SFWR ENG 4E03

Kemal Ahmed

Fall 2015

Instructor: Vincent Maccio

Note: material covered in [Stats 3Y03 Summary](https://drive.google.com/file/d/0BxW61uJyyN8TNy1iUFE0ZlRMLTg/view) will not be covered in this summary. To find a unit CTRL-F “[<unit>]”, e.g. for Number of jobs in system, CTRL-F “[N]”

Contents

[Statistics 2](#_Toc436247953)

[Variance 2](#_Toc436247954)

[Exponential 2](#_Toc436247955)

[Uniform 2](#_Toc436247956)

[Binomial 2](#_Toc436247957)

[Operations Analysis 3](#_Toc436247958)

[Visitation Trick 4](#_Toc436247959)

[Summation Equations 4](#_Toc436247960)

[DTMC 5](#_Toc436247961)

[Balance Equations 5](#_Toc436247962)

[Matrices 6](#_Toc436247963)

[CTMC 6](#_Toc436247964)

[Poisson Process 6](#_Toc436247965)

[Kendall notation 6](#_Toc436247966)

[Variations 6](#_Toc436247967)

[M/M/1 7](#_Toc436247968)

[M/M/1/N 7](#_Toc436247969)

[M/M/C 7](#_Toc436247970)

[M/M/∞ 7](#_Toc436247971)

[Queuing 8](#_Toc436247972)

[Square Root Staffing Rule 9](#_Toc436247973)

[e.g.) 9](#_Toc436247974)

[Jackson Networks 9](#_Toc436247975)

[Traffic Equations 10](#_Toc436247976)

[Mean Value Analysis 10](#_Toc436247977)

# Statistics

**Poisson parameter** [λ]: rate

**Service rate** [μ]:

**Continuous Random Variable (CRV)**:



*Think chemistry, i.e. cancelling units*

## Variance



* Don’t change probability, but square X for calculation only



**The higher your variance, the worse your system will perform.**

## Exponential

* **Mean** [E[X]]: 1/λ
  + a.k.a. Expected value
* **Variance**: 1/λ2
* **Probability Distribution Function (PDF)** [P(X=x)]: λe–λx/x!
* **Cumulative Distribution Function (CDF)** [f(x)]: CDF = ∫PDF, i.e. 1 – e–λx
* Memoryless
* not always for time

## Uniform

* **Variance**: (b–a)2/12
* **Mean**: (a+b)/2
* **PDF**: 1 / (b–a) , a ≤ x ≤ b
* **CDF**: x–a/b–a
* **Uniform Distribution**: no memoryless property

## Binomial

* **Mean** [E[X]]: n × probability
* **Variance**: n × p × (1 – p)
* **Probability Distribution Function (PDF)** [P(X)]: (n c x)px(1–p)n–x
* **Cumulative Distribution Function (CDF)** [f(x)]: 

# Operations Analysis

**Device** [i]: units that are in terms of *i* are specific to an individual device or node within a system

**Total devices** [k]:

**Service Time** [S]: time per specific job

1/μ

**Visitation** [V]: given or projected visits/jobs (closed system); cannot be calculated; basically a probability

[E(V)]: calculated visit/job ratio

P(visit)∙total visits in previous node

**Demand** [D]: total service time for all jobs





**Bottleneck** [Dmax]: device with largest demand, utilization

**Time in system** [T]: time the job is in the system





If E[Z] = 0, T = R

**Response Time** [R]: time the job is *being processed* in the system

If E[Z] = 0, R = T

M/M/1: E[R] = 1/(μ–λ)

M/M/1/N: E[R] = E[N]/λ'

M/M/C: E[R] = E[RQ] + E[S]

**Users** [M]:

**Optimal users** [M\*]:



**Total Jobs** [N]: N=M in a closed system

* *Little’s Law*:
* 
* M/M/1:
  + E[N] = λ/(μ–λ) = *ρ*/(1–*ρ*), if you have overall system λ
  + E[N] = ← probability × #jobs, if your λ or μ is different for each state
* M/M/1/N: E[N] is expected # jobs, N is max # jobs
* M/M/C: go through Little’s law
  + E[N] = E[NQ] + ρ
* M/M/∞:
* Jackson Network: E[N] = ΣE[Ni] = ΣPλ/(μi–Pλ) = Σ(λi/(μi–λi))

**Think time** [Z]: time it takes the user to put a request in and start, it’s kinda like the frequency that users put in requests (seconds / request)



**Throughput** [X]: out-rate, max jobs / hour of full system



Note: andconverge at their lowest point, so equate them



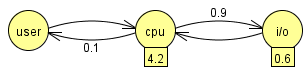
**Utilization** [ρ]: ratio that the time is busy



ρ = λ/ciμ

## Visitation Trick

If determining visitation at a node, establish a reference node from one of the incoming nodes, usually the user node, that has a returning percentage



Vuser = 1 = 0.1 ∙ VCPU

## Summation Equations

**Geometric Series**: , where 0 ≤ r ≤ 1 (because otherwise it would be unstable)



**Geometric Sequence**: 



Removing the annoying factors:



Take out a value so the integral takes out the i and i+1



# DTMC

**Discrete Time Markov Chains (DTMC)**:

[n]: number of tasks in queue / system

Steady state: n->∞

For discrete: use the sum of the X’s, so E[X] = Σ(P(X=i)∙Xi) and E[X2] = Σ(P(X=i)∙Xi2)

## Balance Equations



^think of it like series / parallel, where you add multiple connections out in different directions (parallel) and multiply connections stacked onto each other (series)

OR jobsin = jobsout

## Matrices

Rows: equations for nodes going out (add up to 1)

Columns: equations for nodes coming in

# CTMC

## Poisson Process

**Counting Process**: a way of determining the time between consecutive occurrences of an event

**Poisson Process**: a *counting process*, whose time between arrivals uses Exponential Distribution

* λtotal = Σλi
  + you can also split up λ into multiple λs
* Not only do you see each second as time independent, each stream of probabilities is independent
* P(x;λ) = e–λλx/x!
  + [x]: things will happen
  + [λ]: rate; λ = αt
* [α]: expected number of events during unit interval
* [t]: time interval length
* 

## Kendall notation

**Job Processing time** [μ]: rate of jobs leaving system (jobs/sec)

μ = 1/ processing\_time\_per\_job

M/M/1 Queue

[M]: time between arrivals is Markovian (Memoryless) ~ exp(λ)

[M]: job processing times are Markovian (Memoryless) ~ exp(μ)

[1]: single server

(Σpout)×πi = Σpjπj, j=0..n, j≠i

π0: percent of time that the queue is empty

Attributes:

* FIFO
* Infinite buffer

### Variations

* M/M/2 Queue: same, except 2 servers
* M/M/C Queue: *C* servers
* M/Ek/C: Erlang k, i.e. series of exponential
* H()/M/C: hyperexpontial distribution
* PH/M/C: phase type, i.e. any combination of any number of exponentials with any rate
* M/G/C: General distribution
* G/G/1: has not been solved yet
* M/M/1/1: 1 server, 1 job

[c]: number of servers

Think: one queue goes to multiple servers



### M/M/1

π0 = 1 – λ/μ

### M/M/1/N

When you can only have up to N jobs in system queue.

[λ']: rate jobs enter the system, until the queue is full

λ' = λ(1 – πN)





**Waiting**: jobs put into the queue

**Blocked**: jobs not allowed in the queue

### M/M/C

Useful if multiple jobs are sharing the same queue

Does the μ you use for equations double in M/M/2? No, but you’ll see jobs coming out of a system at a rate of c∙μ.



****

### M/M/∞

Same as M/M/C, except:



and just find the unit

## Queuing

**Blocking Probability** [PQ]: probability that a process will be blocked when entering the system and be placed in the queue

Erlang-C Equation: 

Given λ and μ, what should c be so PQ < ρ

**Waiting time in queue** [RQ]: response time of queue



M/M/1: 

M/M/C: 

M/M/∞: E[RQ] = 0

**Number of jobs in queue** [NQ]:

M/M/1: ρ2/(1 – ρ)

M/M/C: 

M/M/∞: E[NQ] = 0

You need to know what is in the progression of each step

#### e.g.

When you have varying





## Square Root Staffing Rule

Given an M/M/c queue with arrival rate, λ, server speed, μ, and ρ is *large* (assume this means over 100, but we don’t actually know what it means), α is a bound on PQ, let denote the least # of servers needed to ensure that PQ < α. Then

, where k = is the solution to

, where Φ(∙) is the CDF of the standard normal and φ() is its pdf

[K]: minimum # servers to stay stable λ/μ or ρ

[k]: a constant…just assume 1 for now

### e.g.)

|  |  |  |
| --- | --- | --- |
| α | k | ρ + k√ρ |
| 0.8 | 0.178 | 10, 018 |
| 0.5 | 0.506 | 10, 051 |
| 0.2 | 1.06 | 10, 106 |
| 0.1 | 1.42 | 10, 142 |

[Q]: transition matrix



Replace i <--> j to get qjj and qji.

# Jackson Networks

P(N1 = n1) =



π**n~** = P(state of system n~)P(n jobs at node i)



**Poisson Arrivals See Time Averages property (PASTA)**: the probability of the state (i.e. πi) as seen by an outside random observer is the same as the probability of the state seen by an arriving customer

λtotal = Σλin,i

## Traffic Equations

For each node, what is the number of jobs entering?

λx = R + ΣPi,entering ∙ λi,entering

response rate + probability of each job entering

# Mean Value Analysis

* Performs better than balance equations or Jackson Network, but can’t find steady state distribution or PDF
* Recursive algorithm
* Only finds E[N], i.e. mean queue length

The higher your variance, the worse your system will perform.

**Pareto distribution**: an exponential which doesn’t start at 0 (a.k.a. **zipfian**)

Just think: 99% controls 50% and 1% controls the rest