lambda,k>} = \sum_{i=1}^{k} X_{Exp<\lambda>} = -\frac{\sum_{i=1}^{k} \ln(u_i)}{\lambda} = -\frac{\ln(\prod_{i=1}^{k} u_i)}{\lambda}$$

Similarly, an Hypo-exponential distribution (section 4.2.4) can be generated as the sum of n samples of Exponential distributions, each with its rate λ_i:

$$X_{Hypo-Exp<\lambda_1,\ldots,\lambda_n>} = X_{Exp<\lambda_1>} + \cdots + X_{Exp<\lambda_n>} = -\frac{\ln(u_1)}{\lambda_1} - \cdots - \frac{\ln(u_n)}{\lambda_n}$$

A Hyper-exponential distribution (section 4.2.3) can be generated by:

- first, using a discrete random variable, characterized by probabilities $p_1, \ldots, p_n$ to select the stage,
- then, determine the final sample using the correct exponential distribution.

for $k = 1 : n$ **do**
  if $\sum_{j=1}^{k-1} p_j < u_1 \leq \sum_{j=1}^{k} p_j$ **then**
    return $-\frac{\ln(u_2)}{\lambda_k}$
  end if

end for

Finally, a Hyper-Erlang distribution (section 4.2.7) can be generated by combining a discrete distribution to select the component and the Erlang that we have seen above to generate the sample from the selected stage.

5.3 Generating a Distribution With Given Statistics

There are some cases in which we want to find a distribution that has some given statistics (*e.g., average, skewness, kurtosis, ...*).
In this case, we typically do not perform the fitting of the trace, but we compute the parameters starting from the statistics computed on the trace.

One of those cases is the one of the Normal Distribution (section 4.2.10).
Because of its structure, there is no simple formula to compute $F^{-1}(\cdot)$. However, some languages have a built-in function to compute it (*Matlab uses the* **norminv(·)** *function*).
Several algorithms have been proposed to generate samples of a Normal distribution starting from one (or more) Uniform distribution(s). One such algorithm is the **Box-Muller method** that can generate two independent samples of a Standard Normal distribution starting from two samples u and v of an Uniform distribution:

$$\begin{array}{cc} u_i \sim X_{Unif<0,1>} & v_i \sim X_{Unif<0,1>} \\ x_i = \sin(2\pi v_i)\sqrt{-2\ln(u_i)} & \\ y_i = \cos(2\pi v_i)\sqrt{-2\ln(u_i)} & \\ x_i \sim X_{N<0,1>} & y_i \sim X_{N<0,1>} \end{array}$$

In the case of a Truncated Normal distribution (section 4.2.11), the samples can be generated using a simple rejection algorithm:

repeat
  $x_i = \text{GenNormalRandomSample}(\mu, \sigma^2)$
until $x_i \geq 0$

Notice that, depending on the parameters, this algorithm will run for different amounts of time.

Another case is that of the Log Normal distribution (section 4.2.12), in which the parameters of the distribution can be computed starting from the target average $E[X]$ and the coefficient of variation c_v as:

$$\mu = \ln\left(\frac{E[X]}{\sqrt{1+c_v^2}}\right) \qquad \sigma^2 = \ln(1+c_v^2)$$

From an algorithmic perspective, we can simply start from a Normal sample generated by the Box-Muller technique and use a simple algorithm as such:

$x_i = \exp(\text{ GenNormalRandomSample}(\mu, \sigma^2)\ )$

In the case of a Gamma distribution (section 4.2.13), the parameters can be easily obtained starting from the mean and the coefficient of variation:

$$k = \frac{1}{\sqrt{c_v}} \qquad \theta = \frac{E[X]}{k}$$

The Gamma distribution can be used to generate samples with a given mean and distribution as an alternative to previously seen distributions.
However, the generation of samples is not a trivial task and for this reason we will not consider it.

In the case of the Pearson family of distributions (section 4.2.14), the four parameters of a distribution of such family can be directly related to the first four moments with analytical formulas that consider two values

- β_1, that correspondes to the square of the skewness
- β_2, that denotes the kurtosis (without the "-3")

$$b_0 = \frac{4\beta_2 - 3\beta_1}{10\beta_2 - 12\beta_1 - 18}\sigma^2$$

$$a = b_1 = -\frac{4\beta_2 - 3\beta_1}{10\beta_2 - 12\beta_1 - 18}\sigma\sqrt{\beta_1}$$

$$b_2 = -\frac{2\beta_2 - 3\beta_1 - 6}{10\beta_2 - 12\beta_1 - 18}$$

5.4 Confidence Intervals

The reason why we want to be able to generate synthetic traces is that, starting from the real trace, we are able to identify the distribution and then reproduce a synthetic trace, which in turn can be used to study and predict some of the performance of the system.

A problem with this process is that in order to correctly analyze the performance

of the system, we need to define the correct number of samples to generate so that the synthetic trace reflects the performance of a real system.
In order to properly consider the effect of the number of samples when computing performance indices, we typically determine a *confidence interval* instead of a single value.
The user specifies a *confidence level* $0 < \gamma < 1$, and the measures are returned as an interval $[l_\gamma, u_\gamma]$ such that the performance index α has a probability of falling in such interval that is equal to the requested value γ:

$$P(l_\gamma < \alpha < u_\gamma) = \gamma$$

In this way, we solve the issue of determining a specific number of samples that may not correspond to the real system.
In general, we would like the confidence level to be as close to 1 as possible. However, the higher is the confidence required by the user, the harder will it be to solve the problem.

The size of the interval $[l_\gamma, u_\gamma]$ depends on both the confidence level γ and on the number of samples N:

- it becomes large as γ increases
- it reduces with the population size N

Notice that this definition still allows for the technique to return a wrong performance indices estimate, in fact the interval has a probability of $1 - \gamma$ of not including α.

The way in which the interval is computed depends on the requested index and on its statistical properties.

The computation of the interval is pretty easy in the cases in which the measure requires us to compute the average of the samples (*e.g., the average response time*).
In this case, the main idea is to consider each sample as coming from its own distribution X and the average itself as another distribution $\bar{X}$, which can be defined as:

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X$$

Moreover, it has been shown that, if we consider $\bar{X}$ and if we define S^2 as:

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(X - \bar{X}\right)^2$$

and if we call μ the real average that we want to study ($\mu = E[X]$), then the distribution of the following quantity follows a special distribution T_{N-1} that depends only on N and is called *Student N-1 distribution with N degrees of freedom*:

$$\frac{\bar{X} - \mu}{\sqrt{\frac{S^2}{N}}} = T_{N-1}$$

So, a set of samples x_i would determine an instance of both $\bar{X}$ and S^2 and a corresponding instance t_{N-1} of the T_{N-1} distribution:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

$$\frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}} = t_{N-1}$$

Notice that the previous values are instances of the distributions and not distributions themselves.

The T_{N-1} has a symmetrical shape. In particular, an instance t_{N-1} is inside the percentiles $(1-\gamma)/2$ and $(1+\gamma)/2$ with a probability γ.
If we define

$$c_{\gamma,N} = -\theta_{T_{N-1}} \left(\frac{1-\gamma}{2} \right) = \theta_{T_{N-1}} \left(\frac{1+\gamma}{2} \right)$$

then, by definition:

$$P(-c_{\gamma,N} \leq t_{N-1} \leq c_{\gamma,N}) = P \left(c_{\gamma,N} \leq \frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}} \leq c_{\gamma,N} \right) = \gamma$$

From the extreme values of the confidence interval, we can derive the extreme values of the index.
In the worst case possible, $t_{N-1} = -c_{\gamma,N}$ and $\mu = \mu_u$. We can derive μ_u as:

$$\mu_u = \bar{x} + c_{\gamma,N} \sqrt{\frac{s^2}{N}}$$

On the other hand, if $t_{N-1} = c_{\gamma,N}$ and $\mu = \mu_l$, then:

$$\mu_l = \bar{x} - c_{\gamma,N} \sqrt{\frac{s^2}{N}}$$

The actual value of μ will fall between μ_l and μ_u with a probability of γ.

Notice that the Student T distribution tends to the Normal distribution as the degrees of freedom increase. If $N > 30$, the two are almost identical.
To summarize, if we have few samples (*i.e.,* $N < 30$) we compute the percentiles of the Student T and we can express the confidence interval as:

$$\left[\bar{x} - c_{\gamma,N}\sqrt{\frac{s^2}{N}}, \bar{x} + c_{\gamma,N}\sqrt{\frac{s^2}{N}}\right]$$

If we instead have a large number of samples (*i.e.,* $N \geq 30$), and we call $d_\gamma = -\theta_{N<0,1>}((1-\gamma)/2)$ the percentile of the Standard Normal distribution, then the confidence interval is defined as:

$$\left[\bar{x} - d_\gamma\sqrt{\frac{s^2}{N}}, \bar{x} + d_\gamma\sqrt{\frac{s^2}{N}}\right]$$

Notice that the previous mathematical formulas apply only if the considered samples follow a Normal distribution. However, because of the Central Limit Theorem, they become a good approximation for all cases with large enough values of N.
Moreover, for a Normal distribution, the percentiles that correspond to the most common confidence interval do not need to be computed as they are already available,as shown in the following table:

γ	d_γ
99%	2.576
98%	2.326
95%	1.96
90%	1.645

Unfortunately, this technique can be applied only to few measurements:

$$\text{Response time} \longrightarrow R = \frac{W}{C} = \frac{\sum_{i=1}^{C} r_i}{C}$$

$$\text{Average Service Time} \longrightarrow S = \frac{B}{C} = \frac{\sum_{i=1}^{C} s_i}{C}$$

For the other measures that are computed not as averages but as ratios, we can divide the large number of events N into K *runs* of M events each, so that $N = M \cdot K$.
Then, for each run we compute the desired measures. Finally, the confidence interval is computed at the end considering the values obtained in each run as an instance to be averaged.

5.4.1 Checking a distribution

Once we have found a distribution that matches the data of a trace, the corresponding parameters have been fit and we have generated a synthetic trace of samples, we then have to verify if the samples we generated actually follow the distribution we found in the first place.

The techniques that quantify the effectiveness of the fitting process are called *goodness of fit techniques*.
Many techniques exist, however in performance evaluation the choice of the proper technique depends on the specific type of system being modelled.
We will only see a couple of simple graphical procedures.

Notice that in some cases, matching the moments of the distribution is better than matching the maximum likelihood from a performance point of view.

Q-Q plot and P-P plot
To compare two distributions and see if they are similar, we can use the *Q-Q plot* (quantile-quantile plot) and *P-P plot* (probability-probability plot) techniques. They work well either if we compare two analytical distributions, if we compare a distribution with a dataset, and if we compare two datasets.

The Q-Q plot works comparing the two CDFs, considering each of the possible quantiles and using the two measures of quantiles as the coordinates of a point in the Q-Q plot.
Each point in the Q-Q plot is defined as:

$$(x, y) = \left(x, F_Y^{-1}\left(F_X(x)\right)\right) = \left(F_X^{-1}(p), F_Y^{-1}(p)\right)$$

where $x \in \Omega_X = \Omega_Y$ and $p \in [0, 1]$.

The P-P plot starts instead from a value, computes the CDF of the two distributions and uses the two results as the coordinates of a new point in the P-P plot.
Each point in the P-P plot is defined as:

$$(x, y) = \left(p, F_Y\left(F_X^{-1}(p)\right)\right) = \left(F_X(x), F_Y(x)\right)$$

where $x \in \Omega_X = \Omega_Y$ and $p \in [0, 1]$.

For both techniques, the more the two distributions are similar, the more the point will align with the 45◦ line ($x = y$).
However, notice that in the Q-Q plot the points can be in the $(0, \infty)$ range since we are comparing quantiles, whereas in the P-P plot the points are in the (0, 1) range since we are comparing probabilities.
If instead the two distributions are different, in both plots the curve diverges

significantly from the $x = y$ line.

The Q-Q plot can be easily created in we have samples coming from two distributions, assuming that they have the same number of samples N.
We need to simply sort the two set of samples and use the samples with the same indices to create the new points.
This means that this techniques works well in case we want to see if a synthetic traces fits the starting distribution.

If instead we want to compare the CDF of the distribution obtained by fitting the parameters with the empirical CDF obtained from the samples, we need to do the following:

$$|y_1, \ldots, y_N| = \text{sort}(x_1, \ldots, x_N)$$

$$\left(y_i, F^{-1}_{X<p_1,\ldots,p_n>}\left(\frac{i}{N}\right)\right) \qquad \text{(Q-Q plot)}$$

$$\left(\left(\frac{i}{N}\right), F_{X<p_1,\ldots,p_n>}(y_i)\right) \qquad \text{(P-P plot)}$$

Notice that, for the Q-Q plot, we need to have an analytical expression of the CDF.

In general, the Q-Q plot is preferred to the P-P plot since it can show if two variables could become identical after a linear transformation. In such case, the points of the Q-Q plot will still align not on the 45◦ line but on another line.
However, most of the times, the P-P plot is simpler to compute and so it is valid to check if a fitted distribution matches the corresponding dataset.

As we have already seen, also in this case the result of the technique depends on the number of samples that we have available and so we need to generate confidence intervals.

6 State Machines

State machines are one of the most used tools in Engineering to formalize sequential processes.
A *set of states* represents all the different states in which the system can be at any given moment.
Transitions determine the way in which the system can change state.

Usually, state machines are represented using *directed graphs*, where nodes represent the state and arcs correspond to transitions.

In general, transitions are triggered by *external events*. We can model external events (and so, transitions) using various distributions.

Starting from given distributions, we would like to generate a synthetic trace of the system's behavior. In particular, we are interested in a trace that accounts for the state in which the system is at any given point in time until a maximum time T.

6.1 Performance Indices

Let us call T_s the total time the system spent in a given state s. Then, we can define the *fraction of time spent in s* π_s simply as:

$$\pi_s = \frac{T_s}{T}$$

Let us call C_e the count of times a given change of state $a \to b$ has occurred due to a transition e, where a and b are state of the system. Then, we can define the *frequency at which transition e was triggered* ϕ_e simply as:

$$\phi_e = \frac{C_e}{T}$$

Notice that the generation of a synthetic trace of a system is not a trivial task. Several formal ways of presenting and considering state machines in performance modelling have been considered in the literature.

6.2 State Machine Analysis

The analysis of a state machine typically starts by enumerating the states.
We can then consider the trace as a function that for each time point returns the state the system was in. Since a system remains in a state for some time, we can encode the trace as a step function using couples (t_i, s_i), representing that the system entered state s_i at time t_i due to a state change.

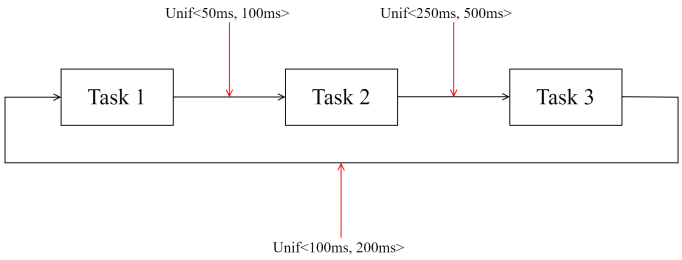


Figure 15: State Machine with sequential tasks.

A general high-level algorithm can be defined as:

```
s = s_0;
t = 0;
while t < T do
    if s = 1 then
        ns = newStateFromStateMachine;
        dt = durationFromDistribution;
    end if
    if s=2 then
        ...
    end if
    ...
    s = ns;        %Move to the next state
    t = t + dt;        %Increase time
end while
```

where, depending on the current state, the next state and the duration need to be computed in a different way.

The way in which we define the next state and the duration of the current state can be one of several alternatives. The main ones are:

- Sequence
- Choice
- Race
- Concurrency

6.2.1 Sequence

Consider a system that can run three tasks.
Each task has associated a quantum of time in which it can run, which is different for each task, depending on their function and priority.

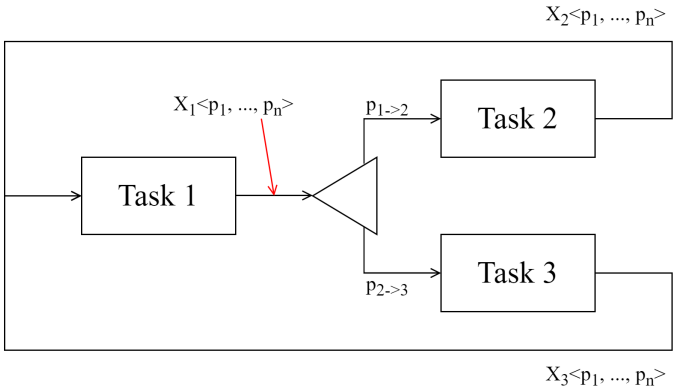


Figure 16: State Machine with a choice transition.

All the tasks are continuously repeated one after the other, an example of which is shown in Figure 15.

In a case such as this one, each state can lead only to a single different state and so we simply need to generate the duration of a state according to the correct distribution.

Also, to compute performance indices, we can simply extend the code with counters.

6.2.2 Choice

In the case of a choice, the next state after a certain state X is not always the same but is defined by a probability of ending up into one of many states, each with its duration distribution and its subsequent states, as shown in Figure 16. As for the other cases, we might want to compute *state probabilities* and the *throughput*. In the case of Figure 16, the throughput is the ratio between the completed cycles and the total time, and a cycle is complete when either task 2 or task 3 performs its computation and ends.

6.2.3 Race

In the case of a race, two or more states compete to be the one that is executed after a certain state X.
The transitions from state X to the states in the race are different for each state. In this case, we might want to compute the victory probability of each state and the average duration of a race.

To decide which is the state that wins the race, we need to compute the time it would take to transition from state X to all the states that are competing

according to the respective distributions, and then select only the smallest time sample, which will determine who is the winner of this race.

A pseudo-algorithm to select the winner of a race between N states can be written as:

```
if s = X then
    dtA = durationFromDistributionA;
    ...
    dtN = durationFromDistributionN;

    dt = min(dtA, ..., dtN);

    if dt == dtA then ns = A;
    end if
    ...
    if dt == dtN then ns = N;
    end if
end if
```

In this case, the choice of the next state is determined using the so called *Race policy*, in which the next state is chosen according to the path that has extracted the minimum time.
However, this is possible only in the few cases in which all the transitions start at the same time, or when we have special event time distributions.

A different case is the one in which the race is not defined as a single event but as a series of events (and so, states) that alternate each other until one player wins the race.

6.2.4 Concurrency

This situation, in which the race is not decided by a single transition, is called concurrency.
Concurrency can occur in many forms and in general it cannot be solved with simple solutions. Typically we need to analyze the system to understand which states of the State Machine will be concurrent ones, which ones will require a choice and which ones have no inherent concurrency.
To handle concurrency, we can usually employ *Discrete Event Simulation*, which the system is encoded not by its states but by its transitions. As it is a complex topic, we will not enter into the details of it.

However, there is one special case of concurrency that we can solve using what we have already seen. This case is the one in which all the timings follow an exponential distribution.
We exploit the *"memory-less property"* of the exponential distribution, which states that no matter how much time we have been waiting for an event, the probability that it will happen afterwards still follows an exponential distribu-

tion.
Thanks to this property, the race policy becomes identical to concurrency. Also the choice can be implemented with race policy, provided that we modify the distributions to consider the probability of a choice (*i.e., since it is an exponential distribution, we can simply multiply λ by the probability of taking a certain choice*).
Notice that, by extending the states, any general distribution can be approximated with exponential stages, making the restriction of using only exponential timings not crucial.

7 Stochastic Processes and Markov Chains

Discrete Event Simulation and State Machines with exponentially distributed transitions are a useful tools with which we can analyze many systems.
However, they are based on random number generation. This implies that the solutions can never be accurate and we need to rely on presenting confidence intervals rather than precise results.
A *Stochastic process* is a set of random variables $X_1, \ldots, X_n$ that operate over the same set.
Since the variables might be correlated, the process must be described with the probability of obtaining a given outcome for a variable i, based on the values of the previous outcomes:

$$P(X_i = a_i | X_1 = a_1, \ldots, X_{i-1} = a_{i-1})$$

However, we can simplify things if we consider smaller levels of correlation among random variables.

The index i of the random variables can be either **discrete** pr **continuous**.
Notice that, if the index is continuous, it usually corresponds to the time.
The set of outcomes can also be discrete or continuous.

In performance evaluation, the index is usually continuous (*i.e., time*) whereas the set of outcomes is discrete, and corresponds to the states of a State Machine.
Continuous Time Markov Chains are a special type of stochastic process with discrete state and continuous time. They are characterized by a set of states (finite or infinite) in which the system can be.
A Continuous Time Markov Chain (CTMC) remains in a state s_j for a random exponentially distributed amount of time, after which the system transitions to another state.
In a Markov Chain, we can define the probability of the system being in any state at a given time t_m depends only of the state in which the system was at time t_{m-1} and how much time has passed since then. We can denote this probabilty as:

$$\pi_i(t) = \Pr\{Z(t) = s_i\}$$

CTMCs are typically drawn as graphs, where the nodes represent the states and the arcs represent the possible transitions between the states.
In practice, there is a one-to-one correspondence between a State machine with exponential transition times and a CTMC.

The set $\Omega = \{s_1, \ldots\}$ of all the possible states the system can assume is called the *state space* of the model.

7.1 Chapman-Kolmogorov Master Equation

Each transition from a state s_i to a state s_j has an associated rate q_{ij} which corresponds to the rate of an exponential random variable.
If there is no arc connecting state s_i to state s_j, then $q_{ij} = 0$.
If there are multiple arcs exiting from state s_i, the system evolves along the path of the event that happens first (*i.e., race policy is applied*).

The transition rate q_{ij} can be seen as the limit of the probability that the system performs a jump in a small time Δt, divided by Δt:

$$q_{ij} = \lim_{\Delta t \to 0} \frac{\Pr(\text{"}\textit{System jumps from } s_i \textit{ to } s_j \textit{ in } \Delta t\text{"})}{\Delta t}$$

If Δt is small enough, the definition can be inverted to obtain:

$$\Pr(\text{"}\textit{System jumps from } s_i \textit{ to } s_j \textit{ in } \Delta t\text{"}) = q_{ij} \cdot \Delta t$$

We can express the probability $\pi_i(t + \Delta t)$ as the sum of the probabilities of following two events:

- The system was already in state s_i at time t and did not move to another state in Δt
- The system was in a state s_j and jumped to state s_i in Δt

What we are interested in is to define those two events.
The second event can be simply defined as:

$$\pi_j(t) \cdot q_{ji} \cdot \Delta t$$

Moving to the first event, how do we define the fact that the system has not jumped from state s_i to another state? We can define this probability as 1 - the probability that the system jumped to another state. So, the probability of the first event can be defined as follows:

$$\pi_i(t) \cdot \left(1 - \sum_{j \neq i} (q_{ij} \cdot \Delta t)\right)$$

CMTC allows us to compute the probability $\pi_i(t)$ for every state $s_i \in \Omega$ at any given point in time t. More formally, from $\pi_i(t)$, the probability $\pi_i(t + \Delta t)$ can be defined as:

$$\pi_i(t + \Delta t) = \pi_i(t) \cdot \left(1 - \sum_{j \neq i} (q_{ij} \cdot \Delta t)\right) + \sum_{j \neq i} (\pi_j(t) \cdot q_{ji} \cdot \Delta t)$$

We can simplify this equation be defining:

$$q_{ii} = -\sum_{j \neq i} q_{ij}$$

and so the equation becomes:

$$\pi_i(t + \Delta t) = \pi_i(t) + \pi_i(t) \cdot q_{ij} \cdot \Delta t + \sum_{j \neq i} (\pi_j(t) \cdot q_{ji} \cdot \Delta t)$$

We can then insert the second term of the sum into the summation, simplifying the equation as:

$$\pi_i(t + \Delta t) = \pi_i(t) + \sum_{j} (\pi_j(t) \cdot q_{ji} \cdot \Delta t)$$

With some computation, we can obtain the following equation:

$$\frac{\pi_i(t + \Delta t) - \pi_i(t)}{\Delta t} = \sum_{j} (q_{ji} \cdot \pi_j(t))$$

and, if we apply the limit $\Delta t \to 0$, we obtain:

$$\frac{d\pi_i(t)}{dt} = \sum_{j} (q_{ji} \cdot \pi_j(t))$$

which is known as the *"Chapman-Kolmogorov master equation"*.

To solve the system of differential equations, we can use a row vector containing the various $\pi_i(t)$ and a matrix Q (called the *Infinitesimal generator*) containing all the values of q_{ij}. Due to the definition of q_{ii}, the elements of all the rows of Q must sum up to 0.
If we multiply the vector and the matrix, we obtain a new row vector containing all the values of the derivatives:

$$\begin{bmatrix} \pi_1(t) & \cdots & \pi_n(t) \end{bmatrix} \begin{bmatrix} q_{11} & \cdots & q_{1n} \\ \vdots & \ddots & \vdots \\ q_{n1} & \cdots & q_{nn} \end{bmatrix} = \begin{bmatrix} q_{11}\pi_1(t) + \cdots + q_{n1}\pi_n(t) & \cdots & q_{n1}\pi_1(t) + \cdots + q_{nn}\pi_n(t) \end{bmatrix}$$

So, the Chapman-Kolmogorov equation can be written in the matrix form as:

$$\frac{d\pi(t)}{dt} = \pi(t) \cdot Q$$

where $\pi(t) = \begin{bmatrix} \pi_1(t) & \cdots & \pi_n(t) \end{bmatrix}$.
If we know the initial state distribution $\pi(0)$ of the system, we can compute the probability distribution at time t by solving the differential equation using $\pi(0)$ as the initial condition.

This is referred to as the *transient solution* of the model.

7.2 Defining the Infinitesimal Generator

In general, we can define the infinitesimal generator Q starting from the graphical representation of the CTMC.
First, we need to enumerate the states and associate them with rows and columns of matrix Q.
Then, we put the transition rates associated to the arcs in the corresponding cells of the matrix.
Then, we compute the values on the main diagonal of Q as the negative of the sum of all the elements of the rows.
Finally, we fill with value 0 all the remaining cells of the matrix (we may avoid this step if the matrix is initialized to 0 by default).

We also need to define the initial condition. In general, we can identify a certain state as the *initial state*, so that the initial condition has all probabilities set to 0 except for the starting state, which has a probability of 1.
However, the only constraint is that the sum of the elements of $\pi(0)$ is equal to 1, which means that we could also use a random vector.

Tools like Matlab offer functions that are already capable of solving this problem. If we are able to define the infinitesimal generator Q and the initial condition p0, then we can solve the problem for each time instant t using the function:

```
[t, Sol] = ode45(@(t, x) Q'*x, [0 10], p0');
```

Notice that the function computes a *matrix by column vector* product, so we need to modify our data structures to adapt to it.

The solution of the ODE is a time-dependent vector $\pi(t)$ that tells us the probability of each state at each time instant t.
We can then use this vector as the basis for computing other performance indices.

Notice that the result of the ODE greatly depends on the initial state. However, all solution for a system tend to the same limiting values, no matter the initial state.

Based on what we have seen in this section, starting from the trace of a real model, if we can assume that all the transitions of the model are exponentially distributed, we can rewrite the model as a CTMC and obtain our results as a value that approximates the real system instead of confidence intervals obtained using random number generation.

The *Matrix Exponential* of a matrix is defined as:

$$e^M = \sum_{k=0}^{\infty} \frac{M^k}{k!}$$

We can apply to Qt and obtain:

$$e^{Qt} = \sum_{k=0}^{\infty} \frac{Q^k t^k}{k!}$$

We can derive that:

$$\frac{de^{Qt}}{dt} = \sum_{k=0}^{\infty} \frac{Q^k}{k!} \frac{dt^k}{dt} = \sum_{k=1}^{\infty} \frac{Q^k}{(k-1)!} t^{k-1} = Qe^{Qt} = e^{Qt}Q$$

This demonstration shown that the Matrix exponential is a solution of the Chapman-Kolmogorov equation. In particular, since $e^{Q \cdot 0} = \mathbb{I}$, we can derive that:

$$\pi(t) = \pi(0) \cdot e^{Qt}$$

Notice that many mathematical packages (*e.g., Matlab*) provide built-in functions to compute the Matrix exponential (in the case of Matlab, the function is `expm(M)`).
In this way, we do not need to store the result of ODE, but we can directly compute the probability at any time t.

7.3 Steady-State and Transient Analysis of CTMC

Under very common hypothesis, the transient probability $\pi(t)$ tends to a fixed vector π as t tend to infinity, regardless of the distribution of the initial state $\pi(0)$.
This limit distribution π is called the *Steady-state distribution* of the CTMC.

However, since the rows of matrix Q sum up to 0, they are not linearly independent and the system has infinite solutions.
We can then use the normalizing condition that the sum of the probabilities of the states is equal to 1 to find a single solution:

$$\begin{cases} \pi \cdot Q = 0 \\ \sum_{i=1}^{n} \pi_i = 1 \end{cases}$$

To compute the stationary solution of a given system, we can consider the matrix Q, replace one of its columns (e.g., the first one) with a column of ones

to express the normalization condition, invert it (in Matlab, we can use the `inv` function), and multiply it with a vector that has a 1 in the same place and 0 in the other places.
In this way, we do not need the initial condition probability of the system, since we are directly computing the steady-state.

7.4 Performance Indices on CTMCs

In Markov Chains, performance indices can be computed by associating rewards to the states and the transitions.
In particular, a model can have several *state rewards* α_k, each one defined by a vector that has a component for each state of the CTMC ($\alpha_k = |\alpha_{k1} \cdots \alpha_{kN}|$). State rewards convey the information of "how much the system gains by staying in state s_i". As such, they can convey information about the properties of the system in a given state (*e.g., the availability of a component*).
Alternatively, we can define *transition rewards* ξ_k, that are defined by matrices and are associated to the transitions of the system. They can convey information about the work performed by the system (*e.g., new jobs entering the system, job completions, ...*).

Starting from the steady-state solution π_i we can compute:

- the average value for state rewards

$$E[\alpha_k] = \sum_{i=1} \pi_i \cdot \alpha_{ki}$$

- the average per time unit for transition rewards

$$E[\xi_k] = \sum_{i=1} \left(\pi_i \sum_{j \neq i} q_{ij} \cdot \xi_{k,ij} \right)$$

Notice that it is possible to transform transition rewards into state rewards.

State rewards are generally used to compute measures such as:

- Utilization
- Average number of jobs in the system / in the queue
- Average energy consumption
- Availability
- Time dependent costs

Transition rewards are generally used to compute measures such as:

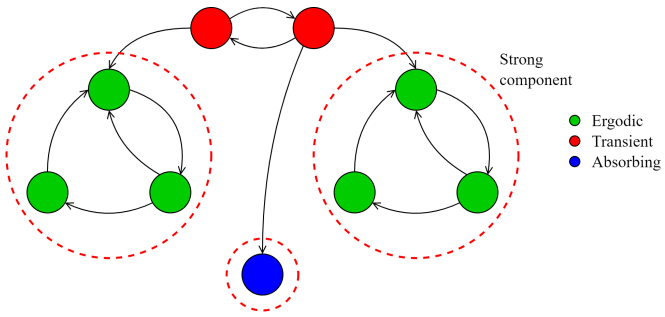


Figure 17: Types of states in a Markov Chain.

- Throughput
- Loss rate / Block rate
- Reliability
- Event dependent costs

Both measures can be computed in steady-state to assess general and long-time performance metrics:

$$E[\alpha_k] = \sum_{i=1} \pi_k \cdot \alpha_{ki} \qquad E[\xi_k] = \sum_{i=1} \left(\pi_k \sum_{j \neq i} q_{ij} \cdot \xi_{k,ij} \right)$$

or in transient, to evaluate properties in a specific point in time t:

$$E[\alpha_k(t)] = \sum_{i=1} \pi_k(t) \cdot \alpha_{ki} \qquad E[\xi_k(t)] = \sum_{i=1} \left(\pi_k(t) \sum_{j \neq i} q_{ij} \cdot \xi_{k,ij} \right)$$

Notice that measures like the response time require more advanced techniques that will not be covered.
However, thanks to *Little's Law* we are able to derive them from previously presented measured in most of the cases.

7.5 State Classification

When we create a CTMC, we might incur in some problems that depend on the recurrence of some states.
In a CTMC, as state can be of one of three types:

- Ergodic

- Transient
- Absorbing

as shown in Figure 17.

An *Ergodic* (or *Recurrent*) state is a state the can occur an indefinite amount of times.
Ergodic states are the "normal" states of a system.
For ergodic states only, it is meaningful to compute steady-state distributions.

Transient states are states for which their probability tends to zero (in finite chains) as the time goes to infinity.
This means that those states will sooner or later be "abandoned" by the system and it will not be possible to return to such states.
Should the system not start from a transient state, the transient states of the Markov Chain will never be reached.
If there are transient states in a system, the Markov Chain may have more than one **strong component**: subsets of states that the system will not be able to exit once it enters them.
If there are more than one strong component, the steady-state solution is not unique and depends on the initial state of the model. Moreover, once a strong component has been chosen, the other strong components will have zero probability. Because of that, the steady-state solution loses some significance. Thus, we want to try to avoid having multiple strong components in our systems.

An *absorbing* state is a state in which there are no output transitions.
An absorbing state is a strong component composed of a single state.
In the Infinitesimal generator Q, the row corresponding to an absorbing component is entirely composed of zeros.
If the system contains a single absorbing state, its steady-state probability is 1, and the steady-state probability of all the other states is 0. If there are more than one absorbing state, their steady-state probabilities depend on the initial state of the model.
The transient probability of absorbing states can be used to study interesting properties of a system, for example the Reliability.

We can study a system containing an absorbing state simply computing $\pi(t)$ as seen before to analyze how long it takes for the system to reach the absorbing state. Alternatively, we can compute the *distribution of the time to absorption* $F(t)$, using the submatrix Q' composed only of the rows and columns of Q that correspond to non-absorbing states:

$$F(t) = 1 - \pi'(0) \cdot e^{Q't} \cdot u$$

where $\pi'(0)$ represents the initial state distribution, excluding the absorbing state, and $u = |1, \ldots, 1|^T$ is a column vector composed of all 1.

Similarly, we can use submatrix Q' to compute the *mean time to absorption* σ:

$$\sigma = -\pi'(0) \cdot (Q')^{-1} \cdot u$$

8 Phase Type Distribution and Markov Arrival Processes

Suppose to have a system with non-exponential distribution. Can we analyze it using CTMC?
The main idea of *Phase-Type distributions* (PH) is to approximate a non-exponential distribution using a sequence of exponentially distributed steps, each called a *"stage"*.
The starting point of such distribution can be one of the stages, chosen with a given probability.
The final state has to be an absorbing one, and the sample of the distribution is considered generated once the Markov Chain has reached it.

This means that any arbitrary distribution can be approximated using a Markov Chain with exponentially distributed transitions.

In general, a PH distribution is defined by the Infinitesimal Generator $\overline{A}$ and the initial state probability vector α:

$$\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}$$

$$\overline{A} = \begin{pmatrix} A & a \\ 0 & 0 \end{pmatrix}$$

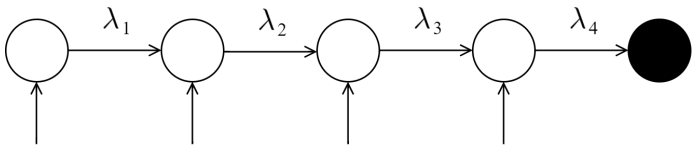
where:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

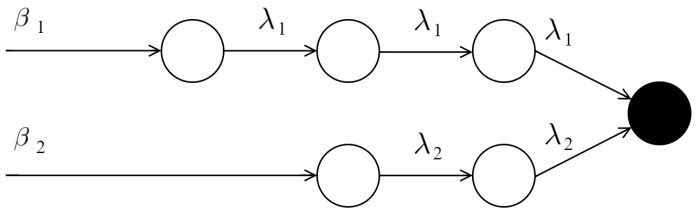
$$a = -A\mathbb{I}$$

Using this definition, the Probability Density Function $f(x)$, the Cumulative Distribution Function $F(x)$, the Laplace transform $\tilde{F}(s)$ and the k-th moment $E[X^k]$ of the distribution can be computed as:

$$f(x) = \alpha e^{Ax} a$$
$$F(x) = 1 - \alpha e^{Ax} \mathbb{I}$$
$$\tilde{F}(s) = \alpha_{n+1} + \alpha(s\mathbb{I} - A)^{-1} a$$
$$E[X^k] = k! \alpha (-A)^{-k} \mathbb{I}$$



(a) A PH distribution in CF-1 form



(b) Hyper-Erlang distribution with two branches

Figure 18: The two most common canonical PH forms.

Although any Markov Chain with a single absorbing state can be used to model a PH distribution, it has been proven that most distributions can be expressed in some canonical forms.
Two of the most known are the *Cumani's Canonical Form 1* (CF-1) and the *Hyper-Erlang* forms, as shown in Figure 18.

Exponential
The PH corresponding to the Exponential distribution with parameter λ is composed of a single state and an absorbing state and is defined as:

$$A = \left| \begin{array}{cc} -\lambda & \lambda \\ 0 & 0 \end{array} \right| \qquad \alpha = | \ 1 \quad 0 \ |$$

as shown in Figure 19.a).

Hypo-Exponential
The PH for the Hypo-Exponential has as many states as the stages plus the absorbing state, and the rate parameters of the distribution are assigned to state transitions. The CTMC always starts from the first state. It is defined as:

$$A = \left| \begin{array}{ccc} -\lambda_1 & \lambda_1 & 0 \\ 0 & -\lambda_2 & \lambda_2 \\ 0 & 0 & 0 \end{array} \right| \qquad \alpha = | \ 1 \quad 0 \quad 0 \ |$$

as shown in Figure 19.b).

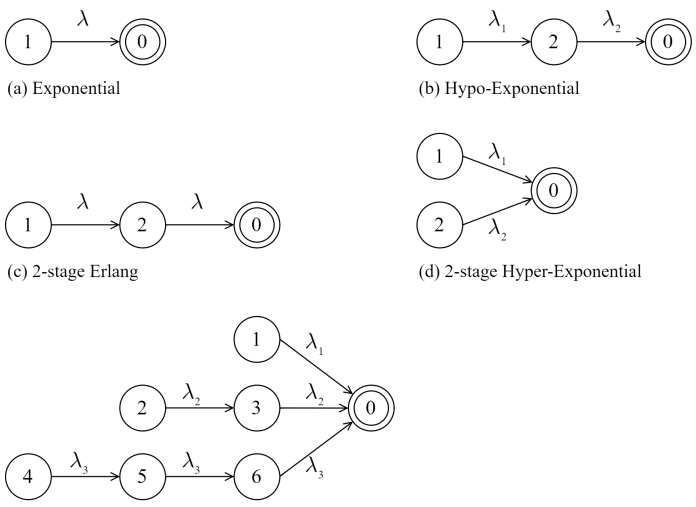


Figure 19: CTMC of the basic distributions.

Erlang
The PH for the Erlang distribution is very similar to the PH of the Hypo-Exponential: the only differences are the jump rates. A 2-stage Erlang can be defined as:

$$A = \begin{vmatrix} -\lambda & \lambda & 0 \\ 0 & -\lambda & \lambda \\ 0 & 0 & 0 \end{vmatrix} \qquad \alpha = \begin{vmatrix} 1 & 0 & 0 \end{vmatrix}$$

as shown in Figure 19.c).

Hyper-Exponential
In the case of an Hyper-Exponential, the PH is composed of as many branches as the parameters, with each jumping directly to the absorbing state. The starting state is chosen according to the given probability. A 2-stage Hyper-Exponential can be defined as:

$$A = \begin{vmatrix} -\lambda_1 & 0 & \lambda_1 \\ 0 & -\lambda_2 & \lambda_2 \\ 0 & 0 & 0 \end{vmatrix} \qquad \alpha = \begin{vmatrix} p_1 & p_2 & 0 \end{vmatrix}$$

as shown in Figure 19.d).

Hyper-Erlang
The PH of a Hyper-Erlang distribution can be obtain by combining the CTMC

of the Hyper-Exponential with the CTMC of the Erlang. For example, a Hyper-Erlang with three branches with 1, 2, and 3 stages respectively, can be defined as:

$$A = \begin{vmatrix} -\lambda_1 & 0 & 0 & 0 & 0 & 0 & \lambda_1 \\ 0 & -\lambda_2 & \lambda_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\lambda_2 & 0 & 0 & 0 & \lambda_2 \\ 0 & 0 & 0 & -\lambda_3 & \lambda_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\lambda_3 & \lambda_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\lambda_3 & \lambda_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$\alpha = \begin{vmatrix} p_1 & p_2 & 0 & p_3 & 0 & 0 & 0 \end{vmatrix}$$

as shown in Figure 19.e).

If we have the trace of a system, we can directly fit the Phase Type distribution using tools such as *HyperStar*.

Notice that, in the CTMCs of general distributions, the state selection probability vector α is implemented in the transitions that "enter" the states where the general distribution starts.
If we have multiple general distributions, we start by expanding each distribution into a CTMC. Then, the absorbing state of each distribution is merged with the starting state(s) of the subsequent one. We also need to consider the initial probability α of the next general distribution and generate as many states as necessary.

Although all the stages are exponentially distributed, the implementation of *Choice*, *Race* and *Concurrency* requires special care, since they are still non-exponential.

In particular, each state must be "multiplied" for the number of phases of each concurrent activity.
Moreover, we need to pay attention to see which events "reset" the memory and restart the activity.

The Choice realized after a general distribution transition is implemented during the ending transition.

In case of Race or Concurrency, since there are several distributions active at the same time, we have to combine each state with all the concurrent ones, in a sort of Cartesian product of the distributions.

State rewards associated with each state of the State Machine are transferred to all the corresponding states of the CTMC obtained through the PH expansion. Transition rewards of the State Machine, instead, are transferred only to the ending transitions of the sub-states of the PH distribution, so that only the transitions that correspond to transitions in the State Machine have a reward associated to them.

8.1 Markov Arrival Processes

Markov Arrival Processes (MAP) extend Phase Type distribution to allow the generation of *correlated random variables*.
Their main characteristic is that they do not have an absorbing state, but instead they differentiate the transitions into two sets:

- State changing transitions
  (normal arrows in the graph)
- State changing + event generation transitions
  (bold arrows in the graph, they mean that a sample of the distribution has been generated)

Notice that state changing + event generation transitions allow a state to jump into itself.

Markov Arrival Processes are described by two matrices:

- $D_0 = |a_{ij}| \rightarrow$ describes the local movements
- $D_1 = |b_{ij}| \rightarrow$ defines the transitions that generate the considered events

The diagonal of D_0 is defined such that:

$$a_{ii} = -\left(\sum_{j \neq i} a_{ij} + \sum_{j} b_{ij}\right)$$

Notice that, thanks to this definition, the matrix $D = D_0 + D_1$ is a proper infinitesimal generator of a CTMC.

For example, we can define the matrices of the Markov Arrival Processes shown in Figure 20:

(a) Exponential

$$D_0 = |-\lambda| \qquad D_1 = |\lambda|$$

(b) Hyper-Exponential

$$D_0 = \begin{vmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{vmatrix} \qquad D_1 = \begin{vmatrix} \lambda_1 p_1 & \lambda_1(1-p_1) \\ \lambda_2 p_1 & \lambda_2(1-p_1) \end{vmatrix}$$

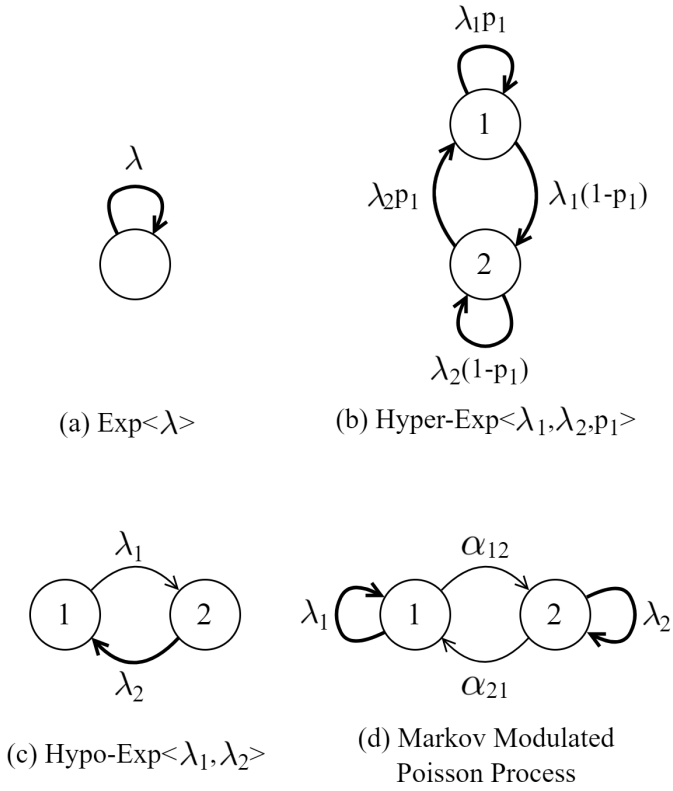


Figure 20: Examples of Markov Arrival Processes.

(c) Hypo-Exponential

$$D_0 = \begin{vmatrix} -\lambda_1 & \lambda_1 \\ 0 & -\lambda_2 \end{vmatrix} \qquad D_1 = \begin{vmatrix} 0 & 0 \\ \lambda_2 & 0 \end{vmatrix}$$

(d) Markov Modulated Poisson Process

$$D_0 = \begin{vmatrix} -\lambda_1 - \alpha_{12} & \alpha_{12} \\ \alpha_{21} & -\lambda_1 - \alpha_{12} \end{vmatrix} \qquad D_1 = \begin{vmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{vmatrix}$$

The possibility of having internal movements (thanks to matrix D_0) and of changing states when an event occurs (thanks to elements of matrix D_1 that are not on the diagonal) allow the model to have a "background" process that modulates the rate and distribution at which events occur.

Special tools and techniques exist for fitting MAP against data traces, allowing to capture both their moments and correlation structures.

9 Queueing Systems

When working with single queue systems, we can use a special notation, composed of 3 to 6 terms, known as *Kendall's notation*.

$$\text{A / S / c [/ k [/ p]] [/ Q]}$$

These six terms (some of which can be omitted if not relevant) represent the following properties:

- **A** → arrival process
  Several acronyms are used in literature. The most common are:
  - M → Exponential / Poisson (Markov)
  - D → Deterministic
  - E_k → Erlang with k stages
  - G → General
- **S** → service time distribution
  The same acronyms as those of arrival processes are employed.
- **c** → number of servers
  Numerical value (or ∞) that defined the number of servers in the system.
- **k** → system capacity
  Maximum capacity of the station (queues + service centers). If the system reaches the total capacity, incoming jobs are either blocked or lost.
- **p** → population size
  Used if the population from which the jobs enter the system is finite. In this way, the arrival rate can be scaled proportionally if the population outside the system becomes comparable to the population inside.
- **Q** → queueing discipline
  Service discipline is generally identified using acronyms:
  - FIFO or FCFS → First-In First-Out
  - LIFO or LCFS → Last-In Last-Out
  - SIRO → Service In Random Order
  - PS → Processor Sharing

Some examples of single queueing system specifications that use Kendall's notation are:

- M/M/1 → Poisson arrival, exponential service time, one server, infinite queue and infinite population.

- M/G/1/K → Poisson arrival, general service time, one server, queue capacity for K jobs and infinite population.
- M/M/2/5/20 → Poisson arrival, exponential service time, two servers, maximum 5 jobs from a population of 20.
- D/G/1/LCFS → Deterministic arrival, general service time, one server, infinite queue and population, last-come first-served policy.

Let's now summarize some algebraic relations involving finite and infinite sums that are required to analyze the models we will consider next.

The finite sum of a term raised at an increasing power is:

$$\sum_{i=1}^{n} x^i = \frac{1 - x^{n+1}}{1 - x}$$

The infinite sum of a term raised at an increasing power is:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1 - x}$$

The infinite sum of a term raised at an increasing index and multiplied by the index is:

$$\sum_{i=0}^{\infty} i \cdot x^i = \frac{x}{(1 - x)^2}$$

The finite sum of a term raised at an increasing index and multiplied by the index is:

$$\sum_{i=0}^{n} i \cdot x^i = x\frac{nx^{n+1} - (n+1)x^n + 1}{(1 - x)^2}$$

9.1 Birth-Death Processes

Birth-death processes are special CTMCs that study the evolution of the count of the members of a population. Their state is used to count the population in a given time instant.
In each state i, the population can increase to state $i+1$ after a state-dependent exponentially distributed time, with rate λ_i. Such transitions are called *births*.
The population in each non-zero state i can also decrease to state $i-1$ after a state-dependent exponentially distributed time, with rate μ_i. Such transitions are called *deaths*.
Notice that all rate can be different in each state.

The number of states can be infinite. However, if the system is stable, the probability of being in a state tends to zero as i tends to infinity, and remains non-negligible only for the first few states.

The infinitesimal generator Q of such CTMC is a simple tri-diagonal matrix defined as such:

$$Q = \begin{vmatrix} -\lambda_0 & \lambda_0 & & & & & \\ \mu_1 & -\lambda_1 - \mu_1 & \lambda_1 & & & & \\ & \mu_2 & -\lambda_2 - \mu_2 & \lambda_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \mu_i & -\lambda_i - \mu_i & \lambda_i & \\ & & & & \ddots & \ddots & \ddots \end{vmatrix}$$

The steady-state probabilities of such CTMC can be explicitly computed solving the so called *balance equations*:

$$\pi \cdot Q = 0$$

that correspond to

$$\pi_i \cdot (\lambda_i + \mu_i) = \pi_{i-1} \cdot \lambda_{i-1} + \pi_{i+1} \cdot \mu_{i+1}$$

It can be proven that the probability π_n of being in a state n is proportional to the following value:

$$\pi_n \sim \frac{\lambda_0 \lambda_1 \cdots \lambda_{n-1}}{\mu_1 \mu_2 \cdots \mu_n} = \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i}$$

that is, there exists a constant α such that:

$$\pi_0 = \alpha$$

$$\pi_1 = \alpha \frac{\lambda_0}{\mu_1}$$

$$\vdots$$

$$\pi_n = \alpha \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i}$$

This constant α can be determined taking into account that π_n must be a valid probability distribution, and its elements must sum up to 1:

$$\sum_{n=0}^{\infty} \pi_n = \sum_{n=0}^{\infty} \alpha \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i} = 1$$

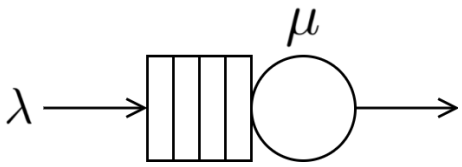


Figure 21: Example of an M/M/1 queue with arrival rate λ and service rate μ.

from which we can derive:

$$\alpha = \pi_0 = \left[\sum_{n=0}^{\infty} \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i} \right]^{-1}$$

$$\pi_n = \pi_0 \cdot \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i}$$

9.2 M/M/1

Let's now focus on an M/M/1 queue with arrival rate λ and service rate μ (*i.e., the average service time is* $D = 1/\mu$), as shown in Figure 21.

This queueing system can be modeled using a birth-death process where the state counts the number of jobs in the system, in which the births have a constant rate λ and the deaths have a constant rate μ.

Using the previous results, we can derive the probability of having n jobs in the system π_n as:

$$\pi_n = \pi_0 \cdot \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i} = \pi_0 \cdot \frac{\lambda^n}{\mu^n}$$

$$\pi_0 = 1 - \frac{\lambda}{\mu}$$

$$\pi_n = \left(1 - \frac{\lambda}{\mu}\right) \cdot \left(\frac{\lambda}{\mu}\right)^n$$

We can define the *traffic intensity* ρ as:

$$\rho = \frac{\lambda}{\mu} = \lambda \cdot D$$

We can see that the queue length is geometrically distributed with parameter $q = 1 - \rho$:

$$\pi_n = (1 - \rho) \cdot \rho^n = P(X_{Geom<1-\rho>=n})$$

As such, we can compute the probability of having more than n jobs in the system as:

$$P(N_{<\rho>} > n) = \rho^{n+1}$$

Using this result, we can derive the utilization U as:

$$U = P(N_q > 0) = \rho$$

and we can compute the average number of jobs in the system as:

$$N = E[X_{Geom<1-\rho>}] = \frac{\rho}{1-\rho}$$

The average response time R can then be computed using Little's Law:

$$R = \frac{N}{X} = \frac{D}{1-\rho} = \frac{1}{\mu - \lambda}$$

If we remove the average service time, we can then find the average time spent in the queue:

$$\Theta = R - D = \frac{\rho D}{1-\rho}$$

Using Little's Law, we can also compute the *average number of jobs in the queue* N_q (jobs that are in the system, but not in service):

$$N_q = \lambda \cdot \Theta = N - U = \frac{\rho^2}{1-\rho}$$

We know from the theory that the sum of a geometrically distributed random number of samples taken from independent exponential distributions, is exponentially distributed with parameter equal to the product of the parameters of the two distributions:

$$\sum_{i=1}^{X_{Geom<\alpha>}} X_{Exp<\beta>} \sim X_{Exp<\alpha \cdot \beta>}$$

From this result we can determine the distribution of the response time of an M/M/1 queue:

$$R_{<\rho>} = X_{Exp<\mu-\lambda>}$$

We can then use this result to compute the probability that the response time is greater or lower than a given value:

$$P(R_{<\rho>} > t) = e^{-\frac{t}{R}}$$

and we can also determine the percentiles of the response time distribution as:

$$\theta_{R_{<\rho>}}(k) = -\log\left(1 - \frac{k}{100}\right) \cdot R$$

9.3 Multiple Servers

A common approach to allow a system to handle a larger number of jobs is to use several servers that share the same queue (*i.e., multiple servers queueing stations*).
This type of model is based on a few assumptions:

- All the jobs wait in a single, common queue.
- All the servers are equivalent.
- As soon as one of the servers becomes available, the next job from the queue immediately enters it.
- At the arrival, if multiple servers are available, the choice is random with equal probability of joining any of the free servers.

Multiple server stations are important for modelling modern computing infrastructures since they are the simplest way to consider multi-core CPUs.

When considering multiple servers, the concept of utilization requires a little more attention. In particular, the utilization law must be used in the proper way.
Notice that each server of a multiple server station has its own busy time, and the busy time $B(T)$ of the system can be defined as the sum of the busy times of the single servers, as shown in Figure 22.

In a multiple server model, the total utilization of the system is a number between 0 and the number of servers c:

$$0 \leq U = \frac{B(T)}{T} \leq c$$

Of more interest is the *average utilization* $\overline{U}$ that focuses on a single one of the c servers:

$$\overline{U} = \frac{U}{c} = \frac{B(T)}{c \cdot T} \qquad 0 \leq \overline{U} \leq 1$$

The Utilization Law assumes the following form when considering also the average utilization:

$$U = X \cdot S \qquad \overline{U} = \frac{X \cdot S}{c}$$

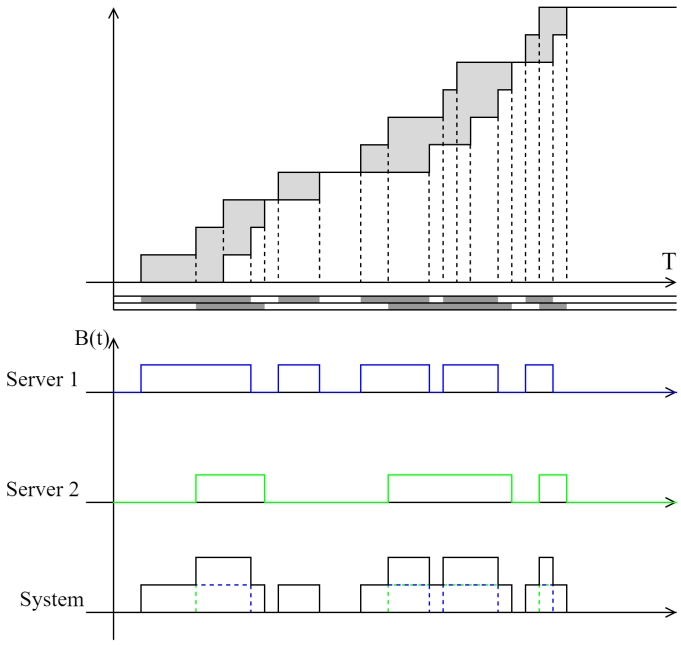


Figure 22: Arrivals, completions and busy time of a two-server station.

From this, we can define a new bound to the maximum arrival rate that a c-server system is capable of handling before becoming unstable:

$$\lambda \leq \frac{c}{S} \qquad S \leq \frac{c}{\lambda} \qquad E[Y] \geq \frac{S}{c}$$

We there are more than one job in service in a system, the events that represent the jobs' completions are concurrent. Because of this, simulating such system requires more attention.
However, if we can assume that the service times are exponentially distributed, the simulation becomes simpler to handle.

9.4 M/M/2

An M/M/2 queue is a station with two servers where both arrival times and service times are exponentially distributed.

If the system has only one customer, it is immediately served by either one of the servers as soon as it enters the queue. In this case the utilization is one, since only one server is not idle. The system will become empty after an exponentially distributed time with rate μ.

If two jobs are in the station, both servers are working. In this case the utilization is 2, and the number of jobs will decrease when the first of the two customers leaves the system. This time is computed as the minimum of two exponential distributions with rate μ.
Because the minimum of a set of exponential distribution is exponentially distributed with rate equal to the sum of the rates, the first job will leave the system after an exponentially distributed time with rate 2μ:

$$f_{\min\{Exp<\lambda_1>,\ldots,Exp<\lambda_n>\}}(t) = f_{Exp<\lambda_1+\cdots+\lambda_n>}(t)$$

If more than two jobs are in the system, a queue will form. In this case, the utilization of the system remains 2, and the number of jobs decreases after an exponentially distributed time with rate 2μ.

The birth-death process that characterizes M/M/2 models has birth rate of λ for all states but has a death rate that is equal to μ if there is one job in the system, and 2μ if there are two or more jobs in the system.
Note that, thanks to the exponential assumption and the equivalence of the servers, there is no need to consider whether a job is being served by the first or the second server.

We can compute the probability of an M/M/2 queue having n jobs in the system π_n as:

$$\pi_n = \pi_0 \cdot \frac{\lambda}{\mu} \cdot \left(\frac{\lambda}{2\mu}\right)^{n-1} \quad n \geq 1$$

and the probability of the system being empty π_0 as:

$$\pi_0 = \frac{2\mu - \lambda}{2\mu + \lambda}$$

The utilization U and average utilization $\overline{U}$ can be derived as:

$$U = \frac{\lambda}{\mu} \qquad \overline{U} = \frac{\lambda}{2\mu}$$

If we define ρ as:

$$\rho = \frac{\lambda}{2\mu} = \overline{U} = \frac{\lambda \cdot D}{2}$$

we can derive that:

$$\pi_0 = \frac{1-\rho}{1+\rho}$$

$$\pi_n = 2 \cdot \frac{1-\rho}{1+\rho} \cdot \rho^n \qquad n \geq 1$$

The average number of jobs in the system can be derived as:

$$N = \frac{2\rho}{1-\rho^2}$$

Using Little's Law we can derive the average response time R:

$$R = \frac{D}{1-\rho^2}$$

and the average time spent in the queue Θ:

$$\Theta = R - D = \frac{\rho^2 D}{1-\rho^2}$$

9.5 M/M/c

An M/M/c queue is similar to the M/M/2 queue, but it has $c > 2$ servers.
In this case, the death rate of the corresponding birth-death process increases linearly up to the state in which all the c servers are busy, after which the death rate remains constant at $c \cdot \mu$.
Similarly to the M/M/2 case, the birth rate is constant for all the states and is equal to λ.

The state probabilities can be defined as:

$$\pi_n = \begin{cases} \frac{\pi_0}{n!} \cdot \left(\frac{\lambda}{\mu}\right)^n & n < c \\ \frac{\pi_0}{c! \cdot c^{n-c}} \cdot \left(\frac{\lambda}{\mu}\right)^n & n \geq c \end{cases}$$

and if we define ρ as:

$$\rho = \frac{\lambda}{c\mu} = \overline{U} = \frac{\lambda \cdot D}{c}$$

we can rewrite the state probabilities as:

$$\pi_n = \begin{cases} \frac{\pi_0}{n!} \cdot (c\rho)^n & n < c \\ \frac{\pi_0 c^c \rho^n}{c!} & n \geq c \end{cases}$$

We can define the probability of the empty system as:

$$\pi_0 = \left[\frac{(c\rho)^c}{c!} \cdot \frac{1}{1-\rho} + \sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} \right]^{-1}$$

Extending the techniques used for the M/M/2 queue, we can define the average number of jobs in the system N:

$$N = c\rho + \frac{\frac{\rho}{1-\rho}}{1 + (1-\rho)\left(\frac{c!}{(c\rho)^c}\right)\sum_{k=0}^{c-1}\frac{(c\rho)^k}{k!}}$$

as well as the average time spent in the queue Θ:

$$\Theta = \frac{\frac{D}{c(1-\rho)}}{1 + (1-\rho)\left(\frac{c!}{(c\rho)^c}\right)\sum_{k=0}^{c-1}\frac{(c\rho)^k}{k!}}$$

and the average response time R:

$$R = D + \Theta$$

9.6 M/M/∞

The number of servers can tend to infinity. This means that every job that arrives in the system never has to wait in the queue as it will always be handled by a server immediately.
Infinite servers can be used to model activities that are performed by the job itself, and result in delaying its movement for a given amount of time, called *waiting time*.

For a queue with infinite servers, the average utilization is not meaningful, since the number of servers tends to infinity.
The classical utilization, instead, can be defined, since the busy time can be accounted for each job that enters the system. In this case, the utilization becomes identical to the average number of jobs in the system, and the average response time corresponds to the average service time:

$$U = N = \lambda \cdot D \qquad\qquad R = D$$

This type of queue can handle any arrival rate.
The birth-death process corresponding to an M/M/∞ queue is characterized by a constant birth rate λ and a linear increasing death rate $i \cdot \mu$.
We can easily express the empty state probability as:

$$\pi_0 = e^{-\frac{\lambda}{\mu}}$$

and the state probability as:

$$\pi_n = \frac{\pi_0}{n!} \cdot \left(\frac{\lambda}{\mu}\right)^n = e^{-\rho} \cdot \frac{\rho^n}{n!}$$

9.7 Finite Queue Capacity

Differently from what we have seen before, in real systems the capacity to hold waiting jobs is always finite.
Finite capacity means that there is a limit to how much the queue can grow. When modelling, we need to take this into accounts to specify what happens when such event occurs.
When a queue with finite capacity reaches its bound, two distinct policies can be applied:

- Loss
- Blocking

Loss
With the *loss* policy, jobs that arrive when the queue is full are lost.
This type of policy is generally used when modelling packet networks, where frames are discarded if a buffer is full.

In this case, a new performance index is defined: the *drop rate*.
If we call $L(T)$ the number of jobs lost up to time T, then the drop rate D_r is defined as:

$$D_r = \lim_{T \to \infty} \frac{L(T)}{T}$$

As this performance index is defined, a new relation is added: the *arrival rate* is equal to the *drop rate* plus the *throughput*.

$$\lambda = D_r + X$$

The asymptotic behavior of a real system is the following:

- The throughput X grows linearly with the arrival rate λ up to the point of saturation of the buffer, after which the throughput is capped at a maximum,
- The drop rate D_r remains at zero up to the point of saturation of the buffer, after which it starts to grow linearly with the arrival rate.

Based on the capacity of the buffer, the actual behavior of the system will be more or less similar to the asymptotic behavior.

Blocking
When the *blocking* policy is used, jobs are never lost.
Blocking stops the arrival of new jobs to the station when its maximum capacity is reached to prevent an overflow of the queue.

If the jobs come from an arrival process, blocking stops the source until the queue has free space to accept new jobs.
If the jobs instead come from another station, blocking sends a signal to stop the service of the upstream node.

However, if a node has more than one input or output, the blocking semantic becomes more complex. For the sake of simplicity, we will not consider such cases.

Several types of blocking are possible. We are interested in two types:

- **BBS** $\rightarrow$ Blocking Before Service
- **BAS** $\rightarrow$ Blocking After Service

Under BBS, if an incoming jobs fills the queue, the upstream node stops servicing new jobs until there is space in the queue for the new results.
As soon as new space is available, the upstream node restarts servicing jobs.
In general, BBS can be applied only if the upstream node or arrival process can be stopped and restarted without any consequences.

Under BAS, the arrival process does not stop immediately when the queue becomes full. However, if the queue receives a new job while the queue is full, the new job waits "outside" the system and the upstream node is stopped.
As soon as there is space in the queue, the job that is waiting can enter the system and the upstream node is restarted.
In general, BAS can be applied only if the upstream node or input process can hold the finished job without any consequences.

Both BBS and BAS have a specific performance index: the *blocking probability*.
Let us call $\beta(T)$ the time during which the arrival process was not working due to the queue being at full capacity.
Blocking probability p_B is then defined as the ratio between the blocking time and the total time:

$$p_B = \lim_{T \to \infty} \frac{\beta(T)}{T}$$

The relation between the arrival rate and the system throughput can then be expressed as:

$$X = \lambda \cdot (1 - p_B)$$

9.8 M/M/1/K

M/M/1/K queues are used to model finite capacity systems.
These processes correspond to birth-death processes with constant death-rate μ and arrival-rate λ until the system reaches its capacity K.
For state $i > K-1$, the arrival-rate is $\lambda_i = 0$, practically removing all the states with a population $i > K$.

Because we are using either loss or blocking, such a system is always stable, even in $\lambda > \mu$.

The state probabilities of an M/M/1/K system use the same expression as the M/M/1 queue, but are limited to a maximum population of K:

$$\pi_n = \begin{cases} \pi_0 \cdot \rho^n & n \leq K \\ 0 & n > K \end{cases}$$

where $\rho = \frac{\lambda}{\mu}$.
Because of this difference, the expression of the probability of an empty system is modified:

$$\pi_0 = \frac{1-\rho}{1-\rho^{K+1}}$$

The utilization of such system is defined as:

$$U = \frac{\rho - \rho^{K+1}}{1-\rho^{K+1}}$$

The average number of jobs in the system is defined as:

$$N = \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}}$$

As we introduced earlier, finite capacity systems employ either the loss or the blocking policy.
Birth-death processes are capable of modelling both.
The probability of having the system full π_K corresponds to either the loss probability p_L or the blocking probability p_B (computed according to the BBS blocking policy):

$$\pi_K = p_L = p_B = \frac{\rho^K - \rho^{K+1}}{1-\rho^{K+1}}$$

We can then compute the drop rate of a system with losses as:

$$D_r = \lambda \cdot p_L = \lambda \cdot \frac{\rho^K - \rho^{K+1}}{1-\rho^{K+1}}$$

The throughput of an M/M/1/K system is defined as:

$$X = \lambda - D_r = \lambda \cdot \frac{1 - \rho^K}{1 - \rho^{K+1}}$$

We can compute the average response time using Little's Law:

$$R = \frac{N}{X} = D \cdot \frac{1 - (K+1)\rho^K + K\rho^{K+1}}{(1-\rho)(1-\rho^K)}$$

9.9 M/M/c/K

If we combine the results of the M/M/c and M/M/1/K models, we can solve also M/M/c/K models.

The state probabilities are defined as:

$$\pi_n = \begin{cases} \frac{\pi_0}{n!} \cdot \left(\frac{\lambda}{\mu}\right)^n & n \leq c \\ \frac{\pi_0}{c! \cdot c^{n-c}} \cdot \left(\frac{\lambda}{\mu}\right)^n & c < n \leq K \end{cases}$$

and the probability of the empty system is defined as:

$$\pi_0 = \left[\frac{(c\rho)^c}{c!} \cdot \frac{1 - \rho^{K-c+1}}{1-\rho} + \sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} \right]^{-1}$$

Since the population is finite, we can compute the average number of jobs in the system and the utilization starting from state probabilities;

$$N = \sum_{i=1}^{K} i \cdot \pi_i$$

$$U = \sum_{i=1}^{c} i \cdot \pi_i + c \cdot \sum_{i=c+1}^{K} \pi_i$$

Loss probability, throughput and drop rate are computed as in the M/M/1/K case:

$$p_L = p_B = \pi_K \qquad X = \lambda \cdot (1 - \pi_K) \qquad D_r = \lambda \cdot \pi_K$$

Using Little's Law we can compute the average response time as:

$$R = \frac{N}{\lambda \cdot (1 - \pi_K)}$$

M/M/c/K models can be implemented using many tools, such as the `JMCH` component of `JMT`, that provides both the computation of the main indices and a visual representation of the system's behavior.

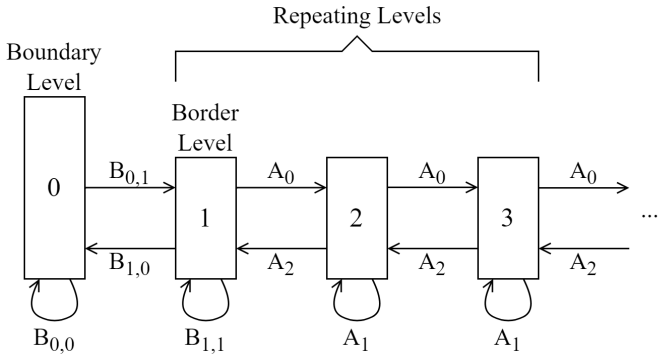


Figure 23: Example of a Quasi-Birth-Death process.

10 Non-Markovian Queueing Systems

Quasi-Birth-Death (QBD) processes are an extension of Birth-Death processes introduced in the previous section.
QBD processes are composed of levels, and each level is divided into a set of sub-states.
A representation of a QBD process is shown in Figure 23.

The transition from one level to another is defined by two matrices: A_0 (forward - birth) and A_2 (backward - death).
Additionally, a matrix A_1 defined the local jumps between sub-states of a level.
Generally, the initial state is allowed to have a different number of sub-states and it can be characterized by different jump matrices $B_{0,0}$, $B_{0,1}$, $B_{1,0}$ and $B_{1,1}$.

Let us call $\pi_{i,j}$ the probability of the system being is sub-state j of level i.
$\pi_i = \mid \pi_{i,1} \quad \cdots \quad \pi_{i,ni} \mid$ is the vector containing all the probabilities of one level.

It can be shown that the stead-state probability solution of the process can be expressed as a function of a matrix R:

$$\pi_n = \pi_1 \cdot R^{n-1} \qquad \text{for } n \geq 1$$

where matrix R is the solution of the following equation:

$$A_0 + R \cdot A_1 + R^2 \cdot A_2 = 0$$

Such equation can be algorithmically solved with a fixed point iterations:

$\mathrm{R}_{\mathrm{old}} = 2\epsilon\mathbb{I}$
$\mathrm{R} = 0$
while $\max|\mathrm{R}_{\mathrm{old}} - R| < \epsilon$ **do**
    $\mathrm{R}_{\mathrm{old}} = \mathrm{R}$
    $\mathrm{R} = -(A_0 + \mathrm{R}_{\mathrm{old}}^2 \cdot A_2) \cdot A_1^{-1}$

end while

Notice that this algorithm converges slowly and faster algorithms have been introduced, however we will not cover those faster algorithms.

The initial solutions π_0 and π_1 can be computed solving this system of equations:

$$\begin{vmatrix} \pi_0 & \pi_1 \end{vmatrix} \begin{vmatrix} B_{0,0} & B_{0,1} \\ B_{1,0} & B_{1,1} + R \cdot A_2 \end{vmatrix} = 0$$

Similar to standard CTMC, this equation has an infinite number of solutions, and the relevant one is chosen such that the probabilities sum up to 1.
If we call u_0 and u_1 two column vectors with as many elements equal to 1 as π_0 and π_1 respectively (since all π_n with $n > 1$ depend on π_1), we can add the following special normalizing condition:

$$\begin{vmatrix} \pi_0 & \pi_1 \end{vmatrix} \begin{vmatrix} u_0 \\ (I - R)^{-1} u_1 \end{vmatrix} = 1$$

The probability $p(n)$ of being in a given level n can then be computed as:

$$p(n) = \begin{cases} \pi_0 \cdot u_0 & n = 0 \\ \pi_1 \cdot R^{n-1} \cdot u_1 & n > 0 \end{cases}$$

We can compute the average population size (*i.e, the level*) of the QBD process as follows:

$$N = \left((I - R)^{-1}(I + (I - R)^{-1}R)\right) u_1$$

QBD processes can be used to study many queueing systems with arrivals and services distributed according to PH or MAP (see Section 8).

10.1 PH/M/1

Let's focus now on a PH/M/1 queue, where the PH distribution is defined by a matrix A and vectors α and a, and the service occurs according to an exponential distribution of rate μ:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ldots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \qquad \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n \qquad a = -A\mathbb{I}$$

Such queue can be studied using a QBD process defined by the following matrices:

$$B_{0,0} = A \tag{1}$$
$$B_{0,1} = A_0 = a \cdot \alpha$$

$$B_{1,1} = A_1 = A - \mu\mathbb{I}$$
$$B_{1,0} = A_2 = \mu\mathbb{I}$$

Using similar techniques and different definitions of the QBD matrices, we are able to analyze MAP/M/1, PH/M/c, MAP/M/c, M/PH/1, M/MAP/1, PH/PH/1 and MAP/MAP/1 queues.
With slightly more complex procedures, we can also analyze finite capacity versions of the previous models.
However, the analysis of multiple server queues with PH or MAP as service time distributions is much more complex due to the concurrency of services.

10.2 M/G/1

M/G/1 systems are characterized by a Poisson arrival rate λ and a general service time distribution.
Let's call X_G the general distribution, characterized by a PDF $f_G(t)$.

We can define the average service time D and the second moment m_2 as follows:

$$D = E[X_G] = \int_0^\infty t \cdot f_G(t) \cdot dt$$
$$m_2 = E[X_G^2] = \int_0^\infty t^2 \cdot f_G(t) \cdot dt$$

Let us also define $\rho = \lambda \cdot D = \lambda \cdot E[X_G]$.
If we assume that the queue is also FCFS, the average response time R can be computed as the sum of three terms: the service time D, the queueing time W and the remaining time of the job in service w:

$$R = D + W + w$$

A job that enters the system will be served after $W + w$ time.
The waiting time W of an arriving job can be computed starting from the number of jobs that are waiting A' (*i.e., the jobs that are in the system but are not in service*):

$$W = A' \cdot E[X_G] = A' \cdot D$$

Thanks to the Poisson arrivals of the M/G/1 queue, the number of jobs that are found in the system by an arriving job is equal to the average number of jobs in the queue $A = N$.
Applying Little's Law, we can express A as a function of the response time:

$$A = X \cdot R = \lambda \cdot R = \lambda \cdot (D + W + w)$$

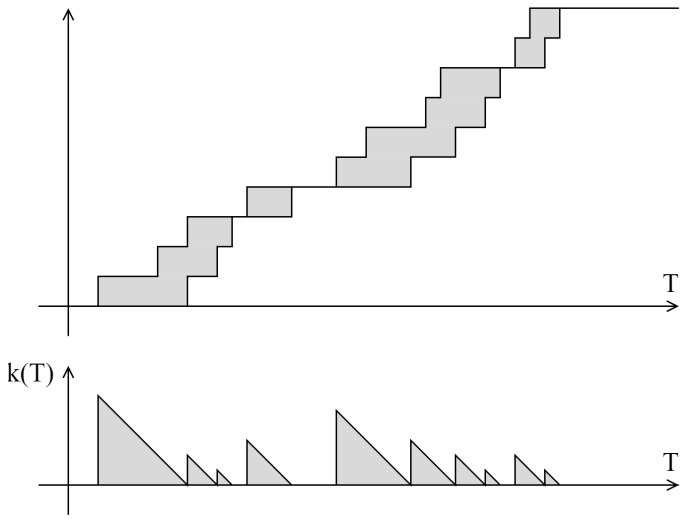


Figure 24: Remaining service time plot derived from the arrival and completions plot.

Because the number of jobs in service is, by definition, the utilization of the system, the total number of jobs found in the system at the arrival can also be expressed as:

$$A = A' + U = \lambda \cdot (D + W + w)$$

that can be simplified thanks to the equation $U = XD = \lambda D$:

$$A' = \lambda \cdot (W + w)$$

Recalling that $\rho = \lambda \cdot D$, we can define the waiting time of the queue as a function of the average remaining time of the job in service:

$$W = \frac{\rho \cdot w}{1 - \rho}$$

and we can then use this result to express the average response time as a function of the remaining service time:

$$R = D + \frac{w}{1 - \rho}$$

If we analyze the system, from the arrival and service plot we can determine the remaining service time at each instant, as shown in Figure 24.
Let us call $k(t)$ the waiting time function. The average waiting time w can be computed as the time average of function $k(t)$:

$$w = \lim_{T \to \infty} \left[\frac{1}{T} \cdot \int_0^T k(t) \cdot dt \right]$$

The jumps in function $k(t)$ correspond to instances $x_i \cong X_G$ of the service time distribution. There is one jump for each completion that happens during time T.
The integral of $k(t)$ can be computed as the area of the corresponding triangles:

$$\int_0^T k(t) \cdot dt = \sum_{i=1}^{C(T)} \frac{x_i^2}{2}$$

If T tends to infinity, then we can derived that:

$$w = \frac{\lambda \cdot m_2}{2}$$

From this result, we can compute the average response time as:

$$R = D + \frac{\lambda \cdot m_2}{2(1-\rho)}$$

Using Little's Law, we can also compute the average number of jobs in the system:

$$N = \rho + \frac{\lambda^2 \cdot m_2}{2(1-\rho)}$$

which is known as the *Pollaczek-Khinchine* formula.
If we recall the relations between the first two moments, we can rewrite the previous equations as:

$$R = D + \Theta \left[\frac{1 + c_v^2}{2} \right] = D + \frac{\rho D}{(1-\rho)} \left[\frac{1 + c_v^2}{2} \right]$$

$$N = \rho + \frac{\rho^2 (1 + c_v^2)}{2(1-\rho)}$$

where $m_2 = D^2(1 + c_v^2)$

10.3 M/G/∞

An M/G/∞ queue has similar results to the M/M/∞.
Such queues are typically used in traffic engineering and telephony.
Their results depend only on the mean of the distribution and are insensitive to the higher moments:

$$U = \lambda \cdot E[S]$$

$$p_n = e^{-U} \frac{U^n}{n!}$$

$$N = U$$

$$R = E[S]$$

10.4 G/M/1, G/M/c, G/G/1, M/G/c and G/G/c

G/M/1 models can be analyzed by solving an equation that involves the Laplace transform of the distribution of the interarrival time and the service rate $\mu = 1/D$.

G/M/c models are also characterized by analytical solutions, and their main idea is that the service process follows a Markovian process. However, the speed of service changes with the length of the queue, as for M/M/c queues.

G/G/1, M/G/c and G/G/c models do not have simple solution techniques. Several techniques are available to determine upper and lower bounds for the systems.

G/G/c systems can be approximated using the *Kingsman formula*, starting from the average interarrival time $T = 1/\lambda$, the average service time D and the coefficients of variation of the two general distributions c_a and c_v:

$$R \cong D + \left[\frac{c_a^2 + c_v^2}{2}\right] \cdot E[\Theta_{M/M/c}]$$

where $E[\Theta_{M/M/c}]$ refers to the expected waiting time of the corresponding M/M/c queue with the same arrival rate and average service time.

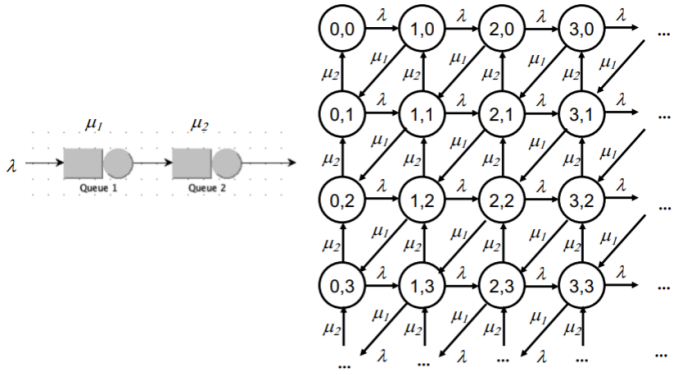


Figure 25: CMTC of a two-station system with exponential service times and Poisson arrivals.

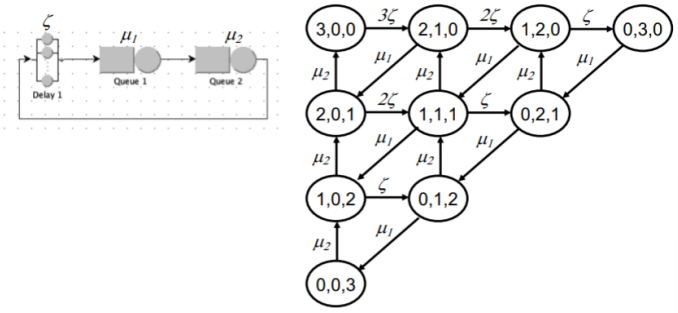


Figure 26: CMTC of a closed time-sharing system with three users.

11 Queueing Networks

The techniques presented in the previous sections can be extended to consider systems composed of more stations.

For example, a system composed of two sequential station, characterized by exponential service times with rates μ_1 and μ_2 respectively, with Poisson arrivals with rate λ, can be modelled as a bi-dimensional infinite CTMC.
In this CTMC, states represent how many jobs are in each of the two stations, as shown in Figure 25.

A similar system with $N = 3$ users and a think time with average Z (rate $\zeta = 1/Z$), can be modelled as shown in Figure 26.

However, such systems generally reach sizes that can no longer be handled, since the complexity is exponential to the number of stations considered.
There are different analytical techniques that can compute the main performance indices indirectly, avoiding the complexity of using a Markov chain.

Notice that, while CTMC models are very general and can be used to describe a large variety of systems, the techniques we will now introduce can be applied only to systems that fulfill special properties.

When more than one service center is involved, we talk about **Queueing networks**.
Queueing networks can either be *open* or *closed*.
In the case of an open model, jobs arrive from outside at a specified arrival rate.
If the model is instead closed, there is a fixed population of jobs that move between the queues inside the system.

11.1 General Topology, Open Models and Closed Models

Open models are characterized by *arrivals* and *departures* from the system. Both events can happen at any station in the network.
In closed models, there is a fixed population of N jobs that continuously circulate inside the system.

In a queueing model with a general topology, several different fluxes can join before entering the same station and a job exiting a station might choose from several paths.
Moreover, loops of any size can be present, meaning that a job can visit a station several times before leaving the system.

Each queue i is characterized by its own average service time S_i.

In open models, there can be several sources of new jobs, each with its own arrival process.
The jobs can also leave the system from different points in the network.

Closed models can be used to describe two types of workload: *batch* and *time-sharing*.
Batch closed models represent systems in which jobs continuously circulate: as soon as a job finishes, it immediately restarts.
Time-sharing models represent systems in which users submit jobs and wait for the response from the system.

The main difference between batch and time-sharing models is that in time-sharing models each job has a "think time": a time interval during which each users (in parallel with the others) examines the results of the previous iteration.

As we have seen before, the think time is identified by a parameter Z and is implemented using a delay station.

The workload of open models is characterized by the arrival rate λ.
The workload of batch closed models is defined by the number of jobs N.
The workload of time-sharing closed models is characterized by the number of jobs N together with the think time Z.

11.2 Visits and Demand

The *visits* v_k represent the average number of times a job passes through a station k from the moment it enters the system up to the moment it leaves the system.
Let us call $C_k(T)$ the number of completions of jobs at station k and $C(T)$ the number of completions of jobs in the system.

Visits v_k can be defined as:

$$v_k = \lim_{T \to \infty} \frac{C_k(T)}{C(T)}$$

Notice that v_k can be greater than one if the model includes loops that make a job pass through a station more than once, and can be less than one if the job has to take one of many exclusive paths.
The value of v_k, whatever it might be, never excludes that a station can be skipped or entered more than once.

The *Service Demand* D_k of a station k accounts for the average time spent by a job at the considered station during all its visits. If we define $B_k(T)$ as the busy time of station k, we can define D_k as:

$$D_k = \lim_{T \to \infty} \frac{B_k(T)}{C(T)} = v_k \cdot S_k$$

where S_k is the average service time that a job spends at station k when it is served.
As for visits, the service demand can be less than, equal or greater than the average service time.

One of the most important relations that we can obtain from the definition of visits is the *Forced Flow Law*:

$$X_k = v_k \cdot X$$

11.3 Response and Residence Time

When we consider nodes characterized by a given visit ratio we can define two permanence times: *Response Time* and *Residence Time*.

The Response Time Φ_k accounts for the average time that a job spends in station k when it enters the node.
The Residence Time R_k account instead for the average time spent by a job at station k during its entire stay in the system.
R_k can greater or smaller than ϕ_k depending on the number of visits. The two values have the same relation as the one between demand and service:

$$R_k = v_k \cdot \Phi_k$$

Notice that, for all single queue systems, $v_k = 1$. This implies that $D_k = S_k$ and $R_k = \Phi_k$.

The Utilization Law and Little's Law can then be defined as:

$$U_k = X \cdot D_k = X \cdot v_k \cdot S_k = X_k \cdot S_k$$
$$N_k = X \cdot R_k = X \cdot v_k \cdot \Phi_k = X_k \cdot \Phi_k$$

11.4 Network Performance Indices

In network models, extra performance indices can be defined:

- System Throughput
- Total System Population
- System Response Time

For what concerns the System Utilization, several definitions can be applied, each with its strengths and weaknesses. For this reason, we will not consider such performance index.

If an open model is stable and no blocking is performed, the *total arrival rate* λ corresponds to the *System Throughput* X (i.e., the average rate at which jobs exit from the system):

$$X = \lambda$$

Open models can be stable only if their utilization is less than one for all their stations:

$$U_k < 1 \qquad \forall k$$

from which we can derive the following relation:

$$\lambda < \frac{1}{D_k} \qquad \forall k \qquad \rightarrow \qquad \lambda < \frac{1}{\max_k D_k}$$

The average total number of jobs in the system can be computed as the sum of the average number of jobs at each station:

$$N = \sum_{i=1}^{K} N_i$$

Notice that this quantity is constant in closed model, and depends on workload λ in open models.

The *Average Response Time* represents the average time spent by the jobs in the system and can be computed as the sum of the Residence Times of the jobs at all nodes:

$$R = \sum_{i=1}^{K} R_i$$

Notice that the system Response Time is the sum of the nodes' Residence Times and not Response Times, because it is influenced by how many times a job passes through a node.

Little's Law continues to be valid using system-wise performance indices:

$$N = X \cdot R$$

11.5 Separable Queueing Networks

Systems that fulfill a certain set of properties are called *Separable Queueing Networks*. Performance indices for such systems can be computed using simple algorithms, starting from the "average number of jobs found in the queue by a new arrival to a station".

All performance indices can be computed using Little's Law and the Utilization Law, starting from the residence time at each station:

$$N = X \cdot R$$
$$N_k = X \cdot R_k$$
$$U_k = X \cdot D_k$$

The Response Time Φ_k of a station k can be computed as the sum of three components:

- the service time of arriving jobs S_k
- the queueing time W_k
- the remaining service time w_k of the job that is being served at the arrival

$$\Phi_k = S_k + W_k + w_k$$

For delay stations, both W_k and w_k are zero because each job is served immediately and in parallel with the others.
Response time Φ_k is then identical to the average service time S_k.

For all the other cases, we assume that $W_k + w_k$ can be computed as the product of the average service time and the average number of jobs found in the system at the arrival:

$$W_k + w_k = A_k \cdot S_k$$

Response time Φ_k then becomes:

$$\Phi_k = (1 + A_k) \cdot S_k$$

The Residence Time R_k for delay station can simply be expressed as:

$$R_k = D_k$$

and, for non-delay stations, as:

$$R_k = (1 + A_k) \cdot D_k$$

The way in which the number of jobs found at the arrival can be computed depends on whether the system is open or closed.
In open models, the average number of jobs found in a queue at the arrival of a new job is identical to the average queue length of the same station:

$$A_k(\lambda) = N_k(\lambda)$$

and the residence time can be expressed as:

$$R_k(\lambda) = (1 + N_k(\lambda)) \cdot D_k$$

Using Little's Law, we can derive the Residence Time as:

$$R_k(\lambda) = (1 + \lambda \cdot R_k(\lambda)) \cdot D_k = \frac{D_k}{1 - U_k(\lambda)}$$

and the average number of jobs in a station as:

$$N_k(\lambda) = \frac{U_k(\lambda)}{1 - U_k(\lambda)}$$

11.6 Routing

When a job, after finishing service at a station, has several possible alternative routes, we need to define an appropriate selection policy.
The policy that describes how the next destination is selected is called *routing*.

One possibility is *probabilistic routing*: each path has a probability of being chosen by a job that has left the upstream station.
Probabilistic routing is particularly effective when modelling the possible alternatives that a job may choose.

Whenever more stations represent identical servers that are used to parallelize a service, we can define two important policies:

- Round robin
- Join the shortest queue

According to the *round robin* policy, the destination chosen by the job rotates among all the possible routes.
Round robin can be used to model systems where the load is balanced by shuffling the possible destination of a job in a fixed order.

In the *Join the Shortest Queue* (JSQ) policy, jobs can query the queue length of the possible destination and choose the one with the shortest length. In multiple destinations have the same queue length, the selection is performed according to a uniform distribution.
JSQ can be used to simulate the intelligence of users that try to minimize their response time.

Routing policies can have a visible impact on the performances computed by the model. For this reason, we need to select the one that matches the real system as close as possible.

For probabilistic routing, it is possible to derive visits starting from the given probabilities.
Let us call p_{ij} the probability that a job exiting station i chooses node j as its next destination. If the considered route is not possible, we set $p_{ij} = 0$.

Probabilistic Routing For Open Models

Let's focus on open models: We call p_{i0} the probability that a job leaves the system after finishing service at station i, and $\lambda_{IN[k]}$ the arrival rate at station k.

We can compute the total system arrival (and the system throughput) starting from the station arrival rates:

$$\lambda_0 = \sum_{k=1}^{K} \lambda_{IN[k]} = X$$

Since $C_k(T) = A_k(T)$, we can derive that $X_k = \lambda_k$. Thanks to the Forced Flow Law, we can obtain the visits as:

$$v_k = \frac{X_k}{X} = \frac{\lambda_k}{\lambda_0}$$

We can determine the arrival rate λ_k to all the stations by solving the following linear system of equations:

$$\begin{cases} \lambda_k = \lambda_{IN[k]} + \sum_{i=1}^{K} X_i \cdot p_{ik} \\ \dots \end{cases}$$

This formula accounts for the jobs that enter the system from outside as well as the jobs that are routed to station k when they exit from other stations in the system. Notice that the summation also includes index k to allow self-loops.

We can compute the visits to each station by exploiting the relation $v_k = \lambda_k/\lambda_0$ in the previous system:

$$\begin{cases} v_k = \frac{\lambda_{IN[k]}}{\lambda_0} + \sum_{i=1}^{K} v_i \cdot p_{ik} \end{cases}$$

This computation can be simplified in most mathematical packages is we use matrix forms:

$$\mathbf{l} = \left| \dots \frac{\lambda_{IN[k]}}{\lambda_0} \dots \right| \qquad \mathbf{v} = |\cdots v_k \cdots| \qquad \mathbf{P} = |\cdots p_{ij} \cdots|$$

$$\mathbf{v} = \mathbf{l} + \mathbf{v} \cdot \mathbf{P} = \mathbf{l} \cdot (\mathbb{I} - \mathbf{P})^{-1}$$

Notice that P does not account for the probabilities of leaving the system, and so its rows do not sum up to one. For this reason $\mathbb{I} - \mathbf{P}$ is invertible.

Probabilistic Routing For Closed Models
Considering now closed models, the computation of visits is more complex.
If we count completions as in open models, then we can define visits as:

$$v_k = \lim_{T \to 0} \frac{C_k(T)}{C(T)}$$

However, it is not easy to define the count of jobs that have completed the execution $C(T)$ since the model is closed.

In order to compute visits and system throughput for a closed model, we must be able to determine when a job has finished its service.
We define that a job has finished its service whenever it exits from a specific station, and we address this special station as the *Reference Station*.
Thus, defining the reference station becomes an important part of the model specification.
For example, in the case of a time-sharing system, the reference station is usually the infinite server that account for the think time of the jobs.
The total number of completions $C(T)$ is defined as the number of completions at the reference station. Also, the system throughput X corresponds to the throughput of the reference station.

Based on the selection of the reference station, the system can have very different performance indices.

Visits in closed models can be computed in the same way as in open models:

$$\begin{cases} \lambda_k = \sum_{i=1}^{K} \lambda_i \cdot p_{ik} \\ \cdots \end{cases}$$

$$v_k = \frac{\lambda_k}{\lambda_{ref}}$$

However, the system has an infinite number of solutions since the rows are not linearly independent.
The solution we are looking for can be obtained by setting $v_{ref} = 1$, which in turn implies that $X = \lambda_{ref}$.

Similar to open models, we can compute visits by solving a system that only uses visits instead of arrival rates:

$$\begin{cases} v_k = \sum_{i=1}^{K} v_i \cdot p_{ik} & \forall k \neq ref \\ v_{ref} = 1 \end{cases}$$

As we have seen for open models, we can define the matrix form of the computation. In this case, we use matrix $\mathbf{P}_0$ as matrix $\mathbf{P}$ from which we remove the column corresponding to the reference station:

$$\mathbf{l} = \left| l_k : \begin{cases} 1 & k = ref \\ 0 & k \neq ref \end{cases} \right| \qquad \mathbf{v} = |\cdots v_k \cdots| \qquad \mathbf{P}_0 = \left| p'_{ij} : \begin{cases} 0 & j = ref \\ p_{ij} & j \neq ref \end{cases} \right|$$

$$\mathbf{v} = \mathbf{l} \cdot (\mathbb{I} - \mathbf{P}_0)^{-1}$$

11.7 Closed Models

As for separable open queueing network models, the solution in closed ones is based on the computation of $A_k(N)$, the average number of jobs at the arrival. The definition of $A_k(N)$ in closed models is different from the one used in open models.
In closed models, we can use the *Arrival Theorem*, which states that "the number of jobs that a customer finds in the queue at its arrival is equal to the average queue length of the system with one less job":

$$A_k(N) = N_k(N-1)$$

The performance indices can then be computed in an iterative way, starting from an empty system and adding one job per iteration. This technique is called *Mean Value Analysis* (MVA):
assuming that a system has been studied up to a workload on $N-1$ jobs, then we can use the arrival theorem to compute the residence time at each station when the population increases by one job:

$$R_k(N) = (1 + A_k(N)) \cdot D_k = (1 + N_k(N-1)) \cdot D_k$$

From these results, we can determine the system response time:

$$R(N) = \sum_k R_k(N)$$

We can then determine the system throughput by inverting the *Response Time Law*:

$$R(N) = \frac{N}{X(N)} - Z \qquad \rightarrow \qquad X(N) = \frac{N}{R(N) + Z}$$

Using Little's Law, we can finally determine the queue length of each station as:

$$N_k(N) = X(N) \cdot R_k(N)$$

At this point, we can then increase the population to $N+1$, repeating the process.

The starting point of the MVA considers an empty system:

$$N_k(0) = 0$$

$$A_k(1) = N_k(0) = 0$$

$$R_k(1) = (1 + A_k(1)) \cdot D_k = D_k$$

We can write the pseudo-algorithm for MVA as follows:

for $k \leftarrow 1$ **to** K **do**
  $Q_k \leftarrow 0$
end for
for $n \leftarrow 1$ **to** N **do**
  for $k \leftarrow 1$ **to** K **do**
    $R_k \leftarrow \begin{cases} D_k & \text{(delay centers)} \\ D_k(1 + Q_k) & \text{(queueing centers)} \end{cases}$
  end for
  $X \leftarrow \frac{n}{Z + \sum_{k=1}^{K} R_k}$
  for $k \leftarrow 1$ **to** K **do**
    $Q_k \leftarrow X R_k$
  end for
end for

MVA computes the solution of a model with complexity $O(N \cdot K)$.
However, as a by-product, it computed also the solution for models with population sizes 1 to $N - 1$.
As such, MVA can be particularly effective when performing sizing studies, where the evolution of the performance against the population is required.

In time-sharing systems, the think time Z is usually not considered in the system response time. As such, Little's Law becomes the Response Time Law to explicitly exclude the think time from the usuale response time:

$$R = \frac{N}{X} - Z$$

11.7.1 JMT

The Java Modelling Tools (JMT) suite includes a component calld JMVA to perform the analysis of open and closed separable models.
The user starts by selecting the classes used in the model, and whether they are open or closed.
As for JSimGraph, the user must also specify either the arrival rate (for open models) or the total population (for closed models).

Since separable models are fully characterized by their demand, the tool does not ask for the topology of the network but only requires the insertion of as

many stations as needed.

JMVA supports both single server queues ("*Load independent*") and infinite server stations.
For each station, the user has to specify the average service time and the visits.
In the case of a time-sharing models, the user specifies also the think time of the terminal station.

For closed models, the user also needs to specify the reference station.

In the "What-if" tab, the user can also setup a range of experiments with a changing workload.

JMVA then computes the performance indices and presents them in a series of panels. If multiple experiments are performed, the tool allows to plot and compare the various performance indices.

11.8 Queue Policies

When a station finishes servicing a job and there are more than one job waiting in the queue, the system must decide which job to serve next.
There are several different policies available, each one has its goal and can work better for a specific application.
There exist two types of queueing policies:

- Non-preemptive
- Preemptive

In *non-preemptive* policies, a job cannot be interrupted after it has started.
In *preemptive* policies, a job in service can be interrupted when some event occurs and the system can start servicing another job.

Non-preemptive queueing policies
Using FIFO (or FCFS) policy, jobs are served in the order in which they arrive.
Using LIFO (or LCFS) policy, jobs are served in the opposite order with respect to their arrival.
With the "Service In Random Order" policy, when a job finishes its service, the next one is chosen randomly among the ones waiting in the queue, with all jobs having the same probability of being chosen.

It can be proven that all non-preemptive queueing policies that do not use the length of the job in service to perform the selection have the same average response time.

However, the policies can have a different variance of the response time, and in particular for the three policies stated above it holds that:

$$Var[FCFS] \leq Var[SIRO] \leq Var[LCFS]$$

Two other non-preemptive queueing policies are the Shortest Job First and the Longest Job First policies, in which the duration of a job is known at the time it enters the station.
In this case, the jobs are ordered according to they service requirement and, once a job leaves the system, the one with the shortest (SJF) or longest (LJF) service time is picked from the queue.

Preemptive queueing policies
The most common preemptive policies are:

- LIFO (or LCFS)
- Round Robin (RR)
- Processor Sharing (PS)
- Shortest Remaining Time First (SRTF)

In preemptive LCFS, as soon as a new job arrives at the station, it stops the one that is currently being served and replaces it. When a job leaves the station, the next one is chosen according to LCFS policy.
Note that a job can be interrupted several times before its completion.
In RR, each job is assigned a maximum service time called *quantum*. If the job does not finish at the end of its quantum, it is interrupted and replaced by another job in the queue. In this case, jobs are selected from the queue in FIFO order.
PS is the equivalent of RR when the length of the quantum tends to zero. In this case, all jobs run concurrently at a lower speed, which depends on the population of the system.
SRTF policy is the equivalent of SJF. If a new job is shorter than the one in service arrives, it immediately interrupts the job being served and replaces it.

When a service is interrupted, there are different ways in which the service can be resumed.
The two main preemption types are:

- Preemptive Resume
- Preemptive Repeat

We will now analyze them for the preemptive LCFS policy (but they can be used for any policy).
In preemptive resume, a job continues from the point it has been stopped, according to the remaining service time at the time of interruption.
In preemptive repeat, a job is restarted from the beginning each time it returns in service.
In preemptive repeat, since the service time is usually sampled from a probabilistic distribution, there are two possibilities:

- Repeat Identical
- Repeat Different

In preemptive repeat identical, the service time is sampled only at the arrival of the job at the station, whereas in preemptive repeat different, a new service time is sampled from its distribution whenever the job is restarted.

12 Separable and Multi-class Models

It has been proven that the analytical results presented for open and closed networks are valid for a large variety of systems.
In particular, those techniques can be applied if a system is a *separable queueing network*.

12.1 Separable Queueing Networks

A queueing network is *separable* if it fulfills the following five assumptions:

1. *service center flow balance*
   Service center flow balance is the extension of the flow balance assumption applied to each service center: the number of arrivals at each center must be equal to the number of completions there.

2. *one step behavior*
   One step behavior asserts that no two jobs in the system "change state" (i.e., finish processing at some device or arrive to the system) at the same time.
   Real systems almost certainly fulfill this assumption.

3. *routing homogeneity*
   Routing homogeneity is satisfied when the proportion of time that a job just completing service at center j proceeds directly to center k is independent of the current queue lengths at any of the centers, for all j and k.

4. *device homogeneity*
   The rate of completions of jobs from a service center may vary with the number of jobs at that center, but otherwise may not depend on the number or placement of customers within the network.

5. *homogeneous external arrivals*
   The times at which arrivals from outside the networks occur may not depend on the number or placement of customers within the network.

Losses clearly violate the 1^{st} assumption, as such models with finite queue capacity that employ the loss policy are not separable.

Probabilistic routing fully satisfies the 3^{rd} assumption. Other routing policies in which the decision depends on the whole state of the system (e.g., JSQ) clearly violate this assumption.
The 3^{rd} assumption has a big implication: as long as nodes have the same demands, two queueing networks have the same performance, independently of their topology.

When we analyze a queueing network, we need to know its topology only to determine the visits and the demands of the different stations. We can then perform the analysis using only the following computed values:

$$X_k = v_k \cdot X \qquad \Phi_k = \frac{N_k}{X_k} = \frac{R_k}{v_k}$$

The 4^{th} assumption is important because it considers the combination of service time distributions, type of servers and queueing policies.
Non-preemptive queueing policies satisfy the 4^{th} property only for exponential service time distributions.
Finite capacity almost always does **not** fulfill the 4^{th} assumption.
Preemptive service policies might satisfy the 4^{th} assumption more often than non-preemptive ones. In particulare, queues that employ the PS policy in open models with exponential arrivals or in closed models, are separable regardless of their service time distribution.

Correlated and burst arrivals in open models generally violate the 5^{th} assumption, making them not separable.
In general, the distribution alone is not enough to determine whether the properties of separable networks are satisfied or not.
In particular, a system might violate or not the 4^{th} and 5^{th} assumptions.

We can add a 6^{th} assumption (reducing the scope of the 4^{th}) then the solutions can be computed with simpler algorithms:

6. *service time homogeneity*
   The rate of completions of jobs from a service center, while it is busy, must be independent of the number of customers at that center, in addition to being independent of the number or placement of customers in the network.

The MVA technique that we introduced in Section 11.7 requires all 6 assumptions.

Multiple servers and server dependencies violate the 6^{th} assumption, so even if separable they require special techniques to be solved.
Infinite server stations (i.e., delay centers) instead never create problems in separable models since they never involve any queueing.

An important note to be made is that, even for systems that do not fulfill the previous properties, the considered techniques can provide meaningful approximations.

12.2 Multi-class Models

Multi-class models distinguish the type of jobs that circulate in the system and allow job-dependent behaviors.

Jobs can be partitioned into *classes*. All the jobs that belong to the same class are characterized by the same properties.
However, each class has different parameters and it is characterized by an independent behavior with respect to the others.

As in single-class systems, multi-class models allow to have:

- Open Models - characterized by external arrivals of jobs belonging to different classes
- Closed Models - networks where each class is characterized by a fixed number of jobs that circulates inside the system, and possibly corresponding different reference stations and think times
- Mixed Models - models in which some classes are populated by jobs coming from external arrivals and some other classes are characterized by a fixed population circulating in the network

Analytical solutions are available for multi-class queueing networks, but extra assumption are required with respect to single-class models.
PS and LCFS with preemptive resume policies can be analyzed without additional requirements.
FCFS, although being the most natural queueing policy, cannot be generally considered for analytical solution of multi-class models. It can be used only if all the classes have the same average service time.

Different classes can be characterized by a different number of visits. This allows the models to have different demands at various resources for the different classes.

Multi-class models are characterized by the following parameters:

- $\lambda_c \rightarrow$ Arrival rate for class c (open classes)
- $N_c \rightarrow$ Population size for class c (closed classes)
- $D_{kc} \rightarrow$ Demand for class c at resource k (including think time in time-sharing models)

If the full set of performance measures is required, then at least one of the following parameters is also needed:

- $S_{kc} \rightarrow$ Average service time for class c at resource k

- $V_{kc} \rightarrow$ Visits for class c to resource k
- $p_{ikc} \rightarrow$ Routing probability for jobs of class c to enter resource k after finishing their service at resource i

Visits should be computed per class, each involving the solution to a specific set of traffic equations.
For **open** classes, the equations to solve are:

$$\begin{cases} \lambda_{kc} = \lambda_{IN[k],c} + \sum_{i=1}^{K} \lambda_{ic} \cdot p_{ikc} \\ \dots \end{cases}$$

$$v_{kc} = \frac{\lambda_{kc}}{\lambda_c}$$

For **closed** classes, the equations to solve are:

$$\begin{cases} v_{kc} = \sum_{i=1}^{K} v_{ic} \cdot p_{ikc} \quad \forall k \neq ref(c) \\ v_{ref(c)} = 1 \end{cases}$$

where each closed class might have a different reference station $ref(c)$.

Global throughput, local throughput, service demand, average service time and visits per class are computed in the same way as in single-class models:

$$D_{kc} = v_{kc} \cdot S_{kc}$$
$$R_{kc} = v_{kc} \cdot \Phi_{kc}$$
$$X_{kc} = v_{kc} \cdot X_c = v_{kc} \cdot \lambda_c (\text{for open classes})$$

Utilization Law and Little's Law are valid for each combination of class and resource:

$$U_{kc} = X_{kc} \cdot S_{kc} = X_c \cdot D_{kc}$$
$$N_{kc} = X_{kc} \cdot \Phi_{kc} = X_c \cdot R_{kc}$$

We can compute station-wise utilization, throughput and average number of jobs by simply summing the metrics for the different classes:

$$U_k = \sum_c U_{kc} \qquad X_k = \sum_c X_{kc} \qquad N_k = \sum_c N_{kc}$$

Little's Law can be applied station-wise: $N_k = X_k \cdot \Phi_k = X \cdot R_k$.
Residence and response times, however, require extra care since they do not simply add up.

To obtain the correct expression, we need to apply Little's Law per class for the entire resource:

$$R_k = \frac{N_k}{X} = \frac{\sum_c N_{kc}}{X} = \sum_c \frac{X_c}{X} R_{kc}$$

$$\Phi_k = \frac{N_k}{X_k} = \frac{\sum_c N_{kc}}{X_k} = \sum_c \frac{X_{kc}}{X_k} \Phi_{kc}$$

System-wise performance indices can be computed per class by summing the corresponding station-wise metrics:

$$N_c = \sum_k N_{kc} \qquad\qquad R_c = \sum_k R_{kc}$$

Notice that Little's Law is also valid system-wise per class: $N_c = X_c \cdot R_c$.

System-wise aggregate performance metrics can be obtained by summing up the performance metrics per resource or per class:

$$X = \sum_c X_c$$

$$N = \sum_k N_k = \sum_c \sum_k N_{kc} = \sum_c N_c$$

$$R = \sum_k R_k = \sum_c \sum_k \frac{X_c}{X} R_{kc} = \sum_c \frac{X_c}{X} R_c$$

Multi-Class Open Models

The stability condition for multi-class open models accounts for the different arrival rates of the different classes:

$$\max_k U_k = \max_k \left\{ \sum_c U_{kc} \right\} = \max_k \left\{ \sum_c X_c \cdot D_{kc} \right\} < 1$$

$$\max_k \left\{ \sum_c \lambda_c \cdot D_{kc} \right\} < 1$$

The relation between residence time and number of jobs found at the arrival is valid for each class c in multi-class models:

$$R_{kc}(\lambda_1, \ldots) = \begin{cases} D_{kc} \cdot (1 + A_{kc}(\lambda_1, \ldots)) & \text{(queue)} \\ D_{kc} & \text{(delay)} \end{cases}$$

For open models, the number of jobs at the station found at the arrival is equal to the average number of jobs. This number however counts all the jobs of all the classes in the station, and is independent from the class:

$$A_{kc}(\lambda_1, \ldots) = N_k(\lambda_1, \ldots) = \sum_c \lambda_c \cdot R_{kc}(\lambda_1, \ldots)$$

We can then derive R_{kc} from the previous expressions as:

$$R_{kc}(\lambda_1, \ldots) = D_{kc} \cdot \left(1 + \sum_{c'} \lambda_{c'} \cdot R_{kc'}(\lambda_1, \ldots)\right)$$

However, we cannot immediately compute R_{kc} since it is present in the summation in the right side of the equation together with the residence times of other classes at the same station.
But we can derive that at each station k the ratio between any two residence times of two classes c and c' has a fixed proportion and in particular is proportional to the ratio of their demands:

$$\frac{R_{kc'}(\lambda_1, \ldots)}{R_{kc}(\lambda_1, \ldots)} = \frac{D_{kc'}}{D_{kc}}$$

We can now derive the residence time as:

$$R_{kc'}(\lambda_1, \ldots) = \frac{D_{kc'}}{D_{kc}} \cdot R_{kc}(\lambda_1, \ldots)$$

We can exploit this result to compute the average response time for a class c at a station k as:

$$R_{kc}(\lambda_1, \ldots) = \frac{D_{kc}}{1 - U_k(\lambda_1, \ldots)}$$

Multi-Class Closed Models
Analytical solutions for closed models exist but their implementation is quite complex. Generally, they are solved using tools such as JMVA.
For closed models, $A_{kc}(\ldots)$ corresponds to the average number of jobs at the considered station k when in the system there is one less job of the target class c:

$$A_{kc}(N_1, \ldots) = N_k(N_1, \ldots, N_c - 1, \ldots)$$
$$R_{kc}(N_1, \ldots) = D_{kc} \cdot (1 + N_k(N_1, \ldots, N_c - 1, \ldots))$$

where $(N_1, \ldots)$ represents the population of each class in the system.

From the residence time of each class at each station, we can determine all the required residence times and the throughputs:

$$R_c(N_1, \ldots) = \sum_{k=1}^{K} R_{kc}(N_1, \ldots)$$

$$X_c(N_1, \ldots) = \frac{N_c}{Z_c + R_c(N_1, \ldots)}$$

Because we have computed the throughput X_c for all the classes, we can now determine the average number of jobs at each station for all the classes:

$$N_{kc}(N_1, \ldots) = X_c(N_1, \ldots) \cdot R_{kc}(N_1, \ldots)$$

We can then determine the entire population at each resource k by simply summing up the results we just obtained:

$$N_k(N_1, \ldots) = \sum_{c} N_{kc}(N_1, \ldots)$$

The main difference with respect to single class models is that the residence time depends on as many configurations as classes.
Each configuration is computed starting from a set of possible configurations with one less job in the system.
Some configurations are obviously shared and can be used in several steps.

The algorithm to compute such models starts from an empty system and adds one job at a time to different classes, maintaining a constant total population at each step (i.e., first all configuration with $N = 1$ are computed, then all the configurations with $N = 2$ and so on). In this way, the algorithm can proceed to as many iterations as required.

Multi-Class Mixed Model
Differently from closed models, mixed models can be solving using MVA on the closed classes after a pre-processing and a post-processing step that account for the open classes.
In particular, the open classes preempt the resources, leaving only the remaining time to the closed classes.
A mixed model is thus stable if it can satisfy all the requests for the considered open classes.

Solution for mixed models is carried out in three steps:

1. Open classes are considered, determining their utilization.
2. The demand of the closed classes is inflated to account for the time resources are preempted by open classes, and a solution for the inflated closed classes is computed using MVA.

3. The residence times of the open classes are computed considering both their utilization and the jobs at the stations belonging to the closed classes.

Formally, the solution steps are: For every station k, for every open class c we determine the utilization U_{kc}:

$$U_{kc} = \lambda_c \cdot D_{kc} \qquad \forall k, \forall c \in Op$$

Then, for each station we compute the overall utilization caused by open classes U_{k0}:

$$U_{k0} = \sum_{c \in Op} U_{kc} \qquad \forall k$$

With the utilization of the open classes, we can determine the *inflated demand* D'_{kc} for the closed classes:

$$D'_{kc} = \frac{D_{kc}}{1 - U_{k0}} \qquad \forall k, \forall c \in Cl$$

Then using the MVA with the inflated demands D'_{kc}, we can determine the performances of the closed classes:

$$R_{kc}, N_{kc}, X_c \qquad \forall k, \forall c \in Cl$$

We can then compute the utilization for all resources for all closed classes:

$$U_{kc} = X_c \cdot D_{kc} \qquad \forall k, \forall c \in Cl$$

Finally, we can compute the average residence time of jobs in the open classes considering the influences of the closed classes:

$$R_{kc} = \frac{D_{kc} \cdot \left(1 + \sum_{d \in Cl} N_{kd}\right)}{1 - U_{k0}} \qquad \forall k, \forall c \in Op$$

and the average number of jobs in the open classes using Little's Law:

$$N_{kc} = \lambda_{kc} \cdot R_{kc} \qquad \forall k, \forall c \in Op$$

Notice that JMVA allows us to consider multi-class models, where each class can either be open or closed, allowing us to define mixed models.
In this case, we need to specify services, visits, demands and reference stations for each class.
Performance indices are then computed per class per resource.

13 Advanced Queueing Network Features

Besides the features we have seen in previous sections, there are also other properties that can characterize a queueing network to model special behaviors that have a big impact on the performances of a system.
These advanced techniques, however, very rarely have analytical solutions or numerical techniques that can compute relevant performance indices.
In most cases, models that exploit these properties are solved via discrete event simulation.

The JMT suite contains a tools called *JSimGraph* capable of solving such models.

One such feature is that of *Map-Reduce applications*, in which a job must be split into several identical subtasks that must be performed in parallel. The job is then considered completed only when all the corresponding subtasks have finished.
This features is modelled using two special primitives called *Fork* and *Join*.
A Fork needs to specify a factor called "Fork degree", denoting how many tasks are created for every possible destination when a job enters the Fork node. When all the tasks reach the Join node, they are then merged back into the original job.

Advanced fork strategies allow to split a job in tasks that belong to different classes, to have a different number of tasks in the dfferent destinations, and to generate a random number of tasks.
Similarly, advanced join strategies allow to implement *KooN* system, where a job is considered to be finished as soon as a subset of the composing task are completed.

Queue Length Dependency
In some systems, the service time distributions or the arrival rates are not constant, and they can vary as a function of the number of jobs in the system.

An example of this are HDDs, sort and serve requests according to their track in the direction of the rotation. This makes the average service time smaller as the queue length increases.

Another example is that of flow control mechanisms, such as *Random Early Detection* (RED), that can reduce the arrival rate in a communication channel by randomly dropping some packets as the queue increases.

To model such mechanisms, we need to provide a different distribution (of service time or interarrival time) for each possible queue length.
This feature is called **queue length dependency**.

When we consider queue length dependency, the utilization of the model is difficult to determine, and we must employ complex solution techniques. Moreover, there are no straightforward rules to determine whether the system is stable or not.

JMT only supports queue length dependent distributions for the service time. In this case, the user must supply a set of queue length ranges and a distribution for each range.
We can then analyze performance indices considering the various workloads or the queue dependent one.

Finite Capacity Regions
Another feature is the Finite Capacity Regions (FCR) that extends the concept of finite buffers to a set of stations, meaning that we can model a finite number of resources to be shared by a set of jobs.

Class Switch
Another feature is Class Switch, according to which a job might need to be served by the same service center multiple times, but each time with a different requirement, effectively changing the service class of the job.
In particular, each class can have different probabilities of transforming into a different class, either open or closed.

13.1 Stochastic Petri Nets

Queueing networks are perfect for modelling systems where jobs are executed through a set of stations. They are characterized by high-level performance indices such as throughput, response times and utilization.

However, they cannot easily model resource contentions and concurrency.
There are other solutions, such as *Stochastic Petri Nets* that can be used to model systems characterized by such features.

Petri Nets are bi-partite graphs, characterized by two sets of elements: *places* and *transitions*. An example of a Petri Net is shown in Figure 27.

Places are used to describe the state of the system and are represented as circles.

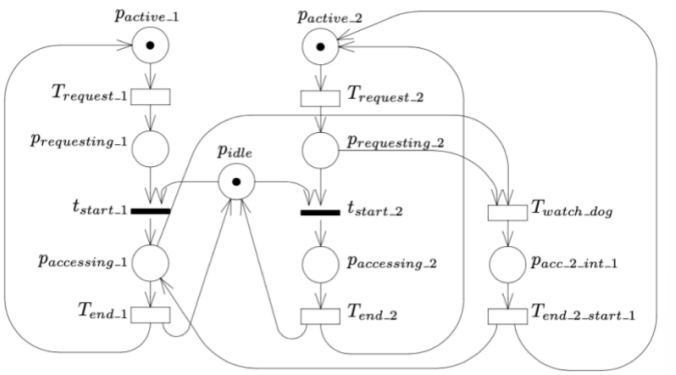


Figure 27: Example of a Petri Net.

Transitions model actions that can change the state of the model. They are represented with bars and boxes.

Each place contains a non-negative number of tokens. Token can either be drawn in the circle or represented with a number or a letter, denoting a parameter.

Arcs are used to connect either places to transitions, or transitions to places.
Focusing on a transition, arcs that end on the transition are called *input arcs* and arcs that start from the transition are called *output arcs*.
Similarly, still focusing on a transition, places connected to it via an input arc are called *input places* and places that are connected to it via an output arc are called *output places*.

Each arc is characterized by a *weight*, a positive integer number. If not specified, the default value is one.

There are two different types of arcs: *normal* (ending with an arrow) and *inhibitor* (ending with a circle).
Inhibitor arcs can only connect places to transitions.

The arcs can have the following different meanings:

- *Input arcs* - model preconditions (i.e., conditions that must be satisfied for an event to take place)
- *Output arcs* - specify the effect of the action
- *Inhibitor arcs* - prevent an event from taking place

If a model is created using a Petri Net, the tokens are used to represent the jobs,

the places define queues or resource availability and transitions correspond to services.

A transition is said to be **enabled** if:

- Each input place has at least as many tokens as the weight of the corresponding input arc.
- Each place connected with an inhibitor arc has less tokens than the weight of the connection.

Enabled transition might **fire**.
The firing of a transition:

- Removes as many tokens as the weights of the input arcs from input places
- Puts as many tokens as the weights of the output arcs to output places

Notice that places connected by inhibitor arcs are unaffected by the firing of a transition.
If the tokens of input places are enough, a transition might fire multiple times. The *enabling degree* of a transition is the maximum number of times a transition might fire before having removed all the token required for another firing from its input places.

A Petri Net is also characterized by an *initial state*: the set of tokens for each place from which the evolution of the system will start.

In a given state of the system, there could be several transitions enabled at the same time. In this case we say that there is a **conflict**.
Stochastic Petri Nets associate timing information to transitions (i.e., a distribution for each transition), and use this information to solve the conflicts. Transitions that have timing informations associated to them are represented with boxes and are called *timed transitions*.

The original definition of Stochastic Petri Nets associate to each transition an exponentially distributed random firing time.
Extended models, called *Non-Markovian Stochastic Petri Nets*, allow the use of general firing time distributions.

Conflicts are solved according to the race policy: the system evolves according to the transition that fires first.

Transitions can be single, multiple or infinite servers.
Depending on the enabling degree of a transition and its server count, the transition might schedule concurrently more than one firing event.

There are also special transitions that have a deterministic time of zero duration. Such transitions are called *immediate transitions* and are drawn with bars instead of boxes.
Conflicts among immediate transitions are solved using weights and priority instead of race policy:

- An immediate transition can fire only if no other immediate transition with a higher priority is enabled
- If several immediate transitions with the same priority are enabled, the one that fires is chosen at random, with a probability proportional to its weight

General transitions always have priority over timed ones.
Models that use only timed transitions with exponential firing times and immediate transitions are called *Generalized Stochastic Petri Nets* (GSPN).

It can happen that a model reaches a state in which no transition is enabled. This situation is called a **deadlock**, and is considered a modelling error.

For Petri Nets, only three types of performance indices are defined:

- *Transition Throughput*
  Counts the average number of firings per time unit per transition.
- *Probability Distribution of Having a Number of Tokens in a Place*
- *Average Number of Tokens*

13.1.1 Colored Petri Nets

In *Colored Petri Nets*, tokens are divided into classes called **colors**. Each place can contain a different number of token for each color.

Each transition can fire in different **modes**: each mode is characterized by a different timing distribution, different enabling and firing rules.
The enabling and inhibition of transitions can require different amounts of different tokens based on the mode. The firing of transitions can also produce different amounts of different tokens based on the mode.

JMT has a partial support for Petri Nets. Job classes are used to introduce colors and the starting population represents the initial number of tokens of the considered color.

13.2 Multiformalism Models

Multiformalism models allow to exploit different formalisms to describe different components of a system.
It is possible to use the most appropriate modelling primitive for each part and combine different languages.

One of the most popular examples is the combination of Queueing Networks and Petri Nets.