Homework#2 (h2) CSC148 Prof Mitchell Posted: 9/22/2019 Revision#1, posted 9/24

Due dates 1) ExtraCredit: start OfClass ,Thursday Sep 26 NOExtraCredit, Tuesday Oct 1

#1 - Introduction

A first example hps model was covered in class. It is a special case of hps service because each cj requests exactly one server. For many systems in daily life, cj requests a random (small) number of servers rather than just one server.

In h2, the following significant extensions of the class model example are implemented:

1. Each arriving cj requests a (small) random number of free servers from the hps degree total
2. Two different cj service configurations are compared:

* In the first configuration, cj service duration is not affected by current hps “load”;

(load means the kinds and rate of service requests)

* In the second configuration, cj service duration is also dependent on current number of busy servers;

service duration is slower or faster depending on whether many vs. very few servers are busy; this model is more realistic; most real hps systems use internal resources for implementing services, and

such resources are always limited in physical systems *(this matches common experience in, say a multi-user & multi-core computer server – when the system is running slowly, you conclude that there is a heavy load of users on the system)*

Let degree = number of hps servers, and nrs = the number of servers requested by cj (nrs is a random variable).

During model execution, the gpss built-in function call r$hpsName returns the number of busy servers, where hpsName is the label on the STORAGE statement associated with this hps *(that is, the hps’s name in the source code)*

h2 uses a simple expression for the time delays caused by the servers themselves competing for resources: (.45)**(r$hpsName)2/degree, abbreviated sc**. *The value of sc is interpreted as sc t.u.s*

*Note that 0 <=* ***sc*** *<= (.45) (degree2)/degree = (.45) degree*.

**Waiting for, and starting a cj service**

The class example hps model is less general than variable cj request sizes in the h2 model. For cj constant request size 1, a GATE block works well. But, in h2, replace GATE with a Refuse mode form of the TEST block:

TEST O A,B

Consult the gpss Reference Manual for O operator and A and B operands needed. Because the optional C operand is missing in this TEST block format, when the tested condition fails, cj is suspended at this TEST block, otherwise cj proceeds to the block following this TEST block. This does for h2, using only one block, what the GATE did for our example hps model in course module P2.

*Hints – it suffices to test (the current number of free servers) >= (cj request size). Thus, relational operator O should be GE.*

*s$hpsName is a gpss built-in function call returning the current number of free servers.*

There are 2 (operational) Configurations / Use Cases for cj service:

For each of 1) and 2) below, each arriving cj requests a number of servers = RN2@3+1 (Refer to such a value as x)

1. Intra-server congestion is not represented ; service duration is modeled as in h1
2. Intra-server congestion is modeled by adding term sc to the Exponential() call for service duration in 1) above

Configuration 2 has a significant complication. When cj is allocated x free servers using a LEAVE block, then cj must also, after finishing service, give back those x servers with an ENTER block. Clearly, returning y different from x servers would invalidate specification of cj behavior (and, sooner or later, the model will become unstable).

Use a **tr parameter** that stores the value x that was allocated via the LEAVE block. Each tr parameter acts like a “private local variable” to that tr, and exists only for the duration of that tr (*tr parameter details below*)

Some thought should convince you that using a savevalue to store x during cj’s lifetime will not work. The reason is that each cj will need a savevalue to store x (so that x servers can be correctly returned after use). For significantly large hps degree, and unpredictable processing queueLength, the number of savevalue entities needing initialization via an initial statement is unpredictable => either many savevalues are never used OR the number of pre-initialized savevalues is exceeded in long model runs.

Using a tr parameter solves this problem: each cj tr (whether being served or waiting) has its x value in local tr memory.

*A coding/readability requirement is: use a user-defined symbol for parameters. Do not use the legal, but meaningless name p1, or worse yet, an integer (which is legal) when referencing the tr parameter needed by cj’s tr.* *Doing so is poor documentation practice.*

**gpss tr parameters - details**

For each parameter that a tr type uses, define a symbolic name for it

parmName1 EQU 1 ; Short comment on the meaning of this parameter

parmName2 EQU 2 ; xxx

The only way that a tr can store a value in a tr parameter is by a block like the following: assign parmName,value

The way that the value currently stored in a tr parameter is retrieved is with the expression p$parmName.

*To repeat, if >=2 cj are currently being served, and they all have stored a value in their parameter named myParm,*

*these parameters are separate entities (that is, different local variables, visible only to blocks in each cj’s tr.*

#2 - Model runs

h2 runs use: fixed {ia} mean = 1/ = 1.0 t.u., and service duration mean = 1/t.u. Each model run uses Start 10000

Also, EVERY run uses DIFFERENT RN1, RN2 seed value pairs = > avoids reliance on one RNk for producing each run’s pseudo random numbers. All runs use exponential {ia} and service process distributions.

#3 – HandIns

h2 executions are divided into 2 groups of runs corresponding to the 2 Configurations described in section #1.

First HandIn – Fill in Table 1 and Table 2 as described below

Table 1 – Runs without server congestion

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ****  **fixed** | ****  **0.25 fixed** | **hps degree** | **Theoretical ** | **gpss result ** | **avg(Wcj)** | **L = avg(n(t))** | **MAX**  **L in S** | **RN1 seed** | **RN2 seed** |
|  |  | **8** | **-** |  |  |  |  |  |  |
|  |  | **16** | **-** |  |  |  |  |  |  |
|  |  | **32** | **-** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Make runs, starting with degree = 8, and double the degree on successive runs. Continue until 2 successive runs result in L <= 4.0 In the row for each run’s results fill in the third column (as needed) and columns 5 through last column.

Table 2 – Runs with server congestion

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ****  **fixed** | ****  **0.25 fixed** | **hps degree** | **Theoretical ** | **gpss result ** | **avg(Wcj)** | **L =**  **avg(n(t))** | **MAX**  **L in S** | **RN1 seed** | **RN2 seed** |
|  |  | **8** | **-** |  |  |  |  |  |  |
|  |  | **16** | **-** |  |  |  |  |  |  |
|  |  | **32** | **-** |  |  |  |  |  |  |
|  |  | **64** | **-** |  |  |  |  |  |  |
|  |  |  | **-** |  |  |  |  |  |  |
|  |  |  | **-** |  |  |  |  |  |  |

Make runs starting, with degree = 8 until a run occurs with L <= 5.0 The number of runs needed

is unpredictable. Fill in Table 2 columns as done for Table 1.

Second HandIn – For the NON congestion Configuration case only, for hps degree 16 only, a) the histogram of distribution of L values and b) the histogram of the number of servers requested by the cj

For the congestion Configuration case only, for hps degree 16 only, the histogram of distribution of L values.

Third HandIn - For the congestion Configuration hps degree 16 only a) the source code and b) only the Report page containing the QUEUE and STORAGE sections; to repeat, do not print that entire Report file of several pages

*================================================================================================*

*An overall hint/suggestion – you do not need to code 2 separate models for the 2 Configurations. You can comment out blocks & statements that are needed or not needed, per Configuration (See h1 solution as an example).*