УДК 004.942

М. О. Доброхвалов, П. В. Корытов, С. И. Степанова,
А. А. Тарасова, Ар. Ю. Филатов
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Анализ подходов к моделированию систем массового обслуживания

Теория массового обслуживания имеет широкое практическое применение, поэтому оптимизация работы систем массового обслуживания (СМО) играет значительную роль в современном мире.
Для моделирования и анализа характеристик очереди СМО можно использовать среду имитационного моделирования GPSS. Ее недостатки – закрытый исходный код, требования совместимости системы со средой, отсутствие API и др., ведут к проблемам ее применения при разработке собственных программ с использованием СМО.
Предложено три решения, позволяющих моделировать одноканальную СМО. Они лишены перечисленных недостатков, но требуют большего времени для проведения моделирования. Два решения (реализации на GNU Octave и на Python), исходя из приведенных в работе графиков, удобны для моделирования и анализа СМО, так как время их работы не зависит ни от среднего значения распределения, ни от числа поступающих в работу заявок.
Третье решение, для которого описаны две реализации на Python, в обоих случаях показывает линейную зависимость времени выполнения за счет обработки заявок в реальном времени. Такой подход позволяет получать практическое время обработки заявок без учета затрат на их создание и удаление, что повышает качество сравнения характеристик очереди с теоретическими. В то же время, такое решение может быть внедрено в существующие СМО реального времени с целью получения характеристик очереди.
Также показано, что практические значения, полученные в результате работы программ, сходятся к теоретическим.

СМО, система массового обслуживания, GPSS, очередь, имитационное моделирование

Система массового обслуживания (СМО) [1] – это система, которая производит обработку поступающих в нее заявок. Концепция системы массового обслуживания прослеживается во многих сферах нашей жизни: кафе и рестораны, колл-центры, электронная очередь и т. д. В общем виде модель системы массового обслуживания можно представить, как показано на рис. 1.

Источник заявок И генерирует заявки с некоторой интенсивностью λ (среднее количество событий потока, происходящих в единицу времени). Заявки поступают в очередь *Q*. Очередь может быть как бесконечной, так и конечной. Если очередь конечна и заполнена, то при поступающая заявка отбрасывается. Если при поступлении заявки в очередь обработчик О свободен, то происходит процесс обработки. После обслуживания заявки обработчик берет из очереди следующую, если такая есть, иначе – переходит в состояние ожидания. Интенсивность потока обслуживания (количество заявок, которое обслуживает обработчик в единицу времени) обозначается μ.

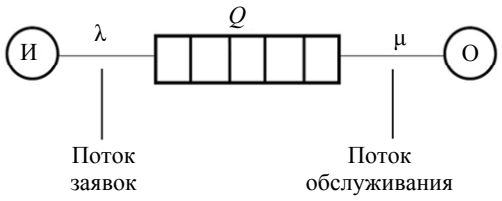

Рис. 1

Цель исследований теории массового обслуживания заключается в рациональном выборе структуры системы и процесса обслуживания на основе изучения потоков заявок, поступающих в систему и выходящих из нее, длительности ожидания и длины очередей. Грамотный выбор параметров системы может помочь в сокращении расходов ресурсов и повышении скорости обслуживания. СМО имеют прикладной характер, поэтому они включены в некоторые программы вузов. В качестве одного из примеров исследования подобных задач можно указать систему моделирования общего назначения GPSS (General Purpose Simulation System) [2].

Цель статьи состоит в создании приложения с открытым кодом для отладки системы массового обслуживания. Код может быть встроен в существующую СМО, что позволит моделировать потоки заявок и обслуживания, а также предоставит возможность отладки систем с очередью. Данные приложения могут использоваться и для обучения основам моделирования СМО благодаря их открытости.

Существующее решение, которое может быть использовано для отладки СМО, – среда GPSS. Основные проблемы в ее использовании – закрытый код и отсутствие какого-либо API, что ограничивает возможность интеграции с другими программами. Например, это означает, что GPSS нельзя встроить в существующую систему.

Другие сложности – необходимость совместимости компьютеров со средой [3], трудоемкость обучения специализированному языку, высокие требования к теории [4], отсутствие визуализации моделируемых процессов во многих версиях [5].

В данной работе изучаются программные реализации систем массового обслуживания. Разработанные авторами алгоритмы расчета их качественных и количественных характеристик, рассматриваемые в качестве предмета исследования, сравниваются по скорости выполнения.

Статья включает в себя: обзор существующих решений; обзор разработанных решений; их сравнение и анализ, где демонстрируются преимущества и недостатки различных подходов и решений; заключение и выводы по выполненной работе.

Обзор существующих решений. Существуют различные реализации систем массового обслуживания, но основным конкурентом предлагаемых подходов является GPSS – язык и среда имитационного моделирования систем общего назначения.

Полезность GPSS обусловлена тем, что языки программирования общего назначения (например, FOTRAN, Pascal, доступные в годы создания GPSS) сложно применять для моделирования систем. Системы DYNAMO, CSMP направлены на моделирование систем с непрерывными величинами, но плохо приспособлены для дискретных моделей. GPSS, напротив, является средством дискретного моделирования систем, поэтому может применяться в том числе для исследования систем массового обслуживания.

Использование специализированного языка упрощает процесс моделирования. В частности, GPSS предоставляет конструкции для моделирования сущностей системы (вроде центров обслуживания) и их атрибутов. С использованием этих структур можно наблюдать изменение параметров модели во время симуляции [2].

Доступно множество различных версий GPSS, но все они поставляются на коммерческой основе [5]. В данной статье рассматривается версия GPSS World [6] как наиболее приспособленная для исследования.

Обзор собственных решений. У всех алгоритмов одинаковые входные параметры:

– распределение интервалов следования заявок (*X_in*);

– распределение интервалов обслуживания заявок (*X_out*);

– максимальная длина очереди (*X*_max).

Интервалы времени *X_in* и *X_out* представляют собой случайные величины, распределенные по некоторому закону. Параметры распределения задаются пользователем через переменные и затем передаются на вход функциям.

К главным достоинствам представленных далее решений относится возможность встраивания их в существующие системы. Также код программ понятен как новичкам, так и профессионалам в данной области, что позволяет быстрее получить первые результаты.

Решение 1. Моделируется состояние очереди (*queue*) в момент времени *time*. В один момент времени могут произойти следующие события:

1) прекращение обработки заявки (STOP);

2) поступление новой заявки (PUSH).

Во втором случае заявка записывается в очередь (*queue*), если есть там есть место; в очередь событий (*events*) записывается следующее событие поступления заявки.

Если очередь событий пуста, но остались необработанные значения *X_in*, также записывается следующее событие поступления заявки. Если очередь не пуста и ни одна заявка не обрабатывается, заявка берется из очереди и начинается ее обработка.

В каждый момент времени сохраняется состояние очереди. После завершения обработки событий в момент *time* время ставится как ближайшее из очереди событий (*events*).

Алгоритм 1 моделирования СМО представлен на рис. 2.

Алгоритм 1.

events $\neq \emptyset$ **or** $X\_in \neq \emptyset$ **forall** *event в момент time* **do**
  if *event.type* $==$ *STOP* **then**
    Записать окончание обработки заявки;
  else if *event.type* $==$ *PUSH* **then**
    Записать поступление новой заявки или отказ (при переполнении);
    if $X\_in \neq \emptyset$ **then**
      Добавить в *events* следующее поступление заявки
if *events* $\neq \emptyset$ ***and*** $X\_in \neq \emptyset$ **then**
  Добавить в *events* следующее поступление заявки;
if *Ни одна заявка не обрабатывается* **then**
  Записать начало обработки заявки;
  Добавить в *events* окончание обработки заявки

Рис. 2

Алгоритм 2.

`append_proccess()` Создаем элемент очереди *obj*;
Добавляем *obj* в очередь *queue* или отказ при переполнении;
if $X\_in \neq \emptyset$ **then**
  wait_interval $\leftarrow$ *X_in.popleft()*;
  ожидание в течение *wait_interval*;
  вызов метода `append_proccess`
Function `remove_proccess():`
  if *queue* $\neq \emptyset$ **then**
    удаляем элемент из очереди *queue*;
    proccessing_interval $\leftarrow$ *X_out.popleft()*;
    обработка в течение *proccessing_interval*;
    вызов метода `remove_proccess`
  else if $X\_in \neq \emptyset$ **then**
    ожидание в течение *check_queue_interval*;
    вызов метода `remove_proccess`
Function `main():`
  Инициализация *X_in*,*X_out*, *check_queue_interval*;
  параллельный вызов `append_proccess`, `remove_proccess`

Рис. 3

Преимущество данного подхода – инвариантность относительно значений интервалов, т. е. алгоритм не начнет работать медленнее, если увеличатся значения интервалов следования заявок или обслуживания.

Сохранение состояния очереди на каждом этапе позволяет моделировать процесс графически.

Решение 2. Идея алгоритма заключается в следующем: имеется очередь, в которую один поток отправляет заявки через заданные временные интервалы, а второй параллельно ему обрабатывает эти заявки (время обработки также задается).

Заявка попадает в очередь только в том случае, если в очереди есть место, а иначе – отклоняется. Все поступившие в очередь заявки обрабатываются в обязательном порядке.

Система работает в реальном времени, поэтому ее удобно встраивать в уже существующие СМО с целью получения их характеристик. Недостаток такого подхода состоит в том, что в процессе анализа алгоритма реальное время ожидания приводит к длительному вычислению характеристик при отсутствии полезной работы.

Для данного решения приведено две реализации на языке Python (рис. 3).

Реализация 1. Дополнительным параметром данной реализации является *check_queue_interval* – интервал проверки наличия заявки в очереди на обработку.

Первый процесс добавляет элемент в очередь, если в очереди есть место. После этого из списка интервалов поступления заявок извлекается промежуток времени *append_wait_time*. Процесс ждет *append_wait_time*, после чего снова повторяет свой цикл, если список интервалов поступления заявок не пуст, иначе – заканчивает свою работу.

Второй процесс извлекает элемент из очереди, если она не пуста. Если очередь пуста, а также пуст список интервалов поступления заявок, то все возможные заявки обработаны, и процесс завершает свою работу. Если указанное условие не выполняется, то из списка интервалов обслуживания *X_out* извлекается элемент и происходит его обработка в течение заданного промежутка времени. Если после ожидания в очереди есть заявка на обслуживание, то начинается ее обработка, иначе – происходит повторная проверка через промежуток времени *check_queue_interval*.

Таким образом, алгоритм способен выдавать характеристики СМО в любой момент времени.

Реализация 2. Поступающие на вход величины *X_in* используются только потоком добавления заявок, а *X_out* – только обработчиком. Единственная область памяти, к которой обращаются оба потока, – это блокирующая очередь.

Первый поток работает полностью аналогично реализации 1, однако второй поток переходит в блокирующее ожидание, если очередь пуста. В таком случае следующая заявка будет обработана без очереди. Обработка заявки длится заданное в *X_out* время. Если первый поток, добавляющий новые заявки, завершен, очередь пуста и есть хотя бы одна обработанная заявка (что обходит ситуацию, когда первый поток начинает работу позже второго), обработчик завершает работу.

Механизм исключений в данной реализации помогает при расчете таких характеристик, как число заявок, обработанных без очереди (исключение QueueEmpty), и число отклоненных заявок (QueueFull); ожидание потока-обработчика – пассивное.

Данный подход также позволяет получать информацию о состоянии очереди в любой период времени и прост в добавлении к существующим СМО реального времени для их отладки и анализа.

Решение 3. Решение реализовано с помощью пакета GNU Octave с открытым исходным кодом. Данный выбор был сделан в силу того, что в Octave реализованы функции для генерации случайных величин с заданным распределением и различные математические функции, а также можно выполнять программу по шагам и легко отслеживать значения в переменных и их изменение.

Далее на основе массива с интервалами времени, через которые заявки подаются на вход, формируется новый массив – со временем поступления заявки не относительно предыдущей, а относительно начала работы СМО. Данная процедура необходима для того, чтобы определять, какие заявки пришли за время обработки.

Общая схема работы системы массового обслуживания может быть описана следующим образом: заявки поступают на вход и формируют очередь, а система по очереди обрабатывает их.

Алгоритм работы программы, описанный в виде псевдокода на рис. 4, представляет собой цикл, условием выхода из которого служит совпадение количества обработанных заявок *processed* или, в случае с ограниченной очередью, обработанных и отвергнутых (*rejected*), с количеством заявок, поступивших на вход *N*. За ограниченность очереди отвечает флаг *limited_ queue*, а за готовность обработчика принять новую заявку – флаг *is_open*.

При выполнении данной программы в GNU Octave по шагам можно увидеть, как обрабатыва-

Алгоритм 3.

```
processed + rejected ≠ N if is_open==True then
    if query ≠ ∅ then
        В обработку поступает первая заявка из очереди;
    else
        В обработку поступает первая заявка из потока;
    proccess++;
    is_open=False;
else
    forall заявки, пришедшие за время обработки do
        if not limited_queue or в очереди есть место then
            Добавить заявку в очередь;
        else
            rejected++;
    is_open=True;
```

Рис. 4

ется та или иная заявка и как изменяются при этом основные показатели СМО. Время работы программы не зависит от среднего времени обработки заявки или от средних интервалов времени поступления заявок на вход.

Анализ. Для сравнения решений, приведенных в разделе «Обзор собственных решений», были выбраны 4 критерия, и для всех реализаций была оценена их выполнимость. Полученные результаты сведены в таблицу.

Критерии сравнения разработанных алгоритмов. Для сравнения описанных алгоритмов и их программных реализаций использованы следующие критерии.

1. Открытость. Цель разработки состоит в том, чтобы созданное решение, аналогичное GPSS, имело открытый код, который может быть встроен в уже существующую систему для обработки очереди. Чем понятнее будет написан код и чем богаче API, тем удобнее будет использование решения в стороннем приложении.

2. Время выполнения / Время моделирования. Для отладки требуется многократно запускать систему, поэтому сокращение времени выполнения будет достоинством приведенного решения. Дополнительным плюсом будет случай, в котором время моделирования не зависит от входных и выходных временных промежутков.

Однако может быть необходимо показать, как ведет себя очередь в зависимости от времени. В таком случае система должна либо обладать отдельным функционалом симуляции времени, либо выполняться в реальном времени. Систему, выполняющуюся в реальном времени, можно интегрировать с реально существующей СМО реального времени, чтобы показать поведение последней на основе модели.

3. Просмотр состояния очереди в конкретный момент времени. Помимо конечных результатов, характеристик моделируемой СМО, важны также промежуточные результаты, в частности просмотр состояния очереди в конкретный момент времени. Данный функционал может помочь в отладке системы, а также при использовании программы в образовательных целях – это позволит лучше понять схему работы.

4. Объем ресурсов (использование CPU, RAM). Данная характеристика отвечает за скорость работы программы в зависимости от количества поступающих на вход заявок, т. е. за то, какое количество операций в среднем уходит на обработку одной заявки. На ее основе формируются системные требования для запуска программы.

Сравнение результатов. Для сравнения решений по численным критериям проведены запуски со следующими параметрами:

1. Измерение времени выполнения t в зависимости от длины очереди N

$$N = [1000,\ 2000,\ ...,\ 100\ 000];\ \ \rho = 0.75,$$

где

$$\rho = \frac{\lambda}{\mu}$$

– приведенная интенсивность, λ – интенсивность потока заявок, μ – интенсивность потока обслуживания [1].

2. Измерение времени выполнения в зависимости от значений распределения

$$N = 5000;\ \ \rho = 0.75;\ \ \lambda = \left[10^{-5}, 10^{-4.9}, ..., 10^{5}\right].$$

Для обоих потоков распределение экспоненциальное.

Выбор в пользу экспоненциального распределения и простейшего (пуассоновского) потока обусловлен свойствами стационарности, однородности и отсутствием последействия [7]. Пуассоновский поток может быть хорошим приближением реальных потоков различного рода [8].

Использование решений на основе [9] и [10] универсально и не предоставляет готовых библиотек с математическими методами. Решения на основе продуктов с открытым исходным кодом (Python и GNU Octave) предоставляют возможности для измерения времени выполнения и интерфейсы для взаимодействия с другими программами (CLI). GPSS не предоставляет ни того, ни другого, поэтому время выполнения измеряется «снаружи» и включает в себя накладные расходы на запуск и закрытие GUI.

Поскольку время на открытие и закрытие GUI подвержено существенной дисперсии, для каждого набора параметров взято минимальное значение по 5 запускам [11]. Также из измерений вычтено минимальное время запуска GPSS на программе с очередью длиной 1.

Таким образом, для GPSS можно оценить характер исследуемых зависимостей; время выполнения в абсолютных числах, скорее всего, неточно.

Результаты первого запуска – зависимость времени выполнения от количества поступающих в обработку заявок – представлены на рис. 5. На рис. 5–8 обозначены: *1* – Python, Discrete Time; *2*/*1* – Python, Real Time; *2*/*2* – Python, Real Time; *3* – Octave, Discrete Time.

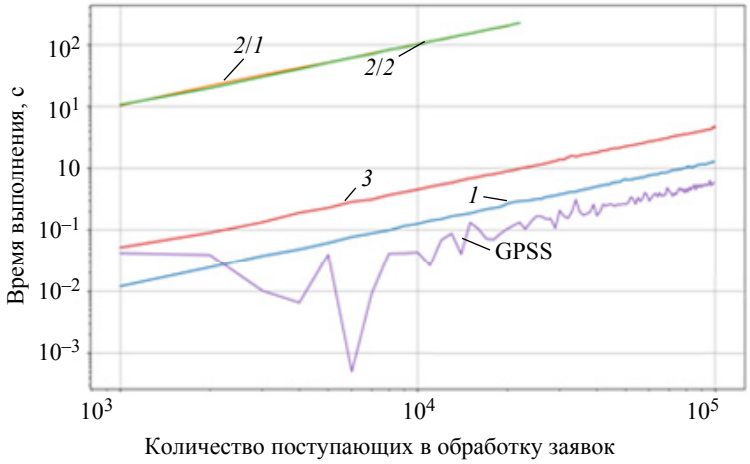

Рис. 5

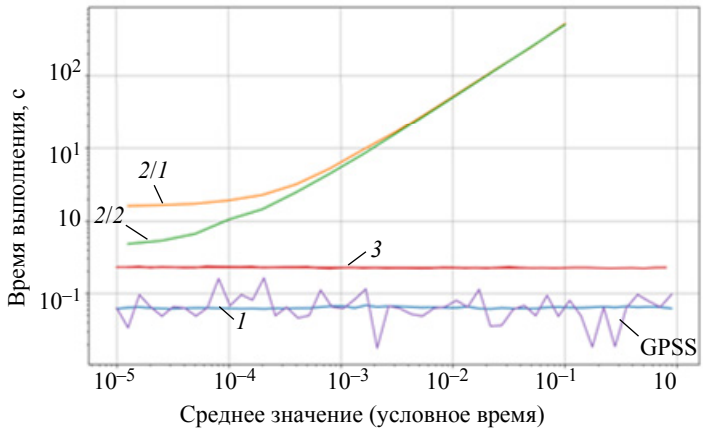

Рис. 6

Как можно заметить, во всех случаях наблюдается линейная зависимость времени выполнения от длины очереди, хотя решения *2/1* и *2/2* существенно медленнее. Решение на Octave оказалось медленнее решения на Python; GPSS быстрее всего, если не учитывать работу с GUI.

Зависимость времени выполнения от количества поступающих в обработку заявок (среднее значение) представлена на рис. 6.

Из рисунка видно, что решения *2/1* и *2/2*, проводящие моделирование в реальном времени, показывают рост времени выполнения при увеличении среднего значения распределения. Для GPSS, решений *1* и *3* такой зависимости не наблюдается.

Для сравнения решений соответствия теоретическим значениям среднего числа заявок *QA* и среднего времени ожидания заявки в очереди *QT* проведены запуски со следующими параметрами:

$$N = [1000, 2000, ..., 100\,000]; \quad \lambda = 100; \quad \rho = 0.75;$$

$$QA = \frac{\rho^2\left(1+\vartheta^2\right)}{2\left(1-\rho\right)};$$

$$QT = \frac{\rho^2\left(1+\vartheta^2\right)}{2\lambda\left(1-\rho\right)};$$

$$\vartheta = \frac{\sigma_t}{t_{\text{об}}},$$

где σ_t – корень дисперсии времени обслуживания; $t_{\text{об}}$ – среднее время обслуживания.

В рамках проведенного эксперимента коэффициент вариации времени обслуживания [1] $\vartheta = 1$. Таким образом, теоретические значения среднего числа заявок *QA* и среднего времени ожидания заявки в очереди *QT* равны 2.25 и 0.0225 соответственно.

Зависимости среднего числа заявок и среднего времени ожидания заявки в очереди от количества поступающих в обработку заявок представлены на рис. 7, 8.

Из этих рисунков можно сделать вывод, что при увеличении общего количества заявок все предложенные решения, как и GPSS, сходятся к теоретическому значению. У решений *2/1* и *2/2* имеется больший разброс практических значений, что связано с моделированием в реальном времени.

Рис. 7

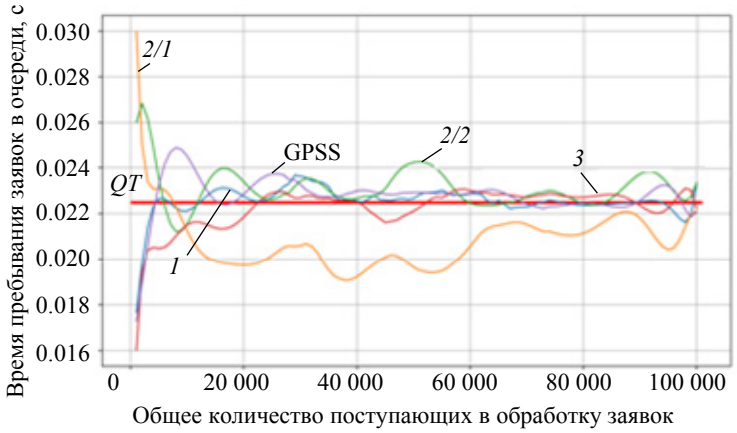

Рис. 8

Критерии	GPSS	Решение *1*	Решение *2/1*	Решение *2/2*	Решение *3*
Зависимость времени выполнения от длины очереди	линейная	линейная	линейная	Линейная	линейная
Зависимость времени выполнения от среднего значения распределения	отсутствует	отсутствует	линейная	линейная	отсутствует
Открытость	–	+	+	+	+
Просмотр состояния очереди в конкретный момент времени	+/–	+	–/+	–/+	+
Требуемый объем RAM, Мбайт	~4	~250	~550	~50	~100
Использование CPU (одно ядро), %	100	100	~30	~5	100

Общие результаты по количественным и качественным критериям представлены в таблице.

В ней содержатся информативные результаты, анализ которых представлен далее.

Анализ. В ходе сравнения программных реализаций СМО были получены следующие результаты.

Во-первых, код программы на GPSS недостаточно прозрачен для тех, кто только приступает к изучению систем массового обслуживания, что может стать причиной больших затрат времени на понимание принципа работы программы и ее запуск. Отладка очереди с использованием разработанных решений лишена этого недостатка – дополнительных знаний о способе моделирования СМО не требуется, для запуска достаточно одной строки кода.

Во-вторых, разработанные решения *1* и *3* позволяют просматривать состояние очереди в конкретный момент времени, в то время как программа на GPSS и решения *2/1* и *2/2* позволяют делать это только при запуске на части данных – на заявках, пришедших до необходимого момента времени. Такой способ крайне неудобен для анализа работы программы и отладки очереди, так как требует повторных запусков. При этом решения *1* и *3* позволяют получать характеристики смоделированной СМО за меньшее время, чем решения *2/1* и *2/2*. Последние моделируют работу

СМО в реальном времени, и на многократные запуски уйдет значительное время. Особенности этих решений позволяют найти реальное время пребывания заявки в системе без учета затрат на ее создание и удаление. Сравнение полученных данных с теорией будет более информативным, чем для решений *1* и *3*.

В-третьих, зависимость времени выполнения от длины очереди для всех случаев линейная. Это говорит о том, что при увеличении количества заявок, поступающих на вход, в каждом случае увеличивается и время работы программы. Однако в силу специфики реализации решений *2*/*1* и *2*/*2* их время выполнения более чем в 100 раз больше, чем для всех остальных решений. Тем не менее, при добавлении этих решений в существующие СМО реального времени получение характеристик очереди не будет приводить к значительным накладным расходам. Программа на GPSS работает быстрее остальных решений, но необходимо принять во внимание, что при измерении времени работы для нее был применен ряд допущений – взято минимальное значение по 5 запускам и вычтено минимальное время запуска GPSS на программе с очередью длиной 1. Поэтому в случае необходимости быстрого получения результатов лучше использовать решение *1* или программу на GPSS.

В-четвертых, у решений *2*/*1* и *2*/*2* наблюдается линейная зависимость времени выполнения от среднего значения распределения. Это означает, что при увеличении среднего значения времени, через которое поступают заявки на вход, и/или при увеличении среднего времени обработки заявки системой время работы также линейно возрастает. Данная ситуация возникает, как уже было ранее сказано, из-за симуляции работы СМО в реальном времени. Поэтому решения *2*/*1* и *2*/*2* следует использовать, только если средние значения входного и выходного распределений достаточно малы, иначе симуляция работы СМО затянется относительно надолго.

В-пятых, решения *2*/*1* и *2*/*2*, моделируя в реальном времени, потребляют меньше мощности CPU. Остальные решения использовали одно ядро CPU на 100 %. GPSS потребляет существенно меньшее количество оперативной памяти, чем решения на Octave и Python. Из разработанных решений наименьшее потребление ресурсов у решений *2/2* и *3*. Следовательно, GPSS и решения *2/2*, *3* наиболее пригодны для использования в условиях ограниченных ресурсов.

В данной статье были рассмотрены четыре программные реализации систем массового обслуживания.

Существующее решение – GPSS – позволяет с помощью небольшого объема кода и системных ресурсов реализовать достаточно быстро работающую симуляцию работы СМО, но закрытый исходный код, множество версий, отсутствие API и, как следствие, ограниченная возможность использования в собственных разработках СМО, представляют его существенные недостатки.

От них свободны предложенные решения. Одно из них написано на Octave и три – на Python, что позволяет встраивать их открытый исходный код в различные системы.

Решения *1* и *3* следует использовать при отладке систем, так как при их работе можно с легкостью отследить состояние очереди в конкретный момент времени. Также преимущество данных решений заключается в скорости работы в силу отсутствия зависимости времени работы от среднего значения входного и выходного распределений. Однако скорость моделирования решения *1* все же выше, чем решения *3*, но решение *1* уступает решению *3* по объему системных ресурсов, необходимых для работы программы.

Преимуществом решений *2*/*1* и *2*/*2* служит простота их внедрения в систему массового обслуживания, работающую в реальном времени. Их также можно использовать в случае, если среднее время, через которое заявки поступают на вход, и среднее время их обработки системой невелики и скорость работы не важна.

Потребление ресурсов у разработанных решений на языках общего применения существенно больше, чем у GPSS.

Реализация разработанных алгоритмов представлена на GitHub [12], тестирование осуществлялось с использованием методов в [13].

СПИСОК ЛИТЕРАТУРЫ

1. Романцев В. В., Филатов Ар. Ю. Анализ, моделирование и оптимизация систем: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2017.

2. Karian Z. A., Dudewicz E. J. Modern statistical systems and GPSS simulation. Florida: CRC Press, 1998.

3. An application of FONWebGPSS in teaching simulation / M. Despotović-Zrakić, B. R. Jovanić, B. L. Radenković, Z. Bogdanović // 18th Telecommunications forum TELFOR 2010. TEFLOR. 2010. P. 1145–1148. URL: https://www.researchgate.net/publication/266631696_A_New_Approach_for_Teaching_Discrete_Event_Simulation_via_Web (дата обращения 11.12.20).

4. Schriber T. J. Perspectives on simulation using GPSS // Proc. of the 21st Conf. on Winter Simulation / Association for Computing Machinery. New York, 1989. P. 115–128.

5. GPSS 50 years old, but still young / I. Ståhl, R. G. Born, J. O. Henriksen, H. Herper // Proc. of the 2011 Winter Simulation Conf. (WSC). Phoenix Arizona, 2011. P. 3947–3957.

6. GPSS World. URL: http://www.minutemansoftware.com/simulation.htm (дата обращения 21.02.2021).

7. Алиев Т. И. Основы моделирования дискретных систем / СПбГУ ИТМО. СПб., 2009.

8. Green L. V., Kolesar P. J., Whitt W. Coping with time-varying demand when setting staffing requirements for a service system // Production and Operations Management. 2007. Vol. 16, № 1. P. 13–39.

9. Беляев С. А., Матросов В. В. Опыт создания среды имитационного моделирования // I-methods. 2018. Т. 10, № 3. С. 14–22.

10. Беляев С. А., Черепкова Ю. С. Архитектура среды моделирования для проведения экспериментов с интеллектуальными агентами // Программные продукты, системы и алгоритмы. 2017. № 3. С. 4.

11. Timeit – Measure execution time of small code snippets. URL: https://docs.python.org/3/library/timeit.html (дата обращения 14.12.20).

12. Исходный код разработанных алгоритмов // Github. URL: https://github.com/QueuingSystems/QueuingSystemsImplementations (дата обращения 22.02.2021).

13. Черная О. С., Федорова Ю. Ю., Беляев С. А. Применение методов и средств автоматизированного тестирования для проверки качества программных комплексов обработки измерительной информации // Изв. СПбГЭТУ «ЛЭТИ». 2013. № 9. С. 55–58.

M. O. Dobrokhvalov, P. V. Korytov, S. I. Stepanova, A. A. Tarasova, Ar. Yu. Filatov
Saint Petersburg Electrotechnical University

ANALYSIS OF APPROACHES TO MODELING QUEUING SYSTEMS

The queuing theory has a wide practical application, therefore, the optimization of the queuing systems (QS) has a significant role in the modern world.

To simulate and analyze the characteristics of the queue on the GPSS simulation environment can be used. Its shortcomings, such as closed source code, requirements for system compatibility with the environment, lack of API and others, lead to problems of its application when developing your own programs using QS.

Three solutions have been proposed that make it possible to simulate a single-channel QS; they are devoid of the listed disadvantages, but they require more time for modeling. Two solutions (implementations in GNU Octave and in Python), based on the charts given in the work, are convenient for modeling and analyzing QS, since their operation time does not depend on the mean value of the distribution, or on the number of applications coming into operation.

The third solution, for which two Python implementations are described, in both cases shows a linear dependence of the execution time due to the processing of requests in real time. This approach allows us to obtain a practical time for processing requests without taking into account the costs of creating and deleting them, which improves the quality of comparison of the characteristics of the queue with the theoretical ones. At the same time, such a solution can be implemented into existing real-time QS in order to obtain the characteristics of the queue.

It is also shown that the practical values obtained as a result of the work of the programs converge to the theoretical ones.

QS, queuing system, GPSS, queue, simulation