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

Computer Network Performance

Author:
Simone Lidonnici

30 september 2025

Contents

1 Introduction

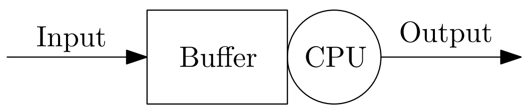
1.1 Notation

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

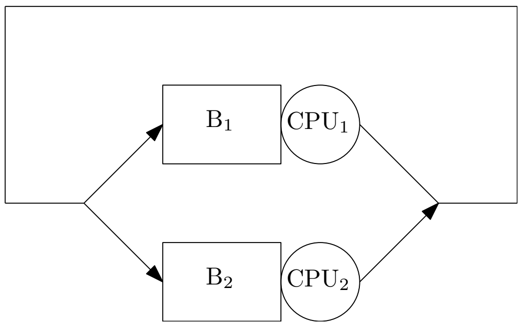
In the case of a system with multiple CPU (each with a personal buffer) we define the probability of a job going in a specific CPU$_i$ with p_i. The total response time of the system is calculated by the formula:

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

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).
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:

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

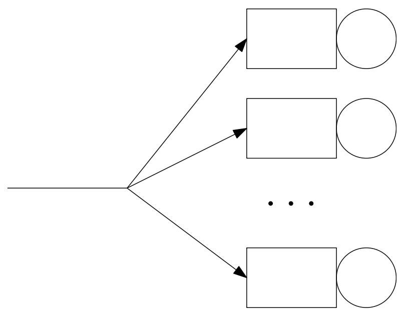
If we change $S_1 = 1.5$ s the average number of jobs in buffer 1 is 0 (1 in the CPU) and in buffer 2 is 4.5 (plus 1 in the CPU). This is caused by the fact that every job going to CPU 1 is completed faster than CPU 2 and the other jobs are stuck in buffer 2. So the average response time in this case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

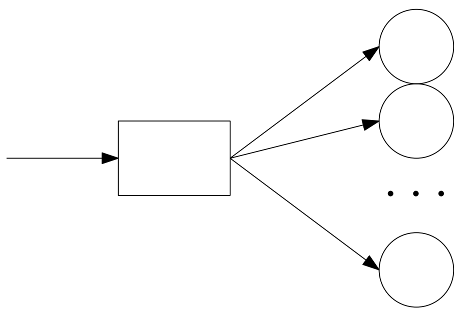
$$R = \frac{1}{2}(1 \cdot 1.5) + \frac{1}{2}(5.5 \cdot 3) = \frac{3}{4} + \frac{33}{4} = 9\text{ s}$$

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.

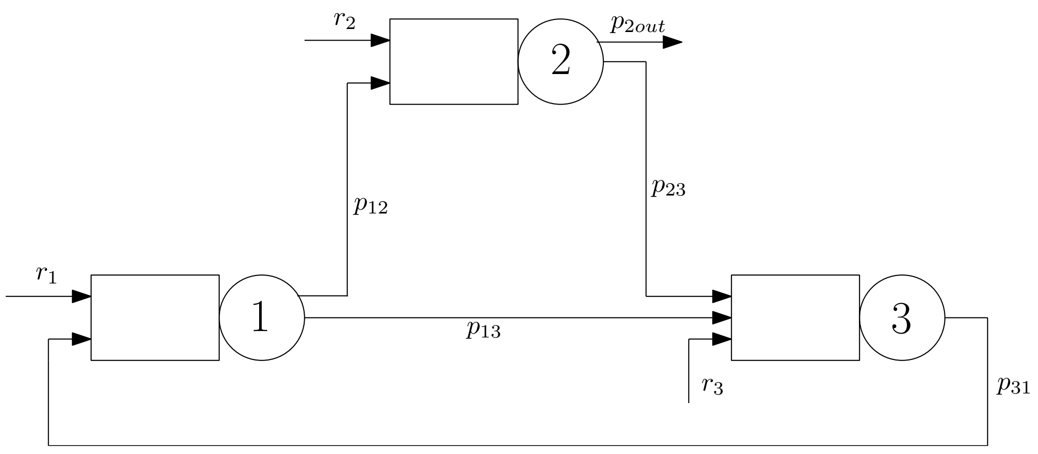
- $A(t)$ is the total number of jobs arrived before time t, and with an infinite buffer is calculated:
$$A(t) = \lambda t$$
- $D(t)$ is the total number of jobs departed before time t, and sure we can say that:
$$D(t) \leq \mu t$$

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.