Faculty of Information Engineering, Informatics and Statistics
Department of Computer Science

Computer Network Performance

Author:
Simone Lidonnici

26 november 2025

Contents

1 Introduction

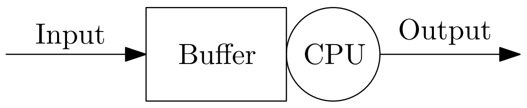
1.1 Notation

To start we will define a bunch of variables that are common to all the systems:

- **CPU Capacity** (μ): expressed as how many jobs can the CPU perform per second on average.
- **Arrival rate** (λ): expressed as how many jobs arrive to the buffer per second on average.
- **CPU Service time (S)**: the mean time a job spends inside the CPU. Can be calculated with the formula:
$$S = \frac{1}{\mu}$$
- **Waiting time (W)**: the mean time a job spends in the buffer.
- **Response time (R)**: the sum of service time and waiting time.
- **Throughput (X)**: expressed as how many jobs are performed by the system per second on average, measured in the output.

1.2 Basic system

We start using a basic system composed by only one CPU and a queue to buffer the jobs that the CPU has to perform. The buffer is supposed to be infinite in size.



In this system we can easy determine the throughput, based on λ or μ:

- $X = \lambda$ if $\lambda < \mu$
- $X = \mu$ if $\mu < \lambda$

The response time can also be calculated with the formula:

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

Example:
If we have a system with $\lambda = 3$ j/s and $\mu = 5$ j/s, the service time will be:

$$S = \frac{1}{\mu} = \frac{1}{5} \text{ s} = 0.2 \text{ s}$$

The response time is:

$$R = \frac{1}{\mu - \lambda} = \frac{1}{5-1} \text{ s} = 0.25 \text{ s}$$

1.3 System with multiple CPU

In the case of a system with multiple CPU (each with a personal buffer) we define the probability of a job going in a specific CPU$_i$ with p_i. The total response time of the system is calculated by the formula:

$$R = \sum_{i=1}^{m} p_i R_i$$

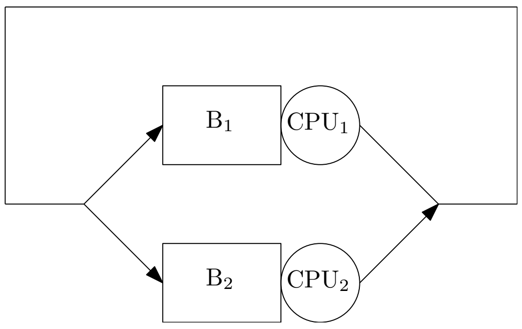
In which m is the number of CPU.

1.3.1 Closed systems

In the case of a closed system an important parameter is the number of jobs in the system (defined with the letter N). There are two types of closed systems:

- **Batch**: a closed system with only servers, where jobs that exit from the CPUs go directly into the buffers.
- **Terminal**: a closed system with terminals, where jobs that exit from the CPUs stay for an average time of Z, called the **think time**, before going into the buffers. We can also define a total system time $T = R + Z$.

Example:
We have a closed system with 2 CPU like the following and a total of $N = 6$ jobs running in the system:



If $S_1 = S_2 = 3$ s the average number of jobs in every buffer is 2 (plus 1 in every CPU), so the response time of the system will be:

$$R = p_1 R_1 + p_2 R_2 = \frac{1}{2}(3 \cdot 3) + \frac{1}{2}(3 \cdot 3) = 9 \text{ s}$$

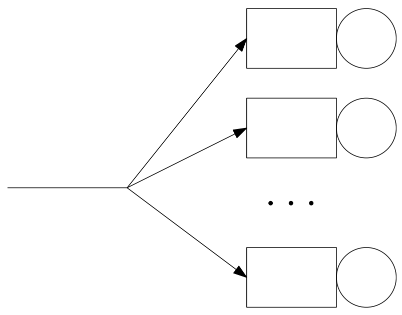
If we change $S_1 = 1.5$ s the average number of jobs in buffer 1 is 0 (1 in the CPU) and in buffer 2 is 4.5 (plus 1 in the CPU). This is caused by the fact that every job going to CPU 1 is completed faster than CPU 2 and the other jobs are stuck in buffer 2. So the average response time in this case is:

$$R = \frac{1}{2}(1 \cdot 1.5) + \frac{1}{2}(5.5 \cdot 3) = \frac{3}{4} + \frac{33}{4} = 9 \text{ s}$$

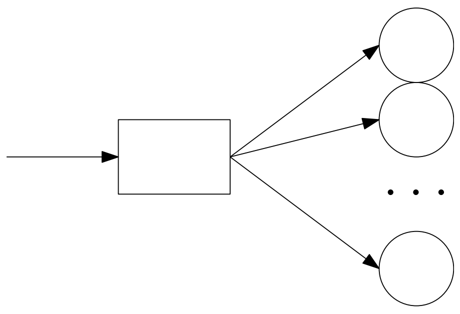
So we can see that upgrading only one CPU on a multi-CPU closed system doesn't change the response time. To lower the response time the routing probabilities must be changed as well.

1.3.2 Open systems

There are 2 different type of open systems with multiple CPUs, with the difference being the number of buffers. If there is one buffer for every server is a **server farm**, otherwise if there is only one buffer in total is a **multiserver system**.



Server farm



Multiserver system

In the case of a server farm we can consider a single branch as a system with only one CPU and arrival rate of $\lambda_i = p_i\lambda$ in which p_i is the probability of a job going on CPU$_i$. If all the branches have the same probability we have that $\lambda_i = \frac{\lambda}{m}$ for a number of CPU m.

2
Utilization of a system

2.1 Average population

We can define the **average population** of a system, that means how many jobs are in the system (counting queue and CPU) on average. We write:

- $\overline{N_S}$ the average population in the CPU, that can also be defined as the probability of the CPU being busy:
$$\overline{N_S} = P(N_S = 1)$$
- $\overline{N_Q}$ the average population in the queue.
- $\overline{N}$ the average population in the system, that is calculated with a sum:
$$\overline{N} = \overline{N_S} + \overline{N_Q}$$

2.1.1 Population over time

We can also define how the population evolve over time as a difference between the total number of jobs that are arrived before time t and all the jobs that have departed before time t.

- $A(t)$ is the total number of jobs arrived before time t, and with an infinite buffer is calculated:
$$A(t) = \lambda t$$
- $D(t)$ is the total number of jobs departed before time t, and sure we can say that:
$$D(t) \leq \mu t$$

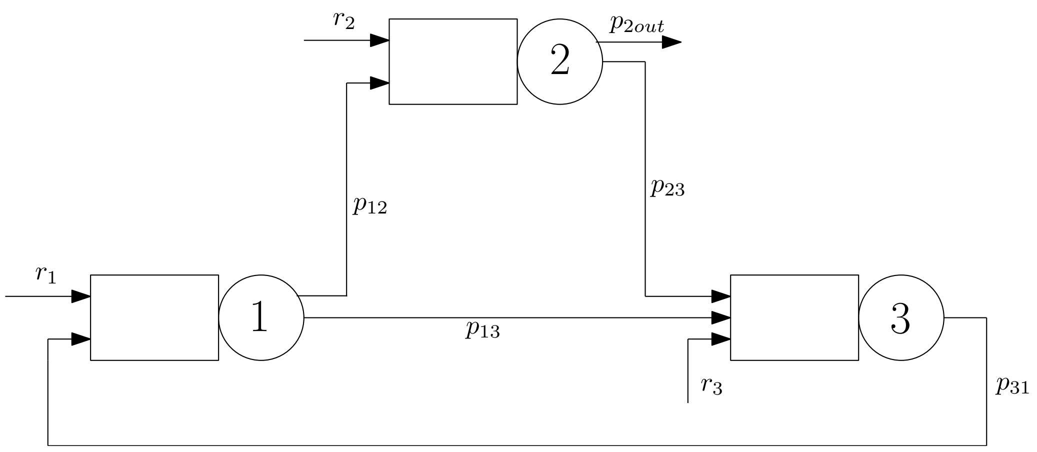
So the population at the time t, denoted as $N(t)$ can be found using the formula:

$$N(t) = A(t) - D(t) \geq \lambda t - \mu t = (\lambda - \mu)t$$

We can define what is called the **stability condition**, a condition in which the queue of the system doesn't grows infinitely. To achieve this condition we need to have $\lambda < \mu$ for every component of the system. If we have $\lambda \geq \mu$ the queue continues to grow and for an infinite amount of time we have:

$$\lim_{t \to +\infty} N(t) = \infty$$

Example:



In the above system we have the following data in which p_{ij} represent the probability of a departed job from CPU$_i$ to go in server j, p_{iout} is the probability of a departed job from CPU$_i$ to exit the system and r_i is the arrival rate from outside the system to server i.:

- $\mu_1 = \mu_2 = \mu_3 = 10$ j/s
- $r_2 = r_3 = 1$ j/s
- $p_{12} = 0.8, p_{13} = 0.2$
- $p_{23} = 0.2, p_{2out} = 0.8$
- $p_{31} = 1$

What value of r_1 keep the system stable?
To do that we need to ensure that every part of the system is stable so:

$$\forall i \; \lambda_i < \mu_i$$

We can write this as a system of equations:

$$\begin{cases} \lambda_1 = r_1 + p_{31}\lambda_3 \\ \lambda_2 = r_2 + p_{12}\lambda_1 \\ \lambda_3 = r_3 + p_{13}\lambda_1 + p_{12}\lambda_2 \end{cases} \implies \begin{cases} \lambda_1 = r_1 + \lambda_3 \\ \lambda_2 = 1 + \frac{4}{5}\lambda_1 \\ \lambda_3 = 1 + \frac{1}{5}\lambda_1 + \frac{1}{5}\lambda_2 \end{cases} \xRightarrow{\text{Resolve for } \lambda_i} \begin{cases} \lambda_1 = \frac{15}{8} + \frac{25}{16}r_1 \\ \lambda_2 = \frac{5}{2} + \frac{5}{4}r_1 \\ \lambda_3 = \frac{15}{8} + \frac{9}{16}r_1 \end{cases}$$

To ensure the stability we have to set every $\lambda_i < \mu_i$:

$$\begin{cases} \lambda_1 = \frac{15}{8} + \frac{25}{16}r_1 < 10 \\ \lambda_2 = \frac{5}{2} + \frac{5}{4}r_1 < 10 \\ \lambda_3 = \frac{15}{8} + \frac{9}{16}r_1 < 10 \end{cases} \xRightarrow{\text{Resolve for } r_1} \begin{cases} r_1 < \frac{26}{5} \\ r_1 < 6 \\ r_1 < \frac{130}{9} \end{cases}$$

So the value of r_1 to keep all the system stable has to be $r_1 < \frac{26}{5}$.

2.2 Utilization

We define the utilization ρ for a device as the fraction of time in which the device is busy. Can be also expressed as the probability of a device being busy, so is equal to $\overline{N_S}$.

Using an observation of τ time, we can calculate the utilization as the sum of times in which the device is busy B divided by the total time τ.

$$\rho = \frac{B}{\tau}$$

Also we can say that the throughput X is the number of completed jobs C divided by the total time τ.

$$X = \frac{C}{\tau} = \frac{C}{\tau} \cdot \frac{B}{B} = \underbrace{\frac{C}{B}}_{\mu} \cdot \underbrace{\frac{B}{\tau}}_{\rho} = \frac{1}{S} \cdot \rho \implies \rho = S \cdot X$$

The formula above is called the **utilization law** and can be also calculated in another way:

$$E(X) = \underbrace{E(X|N_S = 1)}_{\mu} \cdot \underbrace{P(N_S = 1)}_{\overline{N_S}=\rho} + \underbrace{E(X|N_S = 0)P(N_S = 0)}_{0} = \mu \cdot \rho \implies \rho = S \cdot X$$

If we have multiple servers we can calculate the overall utilization of the system simply as the average of all the utilization of the single CPUs:

$$U = \frac{\sum \rho_i}{m}$$

We need to distinguish between 2 cases based on the fact that the buffer is finite or not:

1. If the buffer is infinite and the system is stable, so $X = \lambda$, the utilization can be easily calculated with:
$$\rho = S \cdot X = \frac{1}{\mu} \cdot \lambda = \frac{\lambda}{\mu}$$
2. If the buffer is finite we write $X = \lambda - \lambda_{\text{drop}}$ in which λ_{drop} is the drop rate of jobs because the queue is full. So having an infinite buffer reduce the throughput but also reduce the response time, because the queue can only contain a maximum amount of jobs.

2.3 Slowdown

The **slowdown** of a specific job is the fraction of time the job spends waiting respect to the time it spends being executed:

$$Sl(j) = \frac{R(j)}{S(j)}$$

If we have different type of jobs, we can calculate the expected value of slowdwn:

$$E(Sl) = E\left(\frac{R}{S}\right)$$

Example:
We have an open system with one CPU and $\lambda = \frac{1}{2}$ j/s. We have two types of jobs that require different service times:

$$S = \begin{cases} 1 & p = \frac{3}{4} \\ 2 & p = \frac{1}{4} \end{cases}$$

We know that $E(R) = \frac{29}{12}$.

If we have a FIFO buffer we can calculate the expected value of slowdown:

$$E(Sl) = E\left(\frac{R}{S}\right) = E\left(\frac{S+W}{S}\right) = E\left(1 + \frac{W}{S}\right)$$

The waiting time W is not related to the service time S of the same job, but on the previous jobs, so we can divide the expected values:

$$E(Sl) = 1 + E(W) \cdot E\left(\frac{1}{S}\right) = 1 + E\left(\frac{1}{S}\right)(E(R) - E(S))$$
$$= 1 + \left(\frac{3}{4}\cdot 1 + \frac{1}{4}\cdot\frac{1}{2}\right)\left(\frac{29}{12} - \left(\frac{3}{4}\cdot 1 + \frac{1}{4}\cdot 2\right)\right) = 1 + \frac{7}{8}\left(\frac{29}{12} - \frac{5}{4}\right) = \frac{97}{48}$$

In the case of a different scheduling behaviors like Short Jobs First (SJF) the waiting time would be related to the servise time so the calculation are more complex.

2.4 Little Law

Little Law

The little law says that:

$$R = \frac{\overline{N}}{X}$$

This is valid for every system that is **ergodic**, that respects 3 properties:

1. **Irreducibility**: From every state (defined as the population in the servers) the system can reach every other state and back.
2. **Positive recurrence**: From a state the probability to return in the same state in infinite time is 1.
3. **Aperiodicity**: The system doesn't have a periodic behavior, in which he returns in a specific state after a precise amount of time.

The little low is valid for everything that has an input and output, even a closed system if you consider only the cpu and terminals.

Proof:

To show the little law we can do an example with an open system defining:

- A_R the area with the completed jobs at time t
- A_G the area with the arrived jobs at time t
- A_W the area with all the jobs up to time t

We can define λ and X as:

$$\lambda = \lim_{t\to+\infty} \frac{A(t)}{t} = \lim_{t\to+\infty} \frac{D(t)}{t} = X$$

Also is assured that:

$$A_R \leq A_W \leq A_G$$

If we define T_i as the time spent from job i, we can rewrite the relation as:

$$\sum_{i \in D(t)} T_i \leq \int_0^t N(s)ds \leq \sum_{i \in A(t)} T_i$$

Without modifing the result we can divide by t, putting the limit of t going to inf and multiply by 1:

$$\lim_{t \to +\infty} \frac{\sum_{i \in D(t)} T_i}{t} \cdot \frac{D(t)}{D(t)} \leq \lim_{t \to +\infty} \frac{\int_0^t N(s)ds}{t} \leq \lim_{t \to +\infty} \frac{\sum_{i \in A(t)} T_i}{t} \cdot \frac{A(t)}{A(t)}$$

So we have that the left part is the mean time for a completed job, the right part is the mean time for an arrived job and the middle part is the mean population:

$$XR \leq \overline{N} \leq \lambda R$$

So, because λ and X are equal:

$$\overline{N} = XR \implies R = \frac{\overline{N}}{X}$$

We can do the same process with a closed system, but this time A_W will not be an integral but simply Nt, so with the same procedure we reach that:

$$N = TX \implies T = \frac{N}{X} \implies R = \frac{N}{X} - Z$$

This formula above is called the **Response time of a closed system**.
Example:
We have a closed system with an application layer and a DB layer. The query in the system are $N = 400$ and the average response time of the application layer is $R_A = 150$ ms and $\mu_{DB} = 550$ j/s.
We can calculate R_{DB} using the little law:

$$R_{DB} = \frac{N}{X} - Z = \frac{400}{550} - 0.150 = 0.577 \text{ s}$$

If we double the speed of the application layer, so halving the R_A:

$$R_{DB} = \frac{400}{550} - 0.075 = 0.652 \text{ s}$$

2.5 Forced Flow law

Forced Flow law

For a specific device i in a system we have that:

$$X_i = V_i \cdot X$$

In which V_i is the average number of visits that a job does to device i before exiting the system.

Proof:
We have a device i and we call $D_i(t)$ the number of completed jobs by device i at time t. We also call $V_i(j)$ the number of visits of a specific job j to the device i. we have that:

$$D_i(t) = \sum_{j \in D(t)} V_i(j)$$

We divide by t, adding the limit and multiply by 1:

$$\lim_{t\to+\infty} \frac{D_i(t)}{t} = \lim_{t\to+\infty} \frac{\sum_{j\in D(t)} V_i(J)}{t} \cdot \frac{D(t)}{D(t)}$$

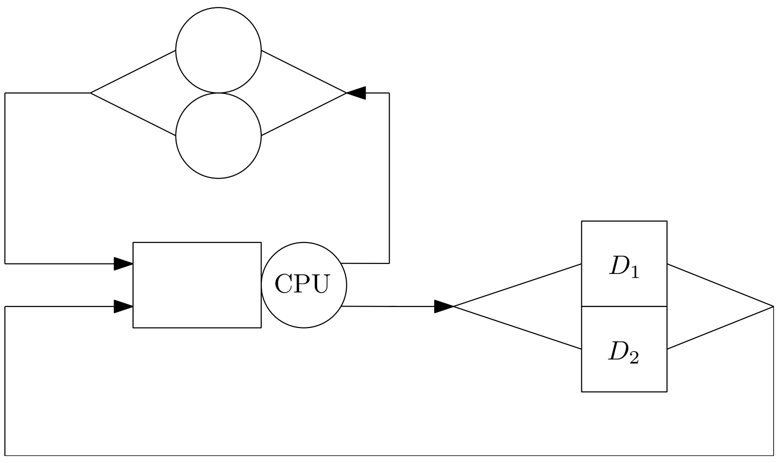
The left part is the throughput X_i and the right part can be rearranged as:

$$\lim_{t\to+\infty} \frac{\sum_{j\in D(t)} V_i(J)}{D(t)} = V_i \qquad \frac{D(t)}{t} = X$$

So we found the formula:

$$X_i = V_i \cdot X$$

Example:



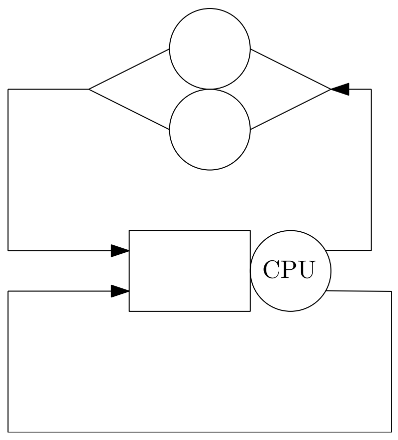
We have that the probablity that a job exiting the CPU goes in each location is:

- Terminals: $p_T = \frac{1}{181}$
- Disk 1: $p_{D_1} = \frac{80}{181}$
- Disk 2: $p_{D_2} = \frac{100}{181}$

The expected number of visit in each components can be calculated:

$$E(V_{CPU}) = \frac{1}{p_T} = \frac{1}{\frac{1}{181}} = 181$$
$$E(V_{D_1}) = E(V_{CPU}) \cdot p_{D_1} = 181 \cdot \frac{80}{181} = 80$$
$$E(V_{D_1}) = E(V_{CPU}) \cdot p_{D_2} = 181 \cdot \frac{100}{181} = 100$$

Example 2:



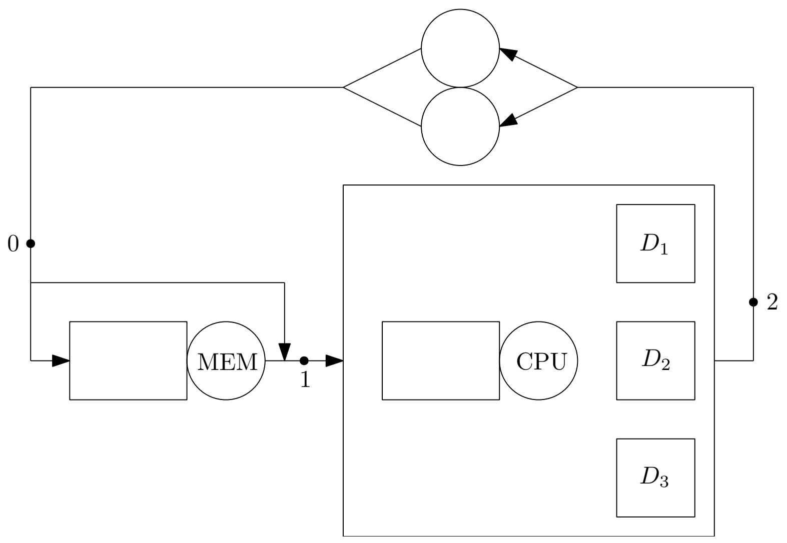
The system has this parameters:

- $N = 25$
- $Z = 18$ s
- $\rho = 30\%$
- $S = 0.025$ s
- $p_T = \frac{1}{20}$

We can calculate:

$$
\begin{aligned}
X_{CPU} &= \frac{\rho}{S} = \frac{0.30}{0.025} = 12 \text{ j/s} \\
X &= \frac{X_{CPU}}{E(V_{CPU})} = X_{CPU} \cdot \frac{1}{p_T} = 12 \cdot \frac{1}{20} = 0.6 \text{ j/s} \\
R &= \frac{N}{X} - Z = \frac{25}{0.6} - 18 = 23.67 \text{ s}
\end{aligned}
$$

Example 3:



We have this system where we don't know how the components are connected inside the box. We have this data:

- $N = 23$
- $Z = 21$ s
- $X = 0.45$ j/s
- $N_{MEM} = 11.65$
- $V_{CPU} = 3$
- $S_{CPU} = 0.21$ s

We need to calculate the average time from point 1 to point 2 (R_{1-2}), and we can do that by doing:

$$R_{0-2} = \frac{N}{Z} - Z = \frac{23}{0.45} - 21 = 30.11 \text{ s}$$
$$R_{0-1} = \frac{N_{MEM}}{X} = \frac{11.65}{0.45} = 25.89 \text{ s}$$
$$R_{1-2} = R_{0-2} - R_{0-1} = 30.11 - 25.89 = 4.22 \text{ s}$$

We also want to calculate the utilization of the CPU ρ_{CPU}:

$$X_{CPU} = V_{CPU} \cdot X = 3 \cdot 0.45 = 1.35 \text{ j/s}$$
$$\rho_{CPU} = S_{CPU} \cdot X_{CPU} = 0.21 \cdot 1.35 = 0.28 = 28\%$$

2.6 Demand time

We define the **demand time** of a job j to a device i as the total time spent from job j in the device i. Can be defined as the sum of all the occurance in which j goes in i before exiting the system:

$$D_i(j) = \sum_{j=1}^{V_i(j)} S_i(j)$$

So, the average demand of a device i will be:

$$D_i = E\left(\sum_{j=1}^{V_i(j)} S_i(j)\right)$$

If we rewrite the formula using the total partition law:

$$D_i = \sum_{n=0}^{\infty} E\left(\sum_{j=1}^{n} S_i \middle| V_i = n\right) \cdot P(V_i = n)$$

If the assume the service time of a job is indipendent from the number of time it goes in device i:

$$D_i = \sum_{n=0}^{\infty} E\left(\sum_{j=1}^{n} S_i\right) \cdot P(V_i = n) = \sum_{n=0}^{\infty} n \cdot S_i \cdot P(V_i = n) = S_i \cdot \sum_{n=0}^{\infty} n \cdot P(V_i = n) = S_i \cdot V_i$$

So, in the end the demand time of a device i can be calculated by:

$$D_i = S_i \cdot V_i$$

2.6.1 Bottleneck law

If we combine the demand time formula with the Forced Flow Law we have the **Bottleneck Law**:

$$D_i = S_i \cdot V_i = S_i \frac{X_i}{X} = \frac{\rho_i}{X} \implies \rho_i = D_i \cdot X$$

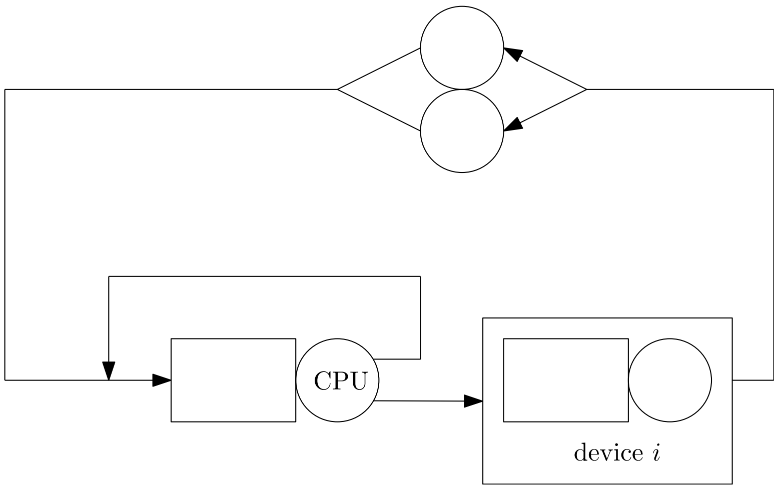
We also have that:

$$X = \frac{\rho_i}{D_i} \quad \forall i$$

So, we can relate the global throughput to the throughput and utilization of every device in the system and because $0 \leq \rho \leq 1$ we are sure that:

$$X \leq \frac{1}{D_i} \quad \forall i$$

The device that does most bottleneck in the system is the one with the higher demand time.
Example:



We have this system where we don't know the device inside the box. We have this data:

- $p_{CPU-CPU} = 0.5$
- $p_{CPU-i} = 0.5$
- $Z = 5$ s
- $S_i = 0.01$ s
- $\rho_i = 0.3$
- $\rho_{CPU} = 0.5$
- $T = 50$ s
- $\frac{V_i}{V_{CPU}} = 10$
- $N_i = 20$

We can calculate the average population in the queue of the CPU N_{CPU-Q}:

$$
\begin{aligned}
&V_{CPU} = \frac{1}{p_{CPU-i}} = 2 \\
&V_i = \frac{V_i}{V_{CPU}} \cdot V_{CPU} = 10 \cdot 2 = 20 \\
&D_i = V_i \cdot S_i = 20 \cdot 0.01 = 0.2 \text{ s} \\
&X = \frac{\rho_i}{D_i} = \frac{0.3}{0.2} = 1.5 \text{ j/s} \\
&X_i = V_i \cdot X = 20 \cdot 1.5 = 30 \text{ j/s} \\
&X_{CPU} = V_{CPU} \cdot X = 2 \cdot 1.5 = 3 \text{ j/s} \\
&R_{CPU} = T - Z - R_i = T - Z - \frac{N_i}{X_i} = 50 - 5 - \frac{20}{30} = 44.33 \text{ s} \\
&N_{CPU} = R_{CPU} \cdot X_{CPU} = 44.33 \cdot 3 = 133 \\
&N_{CPU-Q} = N_{CPU} - \rho_{CPU} = 133 - 0.5 = 132.5
\end{aligned}
$$

2.6.2 Bound on throughput and response time

Through the bottleneck law we know that:

$$X \leq \frac{1}{D_i} \; \forall i$$

So we can obtain an upper bound for the throughput:

$$X \leq \frac{1}{D_{max}}$$

In which D_{max} is the maximum demand time for a device. From this, a lower bound for the response time can be obtained:

$$R = \frac{N}{X} - Z \implies R \geq N \cdot D_{max} - Z$$

Knowing the total demand time of a system D, we are sure that:

$$R \geq D$$

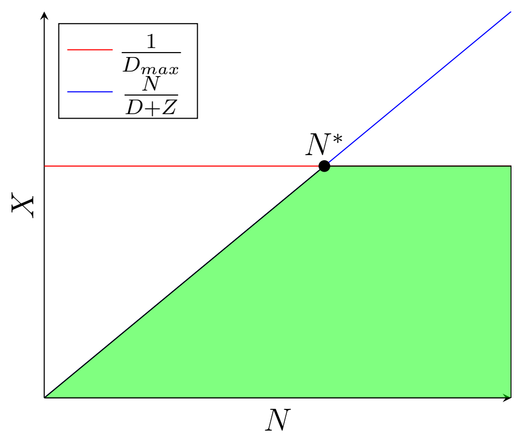
With the same formula as before we can find another bound for the throughput:

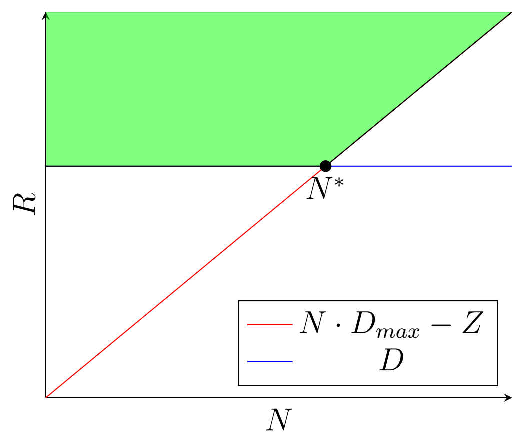
$$R = \frac{N}{X} - Z \geq D \implies X \leq \frac{N}{D+Z}$$

The bound found are:

$$
\begin{gathered}
X \leq \min\left(\frac{N}{D+Z}, \frac{1}{D_{max}}\right) \\
R \geq \max\left(D, N \cdot D_{max} - Z\right)
\end{gathered}
$$

If we plot this bounds:





We can identify the point N^* where the two functions cross, that is the point:

$$N^* = \frac{D+Z}{D_{max}}$$

Based on this number the system can be in two states:

- If $N < N^*$ the system have **low population** and the buond is related to D, so every upgrade to the system will increase the throughput and decrease the response time.
- If $N > N^*$ the system have **high population** and the buond is related to D_{max}, so only an upgrade to the device with higher demand time will increase the throughput and decrease the response time.

2.7 Formulas

- Stability condition of a system:
$$\lambda_i < \mu_i \quad \forall i$$
- Utilization law:
  - Single device:
$$\rho_i = S_i \cdot X_i$$
  - All system:
$$U = \frac{\sum \rho_i}{m}$$
- Slowdown:
$$Sl = E\left(\frac{R}{S}\right)$$
- Little law:
  - Open systems:
$$R = \frac{\overline{N}}{X}$$
  - Closed systems:
$$R = \frac{N}{X} - Z \implies T = \frac{N}{X}$$

- Forced Flow law:
$$X_i = V_i \cdot X$$
- Demand time:
$$D_i = S_i \cdot V_i$$
- Busy time and completions:
$$D_i = \frac{B_i}{C}$$
$$V_i = \frac{C_i}{C}$$
$$S_i = \frac{B_i}{C_i}$$
- Bottleneck law:
$$X = \frac{\rho_i}{D_i} \quad \forall i$$
- Bounds:
$$X \leq \min\left(\frac{N}{D+Z}, \frac{1}{D_{max}}\right)$$
$$R \geq \max(D, N \cdot D_{max} - Z)$$
$$N^* = \frac{D+Z}{D_{max}}$$

3 Markov Chains

Stochastic process

A **stochastic process** is a sequence of states:

$$X_0, X_1, \ldots, X_n$$

in which X_i is the state at time i.

We also define the state space I:

$$\forall i \; X_i \in I$$

3.1 Discrete Time Markov Chains

Discrete Time Markov Chain

A **Discrete Time Markov Chain (DTMC)** is a stochastic process with the following property:

$$P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = j | X_n = i) = P_{ij}$$

So, the probability of next step depends only on the current one, and not on the history (**memoryless**). Also it not depends on the time indexes (**time homogeneity**).
If the state space is finite ($|I| = M$), a **transition probability matrix** P can be defined. The matrix have size $M \times M$. In the matrix we have that:

- $P_{ij} \in [0, 1] \; \forall i, j$
- $\sum_{j \in I} P_{ij} = 1 \; \forall i$

Example:
We have a machine, if the machine is working there is 95% probability that it will work tomorrow, if it's broken there is 40% that it will be repaired tomorrow.
This problem can be expressed as a DTMC. The probability matrix can visualized as a table or an automata:

$$P = \begin{bmatrix} 0.95 & 0.05 \\ 0.4 & 0.6 \end{bmatrix}$$

$$\begin{tikzcd}
W \arrow[loop left, "0.95"] \arrow[r, bend left, "0.05"] & B \arrow[loop right, "0.6"] \arrow[l, bend left, "0.4"]
\end{tikzcd}$$

3.1.1 Probability of multiple steps

Given a DTMC we can define the probability of going from state i to state j in exatly two steps as:

$$P_{ij}^2 = \sum_{k \in I} P_{ik} \cdot P_{ki}$$

Also this formula can be generalized for every number of steps by a formula called **Chapman Kolmogoroff equation**:

$$P_{ij}^{n+m} = \sum_{k \in I} P_{ik}^n \cdot P_{kj}^m$$

For every number of steps n we can calculate the probability matrix for n steps P^n multiplying the original matrix P by itself n times.

3.1.2 Limiting and stationary probabilities

Limiting probability

The **limiting probability** of a state i is the probability that the chain is in state i indipendently to the starting state.

$$\pi_j = \lim_{n \to +\infty} P_{ij}^n \; \forall i$$

Also, we define the **limiting distribution** of being in each state:

$$\pi = (\pi_0, \dots, \pi_M) \text{ where } \sum_{i \in I} \pi_i = 1$$

So, for an arbitrary large n the matrix P^n will have every row equal (the limit could not exists).
Example:
We have the same machine as before but with different probabilities:

$$P = \begin{bmatrix} 1-a & a \\ b & 1-b \end{bmatrix}$$

The probability matrix for n steps P^n is:

$$P = \begin{bmatrix} \frac{b + a(1-a-b)^n}{a+b} & \frac{a - a(1-a-b)^n}{a+b} \\ \frac{b - b(1-a-b)^n}{a+b} & \frac{a + b(1-a-b)^n}{a+b} \end{bmatrix}$$

For $n \to \infty$ we know that $(1-a-b)^n = 0$, so the probability matrix looks like:

$$P = \begin{bmatrix} \frac{b}{a+b} & \frac{a}{a+b} \\ \frac{b}{a+b} & \frac{a}{a+b} \end{bmatrix}$$

So, the probability of being in each state is:

$$\pi_W = \frac{b}{a+b}, \pi_B = \frac{a}{a+b}$$

Stationary distribution

A distribution $\pi = (\pi_0, \dots, \pi_M)$ is **stationary** if:

1. $\pi = \pi \cdot P$
2. $\pi_j = \sum_{i \in I} \pi_i \cdot P_{ij}$

Equalities of limiting and stationary distribution

If a limiting distribution exists then:

1. Is equal to the stationary distribution
2. Is the only stationary distribution

Proof 1:
π is a limiting distribution:

$$\pi_j = \lim_{n \to +\infty} P_{ij}^n$$

Is a limit for n so doesn't change for $n+1$:

$$P_{ij}^{n+1} = \sum_{k \in I} \lim_{n \to +\infty} P_{ik}^n \cdot P_{kj} = \sum_{k \in I} \pi_k \cdot P_{kj} = \pi_j \text{ stationary}$$

Proof 2:
Suppose exists a second stationary probability π' where $\pi' = \pi' \cdot P = \pi' \cdot P^n$. A specific π'_j is calculated as a multiplication between π' and the j-th column of the matrix P:

$$\pi'_j = \pi' \cdot P_j^n = \lim_{n \to +\infty} \pi' P_j^n = \sum_{k \in I} \lim_{n \to +\infty} \pi'_k \cdot P_{kj}^n = \sum_{k \in I} \pi'_k \cdot \left(\lim_{n \to +\infty} P_{kj}^n \right) = \underbrace{\sum_{k \in I} \pi'_k}_{1} \cdot \pi_j = \pi_j$$

Example:
We have the machine again with the probability matrix P:

$$P = \begin{bmatrix} 0.95 & 0.05 \\ 0.4 & 0.6 \end{bmatrix}$$

To find the probability distribution we need to solve a system of equations:

$$\begin{cases} \pi_W = 0.95\pi_W + 0.4\pi_B \\ \pi_B = 0.05\pi_W + 0.6\pi_B \\ \pi_W + \pi_B = 1 \end{cases} \implies \begin{cases} 0.05\pi_W = 0.4\pi_B \\ 0.4\pi_B = 0.05\pi_W \\ \pi_W + \pi_B = 1 \end{cases} \implies \begin{cases} \pi_W = 8\pi_B \\ \pi_B = \frac{1}{8}\pi_W \\ 9\pi_B = 1 \end{cases} \implies \begin{cases} \pi_W = \frac{8}{9} \\ \pi_B = \frac{1}{9} \end{cases}$$

With complex or even infinite state matrix this system of equation can be difficult to define. To help us we can use a flow rule, in which if we draw a box on the automata the probability to enter the box must be equal to the probability to exit the box.

Example:
We have a system with a CPU and a finite queue with 3 spaces, at every time step with probability r a job arrives and with probability q a job finish. The automata of the Markov chain based on the number of jobs in the system is:

$$\begin{tikzcd}
0 \arrow[loop left, "1-r"] \arrow[r, bend left, "r"] & 1 \arrow[loop above, "1-r-q"] \arrow[l, bend left, "q"] \arrow[r, bend left, "r"] & 2 \arrow[loop above, "1-r-q"] \arrow[l, bend left, "q"] \arrow[r, bend left, "r"] & 3 \arrow[loop above, "1-r-q"] \arrow[l, bend left, "q"] \arrow[r, bend left, "r"] & 4 \arrow[loop right, "1-q"] \arrow[l, bend left, "q"]
\end{tikzcd}$$

To calculate π_1 with the formula, we will have that:

$$\pi_1 = r\pi_0 + (1-r-q)\pi_1 + q\pi_2$$

If we draw a box containing the state 0 and its loop, we can say that:

$$q\pi_1 = r\pi_0$$

We can do the same drawing a box containing both state 0 and 1 to write that:

$$q\pi_2 = r\pi_1$$

And so on. So, the system of equations will be:

$$\begin{cases} q\pi_1 = r\pi_0 \\ q\pi_2 = r\pi_1 \\ q\pi_3 = r\pi_2 \\ q\pi_4 = r\pi_3 \\ \pi_0 + \pi_1 + \pi_2 + \pi_3 + \pi_4 = 1 \end{cases} \implies \begin{cases} \pi_1 = \frac{r}{q}\pi_0 \\ \pi_2 = \left(\frac{r}{q}\right)^2 \pi_0 \\ \pi_3 = \left(\frac{r}{q}\right)^3 \pi_0 \\ \pi_4 = \left(\frac{r}{q}\right)^4 \pi_0 \\ \pi_0 \cdot \left(1 + \frac{r}{q} + \left(\frac{r}{q}\right)^2 + \left(\frac{r}{q}\right)^3 + \left(\frac{r}{q}\right)^4\right) = 1 \end{cases}$$

Using this we can know different parameters of the system like the utilization and the average population:

$$U = 1 - \pi_0 \qquad \overline{N} = \sum_{i=0}^{4} i \cdot \pi_i$$

3.1.3 Infinite state Markov Chains

In the case of infinite state Markov Chain the system will contain infine equations, so we have to extract a general case to calculate π_i in respect to π_0.
Example:
We have the same system as before but with an infinite queue, so the probability matrix will be:

$$P = \begin{bmatrix} 1-r & r & 0 & 0 & \dots & 0 \\ q & 1-r-q & r & 0 & \dots & 0 \\ 0 & q & 1-r-q & r & \dots & 0 \\ & & \vdots & & & \end{bmatrix}$$

We can write π_0 as:

$$\pi_0 = (1-r)\pi_0 + q\pi_1 \implies r\pi_0 = q\pi_1 \implies \pi_1 = \frac{r}{q}\pi_0$$

And π_1 as:

$$\pi_1 = r\pi_0 + (1-r-q)\pi_1 + q\pi_2 \implies \pi_2 = \frac{r}{q}\pi_1 = \left(\frac{r}{q}\right)^2 \pi_0$$

For every other π_i:

$$\pi_i = q\pi_{i-1} + (1-r-q)\pi_i + r\pi_{i+1} \implies \pi_i = \frac{r}{q}\pi_{i-1} = \left(\frac{r}{q}\right)^i \pi_0$$

We also know that the sum of π_i must be 1, so:

$$\sum \pi_i = 1 \implies \sum_{i=0}^{\infty}\left(\frac{r}{q}\right)^i \pi_0 = \pi_0 \sum_{i=0}^{\infty}\left(\frac{r}{q}\right)^i = \pi_0\left(\frac{1}{1-\frac{r}{q}}\right) = 1 \implies \pi_0 = 1 - \frac{r}{q}$$

We can calculate the average number of jobs in the system and the utilization (from now $\frac{r}{q}$ will be written as ρ):

$$\overline{N} = \sum_{i=0}^{\infty} i\pi_i = \sum_{i=0}^{\infty} i\rho^i(1-\rho) = (1-\rho)\sum_{i=0}^{\infty} i\rho^i = (1-\rho)\underbrace{\frac{1}{(\rho-1)^2}}_{(1-\rho)^2} = \frac{\rho}{1-\rho}$$

$$U = 1 - \pi_0 = \rho$$

We also can calculate the number of jobs in the queue:

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

3.1.4 Existence of the limiting distribution

Periodicity

The **period** of a state j is the greatest common divisor (GCD) of the set of n such that $P^n_{ij} > 0$.
A state is **aperiodic** iwhen it has period 1. A Markov Chain is aperiodic if all its states are aperiodic.

Example:

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The n such that $P^n_{ij} > 0$ are:

$$S_0 = S_1 = \{2, 4, 6, \dots\} \implies \text{GCD}_0 = \text{GCD}_1 = 2$$

If we change P:

$$P = \begin{bmatrix} 0.01 & 0.99 \\ 1 & 0 \end{bmatrix}$$

For every n we have that $P^n_{00} > 0$, so the state is aperiodic.

Existence of limiting distribution

Given an aperiodic, irreducible and finite state DTMC then the limiting probability exists and the rows define π. The system is positive recurrent.

Proof:
With a probability matrix P we define e as a column with only 0 and a 1 on the j-th row:

$$e = \begin{bmatrix} 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{bmatrix}$$

We have that $P^n \cdot e$ is the j-th column of P^n:

$$P^n \cdot e = \begin{bmatrix} 0 & \dots & P^n_{1j} & \dots & 0 \\ 0 & \dots & P^n_{2j} & \dots & 0 \\ & & \vdots & & \\ 0 & \dots & P^n_{Mj} & \dots & 0 \end{bmatrix}$$

Let M_n be the maximum component of $P^n \cdot e$, m_n the minimum component and s the smaller value in P. We can say that:

$$M_n - m_n \leq (1-2s)(M_{n-1} - m_{n-1})$$

Also:

$$\left.\begin{aligned} M_n &\leq (1-s)M_{n-1} + sm_{n-1} \\ m_n &\geq sM_{n-1} + (1-s)m_{n-1} \end{aligned}\right\} \implies M_n - m_n \geq (1-2s)(M_{n-1} - m_{n-1})$$

We know that $s \leq \frac{1}{2}$, so $(1-2s) \in [0,1]$ and $M_n - m_n$ will converge to 0, making all rows equal. The only exception is when $s = 0$, but because of aperiodicity and irreducibility we know that exists a n_0 for which P^{n_0} as no 0. So, we can define P^n as a moltiplication of P^{n_0} and it will converge. The difference is that the decrese will occur only every n_0 moltiplication of P.

Existence of limiting distribution with infinite state

Given an aperiodic, irreducible and infinite state DTMC then the limiting probability exists if the system is positive recurrent (is not sure like with finite states) and the rows define π.

To check if a system with infinite states is positive recurrent, we can calculate the stationary distribution, if it exists then the system is positive recurrent.

3.2 Continuous Time Markov Chains

A **Continuous Time Marckov Chain (CTMC)** is similar to a DTMC but instead of having probabilities to transition between states, it has rates.

To calculate the probability distribution we can't use the system of equation created from $\pi = \pi P$ because P doesn't have probabilities but rates.

3.2.1 CTMC to DTMC

From a CTMC we can construct an embedded DTMC using a very fast clock that represent an observation of the system. The inter-observation time δ has to be low so that the observation rate $\frac{1}{\delta} = \Delta$ is much larger than any other rate in the system. We want that in every observation we see 1 or 0 transition.
Self loops need to be added in the DTMC because Δ is much larger than the other rates, so they will not sum up to 1.
Example:
We have a CTMC like this:

$$\begin{tikzcd}
0 \arrow[r, bend left, "\lambda"] & 1 \arrow[l, bend left, "\mu"]
\end{tikzcd}$$

The corresponding DTMC is:

$$\begin{tikzcd}
0 \arrow[loop left, "1-\frac{\lambda}{\Delta}"] \arrow[r, bend left, "\frac{\lambda}{\Delta}"] & 1 \arrow[l, bend left, "\frac{\mu}{\Delta}"] \arrow[loop right, "1-\frac{\mu}{\Delta}"]
\end{tikzcd}$$

From CTMC to DTMC

Given a CTMC, we can construct a DTMC in which:

$$I_{DTMC} = I_{CTMC}$$

Let λ_{ij} be the rate from state i to state j in the CTMC and Δ the observation rate, the transition probabilities in DTMC are:

$$P_{ij} = \frac{\lambda_{ij}}{\Delta}$$

$$P_{ii} = 1 - \frac{\sum_{j \in I} \lambda_{ij}}{\Delta}$$

This system has self loops, so is aperiodic by definition.

Example:
We have a single server with a queue that can constain M_Q jobs, an arrival rate of λ and a processing rate of μ.

$$\begin{tikzcd}
0 \arrow[r, bend left, "\lambda"] & 1 \arrow[l, bend left, "\mu"] \arrow[r, bend left, "\lambda"] & \cdots \arrow[l, bend left, "\mu"] \arrow[r, bend left, "\lambda"] & M_Q+1 \arrow[l, bend left, "\mu"]
\end{tikzcd}$$

We can apply the barrier technique, so:

$$\pi_i = \left(\frac{\lambda}{\mu}\right)^i \pi_0$$

And π_0 can be calculated:

$$\pi_0 \sum_{i=0}^{M_Q+1} = 1 \xRightarrow{q=\frac{\lambda}{\mu}} \pi_0 \left(\frac{1+q^{M_q+2}}{1-q}\right) = 1 \implies \pi_0 = \frac{1-q}{1+q^{M_q+2}}$$

If the queue is infinite, the situation depends on $q = \frac{\lambda}{\mu}$:

- If $q \geq 1$: we have that $\pi_0 \to 0$ because the system is not stable and the number of jobs in the queue increase infinitely. The stability distribution doesn't exists.
- If $q < 1$: we have that $\pi_0 = 1 - q \implies q = \rho$ (utilization). In this case we can calculate the average number of jobs in the system, the throughput and other metrics:

$$\overline{N} = \sum_{i=0}^{\infty} i\pi_i = (1-\rho)\sum_{i=0}^{\infty} i\rho^i = \frac{\rho}{1-\rho}$$

$$X = \lambda \implies R = \frac{\overline{N}}{X} = \frac{\rho}{1-\rho}\cdot\frac{1}{\lambda} = \frac{\frac{\lambda}{\mu}}{1-\frac{\lambda}{\mu}}\cdot\frac{1}{\lambda} = \frac{1}{\mu-\lambda}$$

3.3 Google PageRank

To rank the pages Google uses the number of citations, so the number of pages that point to that page with a link.
A simple algorithm that does uses this way of ranking is cheatable creating a lot of fake pages pointing to the page that i want high in ranking.
Another method could be giving a weight to the links corresponding to the number of link pointing to it in a recursive way. This is also cheatable with a clique of pages pointing each other.

3.3.1 Good Hypothesis

A good method (not the one used by Google) is to rank a page high if the sum of its backlinks is high. The rank of a page j is:

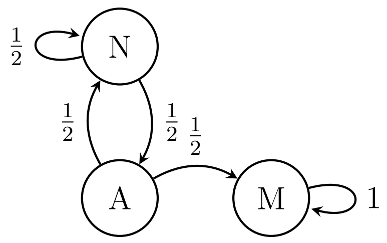
$$\pi_j = \sum_i \pi_i \cdot P_{ij}$$

in which π_i is the rank of page i and P_{ij} is $\frac{1}{k}$ if page i links page j and page i has k links. The algorithm works as follows:

1. Create a DTMC where each page is a state and arrows are links
2. If a page has k links, each has probability $\frac{1}{k}$
3. Solve the DTMC and find the limiting distribution π

4. Rank the pages in descreasing order of π

Example:
We have three pages that create this DTMC:



Solving the DTMC, the limiting distribution is:

$$\begin{cases} \pi_M = 1 \\ \pi_N = 0 \\ \pi_A = 0 \end{cases}$$

This happen because the system is not irreducible, so M is called an **absorbing state** because if we enter in it we cannot exit.

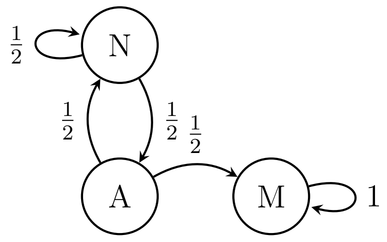
This case cheats the algorithm giving M the max rank. The page M is also called a **spider trap**.

3.3.2 Google algorithm

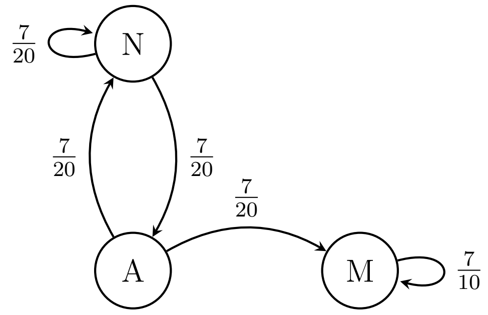
To solve the problem with spider trap we add transition between every pair of states. We decrease every link by a percentage tax t and increase every link equally distrbuting t.

The problem is how to choose the correct tax t, a low value will make slightly changes, leaving the spider trap by far the higher rank and a high value will make every link almost equal, nullifying the algorithm.

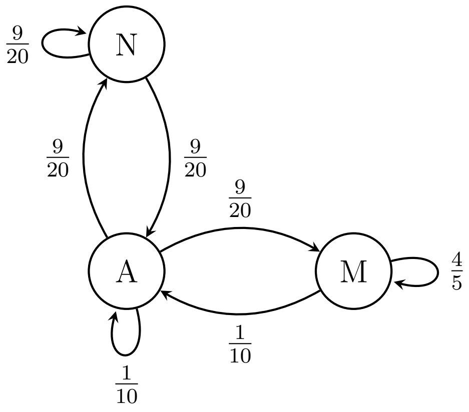
Example:
We have the same DTMC as before:



We choose the tax $t = 0.3$ so we multiply every probability in the existing links by 0.7:



There are three pages, so we will add to every link 0.1:



Calculating now the limiting distribution we have:

$$\begin{cases} \pi_A = 0.19 \\ \pi_M = 0.55 \\ \pi_N = 0.26 \end{cases}$$

4
Da capire

4.1 Poisson distribution

A Poisson distribution describes a rate λ (arrival rate). Knowing the rate, we can calculate the probability of k arrivals before a time t:

$$P(k \text{ arrival before } t) = P(k,t) = \frac{(\lambda t)^k \cdot e^{-\lambda t}}{k!}$$

Let x be the inter-arrival time, we define $F_x(t)$ the distribution of the inter-arrival time. The distribution can be written as:

$$F_x(t) = P(x \leq t) = 1 - P(x > t) = 1 - P(0,t) = 1 - \frac{(\lambda t)^0 \cdot e^{-\lambda t}}{0!} = 1 - e^{-\lambda t}$$

So, the inter-interval time has a **negative exponential distribution** and its expected value is:

$$E(x) = \frac{1}{\lambda}$$

The Poisson distribution has the **memoryless property**, which means:

$$P(x > t + t' | x > t') = P(x > t)$$

The simple proof is:

$$P(x > t + t' | x > t') = \frac{P(x > t + t')}{P(x > t')} = \frac{e^{-\lambda(t+t')}}{e^{-\lambda t'}} = e^{-\lambda t} = P(x > t)$$

4.2 Kendall notation

A system with arrives and departures distributed using a Poisson distribution can be referred as:

$$\text{M/M/}C/M_Q$$

in which:

- The first M stands for memoryless arrives
- The second M stands for memoryless departures
- The C stands for the number of servers
- The M_Q stands for the capacity of the system