ty Simulations & Modelling 15/16

1.

a)

State variables:

n=0 (population of system)

c (buffer size, given)

t = 0 (current time)

totalPackets = 0

droppedPackets = 0

packetDistribution = double[c + 1] = {0.0, 0.0, …, 0.0}

nextArrivalTime = 0

ArrivePacket {

If (n+1) == c {

droppedPackets++

} else {

n++

}

dt = sample from packet arrival distribution

schedule(t + dt, ArrivePacket)

totalPackets++

// We are going to be at state s = n for the next dt seconds (unless there’s a flip)

packetDistribution[n] += dt

// We store this for the purposes of correcting the distribution during a flip

nextArrivalTime = dt

}

Flip {

If n != 0 {

// Take away the remaining time that we were supposed to be in state s = n

packetDistribution[n] -= nextArrivalTime - t

}

// We are going to be at n = 0 until the next arrival

packetDistribution[0] += nextArrivalTime - t

n = 0

dt = sample from buffer flip distribution

schedule(t + dt, Flip)

}

Initialize {

// Schedule first arrival

dt1 = sample from arrival distribution

nextArrivalTime = dt1

schedule(dt1, ArrivePacket)

// Schedule first flip

dt2 = sample from buffer flip distribution

schedule(dt2, Flip)

// 0 packets until first event

packetDistribution[0] += min(dt1, dt2)

}

EndSimulation {

// Normalise packetDistribution by t, the total simulation time, to give probability distribution

for (i = 0; i < packetDistribution.length; i++) {

packetDistribution[i] = packetDistribution[i] / t

}

// Calculate probability of packet being dropped

probPacketDropped = droppedPackets / totalPackets

}

b)

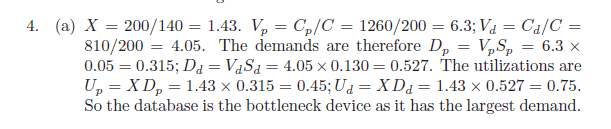
GBEs: lambda \* p0 = mu \* [(sum of i = 1 to c) pi] = mu \* ( 1 - p0) (case with first node for state 0)

(lambda + mu) \* pn = lambda \* p(n-1) (case for 0 < n < c),

mu \* pc = lambda \* p(c-1)

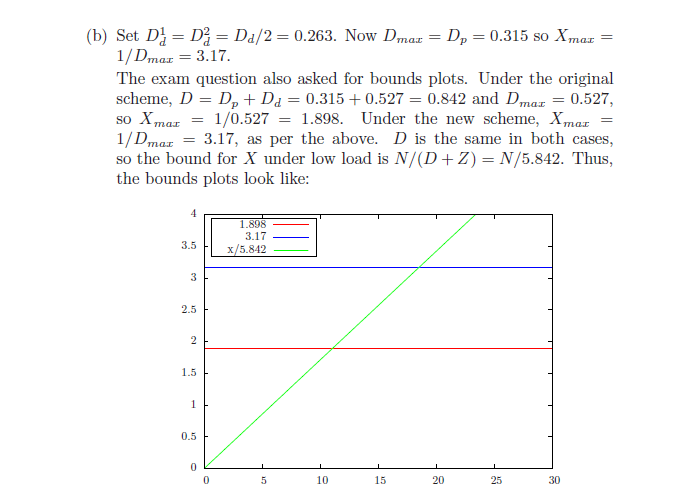
c) Check LHS = RHS using substitution of given solutions.

2a) i) From tutorial 1:



ii) Using N = X(R + Z), N = 20, X = 200/140, Z = 5 => R = N/X - Z = 14 - 5 = 9 seconds.

iii)/iv) From tutorial 1:



2b) i) To apply the Inverse Transform method we take U ~ U(0,1) and determine X = F^-1(U) from our sampled U.

When k = 1, f(x) = 1/theta \* e^-(x/theta) and F(x) = 1 - e^(-x/theta).

Let U = F(x). Then U = 1 - e^(-x/theta) => e^(-x/theta) = 1 - U and so -x/theta = ln(1-U) = ln (U) [as U~U(0, 1)].

Therefore X = - theta \* ln(U).

ii) For k = theta = 2 we have f(x) = ¼ x e^-(x/2).

g(x) = lambda e^(-lambda \* x). We can let lambda be some value >= 0, say ¼.

c = max(f(x)/g(x)).

Let y = f(x) / g(x) = x \* e^(-¼ \* x).

dy/dx = e^(-¼ \* x) \* (1 - ¼ \* x). When dy/dx = 0, x = 4. => c = 4 \* e^(-1).

[Note d^2y/dx^2 = -0.09… at x = 4 (< 0 => max point).]

So c = 4/e and h(x) = 4/e \* g(x) = e^-(¼ x) - 1.

We have found c such that cg(x) >= f(x) for all x >= 0 and so we can sample X from g(x) and take U ~ U(0,1) and find Y = Uh(x). If y <= f(x) then we accept X.

iii) We have p = 1/c and so E(X) = (1-p)/p = 0.472 ( 3sf). We then find 2(E(X) + 1) to account for (X, Y) samples before our success and our final success and we find 2(E(X) + 1) = 2.944 (3dp).

3a) i) Idea: 6 servers for cache [model as an M/M/c with c = 6 to ensure that no jobs will wait at the cache] and one queue for processing server. All servers for cache will link to server. (Server might link to some hard drive or memory before linking back to cache?)

Each rate for the cache servers will be 1/100 and the rate for the processing server will be 1/10./

ii)

**[To amend to consider 6 servers for cache instead of 5]**

Assuming worker threads receive jobs equally we have vi = 1 for i = 1..5 (where we assume v1 = 1 and set the rest as equal). v6 = 1 as there are no loops returning to the cache servers.

Li(0) = 0 for all i in {1..6}.

R(1) = 5 \* (100)[ 1 + 0] + 10 \* [ 1 + 0] = 510 ms.

T(1) = 1/510.

Li(1) = 1/510 \* (100(1 + 0)) = 100/510 = 10/51 for i in {1.. 5}

L6(1) = 1/510 \* 1 \* 10 = 1/51.

[The above mean lengths sum to 1]

R(2) = 5 \* 100 (1 + 10/51) + 10 \* (1 + 1/51) = 10340/17 = 608.24 ms (2dp)

T(2) = 17/10340

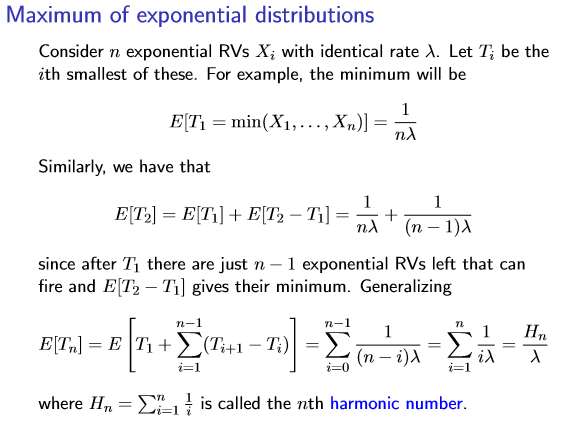
Li(2) = 17/10340 \* 100(61/51) = 305/1551 for i in {1.. 5}

L6(2) = 17/10340 \* 10(52/51) = 26/1551

[The above mean lengths sum to 1]

iii) Unexaminable? I don’t believe we have covered priorities for jobs.

b) i) H(n) = sum(1/i) from i=1 to n.



ii) M = C + n. Dk = Vk/(mu)k so we can sub in Dk for equations to response time and mean length in MVA. (AMVA not examinable)

4a)

i )alpha < 1/2 as routing probabilities leaving any given node must add to 1 (note if alpha = 1 then nothing would ever leave the system and so utilisation on the first node would eventually reach 1).

Traffic equations:

lam\_1 = gamma + alpha \* lam\_2

lam\_2 = lam\_1

We substitute the second equation to the first one and get an expression for lambda 1 in terms of alpha and gamma, i.e. lambda1 = gamma / (1 - alpha). Then use mu1 = 2 gamma and lambda / mu < 1, which gives alpha < ½. (If we consider lambda2 = gamma/(1 - alpha) and mu2 = 4 gamma, we get alpha < ¾ so we take the minimum of the two, which is alpha < ½).

ii) Vector gamma = (gamma, 0)

Q\* = [[0 1]

[alpha 0]]\

We find lambda1 = lambda2 = gamma/(1 - alpha).

p1 = (1 - rho) \* rho^1, rho = lambda2/(mu)2. => rho = ¼ \* 1/(1-alpha). => p1 = (3-4 \* alpha)/(4 - 4 \* alpha)^2.

iii) If we replaced the node with 4 Exp(k \* theta) queues, the jobs arriving at the 2nd, 3rd and 4th would depend on the jobs that arrive in the nodes before them (1st, 2nd, 3rd…) and so there is a dependency between jobs arriving at different nodes. We cannot assume that jobs arrive at nodes independently which is required for Jackson’s theorem. We can still find the mean lengths of each queue by using the 4 exponential dists (which have rate 16). E(X) = sum of E(Xi) from i = 1 to 4 => E(X) = ¼.

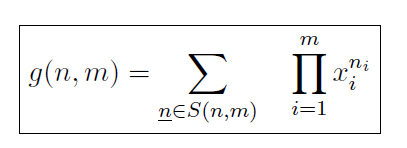
Rate for first queue:

N[i] = rho[i] / (1 - rho[i])

rho[1] =½ => N[1] = 1

For the second queue, model as 4 consecutive M/M/1 queues, each with rate 16 jobs/s. Then the mean queue length of that queue is the sum of the individual queue lengths, which is equal to 4 \* (1/16) / (1 - (1/16)) = 4/15

b) i) Equations from slides for G but with xi replaced with Di, where sum of ni = K.



ii) Uk = Dk \* X(K). As K -> inf and M -> inf, where M/K is constant, then X(K) will increase at the rate M(K) as there are more jobs and more nodes that these jobs can be served at.

Also have (1): g(K, M) = g(K, M - 1) + D\_1 g(K - 1, M) since all D\_k equal

Then as from slides, (2): X(K) = g(K - 1, M) / g(K, M)

Now U\_k = X(k) \* D\_1

= D\_1 g(K - 1, M) / g(K, M) (from (2))

= (g(K, M) - g(K, M - 1)) / g(K, M) (substituting from (1) into numerator)

= 1 - g(K, M - 1)) / g(K, M) (cancelling terms)

= 1 - 1 (as K, M both grow to infinity)

= 0

iii) g(K, M) = 108. (Use Dk as the relative load xk (defs are equivalent), where we index on k)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M/N | 0 | 1 | 2 | 3 |
| 1 | 1 | 3 | 9 | 27 |
| 2 | 1 | 3 + 3(1) = 6 | 9 + 3(6) = 27 | 27 + 3(27) = 108 |