Sapienza
Università di Roma

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

Computer Network Performance

Author:
Simone Lidonnici

17 october 2025

Contents

1

Introduction

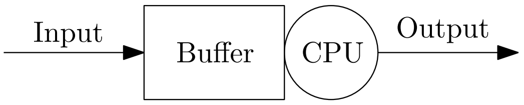
1.1 Notation

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:

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

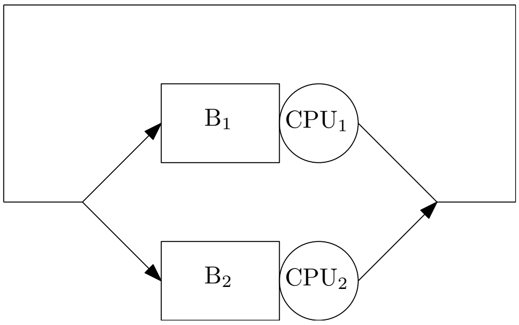
In which m is the number of CPU.

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:

$$R = p_1R_1 + p_2R_2 = \frac{1}{2}(3 \cdot 3) + \frac{1}{2}(3 \cdot 3) = 9 \text{ s}$$

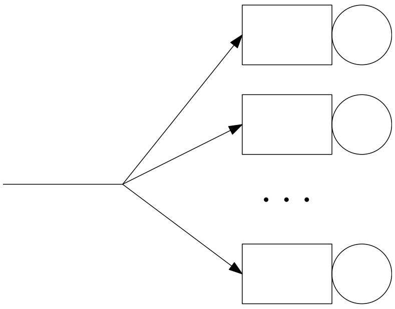
If we change $S_1 = 1.5$ s the average number of jobs in buffer 1 is 0 (1 in the CPU) and in buffer 2 is 4.5 (plus 1 in the CPU). This is caused by the fact that every job going to CPU 1 is completed faster than CPU 2 and the other jobs are stuck in buffer 2. So the average response time in this case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

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

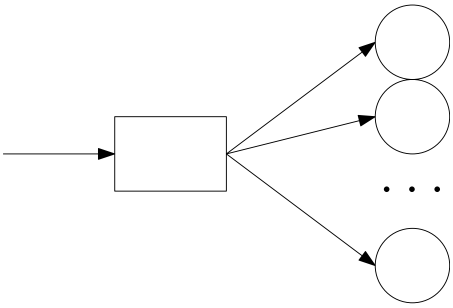
So we can see that upgrading only one CPU on a multi-CPU closed system doesn't change the response time. To lower the response time the routing probabilities must be changed as well.

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:
$$D(t) \leq \mu t$$

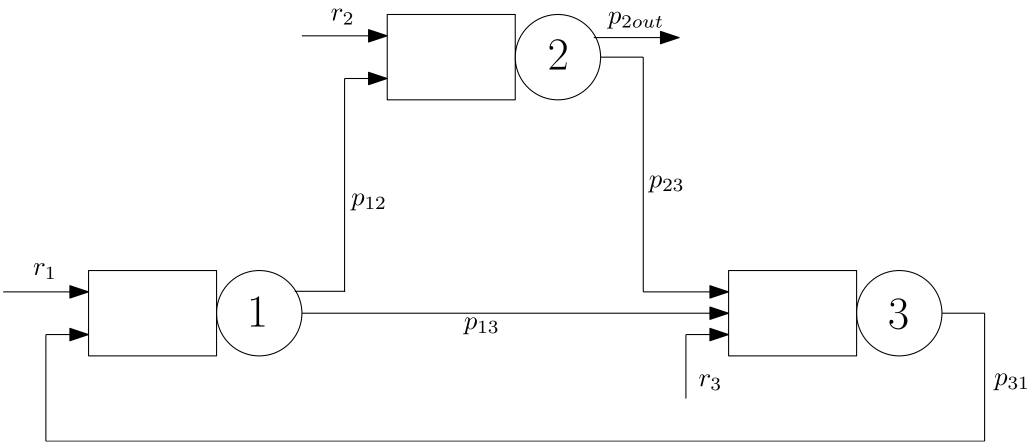
So the population at the time t, denoted as $N(t)$ can be found using the formula:

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

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.

Demonstration:

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:

$$A_R \le A_W \le A_G$$

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 moltiply by 1:

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

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:

$$X_i = E(V_i) \cdot X$$

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

Demonstration:

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 moltiply by 1:

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

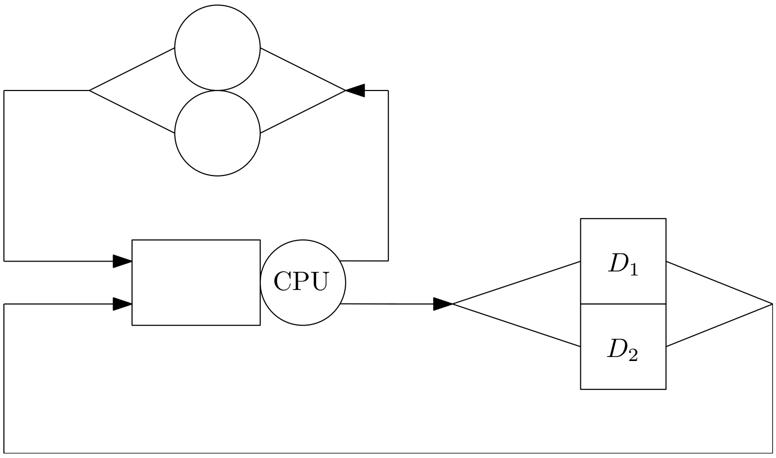
The left part is the throughput X_i and the right part can be rearranged as:

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

So we found the formula:

$$X_i = E(V_i)X$$

Example:



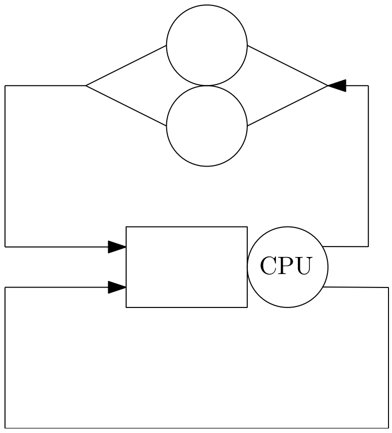
We have that the probablity that a job exiting the CPU goes in each location is:

- 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:



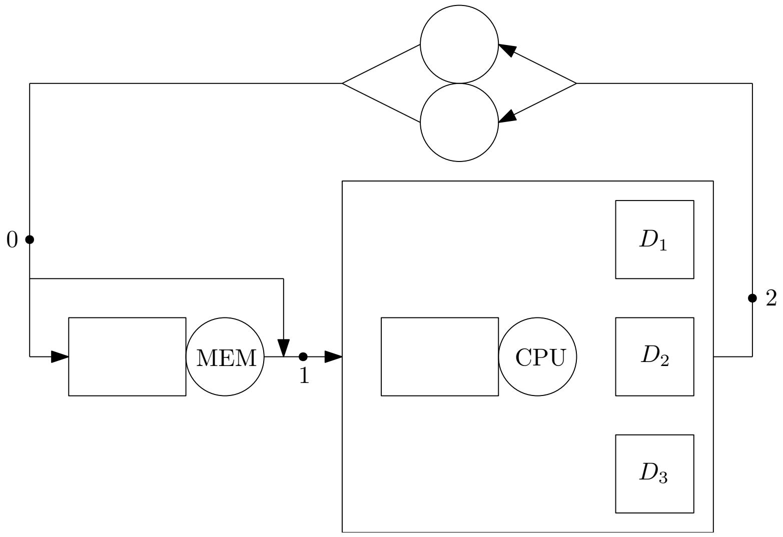
The system has this parameters:

- $N = 25$
- $Z = 18$ s
- $\rho = 30\%$
- $S = 0.025$ s
- $p_T = 0\frac{1}{20}$

We can calculate:

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

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:

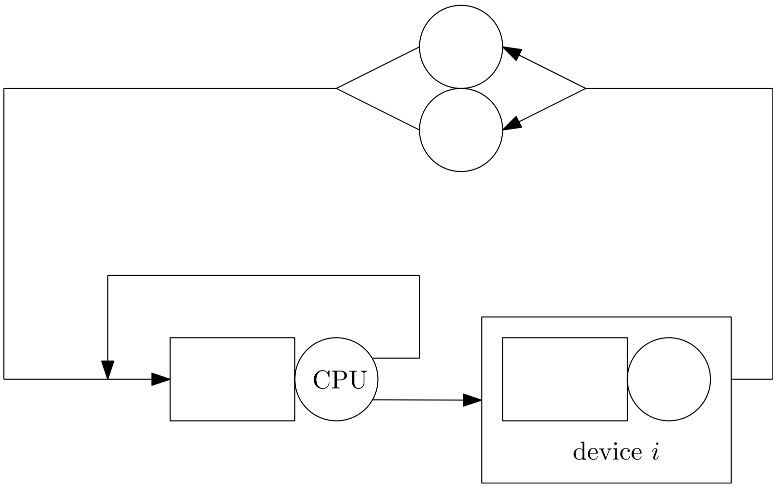
$$X = \frac{\rho_i}{D_i} \forall i$$

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:

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

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}:

$$\begin{aligned}
&V_{CPU} = \frac{1}{p_{CPU-i}} = 2 \\
&V_i = \frac{V_i}{V_{CPU}} \cdot V_{CPU} = 10 \cdot 2 = 20 \\
&D_i = V_i \cdot S_i = 10 \cdot 0.01 = 0.2 \text{ s} \\
&X = \frac{\rho_i}{D_i} = \frac{0.3}{0.2} = 1.5 \text{ j/s} \\
&R_{CPU} = T - Z - R_i = T - Z - \frac{N_i}{X_i} = 50 - 5 - \frac{20}{15} = 31.67 \text{ s} \\
&N_{CPU} = R_{CPU} \cdot X_{CPU} = 31.67 \cdot 1.5 = 47.5 \\
&N_{CPU-Q} = N_{CPU} - \rho_{CPU} = 47.5 - 0.5 = 47
\end{aligned}$$

2.6.2 Buond 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:

$$R \geq D$$

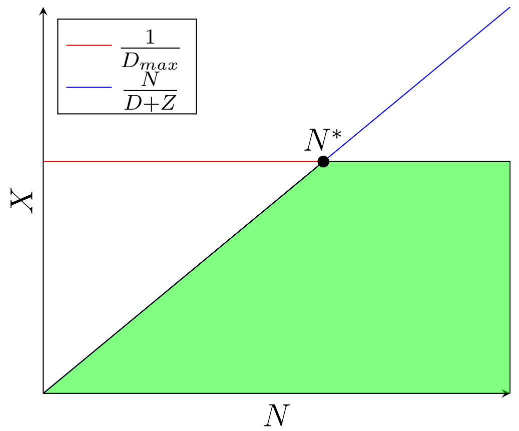
With the same formula as before we can find another bound for the throughput:

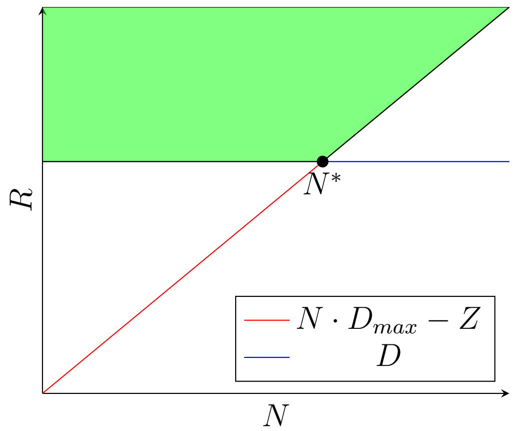
$$R = \frac{N}{X} - Z \geq D \implies X \leq \frac{N}{D+Z}$$

The bound found are:

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

If we plot this bounds:





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

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

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.