Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

Дисциплина: Системный анализ и машинное моделирование

Отчет

по лабораторной работе № 5-6

«Построение и исследование аналитической и имитационной моделей многоканальной системы массового обслуживания со взаимопощью»

Вариант 6

Выполнила:

Карпач Д. В.

Группа 451001

Проверил:

Мельник Н. И.

Минск 2017

Цель работы:

Построение аналитической и имитационной моделей многоканальной системы массового обслуживания со взаимопощью и расчет параметров данной системы с использованием построенной модели.

Условия:

Многоканальная СМО со “взаимопомощью”

μ

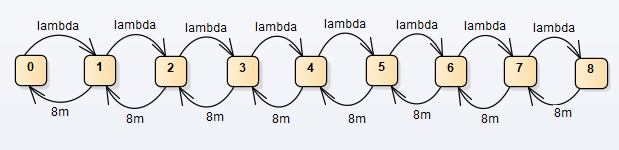
λ

…..

Если в свободную систему поступает заявка, то ее обслуживают совместно все каналы. Если во время обслуживания заявки поступает еще одна, то часть каналов переключается на ее обслуживание и т.д., пока все каналы не окажутся занятыми. Интенсивность совместного обслуживания заявки n каналами nμ. Каналы распределяются равномерно между заявками. На вход поступает простейший поток заявок с интенсивностью λ. Время обслуживания – показательное с параметром μ.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | n | λ | μ | Цель исследования |
| 6 | 8 | 7 | 1 | А, Q |

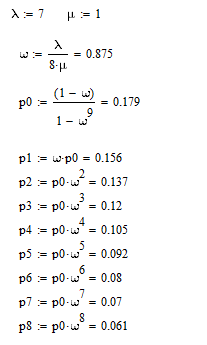
Ход работы:

1. Построение графа состояний
2. Смысл кодировки состояний:

|  |  |
| --- | --- |
| Цифра внутри состояния | Количество заявок, находящихся в СМО |

1. Расчёт теоретических значений показателей эффективности с использованием MathCad.

Исходя из графа состояний, построим СЛАУ для нахождения вероятностей состояний и путём математических преобразований получим следующее:



1. Расчёт величин

* Абсолютная пропускная способность:

A = λ\*(1-p8)

A = 7\*(1-0.061) = 6.573

* Относительная пропускная способность:

Q= (1-p8)

Q = 1-0.061 = 0.939

1. Сравнение теоретических и практических значений.

Результаты работы программы представлены в колонке «Практические значения.

|  |  |  |
| --- | --- | --- |
| Величина | Теоретическое значение | Практическое значение |
| p0 | 0.179 | 0.179 |
| p1 | 0.156 | 0.156 |
| p2 | 0.137 | 0.137 |
| p3 | 0.12 | 0.12 |
| p4 | 0.105 | 0.105 |
| p5 | 0.092 | 0.092 |
| p6 | 0.08 | 0.08 |
| p7 | 0.07 | 0.07 |
| p8 | 0.061 | 0.061 |
| A | 6.573 | 6.572 |
| Q | 0.939 | 0.939 |

1. Код программы

const lambda = 7;

const mu = 1;

const time = 1000000;

// Generates exponentially distributed numbers using param.

const getNextRandom = (param) => -1/param \* Math.log(Math.random());

// Helps to map channels over the invoices.

const channelsDistribution = [

[], // This element is added just for convenient indexing.

[ 8 ],

[ 4, 4 ],

[ 3, 3, 2 ],

[ 2, 2, 2, 2 ],

[ 2, 2, 2, 1, 1 ],

[ 2, 2, 1, 1, 1, 1 ],

[ 2, 1, 1, 1, 1, 1, 1 ],

[ 1, 1, 1, 1, 1, 1, 1, 1 ],

];

const EventType = {

Created: 'created',

Processed: 'processed',

};

let events = [

// Initialize events with first created invoice.

{

type: EventType.Created,

time: getNextRandom(lambda),

},

];

let invoices = [];

const getNearestEvent = (events) => {

let min = Infinity;

let index = -1;

events.forEach((event, i) => {

if (event.time < min) {

min = event.time;

index = i;

}

});

return events[index];

};

const updateEvents = (choosenEvent) => {

// Remove choosen event.

events = events.filter((event) => event !== choosenEvent);

// And change time of all remain events.

events.forEach((event) => {

event.time -= choosenEvent.time;

});

}

let currentTime = 0;

let states = [

0, // S0

0, // S1

0, // S2

0, // S3

0, // S4

0, // S5

0, // S6

0, // S7

0, // S8

];

let processed = 0;

let total = 0;

while (currentTime < time) {

const nearestEvent = getNearestEvent(events);

updateEvents(nearestEvent);

currentTime += nearestEvent.time;

states[invoices.length] += nearestEvent.time;

switch (nearestEvent.type) {

case EventType.Created: {

total += 1;

const invoiceChannelMap = channelsDistribution[invoices.length + 1];

// Don't forget to add event on creation of the next invoice.

events.push({

type: EventType.Created,

time: getNextRandom(lambda),

});

if (!invoiceChannelMap) {

// All channels are processing invoices.

break;

}

// Redistribute channels.

invoices.forEach((invoice, index) => {

const {

event,

channels,

} = invoice;

const newChannelsCount = invoiceChannelMap[index];

// Reverse proportion, because more channels is better:

// 0.3s -- 8channels

// x -- 4channels

// -----------------

// current event time -- channels

// new event time -- new channels count

// -----------------

// x = 0.3s \* 8/4 = 0.6s

event.time \*= channels/newChannelsCount;

invoice.channels = newChannelsCount;

});

// Get channels count for new invoice.

const channels = invoiceChannelMap[invoiceChannelMap.length - 1];

// Add new event.

const event = {

type: EventType.Processed,

time: getNextRandom(channels \* mu),

};

events.push(event);

// Add new invoice.

const invoice = {

event,

channels,

};

invoices.push(invoice);

// Also store reference to invoice to be able to easily remove it from invoices array when needed.

event.invoice = invoice;

break;

}

case EventType.Processed: {

processed += 1;

const { invoice } = nearestEvent;

// Remove processed invoice from invoices array.

invoices = invoices.filter((i) => i !== invoice);

const invoiceChannelMap = channelsDistribution[invoices.length];

// Redistribute channels.

invoices.forEach((invoice, index) => {

const {

event,

channels,

} = invoice;

const newChannelsCount = invoiceChannelMap[index];

// Reverse proportion, because more channels is better:

// 0.3s -- 8channels

// x -- 4channels

// -----------------

// current event time -- channels

// new event time -- new channels count

// -----------------

// x = 0.3s \* 8/4 = 0.6s

event.time \*= channels/newChannelsCount;

invoice.channels = newChannelsCount;

});

break;

}

}

}

// Normalize values using currentTime, because currentTime a little bit bigger than time.

states = states.map((state) => state / currentTime);

console.log(states);

console.log('Q: ', processed / total);

console.log('A: ', processed / currentTime);