Concurrency Is Not Parallelism

Mariana Ávalos Arce
Distributed Computing

Based on the video conference talk *Concurrency Is Not Parallelism* by Rob Pike[1], the following summary is made about the matter.

1 Concurrency Is Not Parallelism

- The world is mainly parallel but computer tools cannot intuitively express such view, so the way to make these tools express parallelism is through concurrency.
- Golang is a concurrent language, which means that it makes the task of implementing concurrency easier and actually useful.
- Concurrency is a way or a method of building a program: it means a composition of independent elements in a program that actually **interact** within each other during their process. It involves **dealing** with many things at once.
- Parallelism is the **simultaneous** execution of interacting or nor interacting elements. It involves **doing** many things at once.
- Concurrency involves structure, while Parallelism is about execution.
- Concurrency needs communication: it is a way of structuring independent things that interact together, and for that you need communication to reach **coordination**: you need to manage one resource to many requestors, for example. Many gophers can transport a book in a separate car, but they will need to drop their book in **one** incinerator. The fact that the gophers transport books on their own is Parallel *only* if they move books at the same time, but the incinerator needs Concurrency to manage a lot of gophers arriving at it at the same time. Therefore, a Concurrent program structure maybe does not use Parallelism too.
- Parallelism usually involves the duplication of processes **and** the execution of these at the same time.
- Concurrency may add more components for the implementation of these communicating elements, but it is used to increase the performance after all. Adding design elements to a program can actually make it more efficient.
- Parallelism mainly means how many duplicated workers are busy **at the same time**.
- To implement Parallelism, you need to have a problem that can be broken down into separate, **independent** elements that work simultaneously to solve to problem together.
- To put the code `function()` means that the program executes the function and it waits until it finished to move on. But if you code `go function()`, the function runs and you get to continue with your program right away. The function calls work concurrently, and maybe in parallel (because of OS threading and scheduling and etc.).

- A Goroutine $\approx$ A Thread. They are not the same because Goroutines are cheaper: they let you run a function without waiting for it to return, but they get indexed and scheduled by the Operating System to give them the threads they need. They make your program use the Thread concept but they are easier to make and you let the OS do the backstage process. Goroutines **feel** like Threads, but they are lighter to process and implement.
- Golang's communication between goroutines is done through Channels, which are like a pipe. This helps coordination because it can block goroutines elements when needed.
- A `for := range` loop through a channel **drains** the channel: it consumes the data in the pipe until it is empty.
- A channel makes it easier to drop old concepts of concurrent synchronization such as **mutex and locking**, because the buffer of a channel and its pipe-like structure makes it for you.
- A distributed program does not necessarily involve multiple machines, it mainly involves workers (could be machines or code elements) that handle a workload to compute at the end one task together.
- In Golang, channels enable communication and can even be passed around like variables, which is like passing around the pipe. This passing around channels directly enables a program to implement **Load Balancing**, if you also add a Heap to represent a Priority Queue to have the Load Balancing idea to include priority work.
- Concurrency enables Scalability.

References

[1] Robert Pike. *Concurrency is not Parallelism by Rob Pike.* URL: `https://www.youtube.com/watch?v=oV9rvDllKEg`.