Людмила Борисовна СЕНКЕВИЧ[1]
Марат Асхатович САБИТОВ[2]

УДК 004.94

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ И ОПТИМИЗАЦИЯ РАБОТЫ ПАРАЛЛЕЛЬНОГО СЕРВЕРА С ОТКАЗАМИ В СРЕДЕ ANYLOGIC

[1] кандидат педагогических наук,
доцент кафедры кибернетических систем,
Тюменский индустриальный университет
lyudmila1@yandex.ru

[2] магистрант кафедры кибернетических систем,
Тюменский индустриальный университет
sabitov.m.a@yandex.ru

Аннотация

Современные научные исследования всё чаще обращаются к проблеме обработки больших массивов данных. Широкое распространение клиент-серверной технологии взаимодействия и облачных вычислений в настоящий момент времени поднимает вопросы эффективности работы параллельного сервера, а также возможности прогнозировать результаты в зависимости от степени загрузки и характеристик оборудования.

В данной статье производится имитационное моделирование параллельного сервера с отказами в среде AnyLogic, а затем производится многомерная оптимизация методом взвешенной суммы. В рамках исследования построена имитационная модель системы массового обслуживания с отказами, содержащая имитатор работы сервера, терминалы, имитатор отказов и сегменты сбора статистики. Используемая модель параллельного сервера является абстрактной и достаточно обобщенной, что позволяет конкретизировать ее путем введения дополнительных зависимостей и уточнения характеристик. Эксперимент с оптимальными параметрами позволил получить следующий выигрыш

Цитирование: Сенкевич Л. Б. Имитационное моделирование и оптимизация работы параллельного сервера с отказами в среде AnyLogic / Л. Б. Сенкевич, М. А. Сабитов // Вестник Тюменского государственного университета. Физико-математическое моделирование. Нефть, газ, энергетика. 2022. Том 8. № 1 (29). С. 126-143.
DOI: 10.21684/2411-7978-2022-8-1-126-143

в показателях эффективности системы: параметр загрузки процессора (по памяти) — 7%; параметр загрузки процессора (по коэффициенту загрузки) — 8%; вероятность простоя терминалов — 5,7%; частота отказов основного компьютера — в 36 раз меньше начальной конфигурации; число прерванных программ — на 7 меньше. При этом необходимо отметить, что общее количество выполненных запросов осталось на том же уровне (462-465, т. к. интенсивность терминалов не варьировалась).

Поскольку оптимизация стохастических моделей основывается на использовании случайных величин, была применена встроенная возможность переменного количества (от 5 до 10) репликаций («прогонов») с доверительной вероятностью 95% и уровнем ошибок 0,5. Полученные результаты позволяют говорить о возможности дальнейшего исследования модели и ее развития в среде AnyLogic.

Ключевые слова

Имитационное моделирование, система массового обслуживания, стохастическая модель, параллельный сервер, многомерная оптимизация, метод взвешенной суммы, AnyLogic.

DOI: 10.21684/2411-7978-2022-8-1-126-143

Введение

Научные исследования на текущий момент всё в большей степени обращаются к вычислительному эксперименту взамен эксперимента натурного, учитывая выигрыш в стоимости, безопасности, широком охвате диапазонов влияющих факторов. Такие эксперименты, отражая стремления к большей точности и адекватности получаемых результатов, становятся всё более требовательными к программному обеспечению и аппаратной составляющей вычислительных систем. На волне такого интереса всё большее распространение получают облачные вычисления на сервере, центры обработки данных, а также суперкомпьютерные системы, которые предоставляют по предварительной договоренности доступ научным группам к вычислительным ресурсам конкретной системы. Поскольку один сервер/суперкомпьютер способен решать параллельно несколько задач, таким образом, возникает ряд взаимодействий типа «клиент — сервер», которые могут перекрываться во времени и существенным способом влиять на качество предоставляемых сервером/суперкомпьютером услуг. Следовательно, работа параллельного сервера требует анализа, направленного на поиск возможных проблем в предоставлении услуг и в целом на выявление наилучшего режима его использования. Под параллельным сервером будем рассматривать вычислительную систему, которая передает каждый запрос отдельному потоку/процессу, после чего он немедленно ожидает следующего входящего запроса [7].

Для описания данного взаимодействия необходимо формализовать понятия клиента, сервера и их взаимоотношений. Можно обратиться к следующей, достаточно абстрактной и редуцированной формулировке задачи, которую, однако, при необходимости несложно расширить на случай конкретного взаимодействия. К серверу подключено четыре клиента, с которых осуществляется решение

задачи. Клиент может использовать только четыре типа операций: операции редактирования, трансляции, планирования и решения. Такое ограничение количества базовых операций возможно, и у него существуют реальные аналоги в существующих информационных системах. Например, при работе с базой данных на языке SQL можно выделить широко известную формулировку CRUD, где рассматриваются также четыре типа операций: Insert, Select, Update, Delete. Этим же методам соответствуют определенные методы протокола HTTP. Далее необходимо отметить, что операции различаются по потреблению ресурсов вычислительной системы: в данном случае таким ресурсом будет считаться оперативная память. Так как взаимодействие клиентов с сервером не является детерминированным, то возможно статистическое представление, в котором задачи поступают согласно закону распределения дискретной случайной величины. Внося неопределенность в работу самого сервера, время выполнения каждой задачи также можно рассматривать как случайную величину; помимо этого, можно ввести в рассмотрение вероятность отказа сервера, указав при этом на наличие сервера резервного. Восстановление сервера занимает также неопределенное время, в течение которого всю работу выполняет резервный сервер. После восстановления основного сервера резервный сервер отключается, и основной сервер продолжает работу с очередной программы. В случае сбоя вычисления производятся по резервному каналу.

Такая модель представляет собой типичную систему массового обслуживания [1]. Системы массового обслуживания можно рассматривать как динамические системы с дискретными событиями. Такие модели могут приближенно описывать поведение многих реальных систем, например, функционирование компьютерных и коммуникационных сетей, а также производственные системы и системы транспортные. В связи с тем, что состояние такой модели изменяется в дискретные моменты времени, эволюция модели представляется в виде последовательности скачков; при этом ввиду использования различных вероятностных распределений в описании эффективности модели чаще всего используются моменты случайных величин. Такое поведение в корне отличается от моделей с непрерывным изменением состояния. Следует отметить, что формулировка задачи не включает в себя детализацию процесса параллельной обработки запросов (с помощью множества потоков или процессов). Для создания подобной модели удобно воспользоваться программой AnyLogic, которая также предоставляет встроенные инструменты для анализа и оптимизации построенных дискретно-событийных имитационных моделей [3].

Методы

Решение поставленной задачи может быть представлено как разработка и организация взаимодействия трех имитационных блоков:

— клиенты,
— имитатор работы сервера,
— имитатор отказов сервера.

Данное решение является предпочтительным по следующим причинам:

— клиенты (пользователи, терминалы) осуществляют выработку запросов независимо друг от друга;

— сервер обрабатывает запросы в зависимости от их порядка и количества свободной оперативной памяти;

— отказы происходят в согласии с заданной функцией вероятности и выражаются в переключении основной линии обработки запросов на резервную.

Таким образом, необходимо разработать каждый блок в отдельности и организовать взаимодействие между этими блоками в соответствии с выданным вариантом.

Модель клиента удобнее всего представить как генератор агентов, где каждый агент представляет собой запрос. Понятие «агент» в данном контексте следует рассматривать как автономный самостоятельный элемент вычислительной модели, имеющий заданные свойства, состояния и поведение. Взаимодействие агентов между собой позволяет описать систему в русле концепции агентного моделирования, а не системной динамики. По мере генерации каждому агенту-запросу (TermTask) присваивается метка клиента-источника, которая представляет собой целочисленный атрибут. Агент TermTask имеет, кроме того, атрибут type, который представляет собой тип запроса (редактирование, трансляция, планирование, решение). Типы запроса определяются согласно дискретному равномерному распределению для интервала [min, max], где как min, так и max входят в интервал значений.

Каждый тип запроса потребляет конкретное количество оперативной памяти. Определим следующие значения для каждого типа: планирование — 100%, решение — 50%, трансляция — 33%, редактирование — 25%. В таком случае оперативную память сервера можно рассматривать в виде некоторого хранилища ресурсов, которые захватываются и освобождаются агентами-запросами. Тогда, установив предельную планку в 120 ресурсов, определим потребление в зависимости от типа запроса в условных единицах: планирование — 120, решение — 60, трансляция — 40, редактирование — 30.

Захватившие ресурсы агенты поступают в очередь по правилу FIFO (в порядке поступления в очередь — по умолчанию). Очередь моделирует блок queue, максимальная длина очереди задана равной 4. Нужно отметить, что величина очереди не зависит от количества клиентов и определяется сугубо на основании технических характеристик сервера. В данном случае выбранное значение соответствует максимальному количеству агентов-запросов, которые могут захватить имеющиеся ресурсы системы массового обслуживания (память).

Обработка запросов может вестись на двух серверах — основном и резервном, поэтому необходим переключатель потоков запросов selectOutput, срабатывающий при выполнении логического условия от имитатора отказов. Сама обработка запроса имитируется блоком delay, который задает задержку агента-запроса в блоке обработки. После обработки запроса необходимо освободить занятые ранее ресурсы — этим занимается блок release.

Имитатор отказов реализован в виде отдельного сегмента модели, основной задачей которого является определение состояния основного сервера (работа/отказ/восстановление). Данная функциональность реализована с помощью двух последовательно соединенных блоков задержки, один из которых реализует интервал до отказа в виде равномерно распределенной случайной величины на интервале от 300 до 350 с (используется встроенная функция uniform, которая генерирует непрерывную случайную величину, равномерно распределенную на интервале [min, max), верхняя граница интервала не включена). Второй блок задержки реализует время восстановления с помощью уже упомянутой функции uniform, но на интервале от 160 до 300 с.

Кроме того, создана булева переменная *A*, которая отвечает за индикацию состояния отказа. По умолчанию переменная *A* равна false (первоначально основной компьютер находится в работоспособном состоянии), при выходе из блока delay01 — true (наступил отказ), при выходе из блока delay02 снова меняется на false (отказ был устранен).

Необходимо учесть еще одну особенность работы резервного канала: если сбои происходят во время вычисления, то запасной сервер запускается за 2 с и вычисляет с самого начала — следовательно, необходимо отслеживать состояние сервера (обрабатывает запрос или нет), а сам обрабатываемый запрос, если таковой имеется на момент отказа, отправлять на резервный сервер. После восстановления основного компьютера резервный компьютер отключается, и основной компьютер продолжает работу с очередной программы — значит, резервный канал завершает обработку последней поступившей заявки (до момента восстановления основного компьютера) и больше запросов не принимает.

Для включения данной особенности в имитационную модель было принято решение использовать дополнительное действие при выходе из блока delay01: прерывание задержки (обработки) всех агентов основным сервером функцией stopDelayForAll().

Это соответствует ситуации, когда поломка произошла во время обработки запроса — тогда запрос должен быть перенаправлен на другой, резервный сервер. При этом должна быть выполнена еще одна условность: время включения резервного сервера в этом случае составит 2 с. Захваченные ресурсы перемещаются вместе с агентом-запросом. Ветвление, связанное с выполнением условий перехода обслуживания на различные серверы (произошел ли отказ основного сервера, выполнено ли было его восстановление), было организовано с помощью блоков selectOutput, установленных после буфера запросов терминалов и после блока C_1.

При входе в блок delay, учитывающий задержку на запуск резервного сервера, было установлено действие, связанное с инкрементированием переменной *B*: эта переменная имеет целочисленное значение и вводится для учета количества прерванных программ.

Итоговая схема имитационной модели представлена на рис. 1.

Теперь необходимо смоделировать работу системы. Моделирование необходимо для определения количественных и качественных характеристик работы системы, таких как:

а) параметр загрузки сервера — может быть выражен двумя способами: средним объемом потребляемой памяти (исходное количество памяти равно 120 у. е.) за время моделирования или коэффициентом загрузки;

б) вероятность простоя клиентов — выражаются через долю времени, когда ни один клиент не выполняет запрос;

в) частота одновременного выполнения операции трансляции с трех клиентов — выражается через долю указанной комбинации выполняемых программ в общем объеме имевших место комбинаций;

г) частота отказов основного сервера — представляет собой отношение числа отказов основного сервера к времени моделирования Δt;

д) число прерванных программ — определяется по значению переменной B после выполнения имитации;

е) функцию распределения времени вычисления одной программы.

Пункт (а) имеет два типа трактовки, первая из которых может быть визуализирована гистограммой с аргументом resourcePool.idle(), а вторая опирается на коэффициент загрузки.

Пункт (в) можно конкретизировать следующим образом: поскольку поступление запросов определяется наличием свободных ресурсов, а поступление трех трансляций блокирует оставшийся терминал, можно измерять время, когда в очереди содержится три запроса и ни одного в обработке или два запроса в очереди

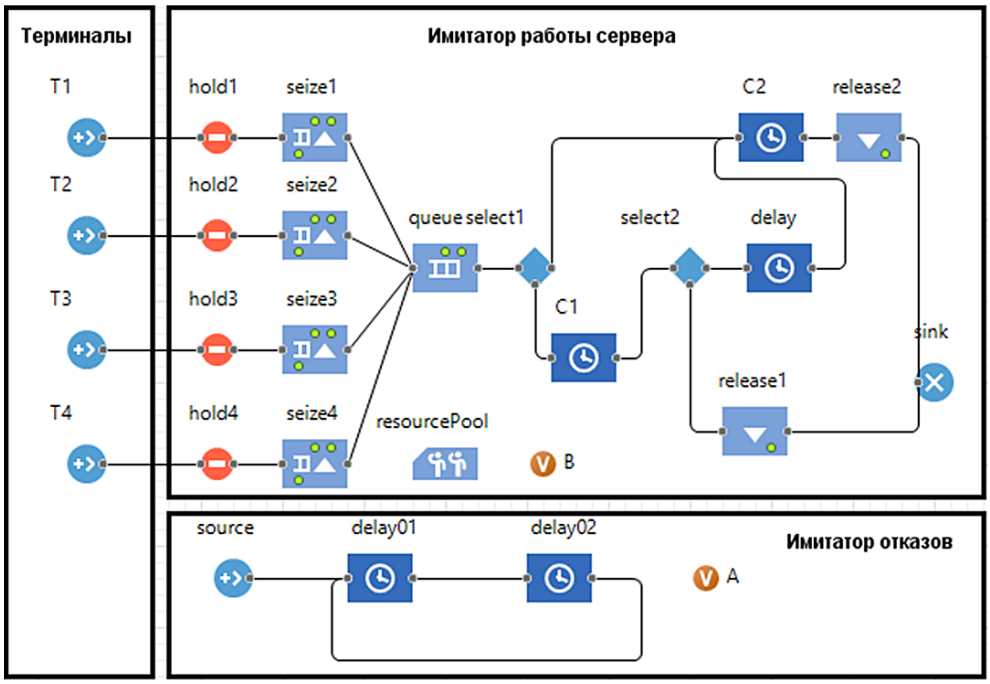

Рис. 1. Полученная имитационная модель в AnyLogic

Fig. 1. The resulting simulation model in AnyLogic

и один в обработке, но при этом свободных ресурсов нет. Доля измеренного таким способом времени от общего времени имитации составит искомую величину. Измерение данного параметра будем проводить, используя дополнительную переменную B_2 (по умолчанию равна 0), которая увеличивается на единицу при входе запроса на трансляцию в блок очереди queue и уменьшается на единицу при входе запроса на трансляцию в блок sink. Если $B_2 = 3$, то в специально выделенном сегменте снимается блокировка счетчика — замеряется искомое время.

С одной стороны, систему можно рассматривать как оборудование с определенным коэффициентом использования, или коэффициентом загрузки, обычно обозначаемым ρ и определяемым как

$$\rho = \lambda/\mu, \tag{1}$$

где λ — интенсивность входящего потока, μ — интенсивность потока обслуживания [6].

С другой стороны, используя определения интенсивности входящего потока и потока обслуживания, можно получить альтернативное выражение коэффициента загрузки:

$$\rho = t_s/t_a, \tag{2}$$

где t_a и t_s есть средний временной интервал между требованиями входящего потока и среднее время обслуживания в канале соответственно. Таким образом, коэффициент использования оборудования можно трактовать как отношение нагрузки на оборудование к максимальной нагрузке, которую может выдержать это оборудование, или отношение времени занятости оборудования к общему времени его функционирования. Параметр t_s можно определить из постановки задачи — это математическое ожидание времени обработки запроса. Второй параметр — t_a — есть отношение времени моделирования к количеству обработанных запросов.

Пункт (е) выбивается из всего ряда, т. к. требует учета времени нахождения каждого агента-запроса в системе с момента его генерации клиентом до удаления из модели. Для определения данного параметра будут использоваться блоки TimeMeasureStart на выходе каждого терминала и TimeMeasureEnd перед блоком sink. Здесь удобнее всего параметризовать TimeMeasureEnd на прием агентов от блоков TimeMeasureStart.

AnyLogic предоставляет пользователю удобные средства для сбора статистики по работе блоков диаграммы процесса. Объекты EnterpriseLibrary самостоятельно производят сбор основной статистики. Для улучшения визуального восприятия динамики загрузки сервера можно воспользоваться автоматически обновляемым графиком зависимости рассматриваемого параметра от времени. Доли каждого типа задач, выполненных сервером, можно визуализировать с помощью круговой диаграммы. Для визуализации функции распределения удобнее всего использовать гистограмму.

Кроме того, средства визуализации позволяют в режиме реального времени отображать текущее использование памяти, средний процент ее использования, а также доли каждого типа обработанных запросов в их общей массе с помощью круговой диаграммы. Общий вид имитационной модели с элементами анализа представлен на рис. 2.

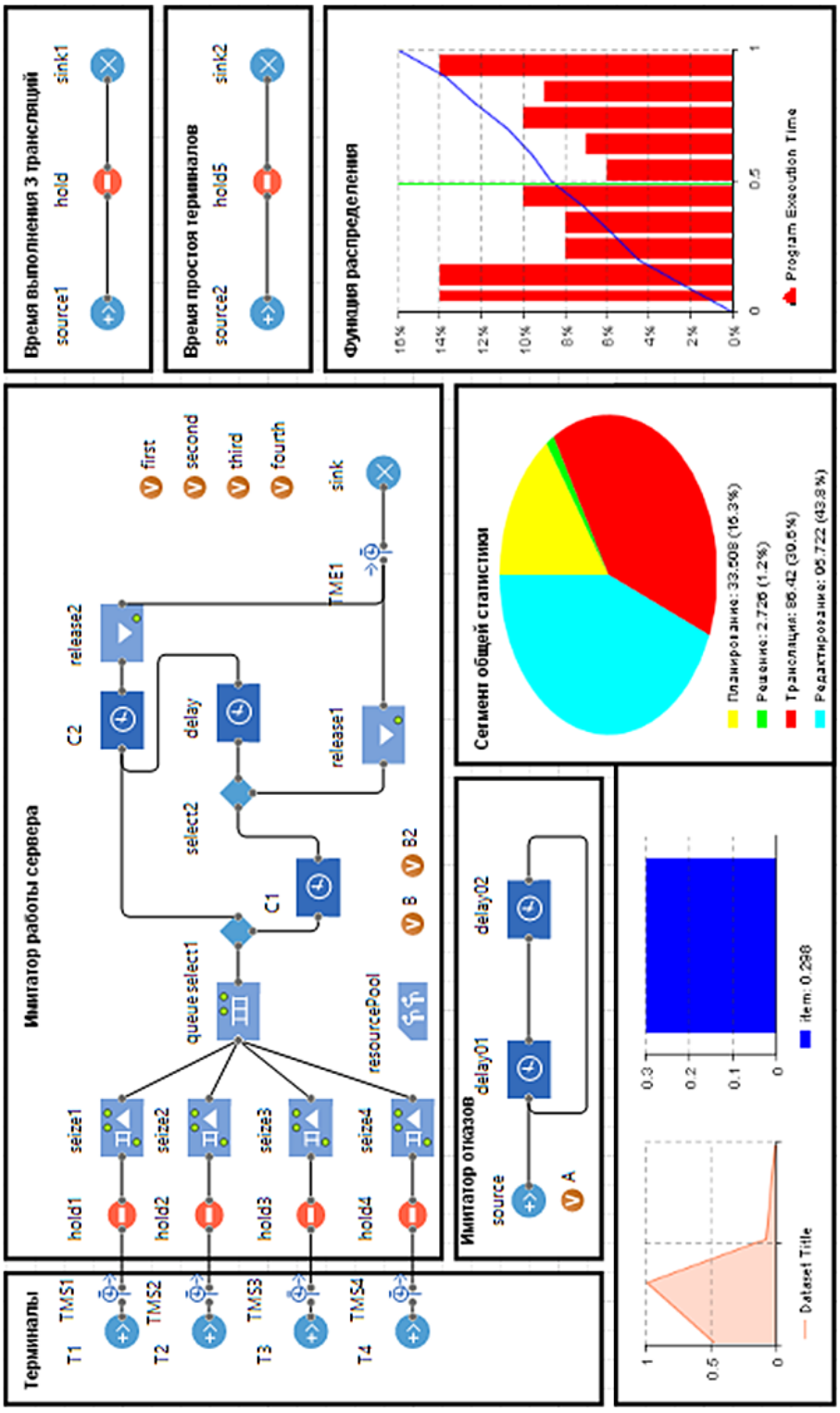

Рис. 2. Имитационная модель в AnyLogic с сегментами анализа

Fig. 2. Simulation model in AnyLogic with analysis segments

Результаты

Для проведения моделирования был создан эксперимент, имитирующий работу системы в течение 5 ч (18 000 с). Функция распределения остается приближенно экспоненциальной, хотя равномерное распределение времени обработки запроса вносит смещение в работу системы. Ключевые результаты эксперимента отражены в таблице 1.

Среда AnyLogic предоставляет достаточно мощные средства оптимизации построенных имитационных моделей [4]. Таким образом, полученная модель может быть изменена «на месте» без необходимости импорта в другую вычислительную среду.

В представленной модели системы массового обслуживания можно выделить семь параметров, влияющих на качество работы системы в целом (таблица 2). Для каждого параметра определен диапазон варьирования и начальное значение.

Оптимизируемые параметры можно определить на основе таблицы 1, при этом можно воспользоваться следующими соображениями:

— параметр загрузки процессора (в эксперименте будет оцениваться только загрузка по памяти) может достигать величины 60% без повышения вероятности преждевременного отказа. Конечно, существует и другой подход, согласно которому параметр загрузки процессора может определяться динамически для увеличения времени работы системы до отказа, но в данном случае такая зависимость не является целевой;

— вероятность простоя терминалов является нежелательным свойством, т. к. определяет время, в течение которого система работает без полезной нагрузки. Регулировать загрузку терминалов можно при централизованном распределении времени обращения каждого клиента (или процесса)

Таблица 1

Результаты моделирования работы параллельного сервера

Table 1

Simulation results of parallel server operation

Параметр	Значение
Параметр загрузки процессора (по памяти)	14%
Параметр загрузки процессора (по коэффициенту загрузки)	26%
Вероятность простоя терминалов	6%
Частота одновременного выполнения трансляции с трех терминалов	0
Частота отказов основного компьютера	0,0018
Число прерванных программ	8
Функция распределения	Экспоненциальное распределение

к вычислительным ресурсам — это типичный пример создания кажущейся параллельной обработки нескольких задач одним вычислительным ядром, свойственный, например, для языка JavaScript (Event Loop). Таким образом, сервер напрямую не контактирует с терминалом, но с его моделью, которая является связующим звеном между клиентской частью и серверной;

— частота отказов основного сервера зависит только от блока имитации отказа, при этом она изначально задана, а сам сервер может быть восстановлен за конечное время;

— число прерванных программ необходимо минимизировать. Это количество напрямую зависит от числа отказов и загрузки сервера, поэтому выражает определенный компромисс между указанными характеристиками;

— частота одновременного выполнения трансляции с трех терминалов не является характеристикой, подлежащей оптимизации, однако нужно оценивать математическое ожидание времени нахождения заявки в системе: его необходимо минимизировать;

— система должна быть в состоянии обработать как можно больше запросов.

Таким образом, стоит задача многокритериальной оптимизации, для которой необходимо разработать соответствующую целевую функцию. Важно отметить,

Таблица 2

Параметры, влияющие на работу системы

Table 2

Parameters affecting system performance

Параметр	Описание	Начальное значение	min	max
$p_{11}, p_{12}, p_{13}, p_{14}$	$1/t$, где t — время между прибытиями запросов	0,00625	0,005	0,025
p_2	среднеквадратическое отклонение времени выполнения программы	0,9	0,1	2,0
p_3	математическое ожидание выполнения программы	10	5	15
p_4	левая граница времени отказа сервера	300	300	300
p_5	правая граница времени отказа сервера	350	350	350
p_6	левая граница времени восстановления сервера	160	50	250
p_7	правая граница времени восстановления сервера	300	300	300
p_8	количество у. е. памяти (меняется дискретно — по 20 ед. в большую сторону)	120	120	180

что, исходя из перечисленного выше, можно воспользоваться концепцией оптимальности по Парето, согласно которой набор решений представляет собой оптимальные компромиссы между конфликтующими целевыми критериями, таким образом, в каждом решении нет преобладающего критерия. Следовательно, ни один целевой критерий не может быть улучшен без ухудшения одного или нескольких других целевых критериев. Для получения целевой функции можно воспользоваться методами скаляризации, которые сводят многокритериальную оптимизацию к оптимизации одной скалярной целевой функцией. Часто пользуются простейшим видом целевой функции, а именно взвешенной суммой — объединением всех критериев в одну функцию приспособленности с использованием линейного соотношения:

$$F\left(\bar{f}(\bar{x})\right) = w_1 f_1(\bar{x}) + \cdots + w_r f_r(\bar{x}), \quad (3)$$

где $f_i(x)$ — i-й критерий, w_i — вес i-го критерия. При этом важно ранжировать критерии по значимости [9]. Следует отметить, что недостатком метода взвешенных сумм в случае выпуклого множества значений целевых функций является невозможность охватить все оптимальные по Парето точки из множества Парето-фронта [5].

Другим возможным вариантом является метод ε-ограничений. В методе ε-ограничений в качестве скалярного критерия оптимальности используется самый важный из частных критериев оптимальности, а остальные частные критерии учитываются с помощью ограничений типа неравенств [10].

В данном случае удобнее всего воспользоваться смешанным определением целевой функции. Очевидно, самым значимым критерием для конечного пользователя является время осуществления запроса, которое напрямую связано с количеством выполненных программ за всё время моделирования. Выражая данный параметр в виде количества выполненных программ и вычитая штраф за прерванные программы, получаем прототип искомой целевой функции, которую необходимо максимизировать. Остальные важные критерии введены в виде ограничений на целевую функцию — это ограничение на количество задействованных системой единиц памяти (до 60%) и ограничение на время простоя терминалов (до 10%). Были выделены также несколько уровней штрафов в зависимости от того, насколько допустимыми являются прерванные программы. Общая характеристика условной оптимизации представлена в таблице 3.

Встроенный в AnyLogic оптимизатор OptQuest позволяет решать задачи многокритериальной оптимизации при заданных ограничениях. Так как оптимизация стохастических моделей имеет дело с варьирующимися значениями величин, для каждой итерации процесса применяется заранее заданное число репликаций, или «прогонов», по результатам которых и строятся результаты итерации. Функционирование системы массового обслуживания во время оптимизационных экспериментов можно анализировать в режиме реального времени с помощью описанного ранее сегмента сбора статистики (рис. 3).

Таблица 3

Характеристика условной оптимизации целевой функции

Table 3

Characteristic of conditional optimization of the objective function

№	Штраф за прерванную программу	Ограничение на простой терминалов, %	Ограничение на использование памяти, %
1	10	до 10	до 60
2	50	до 10	до 60
3	100	до 10	до 60
4	150	до 10	до 60
5	250	до 10	до 60

Задачи динамической стохастической оптимизации основываются на том условии, что решение каждой итерации должно быть получено в результате обработки доступной информации в соответствующий момент времени. Репрезентативность текущих данных можно повысить, увеличив количество репликаций. Следует, однако, учитывать, что данное решение повысит нагрузку на вычислительную систему.

Довольно гибким вариантом решения указанной проблемы является переменное количество репликаций, зависящее от особенностей решаемой задачи. Надо понимать, что в таком случае всегда будет реализовано заданное минимальное количество репликаций. Остальные репликации (вплоть до максимального ограничения) будут выполнены при отсутствии хотя бы одного из следующих условий:

— доверительный интервал для всех репликаций в рамках текущей итерации достаточно мал, чтобы попасть в интервал, заданный степенью доверия;

— значение целевой функции в текущей репликации существенно отстоит от текущего лучшего значения, следовательно, оптимальное решение найдено быть не может [8].

Проведенные оптимизационные эксперименты показали следующие особенности поведения системы:

— дополнительная память выделяется по максимуму — это ускоряет прием новых запросов;

— время выполнения программы достигает промежуточного, а не минимального значения, как это можно было бы ожидать; то же самое касается среднеквадратичного отклонения: оптимизатор находит компромисс между количеством выполненных программ и временем простоя терминалов;

— левая граница времени восстановления системы сдвигается не к минимальному, а к максимальному значению: второй сервер не подвержен риску отказа по условию.

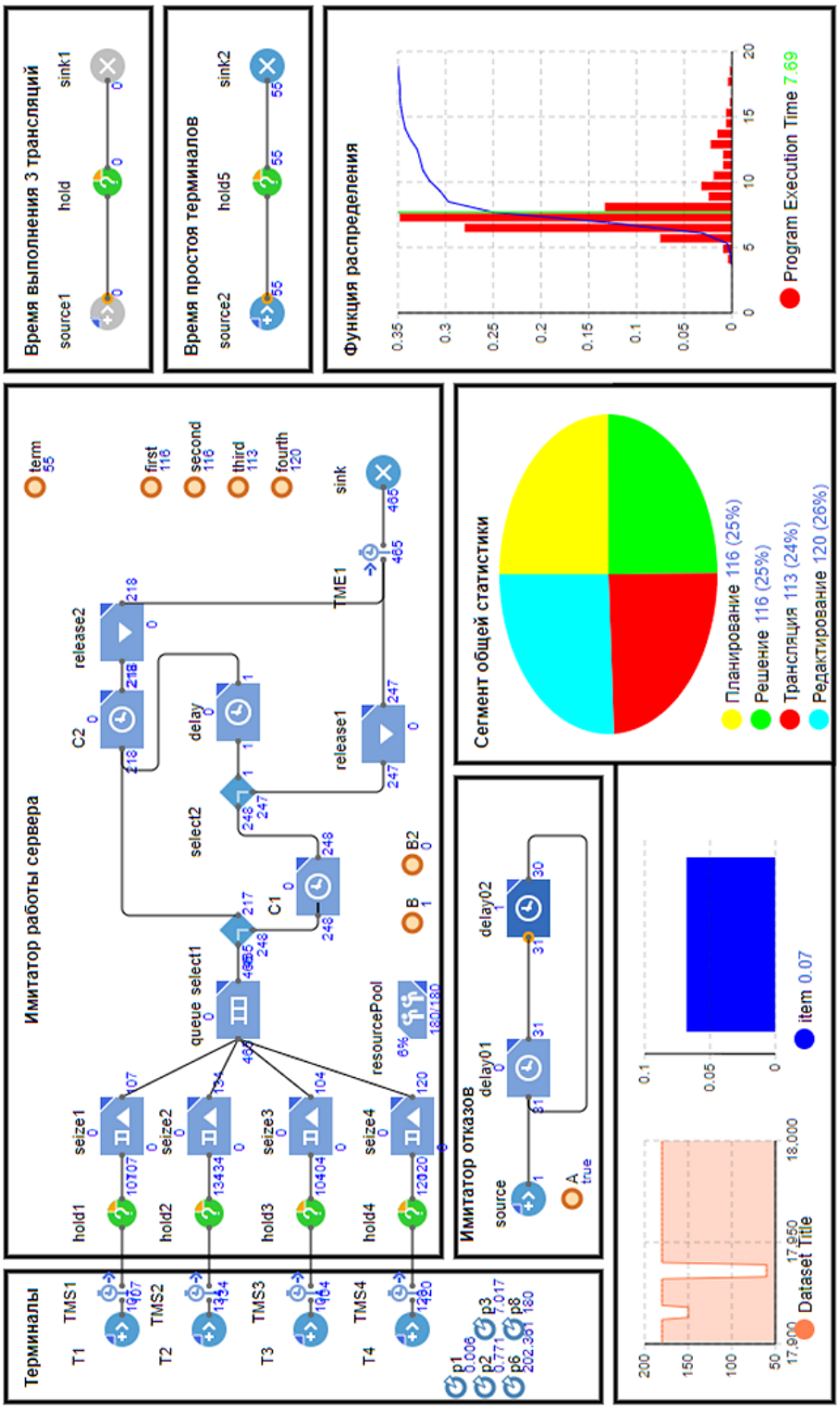

Рис. 3. Анализ модели в режиме реального времени

Fig. 3. Real-time model analysis

Обсуждение

Результаты оптимизационных экспериментов позволяют сделать следующие предположения относительно построенной модели:

— использование модели клиента, ограничивающей обращения к серверу, позволяет довольно гибко подстраивать работу сервера под поступающую нагрузку;

— смещение границ времени восстановления демонстрирует выполнение принципа оптимальности по Парето: выигрыш по одним критериям осуществляется за счет ухудшения других.

Учитывая, что система должна быть готова работать в условиях высокой нагрузки, можно ужесточить ограничение простоя терминалов до 1,5% от общего времени моделирования, а время между прибытиями запросов оставим первоначальным — 160 с. Количество итераций для поиска оптимума оставим равным 250 у. е., а штраф за прерывание программ установим в 150 у. е. Результат оптимизации показан в виде графика (рис. 4), который наглядно демонстрирует стохастическую природу модели и скачкообразные переходы из одного состояния целевой функции в другое. Уменьшенное количество репликаций хотя и повышает погрешность расчетов, помогает сократить вычислительную сложность оптимизации и ускорить процесс получения результата. Полученный в результате данного оптимизационного эксперимента выигрыш в показателях отражен в таблице 4.

Согласно приведенным в таблице 4 данным, оптимизация модели дает существенный результат. Важно выделить тот факт, что текущая модель является достаточно абстрактной, отражающей логику клиент-серверного взаимодействия с ограничением на количество и типы операций, а также ограничением по памяти сервера. Дополнительно могут быть введены ограничения на количество процессоров, предел времени ожидания ответа от сервера и т. д. Следует также отметить, что использование среды AnyLogic не накладывает существенных

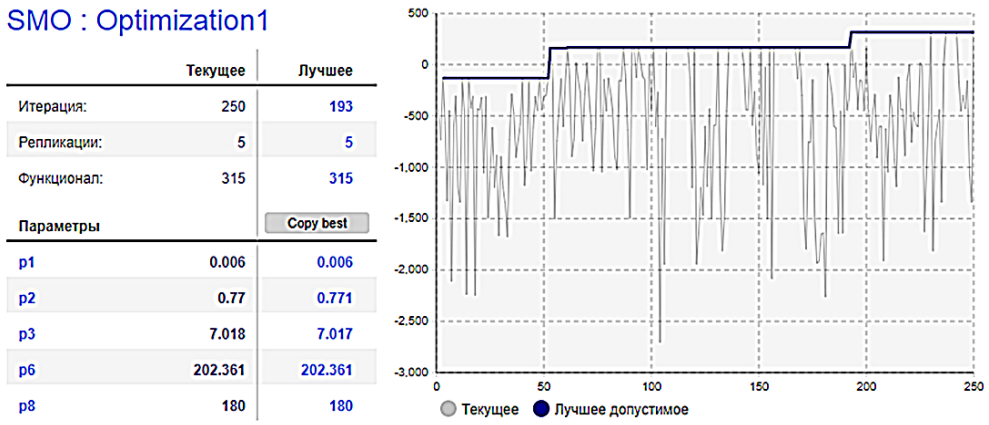

Рис. 4. Оптимизация при ужесточенных условиях

Fig. 4. Optimization under tough conditions

Таблица 4

Характеристика условной оптимизации целевой функции

Table 4

Characteristic of conditional optimization of the objective function

Параметр	До оптимизации	После оптимизации
Параметр загрузки процессора (по памяти)	14%	7%
Параметр загрузки сервера (по коэффициенту загрузки)	26%	18%
Вероятность простоя терминалов	6%	0,3%
Частота одновременного выполнения трансляции с трех терминалов	0	0
Частота отказов основного компьютера	0,00180	0,00005
Число прерванных программ	8	1

ограничений на природу и масштаб описываемых процессов [2], поэтому детализация модели и добавление новых зависимостей остается на усмотрение исследователя. В целом можно выделить следующие уточнения, которые помогут сделать полученную модель более полной и достоверной:

— введение функциональной зависимости между коэффициентом загрузки сервера и частотой отказов. Такая зависимость будет отражать ресурс используемого оборудования максимальным количеством отказов, основываясь на положениях теории надежности;

— введение зависимости между границами восстановления, а также внешними факторами. Время восстановления системы может зависеть как от порядкового номера отказа, так и от сложности самого сервера, причин отказа и множества других факторов, которые здесь трудно было бы перечислить;

— динамическое изменение частоты и порядка поступления запросов разных типов от разных клиентов. Так, в полученной модели все запросы генерируются со случайным типом без учета их возможной логической последовательности, когда, например, при создании пустой базы данных операции чтения для нее являются лишенными смысла. Конечно, такое уточнение должно быть основано на специфике решаемых задач;

— динамическое увеличение количества клиентов и анализ эффективности функционирования сервера в новых условиях. В реальности количество клиентов сервера может постепенно наращиваться, что потребует либо изменения конфигурации самого сервера, либо способа взаимодействия сервера с клиентами. Учитывая, что на такого рода модернизацию будут направлены конкретные финансовые затраты, можно будет выявить пороговое значение клиентов, при котором будет обеспечено заранее обозначенное качество обслуживания;

— разделение каждого типа решаемых задач на микрозадачи. Конечный список микрозадач может использоваться для имитации выполнения задач с помощью технологий OpenMP или MPI, выделения последовательной и параллельной части программ, вычисления показателей, необходимых для оценки эффективности распараллеливания по законам Амдала и Густафсона.

Так или иначе любая модель предусматривает некоторую степень редукции и отражает лишь те черты оригинала, которые наиболее существенны в конкретном исследовании.

СПИСОК ЛИТЕРАТУРЫ

1. Кислицын Е. В. Моделирование систем: дискретно-событийный подход / Е. В. Кислицын, В. К. Першин. Екатеринбург: Урал. гос. экон. ун-т, 2013. 101 с.
2. Макаров В. Л. Разработка цифровых двойников для производственных предприятий / В. Л. Макаров, А. Р. Бахтизин, Г. Л. Бекларян // Бизнес-информатика. 2019. № 13 (4). С. 7-16.
3. Обухов П. А. Исследование эффективности работы сетевых серверов в среде имитационного моделирования AnyLogic / П. А. Обухов, А. Б. Николаев, А. В. Остроух // Международный журнал экспериментального образования. 2015. № 3-3. С. 338-342.
4. Оптимизационный эксперимент. URL: https://help.anylogic.ru/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fexperiments%2Foptimization-experiment.html (дата обращения: 11.10.2021).
5. Ржевский С. В. Исследование операций / С. В. Ржевский. СПб.: Лань, 2013. 480 с.
6. Ротт А. Р. Моделирование и расчеты производственно-технических систем / А. Р. Ротт. Йошкар-Ола: Мар. гос. техн. ун-т, 2010. 224 с.
7. Стин ван М. Распределенные системы / ван М. Стин, Э. С. Таненбаум; пер. с англ. В. А. Яроцкого. М.: ДМК Пресс, 2021. 584 с.
8. Эксперимент Монте-Карло. URL: https://help.anylogic.ru/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fexperiments%2Fmonte-carlo-experiment.html (дата обращения: 11.10.2021).
9. Arora R. Optimization: Algorithms and Applications / R. Arora. Boca Raton: Chapman and Hall/CRC, 2015. 466 p.
10. Luc D. T. Multiobjective Linear Programming: An Introduction / D. T. Luc. London: Springer, 2016. XII, 325 p. DOI: 10.1007/978-3-319-21091-9

Lyudmila B. SENKEVICH[1]
Marat A. SABITOV[2]

UDC 004.94

SIMULATION MODELING AND OPTIMIZATION OF THE OPERATION OF A PARALLEL SERVER WITH FAILURES IN ANYLOGIC

[1] Cand. Sci. (Ped.), Associate Professor,
Department of Cybernetic Systems,
Tyumen Industrial University
lyudmila1@yandex.ru

[2] Master Student,
Department of Cybernetic Systems,
Tyumen Industrial University
sabitov.m.a@yandex.ru

Abstract

Modern scientific research is increasingly raising issues of the processing of large amounts of data. The widespread use of client-server interaction technology and cloud computing at the moment raises questions about the efficiency of a parallel server, as well as the ability to predict results depending on the degree of load and characteristics of the equipment.

This article simulates a parallel server with failures in the AnyLogic environment, and then performs multidimensional optimization by the weighted sum method. As part of the study, a simulation model of a queuing system with failures was built. It contains a server simulator, terminals, a failure simulator and statistics collection segments. The used parallel server model is abstract and rather generalized and makes it possible to concretize it by introducing additional dependencies and refining characteristics. The experiment with optimal parameters allowed to obtain the following gain in system efficiency indicators: processor load parameter (by memory) — a gain of 7%; processor load parameter (by load factor) — a gain of 8%; probability of terminal downtime — a gain of 5.7%; the failure rate of the main computer — 36 times less

Citation: Senkevich L. B., Sabitov M. A. 2022. “Simulation modeling and optimization of the operation of a parallel server with failures in AnyLogic”. Tyumen State University Herald. Physical and Mathematical Modeling. Oil, Gas, Energy, vol. 8, no. 1 (29), pp. 126-143.
DOI: 10.21684/2411-7978-2022-8-1-126-143

than the initial configuration; the number of interrupted programs — 7 less. In addition, it should be noted that the total number of completed requests remained at the same level — 462-465, for the reason that the intensity of the terminals did not vary.

Since the results of replications ("runs") are unique and the values of the optimized function vary for different replications, the built-in possibility of a variable number of replications (from 5 to 10) with a confidence probability of 95% and an error level of 0.5 was used. The obtained results suggest the possibility of further research of the model and its development in the AnyLogic environment.

Keywords

Simulation modeling, queuing system, stochastic model, parallel server, multidimensional optimization, weighted sum method, AnyLogic.

DOI: 10.21684/2411-7978-2022-8-1-126-143

REFERENCES

1. Kislitsyn E. V., Pershin V. K. 2013. System modeling: discrete-event approach. Ekaterinburg: USUE. 101 p. [In Russian]
2. Makarov V. L., Bakhtizin A. R., Beklaryan G. L. 2019. "Development of digital twins for manufacturing enterprises". BusinessInformatics, no. 13 (4), pp. 7-16. [In Russian]
3. Obukhov P. A., Nikolaev A. B., Ostroukh A. V. 2015. "Study of the efficiency of network servers in the AnyLogic simulation environment". International Journal of Experimental Education, no. 3-3, pp. 338-342. [In Russian]
4. Optimization experiment. Accessed on 11 October 2021. https://help.anylogic.ru/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fexperiments%2Foptimization-experiment.html [In Russian]
5. Rzhevskiy S. V. 2013. Operation Research. Saint-Petersburg: Lan. 480 p. [In Russian]
6. Rott A. R. 2010. Modeling and calculations of production and technical systems. Yoshkar-Ola: Volga State University of Technology. 224 p. [In Russian]
7. Steen van M., Tanenbaum A. S. 2021. Distributed systems. Moscow: DMK Press. 584 p. [In Russian]
8. The Monte Carlo Experiment. Accessed on 11 October 2021. https://help.anylogic.ru/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fexperiments%2Fmonte-carlo-experiment.html [In Russian]
9. Arora R. 2015. Optimization: algorithms and applications. Boca Raton: Chapman and Hall/CRC. 466 p.
10. Luc D. T. 2016. Multiobjective linear programming: an introduction. London: Springer. XII, 325 p. DOI: 10.1007/978-3-319-21091-9