

Лабораторная работа № 11

Модель системы массового обслуживания M|M|1

Шуплецов Александр Андреевич

Содержание

0.1	Цели и задачи	3
0.2	Реализация модели системы массового обслуживания $M M 1$ в CPN Tools	3
0.3	Мониторинг параметров моделируемой системы	6
1	Выводы	11

Список иллюстраций

0.1	Декларации модели СМО	4
0.2	Граф и параметры сети системы обработки заявок в очереди . . .	5
0.3	Граф и параметры генератора заявок системы	5
0.4	Граф и параметры процесса обработки заявок на сервере системы	6
0.5	Функция Predicate монитора Ostanovka и функция Observer монитора Queue Delay	7
0.6	График изменения задержки в очереди	8
0.7	Функция Observer монитора Queue Delay Real	8
0.8	Содержимое Queue_Delay_Real.log	9
0.9	Функция Observer монитора Long Delay Time	9
0.10	Периоды времени, когда значения задержки в очереди превышали заданное значение	10

0.1 Цели и задачи

Реализовать в CPN Tools модель системы массового обслуживания M|M|1.

0.2 Реализация модели системы массового обслуживания

M|M|1 в CPN Tools

Зададим декларации системы(рис. [0.1]). Определим множества цветов системы (colorset): - фишки типа UNIT определяют моменты времени; - фишки типа INT определяют моменты поступления заявок в систему. - фишки типа JobType определяют 2 типа заявок — А и В; - кортеж Job имеет 2 поля: jobType определяет тип работы, соответственно имеет тип JobType, поле AT имеет тип INT и используется для хранения времени нахождения заявки в системе; - фишки Jobs — список заявок; - фишки типа ServerxJob — определяют состояние сервера, занятого обработкой заявок. Переменные модели: - proctime — определяет время обработки

заявки; - job — определяет тип заявки; - jobs — определяет поступление заявок в очередь

```
▼ Declarations
  ▼ System
    ▼ colset INT = int;
    ▼ colset UNIT = unit timed;
    ▼ colset Server = with server timed;
    ▼ colset JobType = with A|B;
    ▼ colset Job = record jobType : JobType * AT : INT;
    ▼ colset Jobs = list Job;
    ▼ colset ServerxJob = product Server * Job timed;
    ▼ var proctime : INT;
    ▼ var job: Job;
    ▼ var jobs: Jobs;
    ▼ fun expTime (mean: int) =
      let
        val realMean = Real.fromInt mean
        val rv = exponential ((1.0/realMean))
      in
        floor(rv+0.5)
      end;
    ▼ fun intTime() = IntInf.toInt(time());
    ▼ fun newJob() = {jobType = JobType.ran(), AT = intTime()};
```

Рис. 0.1: Декларации модели СМО

Модель состоит из трех отдельных листов: на первом листе опишем граф системы (рис. [0.2]); на втором — генератор заявок (рис. [0.3]); на третьем — сервер обработки заявок (рис. [0.4]).

Сеть имеет 2 позиции (очередь — Queue, обслуженные заявки — Complited) и два перехода (генерировать заявку — Arrivals, передать заявку на обработку серверу — Server). Переходы имеют сложную иерархическую структуру, задаваемую на отдельных листах модели (с помощью соответствующего инструмента меню — Hierarchy).

Между переходом Arrivals и позицией Queue, а также между позицией Queue и переходом Server установлена дуплексная связь. Между переходом Server и позицией Complited — односторонняя связь.

Граф генератора заявок имеет 3 позиции (текущая заявка — Init, следующая заявка — Next, очередь — Queue из листа System) и 2 перехода (Init — определяет

распределение поступления заявок по экспоненциальному закону с интенсивностью 100 заявок в единицу времени, Arrive — определяет поступление заявок в очередь).

Граф процесса обработки заявок на сервере имеет 4 позиции (Busy — сервер занят, Idle — сервер в режиме ожидания, Queue и Complited из листа System) и 2 перехода (Start — начать обработку заявки, Stop — закончить обработку заявки).

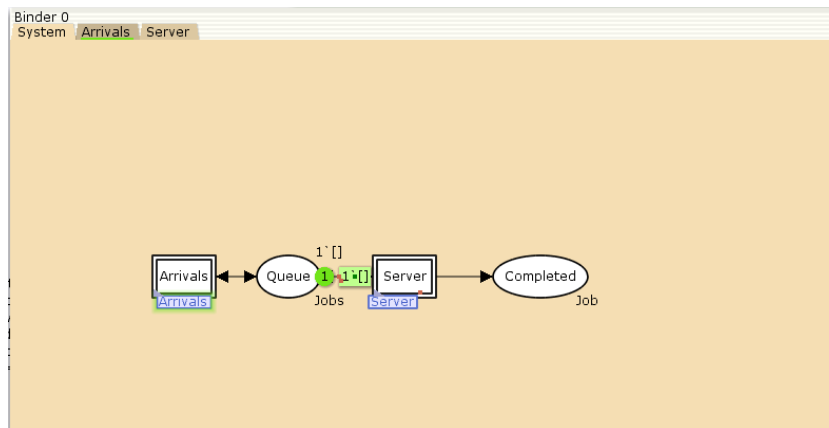


Рис. 0.2: Граф и параметры сети системы обработки заявок в очереди

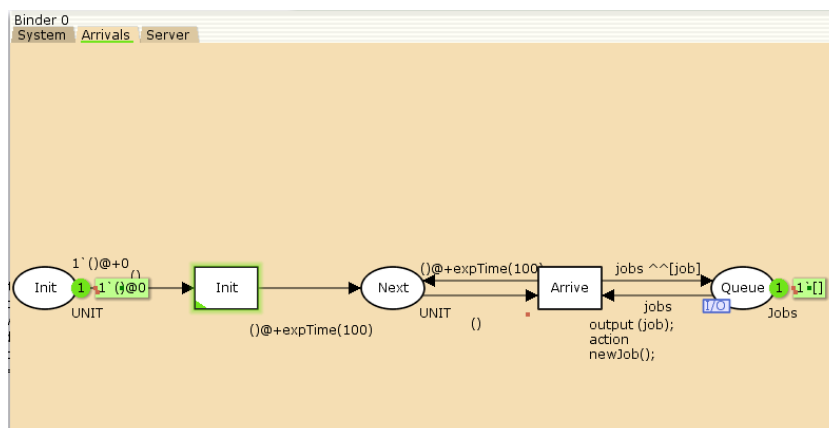


Рис. 0.3: Граф и параметры генератора заявок системы

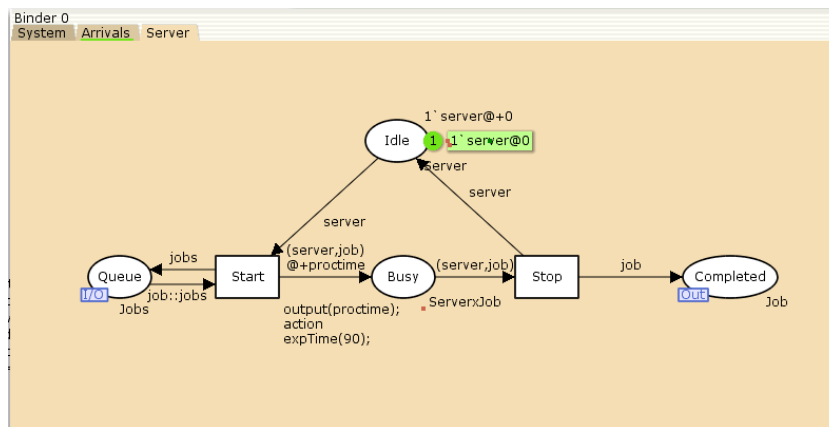


Рис. 0.4: Граф и параметры процесса обработки заявок на сервере системы

0.3 Мониторинг параметров моделируемой системы

Потребуется палитра Monitoring. Выбираем Break Point (точка останова) и устанавливаем её на переход Start. После этого в разделе меню Monitor появится новый подраздел, который назовём Ostanovka. В этом подразделе необходимо внести изменения в функцию Predicate, которая будет выполняться при запуске монитора. Зададим число шагов, через которое будем останавливать мониторинг. Для этого true заменим на `Queue_Delay.count()=200`. Необходимо определить конструкцию `Queue_Delay.count()`. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Появившийся в меню монитор называем Queue Delay (без подчеркивания). Функция Observer выполняется тогда, когда функция предикатора выдаёт значение true. По умолчанию функция выдаёт 0 или унарный минус (~1), подчёркивание обозначает произвольный аргумент. Изменим её так, чтобы получить значение задержки в очереди. Для этого необходимо из текущего времени `intTime()` вычесть временную метку AT, означающую приход заявки в очередь.

В результате функции примут вид(рис. [0.5]):

```

▼ Monitors
  ▼ Queue Delay
    ▶ Type: Data collection
    ▶ Nodes ordered by pages
    ▶ Predicate
    ▼ Observer
      fun obs (bindelem) =
      let
        fun obsBindElem (Server'Start (1, {job,jobs,proctime})) = (intTime() - (#AT job))
          | obsBindElem _ = ~1
        in
          obsBindElem bindelem
        end
      ▶ Init function
      ▶ Stop
    ▼ Ostanovka
      Type: Break point
      ▶ Nodes ordered by pages
      ▼ Predicate
        fun pred (bindelem) =
        let
          fun predBindElem (Server'Start (1,
            {job,jobs,proctime})) = Queue_Delay.count() = 200
            | predBindElem _ = false
          in
            predBindElem bindelem
          end

```

Рис. 0.5: Функция Predicate монитора Ostanovka и функция Observer монитора Queue Delay

После запуска программы на выполнение в каталоге с кодом программы появится файл Queue_Delay.log, содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время. С помощью gnuplot можно построить график значений задержки в очереди (рис. [0.6]), выбрав по оси x время, а по оси y — значения задержки:

```

#!/usr/bin/gnuplot -persist
# задаём текстовую кодировку,
# тип терминала, тип и размер шрифта
set encoding utf8
set term pdfcairo font "Arial,9"
# задаём выходной файл графика
set out 'qm.pdf'
# задаём стиль линии
set style line 2
plot "Queue_Delay.log" using ($4):($1) with lines

```

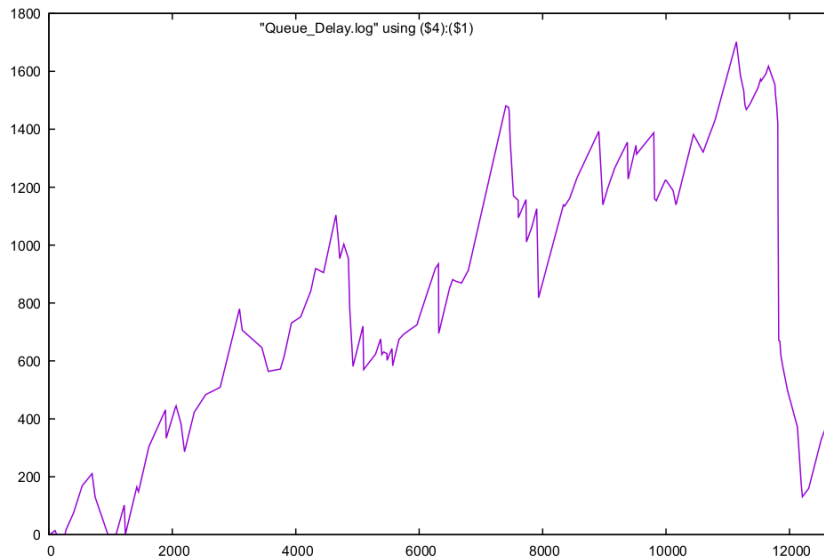


Рис. 0.6: График изменения задержки в очереди

Посчитаем задержку в действительных значениях. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Появившийся в меню монитор называем Queue Delay Real. Функцию Observer изменим следующим образом(рис. [0.7]):

```
Binder 0
System Arrivals Server fun obs <Queue Delay Real> fun obs <Long Delay Time>
let
  fun obs (bindelem) =
    let
      fun obsBindElem (Server'Start (1, {job,jobs,proctime})) = Real.fromInt(intTime()-(#AT job))
        | obsBindElem _ = ~1.0
    in
      obsBindElem bindelem
    end
```

Рис. 0.7: Функция Observer монитора Queue Delay Real

По сравнению с предыдущим описанием функции добавлено преобразование значения функции из целого в действительное, при этом obsBindElem _ принимает значение ~1.0. После запуска программы на выполнение в каталоге с кодом программы появится файл Queue_Delay_Real.log с содержимым, аналогичным содержимому файла Queue_Delay.log, но значения задержки имеют действительный тип(рис. [0.8]):


```

#data counter step time
0.000000 1 3 0
0.000000 2 6 455
3.000000 3 9 514
113.000000 4 14 633
48.000000 5 16 646
164.000000 6 19 786
70.000000 7 21 835
0.000000 8 24 1189
121.000000 9 27 1348
0.000000 10 30 1487
0.000000 11 33 1539
45.000000 12 36 1600
279.000000 13 40 2167
28.000000 14 42 2189
0.000000 15 45 2214
0.000000 16 48 2250
34.000000 17 52 2382
85.000000 18 54 2450
0.000000 19 57 2564
88.000000 20 61 2700
178.000000 21 66 2793
160.000000 22 69 2900
248.000000 23 73 3028
277.000000 24 76 3064
368.000000 25 80 3167
226.000000 26 82 3186
321.000000 27 84 3303
289.000000 28 87 3327
290.000000 29 89 3391
479.000000 30 91 3639
417.000000 31 93 3727
0.000000 32 96 3844
0.000000 33 99 4043

```

Рис. 0.8: Содержимое Queue_Delay_Real.log

Посчитаем, сколько раз задержка превысила заданное значение. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Монитор называем Long Delay Time. Функцию Observer изменим следующим образом(рис. [0.9]):

```

Binder 0
System Arrivals Server fun obs <Queue Delay Real> fun obs <Long Delay Time>
fun obs (bindelem) =
if IntInf.toInt(Queue_Delay.last()) >= (!longdelaytime)
then 1
else 0

```

Рис. 0.9: Функция Observer монитора Long Delay Time

С помощью gnuplot можно построить график (рис. [0.10]), демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200.

```

#!/usr/bin/gnuplot -persist
# задаём текстовую кодировку,
# тип терминала, тип и размер шрифта
set encoding utf8
set term pdfcairo font "Arial,9"
# задаём выходной файл графика
set out 'qm.pdf'
# задаём стиль линии
set style line 2
plot [0:] [0:1.2] "Long_Delay_Time.log" using ($4):($1) with lines

```

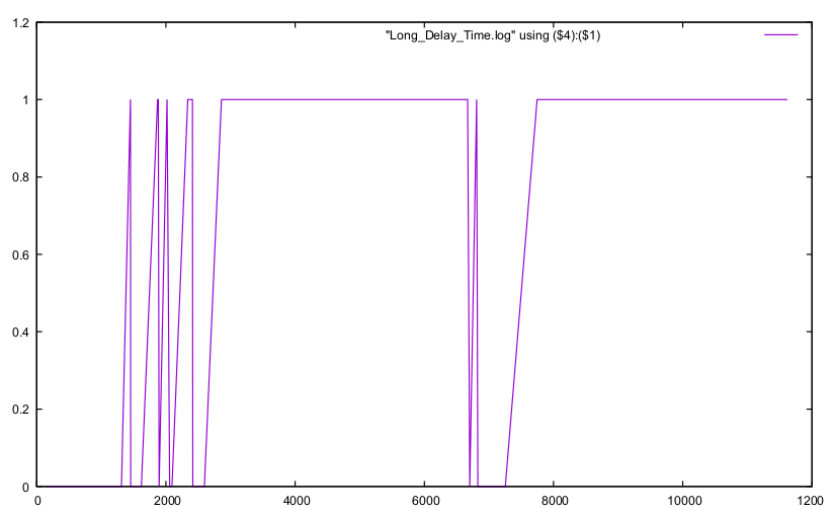


Рис. 0.10: Периоды времени, когда значения задержки в очереди превышали заданное значение

1 Выводы

В результате выполнения работы я реализовал в CPN Tools модель системы массового обслуживания $M|M|1$.