УДК 004.051
DOI 10.25205/1818-7900-2018-16-1-100-112

А. Н. Родионов [1], **Н. В. Эйрих** [2], **О. Я. Дубей** [2]

[1] *Вычислительный центр ДВО РАН*
ул. Ким Ю Чена, 65, Хабаровск, 680000, Россия

[2] *Приамурский государственный университет имени Шолом-Алейхема*
Широкая ул., 70, Биробиджан, 679015, Россия

ran@newmail.ru, nadya_eyrikh@mail.ru, olesya_dubey@mail.ru

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ: ТРАНЗАКЦИОННАЯ МОДЕЛЬ И АРХИТЕКТУРА ИМИТАЦИОННОЙ СИСТЕМЫ [*]

Высокий уровень изменчивости программного и технического обеспечения информационных систем требует получения оценок их производительности за ограниченные сроки на любых стадиях их жизненного цикла. В статье рассматривается унифицированный процесс обработки данных в клиент-серверном приложении, которое взаимодействует с базой данных, состоящей из стандартных компонентов. Формальная модель процесса представлена одноканальной системой массового обслуживания с входящим потоком требований, интегрирующим элементарные потоки, каждый из которых ассоциируется с определенным транзакционным классом. Излагаются актуальные вопросы реализации модели и приводятся результаты имитационных экспериментов, отражающие продолжительность отклика системы на задаваемую рабочую нагрузку.

Ключевые слова: оценка производительности, время отклика, клиент-серверная система, имитационное моделирование, система массового обслуживания.

Введение

Любое многопользовательское приложение баз данных (ПБД), реализованное в клиент-серверной архитектуре, в процессе своего создания или после очередной модернизации должно подвергаться нагрузочному тестированию, главная цель которого – установление фактических значений производительности для заданных параметров рабочей нагрузки, показательной для той предметной области, где ПБД эксплуатируется. Среди используемых сегодня показателей производительности, таких как доступность, загруженность, пропускная способность и продолжительность отклика [1; 2], последний показатель считается основным.

Качество проводимого тестирования во многом определяется тем, насколько адекватно тестовая среда воспроизводит функционирование ПБД, инфраструктуру информационной системы и саму рабочую нагрузку. В подавляющем большинстве случаев [3–10] тестовая среда представлена системами массового обслуживания, с разной степенью детализации

[*] Работа выполнена при поддержке программы фундаментальных исследований ДВО РАН «Дальний Восток» (проект № 18-5-104).

Для выполнения расчетов были использованы вычислительные ресурсы ЦКП «Центр данных ДВО РАН»: сайт. – Хабаровск: ВЦ ДВО РАН. – URL: http://lits.ccfebras.ru

Родионов А. Н., *Эйрих Н. В.*, *Дубей О. Я.* Исследование производительности приложений баз данных: транзакционная модель и архитектура имитационной системы // Вестн. НГУ. Серия: Информационные технологии. 2018. Т. 16, № 1. С. 100–112.

и точности, отражающих процессы обработки данных в ПБД. Несмотря на большое число исследований, выполненных в этой области, на практике эти модели не получили широкого распространения ввиду недостаточной точности их предсказательных возможностей [3], что можно объяснить следующими обстоятельствами.

В основе всех моделей лежат транзакции, представляющие собой запросы к базе данных, и некоторый алгоритм, в соответствии с которым транзакции выполняются той или иной серверной системой управления базой данных (СУБД). При этом номенклатура разновидностей (классов, типов) транзакций может быть сколь угодно большой и варьироваться от одной предметной области к другой в достаточно широких пределах. То же самое относится к интенсивностям входящих транзакционных потоков, меняющихся с течением времени. При этом лишь небольшая часть классов транзакций отличается действительно высокими интенсивностями. Большинство из них инициируются эпизодически и могут не оказывать серьезного влияния на производительность ПБД, если задан и поддерживается, например, корректный регламент обслуживания таких транзакций. Отсюда одна из задач, возникающая при моделировании процессов обработки данных, – локализация и формализация транзакций, характеризующихся наивысшей интенсивностью.

Два других обстоятельствах, сказывающихся на качестве моделирования, – высокие степени неоднозначности и изменчивости среды, которая призвана обеспечить выполнение транзакций.

В последнее время наметилась тенденция к стремлению максимально оградить пользователя от участия в процессах, связанных с обработкой транзакций. По сути, разработчики СУБД инкапсулировали соответствующие алгоритмы и используемые структуры данных и исключили возможность заранее предугадать, насколько быстро и когда будет выполнена конкретная транзакция. В этом проявляется неоднозначность.

Относительно изменчивости отметим следующее. Переход к очередным версиям СУБД или средам разработки прикладного программного обеспечения, если они существенным образом затронули технологии работы с транзакциями, могут свести к нулю и полученные ранее результаты моделирования, и сделать некорректными сами модели.

Все перечисленное делает актуальным построение таких моделей, структура которых, во-первых, представлена неизменяемыми и в то же время существенными элементами, определяющими производительность, и, во-вторых, в состоянии воспроизвести и реальную промышленную нагрузку в виде потоков разнородных транзакций, и саму среду выполнения.

Настоящая работа направлена на решение обозначенных выше задач. Основной результат – формирование рамочной модели, имитирующей процессы обработки данных в многопользовательских приложениях с централизованной базой данных. Модель должна позволять получить оценку продолжительности отклика клиент-серверной системы для заданных интенсивностей входящих потоков требований (SQL-запросов) и мощностей отношений баз данных.

Стандартная архитектура ПБД и показатели производительности

За время непродолжительной эволюции, насчитывающей несколько десятков лет, сформировались представления о том, какой должна быть эффективная организация (архитектура) ПБД. Архитектура, по сути, рамочно задает максимальные возможности системы, в том числе и предельные характеристики ее производительности. Возьмем за основу клиент-серверную архитектуру, ставшую сегодня де-факто стандартом для ПБД, и выделим в ней те элементы, которые будут существенно влиять на значения, принимаемые показателями производительности. (Последние будут перечислены позднее.)

Клиент-серверная архитектура в зависимости от выбранного метода доступа может быть реализована в трех потенциальных вариантах: отсоединенного доступа, прямого доступа и некоторой комбинации первых двух. На сегодняшний день отсоединенный режим считается наиболее предпочтительным по сравнению с режимом прямого доступа, так как позволяет получить значительный выигрыш в скорости выполнения запросов за счет сокращения времени блокировок ресурсов. Поэтому будем на него ориентироваться в дальнейшем.

Отсоединенный доступ предполагает, что клиентское приложение соединяется с базой данных на непродолжительное время только для получения или записи информации и использует собственную подсхему для представления данных. Элементы такой архитектуры показаны на рис. 1. Функции двустороннего обмена и передачи запросов реализует поставщик данных.

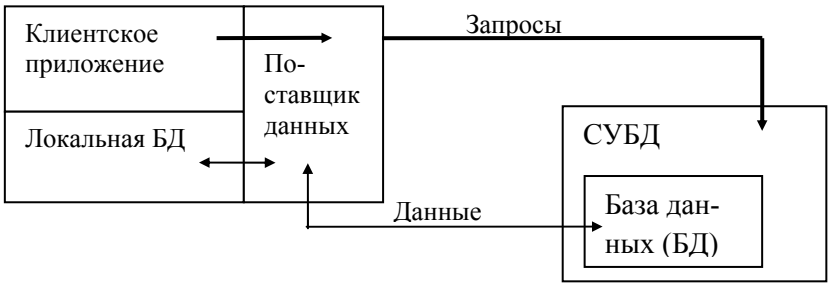

Рис. 1. Элементы клиент-серверной архитектуры ПБД, использующей отсоединенный доступ

Определившись с базовыми элементами и конфигурацией моделируемой системы (ПБД), покажем далее, что будет пониматься под производительностью.

Большинство исследователей и практиков обоснованно считает [1–10], что производительность – это время (продолжительность) отклика системы на запросы, поступающие от пользователей этой системой. Продолжительность в зависимости от структуры запросов и нагрузки на сервер может варьироваться в достаточно широких пределах – от нескольких миллисекунд до часов. ПБД включает большое число разновидностей (классов) запросов q, которые инициируются и выполняются в случайные моменты времени τ_q. Для описания входных потоков требований, каждый из которых ассоциируется с q-запросом, необходимо знать законы распределений случайных величин τ_q, образующих множество $T, T = \{\tau_q\}_1^Q$. Продолжительность выполнения отдельного q-го запроса также будет представлять собой случайную величину ψ_q с неизвестным законом распределения.

Логично для представленной архитектуры ПБД в качестве показателей производительности использовать ряд оценок распределений ψ_q: среднее время выполнения каждого класса (категории) запроса к БД – $\overline{\psi}_q$, а также вероятность того, что эта продолжительность с заданной надежностью γ не выйдет за определенные границы – $P\left(\overline{\psi}_q + D\left(\psi_q\right) \le \gamma\right) = s_q$, где $D\left(\psi_q\right)$ – дисперсия случайной величины ψ_q; γ – предельное значение вероятности.

Последний показатель дает возможность учесть разброс значений ψ_q относительно $\overline{\psi}_q$. Это требование вытекает из практики. Мало кого устроит, например, если 30 % запросов о состоянии банковского счета будет выполняться с продолжительностью 20 и более секунд. Это, конечно, условные цифры. Тем не менее заметим, что значения элементов множеств $\overline{\Psi}_q = \left\{\overline{\psi}_q\right\}_1^Q$ и $S_q = \left\{s_q\right\}_1^Q$ могут рассматриваться в качестве требований к информационной системе (ИС).

Массовые запросы ПБД

Одна из главных сложностей построения моделей и последующего моделирования процессов обработки данных в ПБД – многообразие конфигураций потенциальных запросов, которые могут присутствовать в программах, и, как одно из следствий, трудности с нахожде-

нием частотных распределений τ_q. В действительности же массовые запросы, к которым будем относить запросы с наиболее высокими значениями входных интенсивностей λ_q, отличает стандартная конфигурация, вытекающая из стандартной архитектуры приложений, а также стандартной организации баз данных, представленной типизированными структурами. Поскольку в вопросе унификации массовых запросов типизация структур БД не менее важна, чем выбор архитектуры, изложим соображения и решения, касающиеся одной из возможных типизаций. Будем при этом руководствоваться назначением БД, которая призвана регистрировать факты, и способом, которым это делает СУБД.

Факты – это привязанные ко времени взаимодействия, устанавливающиеся между объектами, из которых состоят предметные области. Факты фиксируются в документах и в базах данных могут размещаться либо в «документальных» структурах, либо в «слабых сущностях». Использование «документальных структур» более предпочтительно, так как исключается необходимость в дублировании однозначных атрибутов, характеризующих документы.

В отличие от фактов, информация об объектах содержится в «справочных» структурах. Одним из признаков последних является присутствие в них семантических сведений либо ссылок на таковые, если используются семантические объекты.

Если не углубляться в дальнейшую типизацию, которая несущественна для задач, решаемых в настоящей работе, структурную часть базы данных в своей основе составляют справочные и документальные структуры. Продемонстрируем сказанное на небольшом примере организации структур, содержащих данные об отметках, получаемых студентами в ходе сдачи ими экзаменов (рис. 2). Возьмем за основу «экзаменационную ведомость». Логическая модель документа представлена двумя документальными структурами: Examination_record_list_Head и Examination_record_list_Details, а также «справочниками», которые поставляют семантику для записей документальных структур.

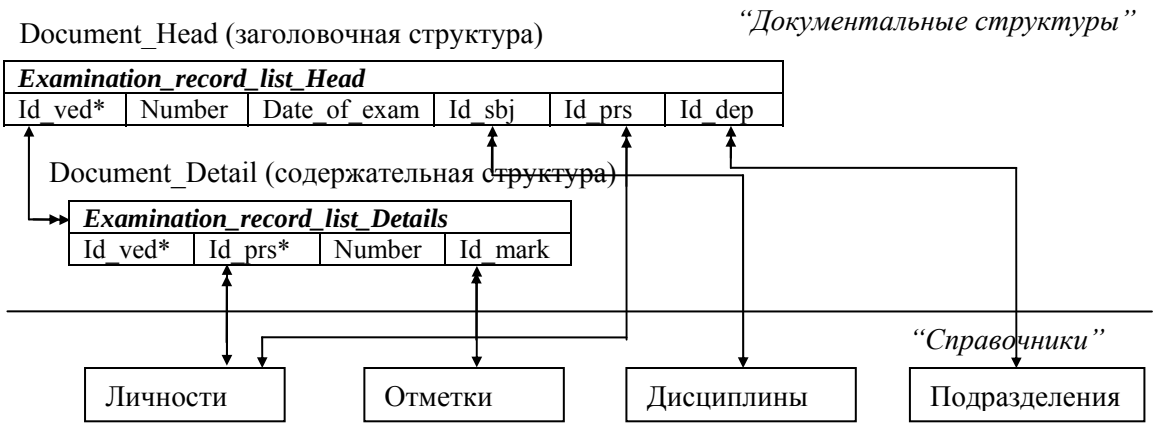

Рис. 2. «Документальные» и «справочные» структуры логической модели базы данных

Корректная работа с документом в информационной системе по умолчанию предполагает, что в произвольный момент времени доступ к документу имеет один и только один пользователь. В противном случае всегда есть риск получения рассогласованных данных. Отсюда элементарными, типичными и одновременно массовыми операциями, выполняемыми в ПБД, будут операции на получение (чтение) и корректировку (запись) данных, сосредоточенных в документальных структурах.

Продемонстрируем общую структуру запросов, производящих чтение и запись данных (рис. 3).

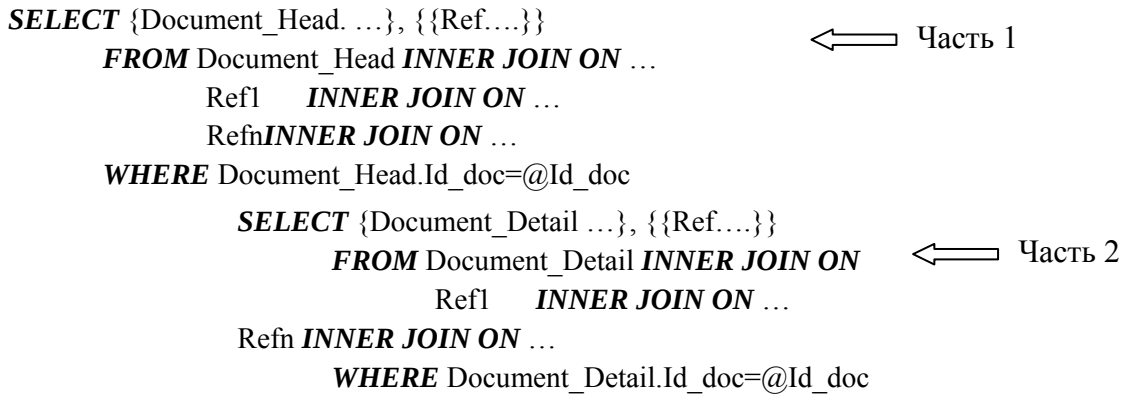

Рис. 3. Структура запроса на чтение данных

Первая часть запроса обеспечивает получение данных из заголовочной таблицы (структуры), вторая – из содержательной. За ссылку на конкретный документ отвечает параметр *WHERE* и переменная @Id_doc, которой предварительно присваивается значение идентификационного кода документа.

Противоположный запрос, предназначенный уже для записи данных, может либо ограничиться инструкциями UPDATE и DELETE, если данные только обновлялись, либо включать два подзапроса: один на удаление (DELETE) и один на вставку данных (INSERT). (Последний вариант может оказаться более эффективным с точки зрения производительности выполнения, так как количество записей, сосредоточенных в одном документе, в основном варьируется в незначительных пределах и в редких случаях превышает 100 единиц.)

Таким образом, каждому классу документа можно поставить в соответствие по одному стандартному запросу: один на чтение данных, другой – на запись.

Модель обработки данных

Присутствие входного потока требований в виде запросов, поступающих на сервер (прибор обслуживания) в случайные моменты времени, и необходимость в оценке продолжительности выполнения (обслуживания) этих запросов свидетельствуют в пользу применения в качестве математической модели исследуемого процесса системы массового обслуживания (СМО).

Согласно документации, описывающей организацию и функционирование таких распространенных СУБД, как ORACLE и MS SQL, СМО включает встроенную очередь с дисциплиной обслуживания FIFO – «первый пришел – первым обслужен». Входящий поток требований представлен совокупностью локальных потоков, каждый из которых соответствует q-му запросному классу (рис. 4).

Применительно к каждому локальному потоку (если брать в расчет характерный для конкретной предметной области временной интервал) можно с высокой степенью уверенности утверждать, что поток обладает всеми признаками простейшего пуассоновского потока: ординарностью, стационарностью и отсутствием последействия. (Несмотря на то что в общем случае, это нестационарный пуассоновский процесс, характеризующийся совокупностью $\lambda_q(t)$, соответствующих различным временным интервалам, для исследования производительности ПБД будем принимать в расчет интервал времени с максимальной интенсивностью.)

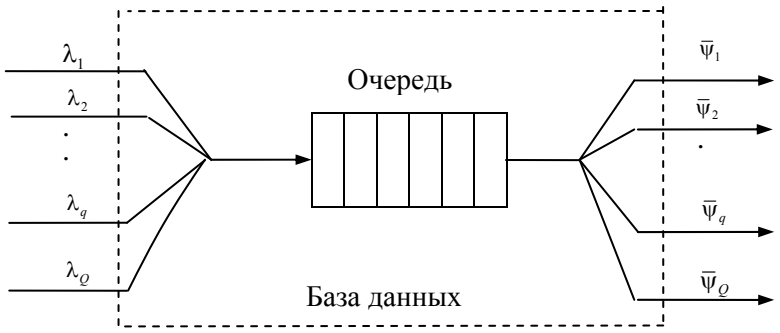

Рис. 4. Элементы СМО, моделирующие клиент-серверное взаимодействие в ПБД

Зная λ, можно получить количество запросов, которые будут поступать на сервер в течение заданного интервала времени. Но для имитации процесса обработки данных необходимы данные другого рода – моменты времени поступления требований. Воспользуемся стандартным преобразованием, согласно которому, если входной поток является простейшим, то вероятность моментов времени поступления требований можно найти на основании экспоненциального распределения: $F(t) = 1 - e^{-\lambda t}$. Тогда, интервал времени t между поступлениями требований составит $t = -\left(\frac{1}{\lambda}\right)\ln(1-R)$, где R – случайное число, равномерно распределенное в интервале [0, 1]. Поскольку $(1 - R)$ – также равномерно распределенное число в том же интервале, можно произвести замену $(1 - R)$ на R.

Неизвестной величиной в представленной модели окажутся случайные величины ψ_q, законы распределения которых предстоит найти в ходе имитационных экспериментов с этой моделью.

Процедура моделирования обработки заявок входящего потока требований

Как уже было замечено, одна из особенностей современных клиент-серверных СУБД состоит в максимальной степени их закрытости, что затрудняет организацию и проведение имитационных экспериментов. Закрытость, в частности, проявляется в том, что СУБД берут на себя полное управление дисциплиной и постановкой запросов в очередь. Поэтому задать время старта очередного запроса, что требуется условиями эксперимента, не прибегая к каким-то нестандартным программным решениям, крайне проблематично.

Решить возникшее затруднение можно двумя способами: либо запустить отдельные экземпляры приложений, включающих текст самого запроса и оператор, инициирующий выполнение запроса, либо воспользоваться потоками, обеспечивающими параллельное выполнение программ. В обоих случаях суммарное число и экземпляров, и потоков будет равно количеству запросов. В настоящей работе был реализован второй способ.

Покажем принципиальное различие в использовании однопоточной и многопоточной архитектур программной системы, имитирующей поступление и выполнение запросов в ПБД (рис. 5).

Пусть t_1^I и t_2^I – случайные моменты времени поступления на сервер запросов q_1 и q_2 соответственно, t_1^S, t_2^S – начало, а t_1^F, t_2^F – завершение выполнения этих запросов. Если продолжительность $\left[t_1^F - t_1^S\right]$ выполнения запроса q_1 (которая также является случайной величиной) меньше $\left[t_2^S - t_1^F\right]$, то, приостановив выполнение программы на $\left[t_2^S - t_1^F\right]$, можно ими-

тировать (посредством какой-либо из команд используемого языка моделирования) старт запроса q_2. В противном случае время старта запроса q_2 будет всегда больше или равно t_1^F, так как команда, инициирующая старт q_2, не может быть выполнена до тех пор, пока не завершится выполнение команды, запускающей q_1. (В однопоточном программном коде все команды выполняются строго последовательно.)

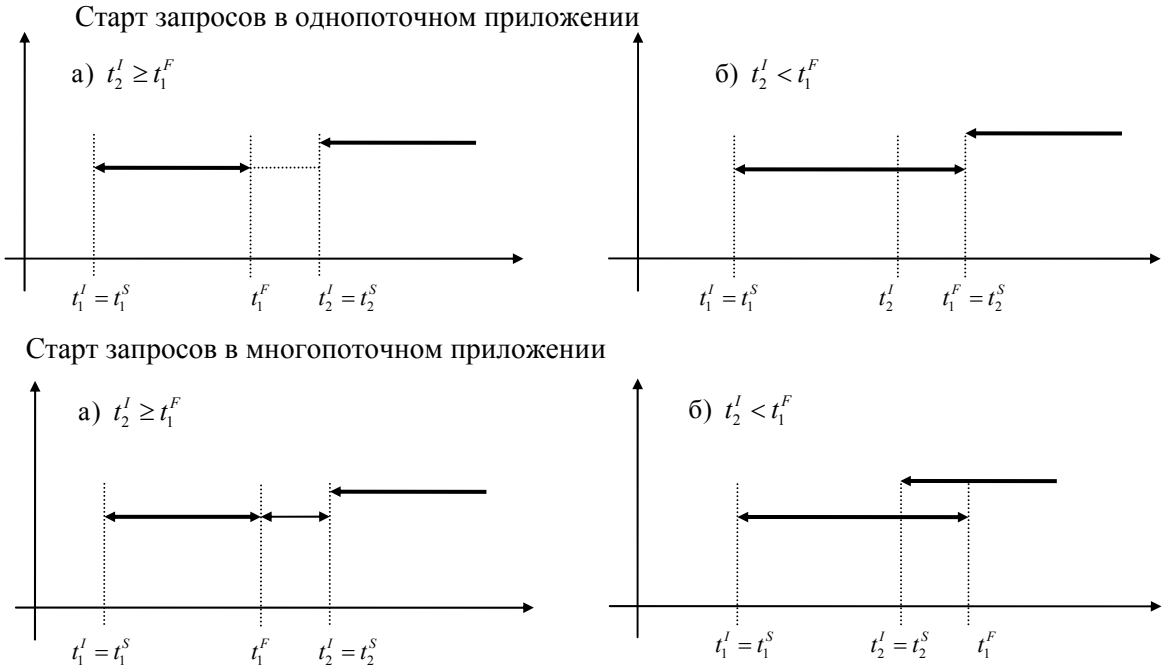

Рис. 5. Использование однопоточной и многопоточной архитектур для выполнения запросов

Использование потоков позволяет избежать возникновения «вынужденной» задержки старта очередного запроса. Каждый запрос может быть выполнен в собственном потоке, и тогда всегда $t_q^I = t_q^S$. (Обратим внимание на то, что в силу наличия серверной очереди время старта очередного запроса t_q^S – это время, когда запрос поступил на сервер и занял место в очереди на выполнение.) Ниже приводится текст программы на языке C# (листинг 1), имитирующей выполнение случайной последовательности запросов (инкапсулированных в транзакции) в потоках, со всеми необходимыми пояснениями. (Моделирующая система разработана в среде Microsoft Visual Studio 2017 с использованием ADO.NET и СУБД MS SQL 2012.)

Листинг 1

```
foreach (DataRowView GG in DV)
   { deltaT = (int)GG["DeltaT"];
      Thread.Sleep(deltaT);
      Th[parCickle] = new Thread(new ParameterizedThreadStart(ThreadProc));
      List_of_ParametrsLoP = new List_of_Parametrs((string)GG["Name_sp"], parCickle,
     (int)GG["Ammountrecords"], (char)GG["Parameters(Y/N)"]);
      Th[parCickle].Start(LoP);
      parCickle += 1;  }
```

```
public static void ThreadProc(Object Paramets)
  { DateTime start;
    string strConn, startQuery, name_sp;
    Random Rand = new Random();
    List_of_ParametrsLoP = new List_of_Parametrs();
    LoP = (List_of_Parametrs) Paramets;
    name_sp = (string) LoP.Name_sp;
    int ammountrecords = (int)LoP.Ammountrecords;
    char parametersYN = (char)LoP.ParametersYN;
    inttransaction_number = (int) LoP.Nomer_transaction;
    strConn = @"Data Source=ISE;Initial Catalog=University;Integrated Security=True;" +
    "Connection Timeout=200; Pooling=False; Asynchronous Processing=True;"+
    “MultipleActiveResultsets=true";
    SqlTransaction myTransaction;
    SqlConnection sqlCon;
    SqlCommand sqlCom;
    sqlCon = new SqlConnection(strConn);
    sqlCon.Open();
    myTransaction = sqlCon.BeginTransaction(Convert.ToString(transaction_number));
    sqlCom = new SqlCommand();
    sqlCom.Connection = sqlCon;
    sqlCom.CommandText = name_sp;
    sqlCom.CommandType = CommandType.StoredProcedure;
    sqlCom.CommandTimeout = 0;
    if (parametersYN=='Y')
        sqlCom.Parameters.AddWithValue("Id_doc",Rand.Next(1,ammountrecords) );
   sqlCom.Transaction = myTransaction;
  sqlCom.ExecuteNonQuery();
  sqlCom.Transaction.Commit();
  sqlCom.Connection.Close();}
```

Перед началом моделирования пользователь, посредством диалогового окна (рис. 6), выбирает из списка хранимых процедур, содержащих откомпилированные тексты запросов, те из них, которые будут участвовать в эксперименте, и задает интенсивность их появления λ_q .

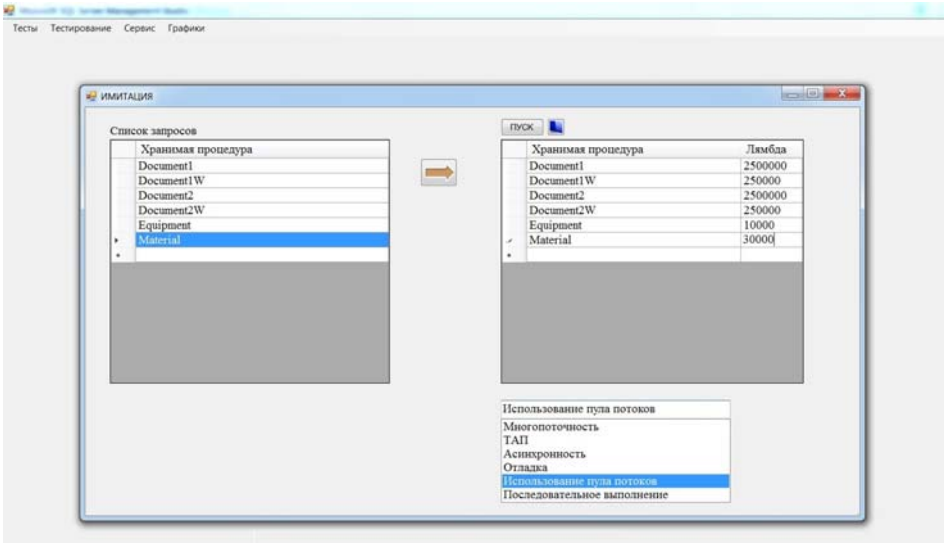

Рис. 6. Интерфейс главного окна программы,
имитирующей работу многопользовательского приложения

Инициация начала эксперимента посредством нажатия кнопки «старт» приводит к выполнению модуля, который рассчитывает моменты времени поступления запросов на сервер. Полученные данные записываются в экземпляр GG класса DataRowView, который представляет собой последовательный список. Далее, в цикле *foreach* на величину *deltaT*, определенную как разность между моментами времени старта смежных запросов, организуется задержка в основном потоке – *Thread.Sleep(deltaT)* и создается отдельный параметризированный поток, которому передается имя процедуры, запускающей на выполнение очередной запрос. Инструкция *Th[parCickle] = newThread(newParameterizedThreadStart(ThreadProc))* создает такой поток, а метод *Start* этого потока *Th[parCickle].Start(LoP)*, принимая набор входных параметров (класс запроса, признак обращения к документальным или справочным таблицам и ряд других, перечисленных в структуре *LoP*), инициирует выполнение запроса, посредством метода *ThreadProc.*

В теле метода *ThreadProc* выполняются следующие действия. Сначала создается набор инструментов, необходимых для выполнения запроса в потоке, а затем, с помощью инструкции *sqlCom.ExecuteNonQuery()*, выполняется сам запрос. Упомянутые инструменты – это команды на создание и открытие соединения с базой данных *sqlCon = new SqlConnection(strConn)* и *sqlCon.Open()*, а также инструкция myTransaction = sqlCon.BeginTransaction(Convert. ToString(transaction_number)), создающая и запускающая внутри открытого соединения транзакцию. Имя хранимой процедуры присваивается свойству CommandText экземпляра sqlCom.

Моменты времени начала и завершения выполнения транзакции (запроса) фиксируются средствами, предоставляемыми утилитой *SQLSERVERPROFILE*, входящей в пакет поставки MS SQL.

Исследование характера и параметров распределения времени обслуживания заявок

Разработанная и реализованная модель обработки запросов для системы, состоящей из централизованной базы данных и клиентских приложений, позволяет решать многие задачи, связанные с организацией информационных систем и поиском эффективных механизмов управления последними. Ограничимся исследованием характеристик случайной величины ψ_q, представляющей собой продолжительность времени обработки запроса, принадлежащего к классу документальных запросов.

Одна из задач, решаемых в ходе экспериментов, – получение распределений ψ_q для различных дискретных значений λ_q и мощностей документальных структур – V. Везде далее значение V будет соответствовать количеству записей, которые содержатся в заголовочной структуре. Количество записей, размещенных в содержательной структуре, составит $V \cdot 10$ (одной записи заголовочной структуры соответствует 10 записей содержательной). В качестве характерного периода для λ_q (индекс q будем далее опускать) примем промежуток времени продолжительностью в один час (3 600 000 мс), который характерен для достаточного большого числа предметных областей: образовательных, почтовых, медицинских, банковских, торговых и многих других.

Эксперименты показали следующее.

1. Ни один из возможных теоретических законов распределения случайной величины $\overline{\psi}$ (нормальное или логнормальное распределения) не может быть использован для аппроксимации экспериментальных данных. Так, например, для $\lambda = 5 \cdot 10^5$ и $V = 5 \cdot 10^4$ проверка по критерию Пирсона на соответствие нормальному закону показала, что $\chi^2_{\text{набл}} = 11367 > \chi^2_{\text{табл}}(\alpha = 0{,}95; k = 13) = 5{,}89$, где α – доверительный интервал, k – число степеней свободы. Для логнормального распределения $\chi^2_{\text{набл}} = 6392 > \chi^2_{\text{табл}}(\alpha = 0{,}95; k = 12) = 5{,}23$.

Вид полученных кривых (рис. 7) также не соответствует экспоненциальному закону распределения продолжительности времени обслуживания, который фигурирует, например, в [3; 6]. Справедливости ради стоит заметить, что на первый план выходят не законы распре-

деления времени обслуживания заявок, а перечисленные ранее характеристики таких распределений, участвующие в формировании критериев производительности ПБД, а также наборы управляющих параметров, которые в состоянии улучшить значения этих критериев.

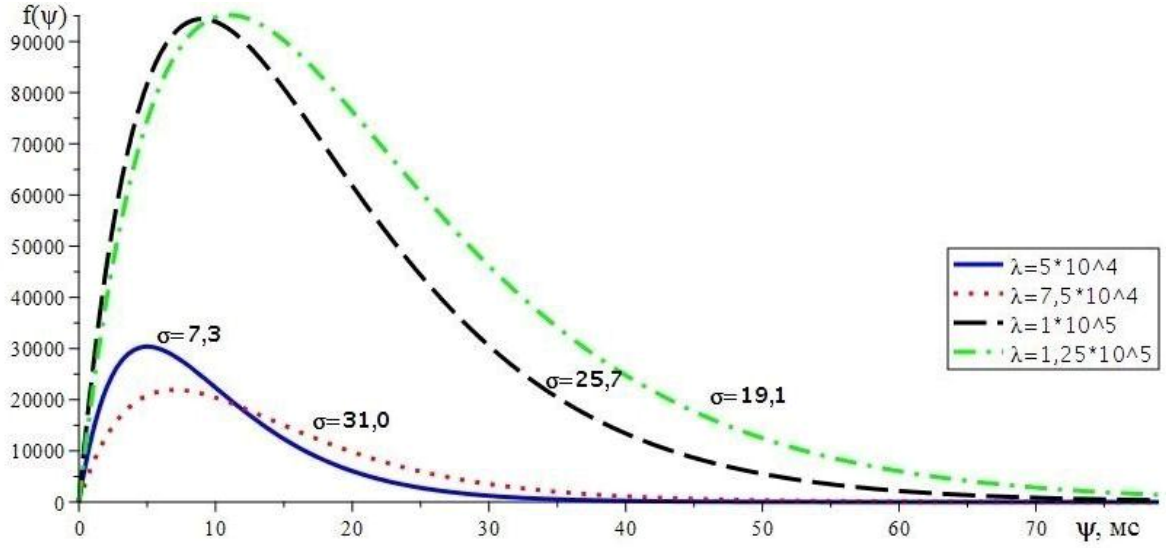

Рис. 7. Эмпирические кривые распределений случайной величины ψ для различных значений λ и $V = 5 \cdot 10^5$

2. $\overline{\psi}$ нелинейно зависит от λ (рис. 8). Это характерно только для тех диапазонов λ и V, которые менялись в процессе экспериментов $\left(\lambda = \left[5 \cdot 10^4, 1{,}25 \cdot 10^5\right], V = \left[5 \cdot 10^4, 1{,}25 \cdot 10^5\right]\right)$.

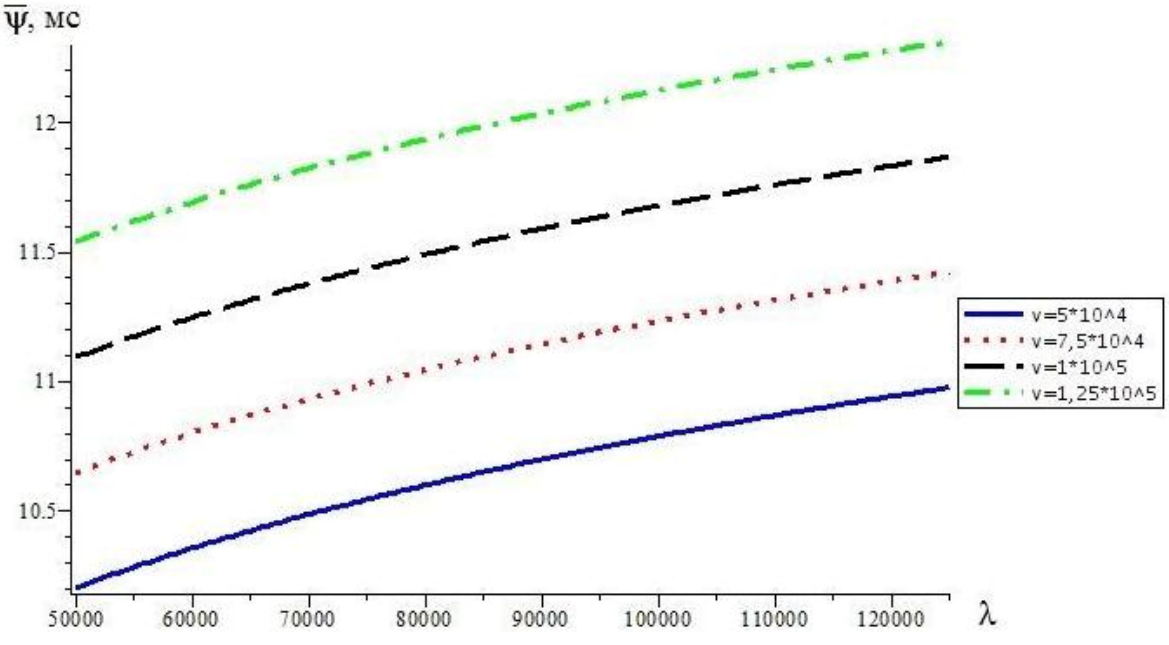

Рис. 8. Зависимость $\overline{\psi}$ от λ при различных значениях V

Логарифмический рост среднего времени выполнения запросов $\overline{\psi}$ объясним двумя моментами: способностью СУБД параллельно выполнять часть запросов и тем, что суммарная продолжительность обслуживания всех запросов для всех значений λ ни разу не вышла за пределы 1 ч (3 600 000 мс). Существенное увеличение λ (от $2 \cdot 10^6$ и выше) приводило к зависанию сервера и соответственно к его неспособности обрабатывать поступающие заявки. Очевидно, что нахождение предельных параметров производительности клиент-серверных приложений – одна из актуальных практических задач, которая может решаться посредством использования разработанной имитационной системы.

3. В связи с ростом накладных расходов операционной системы на поддержку программных потоков возрастание λ (влекущее за собой увеличение общего числа потоков) приводило к нарастанию запаздываний фактических моментов времени старта транзакций (рис. 9) по отношению к расчетному времени, которое находилось для заданных λ (потокам выделяется память, и требуется время, необходимое для их создания, управления и завершения [11; 12]).

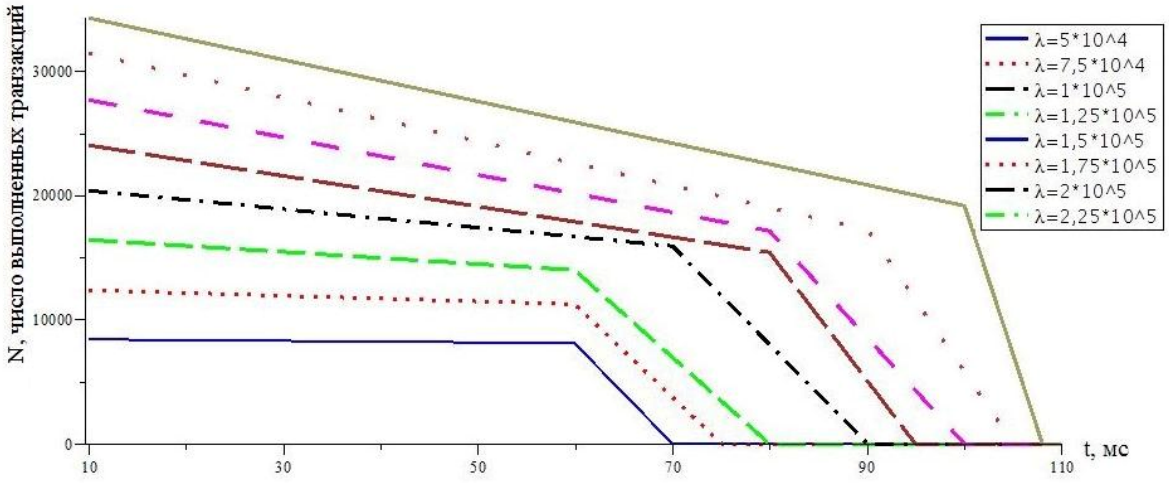

Рис. 9. Интенсивность обслуживания заявок

Эксперименты выполнялись на компьютере со следующими характеристиками: процессор – Intel(R) Core (TM) i5-2400, CPU – 3.1GHz, число ядер – 4, объем оперативной памяти – 4 ГБ. Использование вычислительной системы с более высокими характеристиками увеличивало корректную границу λ, в пределах которой транзакции стартовали в строго расчетное время. Можно заключить, что началу экспериментов с моделируемой системой должно предшествовать определение предельного значения λ для используемого вычислительного оборудования, при котором время старта последней транзакции не превысит 3 600 000 мс. Невыполнение данного условия говорит о необходимости выбора более производительного оборудования или использования иных, непоточных вычислительных моделей.

Заключение

Цель работы состояла в построении адекватной и по возможности простой модели, отражающей существенные аспекты клиент-серверного взаимодействия в приложениях баз данных и позволяющей за ограниченные сроки для заданных входных условий получать оценки производительности таких систем.

Проведенные эксперименты показали пригодность модели для решения поставленной задачи, одновременно обнаружив сложности в ее реализации, связанные с ограниченной возможностью вычислительной системы, состоящей из одного или двух компьютеров, имитировать работу реальной вычислительной среды, включающей сервер баз данных и сколь угодно большое число клиентов.

Возможности разработанной и реализованной в виде программного стенда модели не ограничиваются исследованием производительности ПБД. Перечислим задачи, которые актуальны и требуют решения:

- исследование максимальных (предельных) возможностей программно-технической системы, реализующей функции приложений баз данных;
- управление входными потоками требований, направленное на минимизацию продолжительности отклика системы;
- конфигурирование параметров среды выполнения (таких как блокировки, индексы, буферы, дисциплина очереди и т. д.) под требуемые значения показателей производительности.

Список литературы

1. *Гудсон Д., Стюард Р.* Практическое руководство по доступу к данным. СПб.: БХВ-Петербург, 2013.

2. *Рихтер Д.* CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. СПб.: Питер, 2013.

3. *Molyneaux I.* The art of application performance testing. O'Reilly, 2009

4. *Osman R., Knottenbelt W. J.* Database system evaluation models: a survey // Performance evaluation. 2012. Vol. 69, № 10. P. 471–493.

5. *Koziolek H.* Performance evaluation of component-based software systems: a survey // Performance evaluation. 2010. Vol. 67, № 8. P. 634–658.

6. *Menasce D. A., Gomaa H.* A method for design and performance modeling of client / server systems // IEEE Transactions on software engineering. 2000. Vol. 26. P. 1066–1085.

7. *Gijsen B. M. M., Mei R. D. van der, Engelberts P., Berg J. L. van den, Wingerden K. M. C. van.* Sojourn time approximations in queuing networks with feedback // Performance evaluation. 2006. Vol. 63, №8. P. 743–758.

8. *Kattepur A., Nambiar M.* Service demand modeling and performance prediction with single-user tests // Performance evaluation. 2017. Vol. 110, № 8. P. 1–21.

9. *Urgaonkar B., Pacifici G., Shenoy P., Spreitzer M., Tantawi A.* An analytic model for multi-tier internet services and its applications // Proc. of the 2005 ACM SIGMETRICS International Conference on measurement and modeling of computer systems, ACM. Banff, Alberta, Canada, 2005. P. 391–302.

10. *Menansce D., Bennani M.* Analytic performance models for single class and multiple class multithreaded software servers // Proc. of the international computer measurement group conference. Reno, NV, USA, 2006. P. 475–482.

11. *Elnikety S., Dropsho S., Cecchet E., Zwaenepoel W.* Predicting replicated database scalability from standalone database profiling // Proc. of the 4th ACM European conference on computer systems, ACM. Nuremberg, Germany, 2009. P. 303–306.

12. *Bernstein P. A., Newcomer E.* Principles of transaction processing. San Francisco, CA: Morgan Kaufmann, 2009.

Материал поступил в редколлегию 06.02.2018

A. N. Rodionov [1], **N. V. Eirikh** [2], **O. Ya. Dubey** [2]

[1] *Computer Center FEB RAS*
65 Kim U Chen Str., Khabarovsk, 680000, Russian Federation

[2] *Sholom-Aleichem Priamursky State University*
70 Shirokaya Str., Biribidzhan, 679015, Russian Federation

ran@newmail.ru, nadya_eyrikh@mail.ru, olesya_dubey@mail.ru

A DATABASE PERFORMANCE EVALUATION STUDY: TRANSACTION MODEL AND SIMULATION SYSTEM ARCHITECTURE

A high level of software and hardware platforms changeability dictates the need to expedite deriving divergent critical performance metrics such as response times. In this paper we present uni-

form conceptual data model along with uniform data processing and propose “black box” queuing network model with mass multi-service demands and single queuing node. Every service demand associates with the original transaction class accessing personal documental database objects and characterizes by Poison distribution. Proposal model doesn’t take into account salient features of transaction processing and therefore doesn’t depend on current upgrades of its automation environments. Unlike other performance evaluation models our approach targets on application aspects of application development, deployment and utilization when it needs do performance testing of application en bloc for limited periods.

Keywords: performance evaluation, response time, client/server architecture, simulation, queuing network model.

References

1. Goodsun J., Steward R. Data Access Handbook. St. Petersburg, BHV-Peterburg, 2013. (In Russ.)

2. Richter J. CLR via C#. Programming on the platform Microsoft.NET Framework 4.5 in C#. St. Petersburg, Piter, 2013. (In Russ.)

3. Molyneaux I. The art of application performance testing. O’Reilly, 2009

4. Osman R., Knottenbelt W. J. Database system evaluation models: a survey. *Performance evaluation*, 2012, vol. 69, no. 10, p. 471–493.

5. Koziolek H. Performance evaluation of component-based software systems: a survey. *Performance evaluation*, 2010, vol. 67, no. 8, p. 634–658.

6. Menasce D. A., Gomaa H. A method for design and performance modeling of client / server systems. *IEEE Transactions on software engineering*, 2000, vol. 26, p. 1066–1085.

7. Gijsen B. M. M., Mei R. D. van der, Engelberts P., Berg J. L. van den, Wingerden K. M. C. van. Sojourn time approximations in queuing networks with feedback. *Performance evaluation*, 2006, vol. 63, no. 8, p. 743–758.

8. Kattepur A., Nambiar M. *Service demand modeling* and performance prediction with single-user tests. *Performance evaluation*, 2017, vol. 110, no. 8, p. 1–21.

9. Urgaonkar B., Pacifici G., Shenoy P., Spreitzer M., Tantawi A. An analytic model for multi-tier internet services and its applications. *Proceedings of the 2005 ACM SIGMETRICS International Conference on measurement and modeling of computer systems, ACM*. Banff, Alberta, Canada, 2005, p. 391–302.

10. Menansce D., Bennani M. Analytic performance models for single class and multiple class multithreaded software servers. *Proceedings of the international computer measurement group conference*. Reno, NV, USA, 2006, p. 475–482.

11. Elnikety S., Dropsho S., Cecchet E., Zwaenepoel W. Predicting replicated database scalability from standalone database profiling. *Proceedings of the 4th ACM European conference on computer systems, ACM*. Nuremberg, Germany, 2009, p. 303–306.

12. Bernstein P. A., Newcomer E. Principles of transaction processing. San Francisco, CA: Morgan Kaufmann, 2009.

For citation:

Rodionov A. N., Eirikh N. V., Dubey O. Ya. A Database Performance Evaluation Study: Transaction Model and Simulation System Architecture. *Vestnik NSU. Series*: *Information Technologies*, 2018, vol. 16, no. 1, p. 100–112. (In Russ.)