Нестационарная параллельно-последовательная модель высоконагруженной системы мониторинга

Non-stationary parallel-serial model of a high-load monitoring system

Шардаков / Shardakov K.
Кирилл Сергеевич
(k.shardakov@gmail.com)
ФГБОУ ВО «Петербургский государственный университет путей сообщения императора Александра I» (ПГУПС), аспирант кафедры «Информационные и вычислительные системы».
г. Санкт-Петербург

Бубнов / Bubnov V.
Владимир Петрович
(bubnov1950@yandex.ru)
доктор технических наук, профессор.
ПГУПС, профессор кафедры «Информационные и вычислительные системы».
г. Санкт-Петербург

Ключевые слова: алгоритм генерации матрицы коэффициентов – coefficient matrix generation algorithm; рекурсивный – recursive; Zabbix; ОДУ – ode; нестационарная система массового обслуживания – non-stationary queueing system; система мониторинга – monitoring system; численно-аналитический метод – numerical-analytical method.

В статье описана параллельно-последовательная модель нестационарной системы обслуживания, моделирующая работу системы мониторинга Zabbix на основе распределенной архитектуры. Приведен рекурсивный алгоритм с группировкой, предназначенный для формирования списка состояний такой системы и его блок-схема. Выведены правила переходов между состояниями и представлена диаграмма переходов. Рассмотрен алгоритм формирования матрицы коэффициентов такой модели для системы обыкновенных дифференциальных уравнений Чепмена-Колмогорова и его блок-схема. Описаны особенности программной реализации предложенного алгоритма.

The article describes a parallel-serial model of a non-stationary queueing system that simulates the operation of a Zabbix monitoring system based on a distributed architecture. A recursive algorithm with grouping is presented, algorithm intended to form a list of states of such a system and its flowchart. The rules for transitions between states are derived and a transition diagram is presented. An algorithm for forming the matrix of coefficients of such a model for the system of ordinary differential equations of Chapman-Kolmogorov and its flowchart are considered. The features of the software implementation of the proposed algorithm are described.

Введение

Автоматизированные системы мониторинга являются важной частью любой информационной системы. Мониторинг – это непрерывный процесс наблюдения и регистрации параметров объекта, их обработка, сравнение с пороговыми значениями. Эта система мониторинга должна справляться с возрастающей нагрузкой. Наиболее популярной и легко масштабируемой для адаптации к нагрузке свободно распространяемой системой мониторинга для информационных систем является *Zabbix* [1]. Многие авторов используют для исследования вероятностно-временных характеристик процессов, протекающих в информационных системах, стационарные модели теории массового обслуживания [2–4]. Однако наибольший интерес представляют модели нестационарных систем обслуживания (НСО). Они позволяют исследовать информационные системы в условиях критических (пиковых) нагрузок. Примерами работ, посвященных нестационарному режиму, являются [5–8]. Недостатком работ [5–8] является то, что в них рассматривается только классический численный метод решения систем обыкновенных дифференциальных уравнений (ОДУ) – метод Рунге-Кутты. Это накладывало значительное ограничение на размерность рассматриваемых моделей НСО. В работах [9–11] представлен численно-аналитический метод, скорость и точность которого при решении системы ОДУ, описывающей НСО, превосходят наиболее распространенный при решении данного рода задач метод Рунге-Кутты. Преимуществом этого метода также является рекурсивный алгоритм генерации матрицы коэффициентов системы ОДУ без построения графа состояний и переходов НСО и выведения общего уравнения системы ОДУ как в работах [5–8]. Но этот алгоритм также имеет недостатки, описанные в [12]. Также в [12] предложен последовательный метод генерации матрицы коэффициентов, лишенный недостатков рекурсивного метода. В данной статье рассмотрена ранее не представленная параллельно-последовательная модель НСО. Для расчета модели был использован описанный в [9] численно-аналитический метод, реализованный на *Python*3, но модифицированный в части формирования матрицы коэффициентов. Также в связи с некоторыми ограничениями интерпретатора *Python*3 вычисления не были распараллелены на несколько потоков и проводились в один поток, что не позволило

0-2020.indd 56 11.09.2020 17:00:55

использовать все преимущества представленного в [9, 13] численно-аналитического метода, однако это является недостатком непосредственно интерпретатора, а не метода в целом.

Общее описание параллельно-последовательной модели

Как известно из [1], система мониторинга *Zabbix* позволяет распределять нагрузку и масштабироваться посредством использования прокси-серверов. Каждый прокси-сервер собирает данные с отдельного множества устройств, после чего отправляет данные основному серверу, который занимается обработкой.

Рассмотрим такую модель, в которой имеется два прокси-сервера и один основной. На рис. 1 в общем виде представлена схема взаимодействия компонентов системы.

Информационную систему, взаимодействующую по такой схеме, можно рассматривать как нестационарную систему массового обслуживания. Разделим такую систему на две подсистемы, в первой подсистеме будут находиться прокси-сервера, во второй – основной сервер. Оба прокси-сервера имеют одинаковую пропускную способность, каждый прокси-сервер имеет собственную очередь заявок, поскольку занимается опросом значений с конкретного множества устройств. Иными словами, прокси-сервера работают параллельно. После того как заявка обработана первой подсистемой (любым из прокси-серверов), она сразу же попадает на обработку к основному серверу. После обработки заявки основным сервером заявка считается обработанной. Каждый из серверов считается отдельным каналом обслуживания. На рис. 2 представлена упрощенная схема такой нестационарной системы массового обслуживания (НСМО).

Информационную систему, взаимодействующую по такой схеме, можно рассматривать как НСМО. Разделим такую систему на две подсистемы, в первой подсистеме будут находиться прокси-сервера, во второй – основной сервер. Оба прокси-сервера имеют одинаковую пропускную способность, каждый прокси-сервер имеет собственную очередь заявок, поскольку занимается опросом значений с конкретного множества устройств. Иными словами, прокси-сервера работают параллельно. После того как заявка обработана первой подсистемой (любым из прокси-серверов), она сразу же попадает на обработку к основному серверу. После обработки заявки основным сервером заявка считается обработанной. Каждый из серверов считается отдельным каналом обслуживания. На рис. 2 представлена упрощенная схема такой НСМО.

В каждый момент времени такая НСМО может быть описана векторами $in_system = [in_1, in_2, in_main]$ и $served[out_proxy, out_main]$. Вектор in_system состоит из трех значений. Каждое значение равно количеству заявок в канале с соответствующим номером (in_1, in_2 – прокси-сервера, in_main – основной сервер),

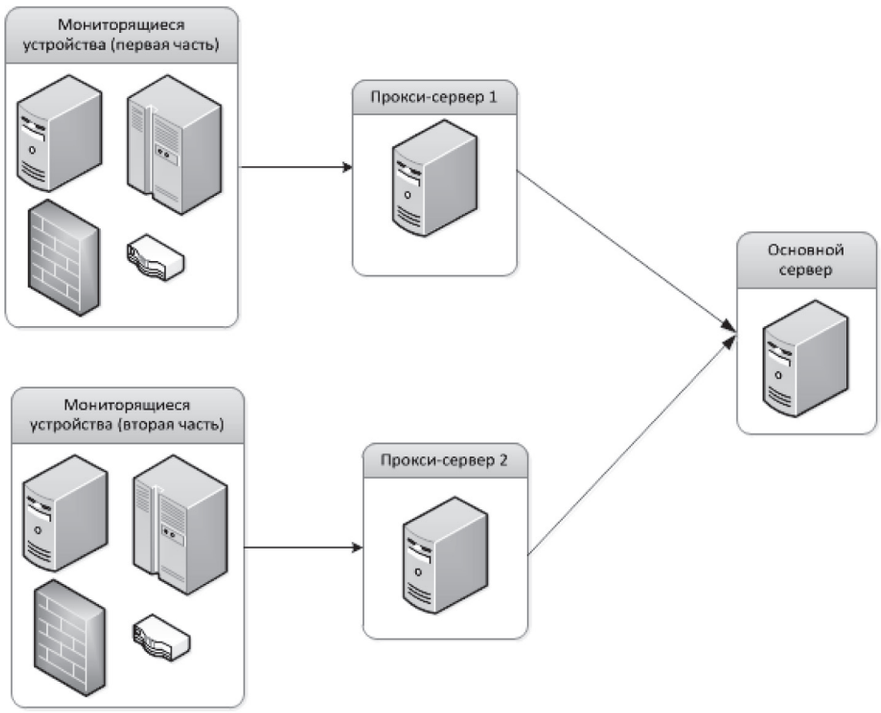

Рис. 1. Упрощенная схема взаимодействий компонентов системы мониторинга

3-2020.indd 57 11.09.2020 17:00:56

при этом $||in_system|| = (0,\overline{N})$.Вектор *served* состоит из 2 значений, где каждое значение равно количеству заявок, обслуженных соответствующей подсистемой, при этом $||served|| = (\overline{0,N-||in_system||})$. Где N – общее число заявок, которые могут поступить в систему. На вход системе последовательно поступают заявки с интенсивностями $\{\lambda_1, \lambda_2, \dots, \lambda_N\}$, которые зависят от номера заявки. Заявки обслуживаются в первой подсистеме с интенсивностями $\{\mu_1, \mu_2, \dots, \mu_N\}$, а во второй подсистеме – с интенсивностями $\{\mu_main_1, \mu_main_2, \dots, \mu_main_N\}$, которые также зависят от номера заявки.

Если каждому состоянию НСМО поставить в соответствие вероятность состояния, то можно записать в векторно-матричном виде систему однородных дифференциальных уравнений Чепмена-Колмогорова (ОДУ)

$$x'(t)=Ax(t),$$

где x(t) – вектор неизвестных функций размерности K, а A – квадратная матрица. Для формирования матрицы необходимо составить полный список всех возможных состояний системы и выявить правила переходов между ними, этот процесс детально описан в следующих пунктах.

В качестве начальных условий обычно выбирают нахождение в состоянии (0,0), то есть состояние, в котором в системе нет ни одной заявки и ни одной заявки не обслужено.

Для решения системы уравнений используется численно-аналитический метод, предложенный в [9]. Получаемое решение позволяет затем найти вероятности нахождения в системе *i* запросов, вероятность обслуживания всех поступивших или не менее некоторого числа запросов и другие характеристики, необходимые для планирования нагрузки на систему мониторинга.

Модификация алгоритма формирования списка состояний

Основной идеей предложенного в [12] метода формирования списка состояний является разбиение всех состояний на группы, а групп на подгруппы так, чтобы некоторое значение в выделенной группе или подгруппе оставалось постоянным, но остальные значения могли меняться. Такой подход позволяет избежать длительного процесса сортировки списка состояний и конечной матрицы коэффициентов по сравнению с подходом, предложенным в [10].

Для рассматриваемой модели необходимо разделить все состояния на N+1 групп так, чтобы в каждой группе значение *served*[*out_main*] являлось постоянным. Каждую группу необходимо разделить на N+1 подгрупп так, чтобы в каждой подгруппе значение *served*[*out_proxy*] являлось постоянным. Каждую подгруппу необходимо разделить на N+1 подгрупп второго порядка так, чтобы в каждой подгруппе второго порядка значение $||in_system||$ являлось постоянным. Пример такой группировки представлен на рис. 4.

Далее, согласно [12], необходимо перебирать все возможные комбинации параметров и проверять их на соответствие некоторым условиям. Состояния, соответствующие этим условиям, записываются в определенное место в созданной структуре групп. Однако в ходе дальнейших исследований подход с перебором возможных комбинаций был признан неоптимальным, поскольку с увеличением количества прокси-серверов и заявок в модели количество возможных комбинаций возрастает экспоненциально. Что приводит к значительному увеличению времени генерации полного списка состояний. Был сформирован новый метод генерации списка состояний, основанный на [10] и [12] – рекурсивный с группировкой. Сгенерированные состояния по-прежнему хранятся в предложенной в [12]

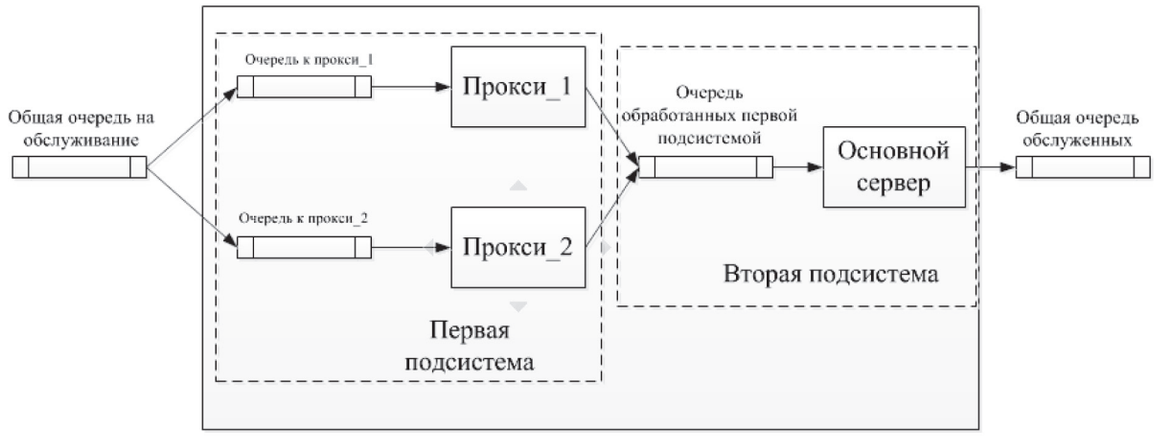

Рис. 2. Упрощенная схема параллельно-последовательной НСМО

3-2020.indd 58 11.09.2020 17:00:56

и описанной выше структуре групп, что позволяет избежать затрат ресурсов на сортировку.

Изменением, привнесенным рекурсивным с группировкой методом, является рекурсивное генерирование новых состояний вместо последовательного перебора комбинаций. Существует три события, инициирующих переход системы из текущего состояния в следующее:

1. В систему попадает новая заявка и поступает на прокси-сервер.

2. Заявка обслужена на прокси-сервере и поступает на основной сервер.

3. Заявка обслужена на основном сервере и покидает систему.

Таким образом, вызывая эти события последовательно, если это возможно для текущего состояния, формируется следующее состояние системы и добавляется в сформированную структуру групп. Если вызвать ни одно из этих событий для текущего состояния нельзя – это означает, что текущее состояние является поглощающим. В таком случае необходимо вернуться на предыдущий уровень рекурсии и продолжить вызов событий, инициирующих переход между состояниями. Такой подход позволяет избежать генерирования большого количества неверных комбинаций, не являющихся описанием состояния системы, и отбрасывания их при проверке. Вместо этого генерируются только реальные состояния, в которые система может перейти из текущего состояния. В таблице 1 приведено сравнение временных затрат на генерацию списка состояний последовательным и рекурсивным с группировкой способами. Оба способа реализованы на языке *Python*3 и выполнялись на одной и той же машине.

После заполнения списка состояний с помощью вышеописанного алгоритма, все состояния в списке будут находиться по порядку при сквозной нумерации, что позволяет не сортировать список и матрицу коэффициентов дополнительно, а также предоставляет удобный способ описания возможных переходов между состояниями. На рис. 3 представлена блок-схема алгоритма формирования списка состояний для параллельно-последовательной модели.

Формирование правил переходов и матрицы коэффициентов

На предыдущем шаге получен структурированный список всех возможных состояний. Благодаря такой группировке можно легко вывести правила переходов между состояниями. Стоит заметить, что основные правила переходов между состояниями можно представить как жизненный цикл заявки, проходящей через НСО:

1. Новая заявка попадает в систему на любой из прокси-серверов, рассматривается как выход системы из текущего состояния и вход в следующее.

2. Заявка обслужена в первой подсистеме (любым прокси-сервером) и поступила в очередь к основному серверу, рассматривается как выход системы из текущего состояния и вход в следующее.

3. Заявка обслужена во второй подсистеме (основным сервером), рассматривается как выход системы из текущего состояния и вход в следующее.

Также благодаря группировке списка состояний можно вывести несколько правил переходов:

1. Переход внутри одной подгруппы второго уровня невозможен.

2. Внутри подгруппы *subgroup* возможен переход из подгруппы второго уровня *under_subgroup* в следующую по порядку подгруппу второго уровня *under_subgroup*+1, при условии, что *under_subgroup* не является последней непустой подгруппой второго уровня подгруппы *subgroup*. Триггером для такого перехода является поступление новой заявки в первую подсистему.

3. Внутри одной группы *group* возможен переход из любой подгруппы *subgroup* в подгруппу *subgroup*+1. При этом переход осуществляется из подгруппы второго

Таблица 1

Сравнение времени генерации списка состояний

Метод генерации списка	Прокси-серверов в модели, шт.	Заявок в модели, шт.	Количество возможных состояний системы, шт.	Время генерации списка, сек.
Последовательный	5	10	19448	45.67
Рекурсивный с группировкой	5	10	19448	0.24

3-2020.indd 59 11.09.2020 17:00:56

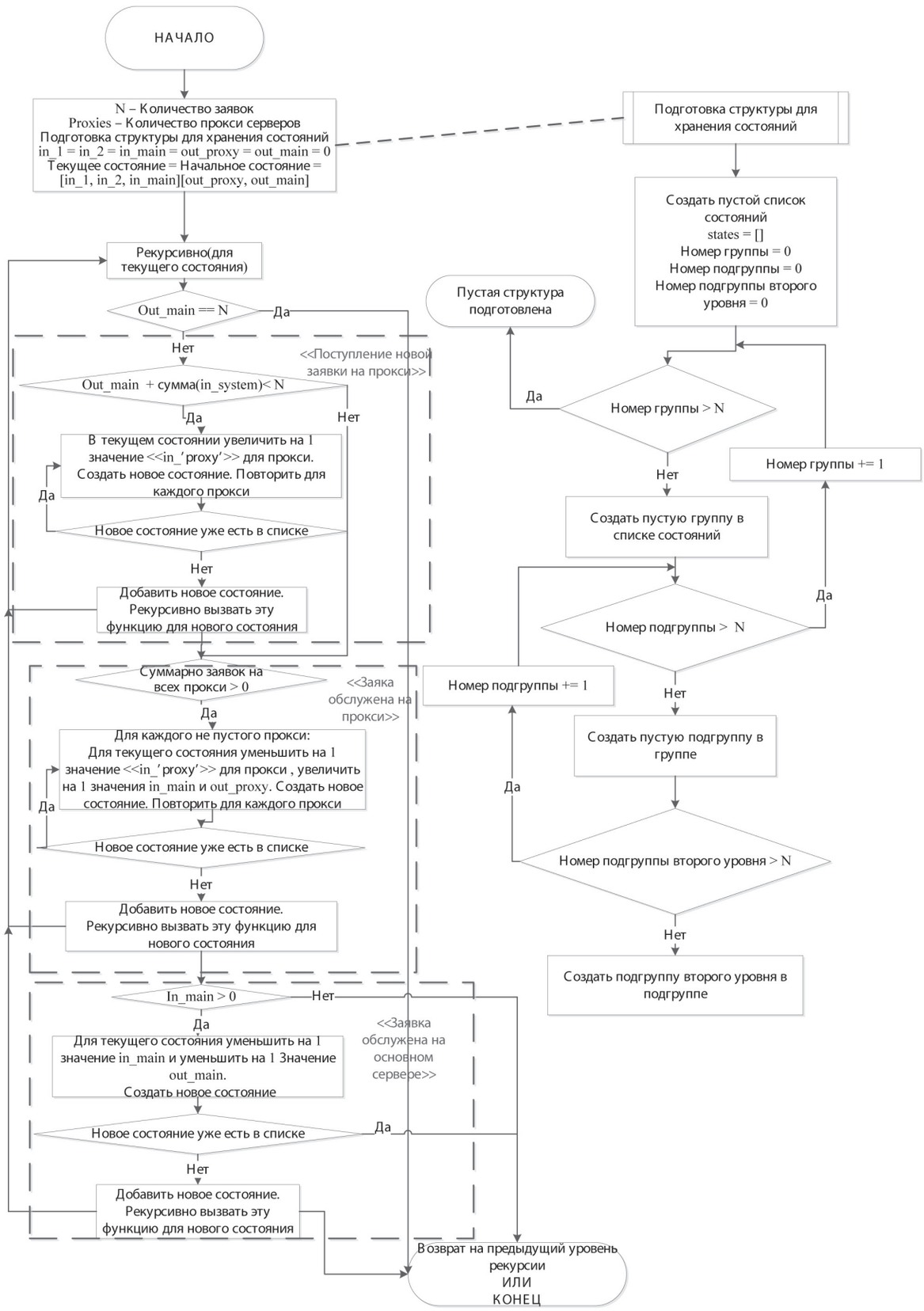

Рис. 3. Блок-схема алгоритма формирования списка состояний

3-2020.indd 60 11.09.2020 17:00:57

уровня *under_subgroup* группы *subgroup* в подгруппу второго уровня *under_subgroup* группы *subgroup*+1. Такой переход невозможен из первых непустых подгрупп второго уровня в подгруппе. Триггером такого перехода является завершение обслуживания заявки в первой подсистеме и переход ее во вторую подсистему.

4. Внутри списка состояний *states* возможен переход из группы *group* в группу *group*+1. Переход осуществляется из подгруппы второго уровня *under_subgroup* подгруппы *subgroup* группы *group* в подгруппу второго уровня *under_subgroup*-1 подгруппы *subgroup* группы *group*+1. Такой переход невозможен из первых непустых подгрупп в группе. Триггером такого перехода является завершение обслуживания заявки на основном сервере.

На рис. 4 представлена упрощенная диаграмма переходов между состояниями для параллельно-последовательно модели из двух прокси-серверов и одного основного сервера при N=3. Как можно заметить, все состояния являются невозвратными.

Последовательно перебирая состояния из сгенерированного списка и применяя выявленные правила к каждому подходящему под условия состоянию, становится возможным в то же время заполнять матрицу коэффициентов. Когда матрица коэффициентов заполнена, остается только решить полученную систему обыкновенных дифференциальных уравнений Чепмена-Колмогорова при заданных начальных условиях численно-аналитическим методом, предложенным в [8].

На рис. 5 представлена обобщенная блок-схема алгоритма генерации матрицы А, представляющая из себя последовательность подзадач для эмуляции продвижения заявки по системе. Основная идея такого алгоритма – эмуляция перехода в следующее возможное состояние и эмуляция возврата в такое прошлое состояние, из которого можно попасть в текущее. Эмуляция

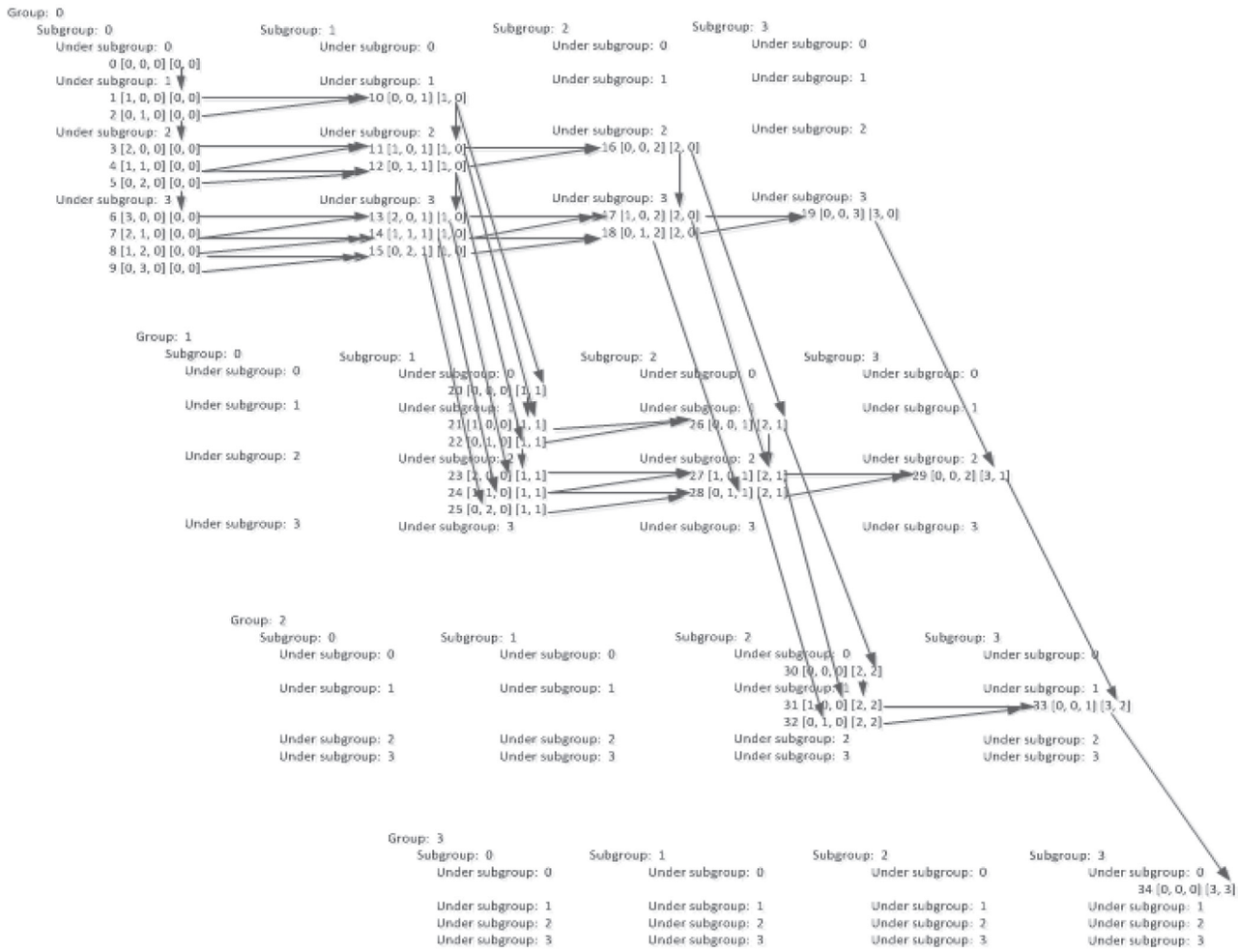

Рис. 4. Упрощенная диаграмма переходов между состояниями при N=3

3-2020.indd 61 11.09.2020 17:00:57

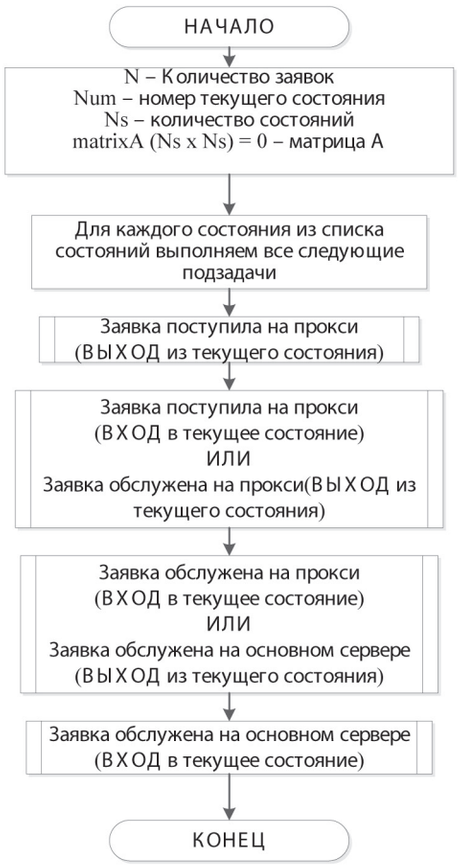

Рис. 5. Обобщенная блок-схема алгоритма генерации матрицы коэффициентов

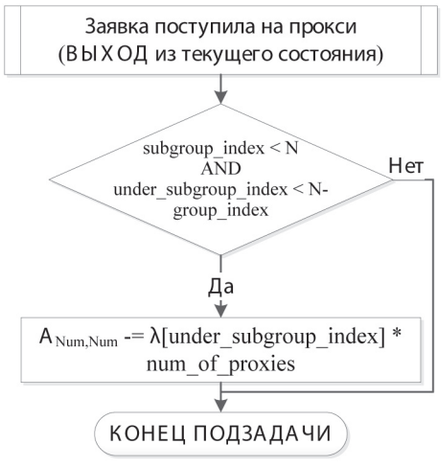

Рис. 6. Блок-схема алгоритма подзадачи, выход из состояния при поступлении заявки на прокси

3-2020.indd 62 11.09.2020 17:00:57

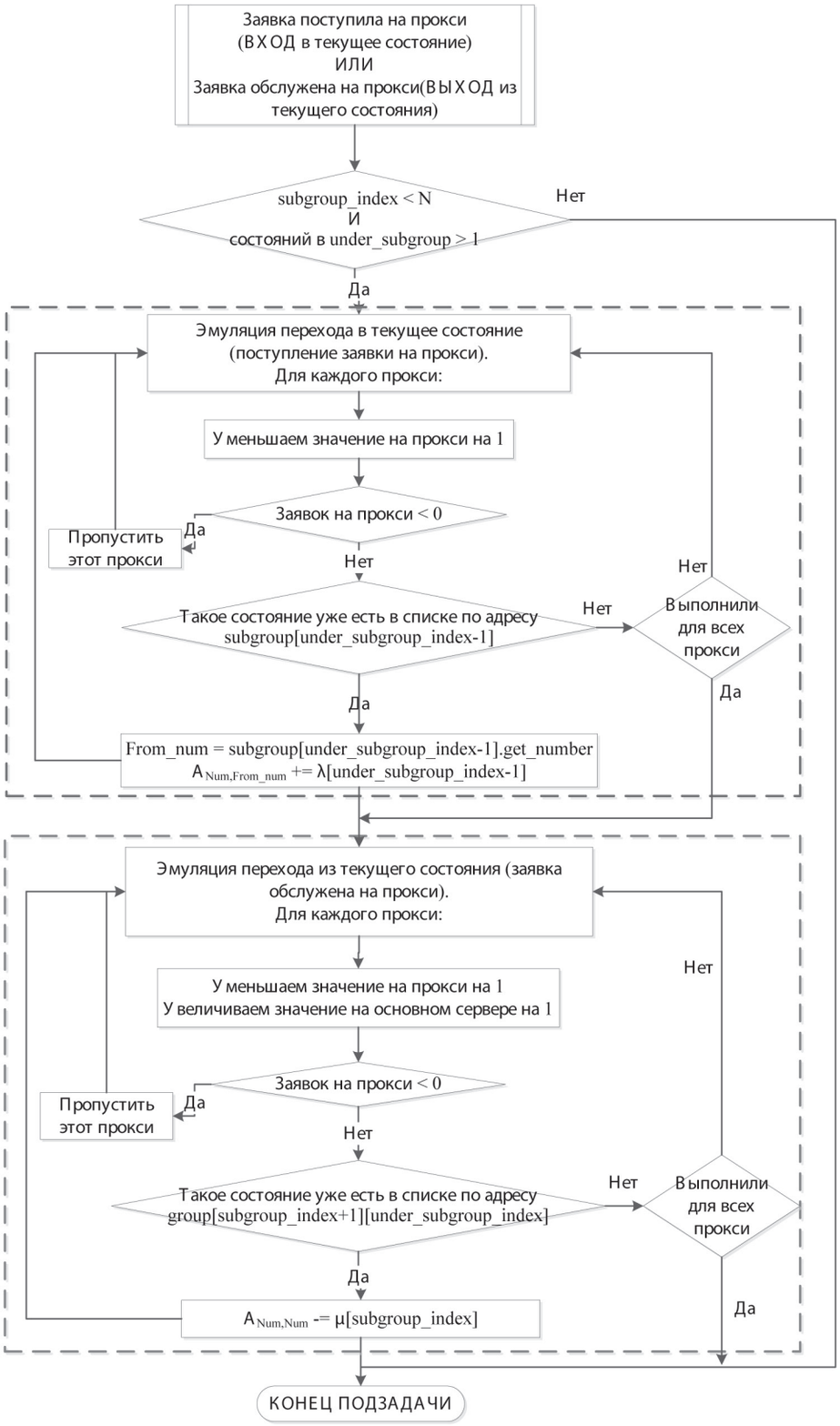

Рис. 7. Блок-схема алгоритма подзадачи, вход в состояние при поступлении заявки на прокси и выход из состояния при завершении обслуживания заявки на прокси

3-2020.indd 63 11.09.2020 17:00:58

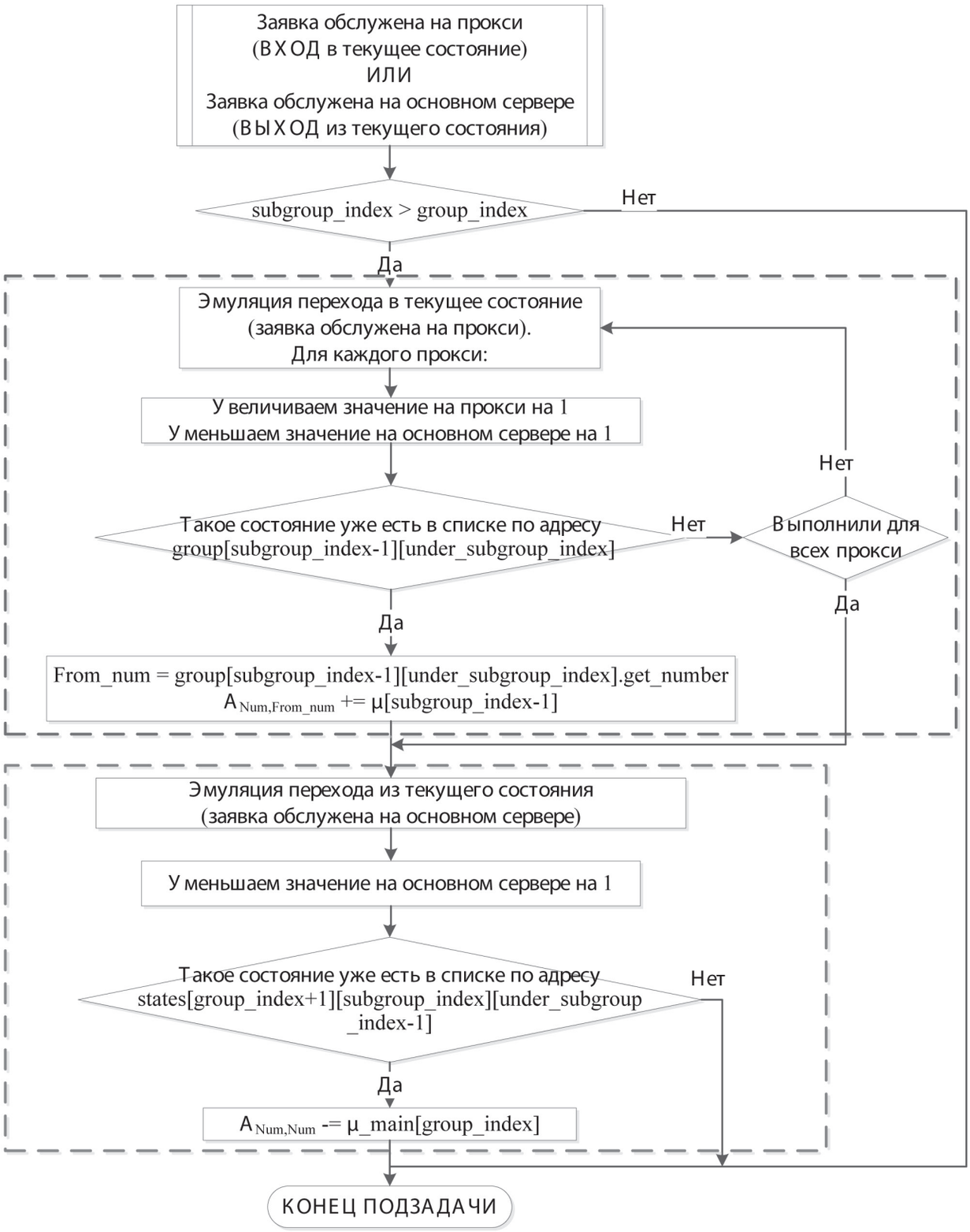

Рис. 8. Блок-схема алгоритма подзадачи, вход в состояние при завершении обслуживания заявки на прокси и выход из состояния при завершении обслуживания заявки на основном сервере

3-2020.indd 64 11.09.2020 17:00:58

достигается путем применения правил переходов и последующей проверки существует ли состояние, полученное после применения правила, в списке уже сгенерированных. Наличие такого состояния говорит о необходимости внести изменения в матрицу А. Плюсы такого подхода:

· всегда известен номер текущего состояния (одна из координат для внесения изменений в матрицу А);

· благодаря использованию типа данных словарь (детальнее описано в главе с особенностями программной реализации) всегда легко можно найти возможные предыдущие или будущие состояния и получить их номер, который является второй координатой для внесения изменений в матрицу А.

Каждая из подзадач, представленных на рис. 5, детальнее представлена на рис. 6–9. Краткое описание значений на блок-схемах:

· *group_index, subgroup_index и under_subgroup_index* – индексы группы, подгруппы и подгруппы второго уровня в структуре хранения состояний соответственно, описывающих место обрабатываемого в текущий момент состояния в структуре хранения;

· *num_of_proxies* – параметр модели, описывает количество прокси-серверов в модели;

· *Num* – номер текущего состояния;

· *From_num* – номер состояния из которого можно перейти в текущее.

Особенности программной реализации

В процессе реализации алгоритма предложенная в [12] структура групп претерпела изменение. В первой реализации список состояний хранил в себе списки с группами, подгруппами и подгруппами второго уровня. Каждая из этих сущностей также являлась списком. В текущей реализации подгруппы второго уровня являются словарями, то есть сущностями, хранящими пары {«ключ»: «значение»}, где ключом является вектор *in_system*, являющийся уникальным для состояний в рамках одной подгруппы второго уровня, а значением является состояние системы. Преимуществом такого подхода являются:

1. Отсутствие зависимости от порядкового номера состояния в подгруппе второго уровня. Это освобождает от соблюдения сложных правил переходов между состояниями разных групп и подгрупп.

2. Отсутствие необходимости перебирать состояния в подгруппе второго уровня при поиске состо-

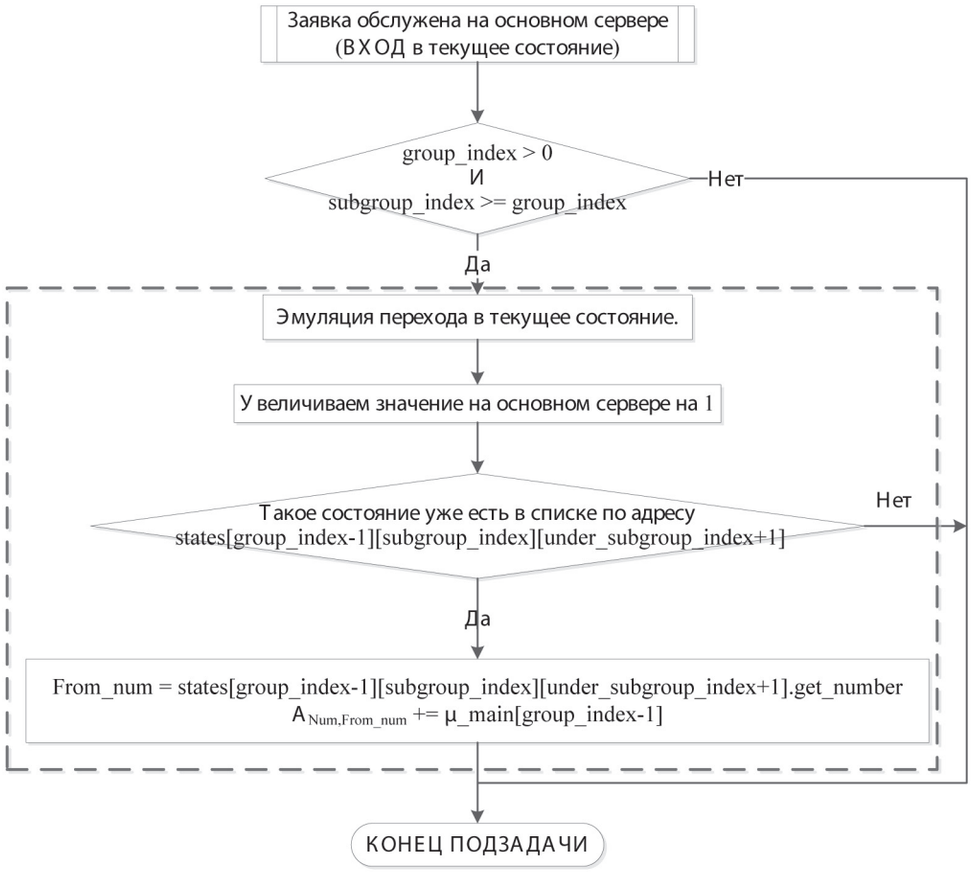

Рис. 9. Блок-схема алгоритма подзадачи, вход в состояние при завершении обслуживания заявки на основном сервере

3-2020.indd 65 11.09.2020 17:00:58

яния. Вместо этого можно обратиться непосредственно к искомому состоянию по его ключу и таким образом быстро получить значение этого состояния либо понять, что такое состояние отсутствует в данной подгруппе второго уровня.

Недостатком такого подхода является увеличение объема памяти, используемого ЭВМ для хранения списка состояний, поскольку приходится дополнительно хранить ключи. Это не является критическим недостатком, учитывая объем памяти, устанавливаемый в современных ЭВМ.

Для борьбы с погрешностью вычислений для хранения чисел использовался тип данных *Decimal*, предоставляющий возможность использовать разрядность чисел, ограниченную только размером памяти ЭВМ. Этот тип данных в *Python*3 хранит значения тем же способом, что и предложенный в [10, 13] тип данных *BigDecimal* в *Java*. Для контроля погрешности в программе также присутствует возможность задания количества разрядов в мантиссе.

Для проверки точности вычислений вероятностей после их вычисления производится расчет их суммы, которая должна быть равна 1. Дополнительно введены граничные проверки, сообщающие о наличии вероятностей с отрицательными значениями или значениями больше 1. Такие проверки вкупе позволяют оценить верность расчета.

К сожалению, в связи с особенностями интерпретатора *Python*3, многопоточная реализация части метода, решающей уже составленную систему ОДУ, аналогично предложенной в [10, 13], производит расчет вероятностей медленнее однопоточной, которая, в свою очередь, работает медленнее предложенной в многопоточной реализации на *Java*. Это связано с использованием GIL (*Global Interpreter Lock*) в *Python*3, который не позволяет потокам приложения в рамках одного процесса использовать одновременно все доступные на исполняемой платформе ядра процессора для распараллеливания потоков в отличие от *Java*. Как результат, все потоки приложения запускаются на одном ядре процессора в конкурентном режиме, на том же, что и родительский процесс самого интерпретатора. Таким образом, мы получаем псевдопараллелизм с обработкой на одном ядре каждого потока последовательно в цикле в течение короткого кванта процессорного времени. Для решения этой проблемы рассматривалась замена многопоточного режима на многопроцессорный. Этот режим позволяет вместо создания дочерних потоков в рамках одного процесса и выделенного ему изолированного пространства ресурсов создавать дочерние процессы с выделенными им собственными изолированными пространствами ресурсов. Такие дочерние процессы могут быть распределены по нескольким ядрам одного процессора. В этом случае распределение никак не ограничивается интерпретатором *Python*3. Однако такой подход не оправдал себя в связи с гораздо большими затратами на межпроцессорное взаимодействие между дочерними процессами, по сравнению с затратами на взаимодействие между потоками в рамках изолированного пространства одного процесса.

Стоит отметить, что проблема многопоточности присуща только реализации на *Python*3 и не присуща реализации на *Java*. Поскольку иных изменений в алгоритме решения уже составленной системы ОДУ не было и алгоритм может быть реализован на любом другом выбранном языке программирования, поддерживающем параллельное выполнение потоков, проблема не может быть расценена как проблема алгоритма и не может влиять на его производительность в общем контексте.

Заключение

Впервые построена параллельно-последовательная модель нестационарной системы обслуживания, моделирующая работу системы мониторинга *Zabbix* на основе распределенной архитектуры. Для построения модели был использован модифицированный алгоритм генерации матрицы коэффициентов системы обыкновенных дифференциальных уравнений Чепмена-Колмогорова с использованием рекурсивного с группировкой метода формирования списка состояний, что ускорило процесс формирования матрицы в несколько раз. В модифицированном алгоритме использовались выведенные в работе правила переходов между состояниями. Описаны ключевые особенности программной реализации предложенной модели.

Поддержка исследований. Исследования, выполненные по данной тематике, проводились в рамках бюджетной темы №№0073–2019–0004.

Литература

1. Шардаков, К. С. Сравнительный анализ наиболее популярных существующих систем мониторинга сетевого оборудования, распространяемых по лицензии GPL / К.С. Шардаков // Интеллектуальные технологии на транспорте. – 2018. – № 1. – С. 44–47.

2. Zegzhda, P. D. Using graph theory for cloud system security modeling / P.D. Zegzhda, D.P. Zegzhda, A.V. Nikolskiy // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). – 2012. – Vol. 7531. – P. 309–318.

3. Osogami, T. Analysis of transient queues with semidefinite optimization / T. Osogami, R. Raymond // Queueing Systems. – 2013. – No. 73. – P. 195–234.

4. Upadhyaya, S. Queueing systems with vacation: an overview / S. Upadhyaya // International journal of mathematics in operational research. – 2016. – Vol. 9, No. 2. – P. 167–213.

5. Смагин, В. А. О загрузке вычислительной системы с изменяющейся интенсивностью поступления заданий / В.А. Смагин, В.П. Бубнов, В.И. Сафонов // Автоматика и вычислительная техника. – 1987. – № 6. – С. 19–22.

3-2020.indd – 66 11.09.2020 – 17:00:59

6. Бубнов, В. П. Разработка динамических моделей нестационарных систем обслуживания / В.П. Бубнов, В.И. Сафонов. – СПб.: Издательство «Лань», 1999. – 63 с.

7. Bubnov, V. P. Software reliability model with coxian distribution of length of intervals between errors detection and fixing moments / V.P. Bubnov, A.D. Khomonenko, A.V. Tyrva // Proceedings – International Computer Software and Applications Conference. 35th Annual Computer Software and Applications Conference. – 2011. – P. 310–314.

8. Bubnov, V. P. Model of reliability of the software with coxian distribution of length of intervals between the moments of detection of errors / V.P. Bubnov, A.V. Tyrva, A.D. Khomonenko // Proceedings – International Computer Software and Applications Conference. 34th Annual IEEE International Computer Software and Applications Conference. – 2010. – P. 238–243.

9. Бубнов, В. П. Особенности программной реализации численно-аналитического метода расчёта моделей нестационарных систем обслуживания / В.П. Бубнов, А.С. Еремин, С.А. Сергеев // Труды СПИИРАН. – 2015. – № 38. – С. 218–232.

10. Бубнов, В. П. Рекурсивный метод генерации матрицы коэффициентов системы однородных дифференциальных уравнений, описывающих нестационарную систему обслуживания / В.П. Бубнов, А.Д. Хомоненко, С.А. Сергеев // XVIII Международная конференция по мягким вычислениям и измерениям SCM 2015. – Т. 1. – С. 164–166.

11. Сергеев, С. А. Метод составления систем однородных дифференциальных уравнений для расчёта вероятностно-временных характеристик нестационарных систем обслуживания / С.А. Сергеев // Интеллектуальные технологии на транспорте. – 2015. – № 2. – С. 32–42.

12. Shardakov, K. S. Generating of the Coefficient Matrix of the System of Homogeneous Differential Equations / K.S. Shardakov, V.P. Bubnov, A.N. Pavlov // Workshop Computer Science and Engineering in the framework of the 5th International Scientific-Methodical Conference "Problems of Mathematical and Natural-Scientific Training in Engineering Education". – 2018. – P. 42–47.

13. Сергеев, С. А. Программный комплекс аналитико-имитационных моделей для расчёта вероятностно-временных характеристик нестационарных систем обслуживания / С.А. Сергеев, В.П. Бубнов, В.В. Бубнов // Свидетельство о регистрации программы для ЭВМ RU 2015617625, 16.07.2015. Заявка № 2015614100 от 19.05.2015.

3-2020.indd – 67 11.09.2020 – 17:00:59