Lecture 32 — Convergence, Ergodicity, Applications

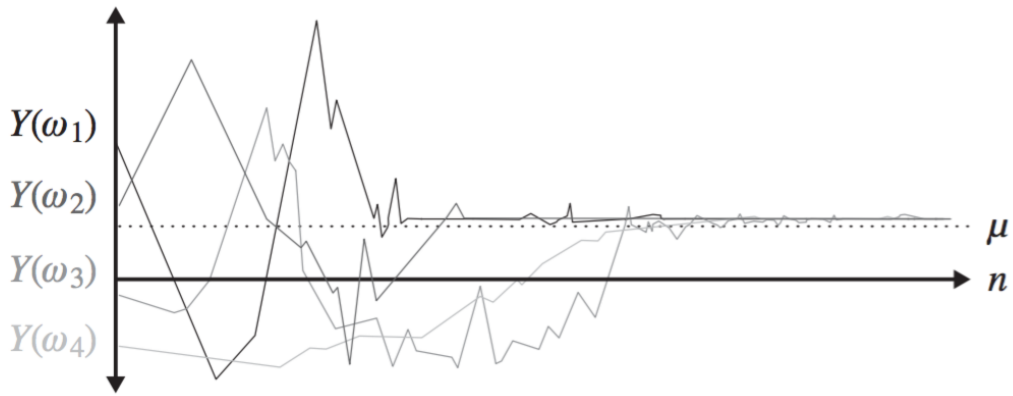
Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 9, 2022

Convergence

An image of what convergence looks like:



We won't concern ourselves with systems where there is no convergence.

Tim and Enzo

A small but important digression on the subject of sampling, measurement, and testing.

You have an idea of what an average is, but there are two different types of average—the time average and ensemble average.

Tim and Enzo Scenario

Let us just focus on having a single First-Come-First-Serve queue.

Every second, a new job arrives with probability p and if there is any work to do, the job being worked on is completed with probability q (and $q > p$).

As a definition, let $N(v)$ equal the number of jobs in the system at a time v.

In the story, Tim and Enzo are trying to simulate the FCFS system to determine what is the average number of jobs in the system.

The Tim Approach

Tim decides he's going to run it as one really long simulation.

He simulates the queue over a very long period, logging as he goes, taking a million samples.

Then he takes the average value over those samples to get the average number of jobs.

The Enzo Approach

Enzo does something slightly different: instead of having one super long simulation, he does 1000 shorter simulations.

He waits until the simulation has run for 1000 seconds and then samples the queue at exactly that point, obtaining one value.

This experiment is restarted with a new random seed.

So after obtaining a thousand samples, he averages these, and Enzo produces another average number of jobs.

Tim and Enzo

So—who has done this correctly, Tim or Enzo?

Time Average vs Ensemble Average

The time average has potential problems because we are only looking at a single sequence and maybe something very unusual has happened here in this single run.

The ensemble average is more likely what we talk about when we talk about the system being at “steady state” (i.e., past the initial conditions).

So we kind of like the Enzo approach. Tim’s approach still has some merit though.

Initial Conditions

Both the Tim and Enzo approaches here require caring about the initial conditions.

Enzo needs to make sure that the initial conditions (startup costs etc) have attenuated before the measurement point.

Tim needs to ensure that the initial conditions impact a sufficiently small portion of all his measurements.

Everyone's a Winner

If we have a nicely behaved system, the time average and the ensemble average are the same (so both Tim and Enzo can be correct).

What is a nicely behaved system? The word for this is ergodic.

That probably did not help, so what is an ergodic system?

It is a system that is positive recurrent, aperiodic, and irreducible.

Irreducibility

Irreducibility means a process should be able to get from one state to any other state (where state is the number of jobs in the system).

This means the initial state of the system does not matter.

So if we started at 0 jobs or 10 we could still get to any state in the system (jobs at 2 or 27)...

Positive Recurrence

Positive recurrence means that given an irreducible system, any state *i* is revisited infinitely often, and the time between visits to that state are finite.

So we can define a certain state as being a “restart”.

The logical choice in the case of a queue or similar is the idea of the queue being empty.

Every time the queue gets down to zero jobs, it’s a “restart” of sorts.

Positive Recurrence

This is what makes Tim's view and Enzo's view potentially the same.

A single long run (Tim's view) is just like a number of independent runs (Enzo's view).

Every time we get down to zero jobs in the queue, it's a restart.

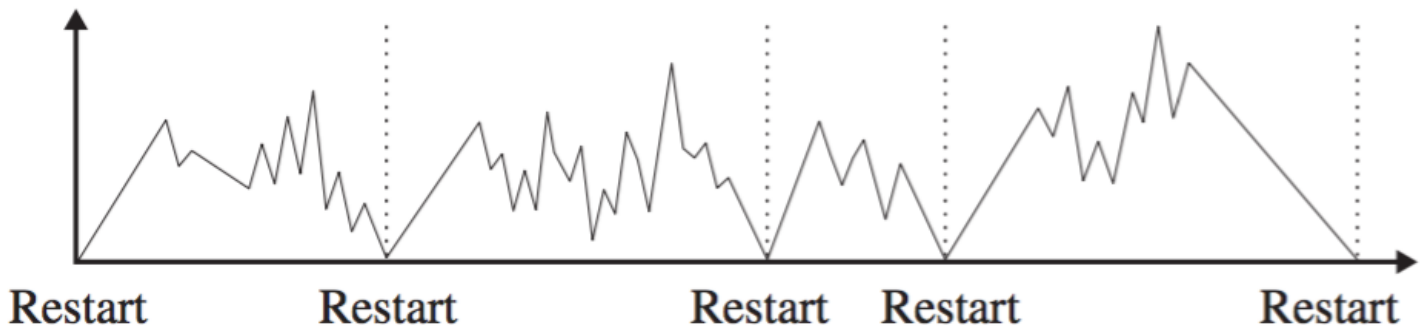
Aperiodicity

The aperiodicity condition is required for the ensemble average to make sense or exist.

That is to say, the state of the system should not be related to the the time.

Otherwise the way Enzo chooses to sample, i.e., $t = 1000$, is potentially going to skew the result.

Tim as Enzo



So both Tim and Enzo are correct, given an ergodic system.

How Long Jobs Are in the System

We could compute either the time or ensemble average.

$$\text{Time Average} = \lim_{t\to\infty} \frac{\sum_{i=1}^{A(t)} T_i}{A(t)}$$

where $A(t)$ is the number of arrivals by time t and T_i is the time in the system of arrival i. The average is taken over one sample path.

How Long Jobs Are in the System

$$\text{Ensemble Average} = \lim_{t \to \infty} E[T_i]$$

where $E[T_i]$ is the average time in the system of job i, where the average is taken over all sample paths.

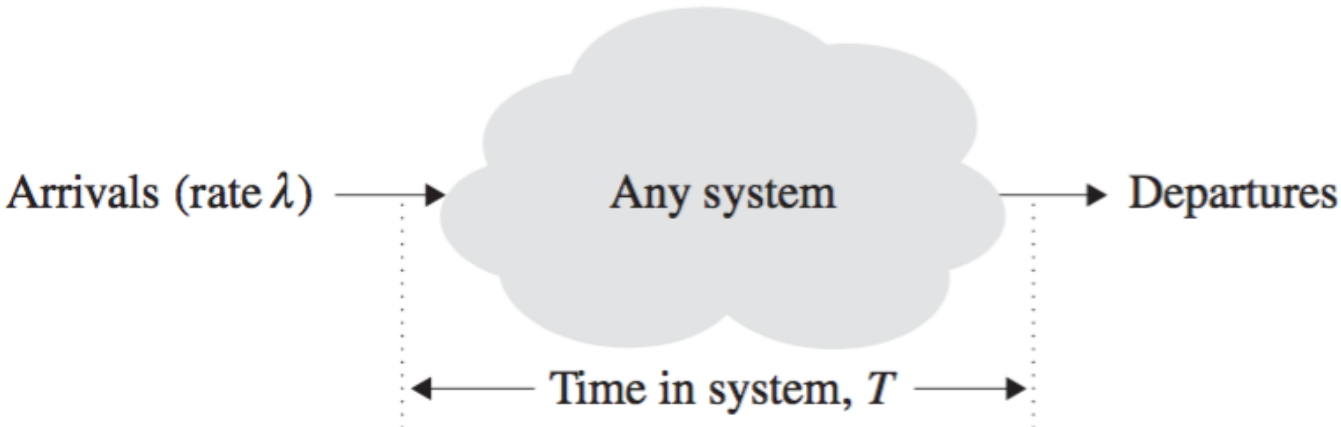
Little's Law

Little's Law says the average number of jobs in the system equals the product the average arrival rate into the system and the average time in the system.

Let's start with an open system. The law, written more formally:

$$E[N] = \lambda E[T]$$

Little's Law


Arrivals (rate λ)
Any system
Departures
Time in system, T

Ignorance is Bliss

What we don’t need to know about:

- the arrival process;
- the service time distribution;
- network topology;
- etc.

It seems intuitive that this is the case (or it should be).

Restaurant Analogy

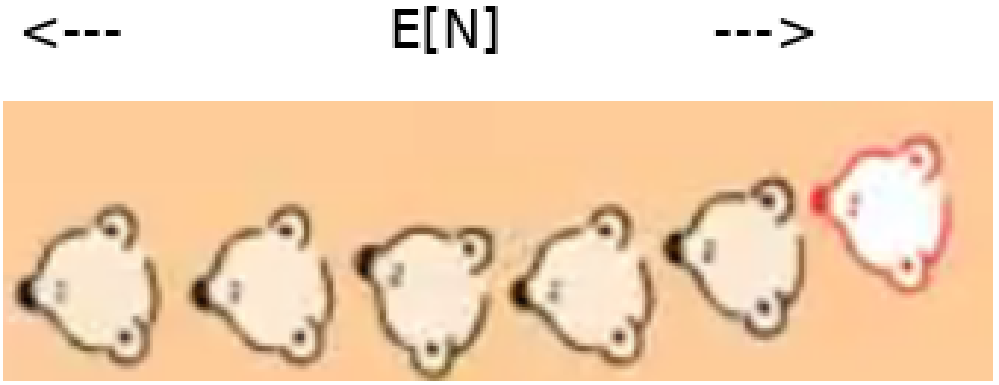
Key: Quick turnaround time.

Get people out quickly (low $E[T]$)
$\Rightarrow$ don't need a lot of seating (low $E[N]$)

Opposite situation:

people leave slowly (high $E[T]$)
$\Rightarrow$ need much seating (more $E[N]$).

Single FCFS queue:



Expected time for each customer to complete: $1/\lambda$
because the average rate of completions is λ.

So we can approximate $E[T]$ as roughly $\frac{1}{\lambda}E[N]$.

Little's Law for Closed Systems

For closed systems:

- N jobs in process at any given time
  (multiprocessing level of the system);
- X throughput rate.

If the system is ergodic, then $N = X \cdot E[T]$.

Assumes that there is zero think time.

Users Ruin Everything

If we do have to deal with users and think time, then we care more about the response time $E[R]$.

So for a terminal-driven system, expected response time is

$$E[R] = \frac{N}{X} - E[Z].$$

Probabilistic processes are described according to their models, which will probably be one of:

1. Deterministic (D)
2. Markov (M)
3. General (G)

Markov Processes

We'll focus on Markov processes.

- Number of arrivals follows Poisson distribution.
- Inter-arrival times follow exponential distribution.
- Service times follow exponential distribution.

Those letters we saw (D, M, G) are part of Kendall notation:

Six symbols, written in a specific order, separated by slashes.

The order is $\alpha/\sigma/m/\beta/N/Q$.

Kendall Notation

Symbol	Meaning
α	The type of distribution (Markov, General, Deterministic)
σ	The type of probability distribution for service time
m	Number of servers
β	Buffer size
N	Allowed population size (finite or infinite)
Q	Queueing policy

Why is abbreviation such a long word?

We often leave off the last three
assuming that there is an infinite buffer, infinite population, and a FIFO queueing policy.

If that is the case, then we have only three values.

This produces the “M/M/1” and “M/M/k” symbols.

“M/M/1” means a Markov arrival process, exponential queueing system, and a single server.

- When there are k servers, use k instead of 1.

Utilization

$$\rho = \text{utilization} \in [0, 1]$$

the amount of time that the server is busy.

We talked about this earlier in an informal way, but now we can actually calculate it!

$$\rho = \lambda \times s.$$

M/M/1 Formulæ

For M/M/1 systems:

- completion time average, T_q, is $\frac{s}{(1-\rho)}$
- average length of queue, W, is $\frac{\rho^2}{1-\rho}$

Queuing Theory Example

Server completes a request, on average, in 10 ms.

Time to complete request is exponentially distributed.

Over 30 minutes, 117 000 jobs arrive.

$\Longrightarrow$ a M/M/1 situation.

- How long did it take to complete the average request?
- What is the average queue length?

Queueing Theory Example

Given: service time s is $0.01s$,
arrival rate is 65 requests per second.

We calculate $\rho = 0.01 \times 65 = 0.65$.

Using the above formulæ:

- Time to complete the average request is 28.6 ms.
- Average length of the queue is 1.21.

Queueing Theory Example

What about the number of jobs in the system?

$Q =$ average number of jobs,
including the waiting jobs and the ones being served.

The probability that there are
exactly x jobs in the system at any time is:

$$(1 - \rho)\rho^x.$$

The probability that the number of jobs is less than or equal to n is:

$$\sum_{i=0}^{n}(1-\rho)\rho^{i}.$$

For more than n at a time, from $n+1$ to infinity…Or …

$$1-\sum_{i=0}^{n}(1-\rho)\rho^{i}.$$

Next: multiple servers.

Jobs arrive at a single queue.
When a server is ready,
it takes the 1st job from front of queue.

Servers are identical, jobs can be served by any server.

So far, so simple. Sadly, the math just got harder.

The server utilization for the server farm is now

$$\rho = \lambda s/N.$$

To make our calculations a little easier, we want an intermediate value K:

$$K = \frac{\sum_{i=0}^{N-1} \frac{(\lambda s)^i}{i!}}{\sum_{i=0}^{N} \frac{(\lambda s)^i}{i!}}.$$

K is always less than 1. It has no intrinsic meaning.

What is the probability that all servers are busy?

$C =$ probability a new job will have to wait in queue.

$$C = \frac{1 - K}{1 - \frac{\lambda s K}{N}}.$$

The M/M/k formulæ:

$$T_q = \frac{Cs}{k(1-\rho)} + s \qquad W = C\frac{\rho}{1-\rho}$$

Printer Example

Our printer completes an average print job in 2 min.

Every 2.5 minutes, a user submits a job to the printer.

- How long does it take to get the print job on average?

For a single printer, the system is M/M/1.

Service time s is 2 minutes;
arrival rate λ is $1/2.5 = 0.4$.

- $\rho = \lambda \times s = 0.4 \times 2 = 0.8$.
- $T_q = s/(1-\rho) = 2/(1-0.8) = 10$.

Ten minutes to get the print job. Ouch.

Printer Example

Executive Decision

Here we have an opportunity to use the predictive power of queueing theory.

Management is convinced that ten minute waits for print jobs is unreasonable, so we have been asked to decide what to do.

- Should we buy a second printer of the same speed?
- Or should we sell the old one and buy a printer that is double the speed?

The faster printer calculation is easy enough.

Now $s = 1.0$ and λ remains 0.4, making $\rho = 0.4$.

So rerunning the calculation:

$$T_q = s/(1 - \rho) = 1/(1 - 0.4) = 1.67.$$

1:40 is a lot less time than 10:00!

The two printer solution is more complicated.
So let us calculate intermediate value K.

$$K = \frac{\sum_{i=0}^{N-1} \frac{(\lambda s)^i}{i!}}{\sum_{i=0}^{N} \frac{(\lambda s)^i}{i!}} = \frac{\frac{(\lambda s)^0}{0!} + \frac{(\lambda s)^1}{1!}}{\frac{(\lambda s)^0}{0!} + \frac{(\lambda s)^1}{1!} + \frac{(\lambda s)^2}{2!}} = 0.849057.$$

Calculate!

Now C is 0.22857 and T_q is 2.38 minutes.

Two observations jump out at us:

1. we doubled the number of printers, but now jobs are completed almost four times faster; and
2. the single fast printer is better, if utilization is low.

Conditions

That is an important condition: if utilization is low.

At some point will the two printers be a better choice than the single fast one?

What if both printers are used to the max (100% load)...?

The basic process is:

1. Convert to common time units.
2. Calculate the visitation ratios V_i.
3. Calculate the device utilization ρ_i.
4. Calculate the CPU service time.
5. Calculate the device time.
6. Find the bottleneck device.
7. Calculate the maximum transaction rate.
8. Calculate the average transaction time.

Let us execute this process on a web server system that serves 9 000 pages per hour. Here are the known values:

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Webpages	9 000					
CPU					42%	
Disk 1	108 000		11ms			
Disk 2	72 000		16ms			
Network	18 000		23ms			

Step one is to convert to common time units; in this case, seconds.

Let's also look at the λ values—reported counts divided by seconds in the reporting period.

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Webpages	9 000	2.5				
CPU					42%	
Disk 1	108 000	30	0.011s			
Disk 2	72 000	20	0.016s			
Network	18 000	5	0.023s			

The visitation ratio is the number of times a device is used in each transaction; divide use by number of transactions to get V_i (you could also log this sort of thing).

The visitation ratio of the CPU is the sum of all other visitation ratios.

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Webpages	9 000	2.5		1		
CPU	207 000	57.5		23	42%	
Disk 1	108 000	30	0.011s	12		
Disk 2	72 000	20	0.016s	8		
Network	18 000	5	0.023s	2		

Next, calculate device utilization: $\rho = \lambda \times s$. That is, arrival rate times service time.

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Webpages	9 000	2.5		1		
CPU	207 000	57.5		23	42%	
Disk 1	108 000	30	0.011s	12	0.33	
Disk 2	72 000	20	0.016s	8	0.32	
Network	18 000	5	0.023s	2	0.115	

We can also get the service time of the CPU by rearrangement of the utilization formula to $s = \rho/\lambda$.

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Webpages	9 000	2.5		1		
CPU	207 000	57.5	0.0073s	23	0.42	
Disk 1	108 000	30	0.011s	12	0.33	
Disk 2	72 000	20	0.016s	8	0.32	
Network	18 000	5	0.023s	2	0.115	

And the device time is the final thing we can fill in for this table: $V_i \times S_i$ (just like the column header says!).

Device	**Data/Hour**	λ	S	V	ρ	$V \times S$
Webpages	9 000	2.5		1		
CPU	207 000	57.5	0.0073s	23	0.42	0.168
Disk 1	108 000	30	0.011s	12	0.33	0.132
Disk 2	72 000	20	0.016s	8	0.32	0.128
Network	18 000	5	0.023s	2	0.115	0.046

Is It Useful?

Did we need to complete the whole table? Probably not.

We cared most about ρ—utilization.

The bottleneck device, i.e., the one that limits our maximum throughput, is the one that is the busiest.

Thus, the one with the largest utilization.

This application appears to be CPU bound; it has the highest utilization at 42%, well ahead of disk 1 and disk 2.

Predicting the Maxes

Since the CPU is the bottleneck, we can predict the maximum rate of transactions (web page requests) we can serve.

$$\frac{1}{S_i V_i}$$ or in this example, 5.95.

This is also called saturation.

If λ exceeds saturation, we will not be able to keep up with incoming requests.

With this table we can also calculate the average transaction time: it is the sum of the $S_i V_i$ columns.

In this example, it is 0.474 seconds.

It Gets Worse!

Typical assumption:

we know the service times for each device.

Unfortunately this is not true;

perf monitoring gives average size of device queue.

So we had better apply queuing theory here.

The average size of a device's queue is W, and for an M/M/1 queue then

$$W = \frac{\rho^2}{1-\rho}.$$

Combining the known W with the average arrival rate λ, we can work out the service time.

$$\begin{aligned} W &= \frac{(\lambda s)^2}{1-\lambda s} \\ s &= \frac{-w \pm \sqrt{w^2+4w}}{2\lambda} \end{aligned}$$