УДК 004.051

А. Н. Родионов [1], **О. В. Решетникова** [2]

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

[2] *Дальневосточный государственный университет путей сообщения*
ул. Серышева, 47, Хабаровск, 680021, Россия

ran@newmail.ru, ov13r@yandex.ru

НЕКОТОРЫЕ АСПЕКТЫ ИССЛЕДОВАНИЯ ДИСЦИПЛИНЫ ВСТРОЕННОЙ ТРАНЗАКЦИОННОЙ ОЧЕРЕДИ СЕРВЕРНОЙ СУБД

Повысить производительность клиент-серверных приложений можно множеством способов. Управление транзакционной очередью – один из них. Наиболее распространенные СУБД, такие как MS SQL или ORACLE, практически запрещают вмешиваться в порядок обслуживания транзакций, образующих очередь. Тем не менее такая возможность все-таки существует. Статья посвящена решению нескольких задач, формирующих базу для последующего вмешательства в работу системной очереди с целью улучшения ее обслуживания. Прежде всего, это установление факта истинности декларируемого параллелизма выполнения транзакций, а также конфигурирование службы Service Broker для построения перехватывающей очереди, порядком обслуживания которой можно будет впоследствии управлять.

Ключевые слова: транзакционная и перехватывающая очереди, приоритет обслуживания, параллельная обработка.

Введение. Постановка задачи исследования

Работы по составлению расписаний [1; 2; 3], в соответствии с которым следует обрабатывать транзакции, с заметной регулярностью появляются в самых разнообразных источниках научно-технической информации. Все они, по умолчанию, предполагают, что последующая реализации полученных решений не должна вызывать сколь-нибудь значительных трудностей. Однако на практике наблюдаются обратные эффекты. Одна из причин состоит в том, что все мы, в какой-то степени, оказываемся «заложниками» тех решений (и хороших, и плохих), которые производители систем управления базами данных (СУБД) закладывают в очередные версии своих программных продуктов.

Но это не единственное, что мешает (или, напротив, что также следует признать) улучшить качество прикладных программных систем, в том числе находящееся в зависимости от оптимальной настройки компонентов базы данных.

Можно принять «на веру» то, что зафиксировано в документации, сопровождающей СУБД [1], или воспользоваться многочисленными советами, содержащимися в публикациях [4; 5]

[1] См., например: Электронная документация по SQL Server 2014. URL: https://msdn.microsoft.com/ru-ru/library/ms130214(v=sql.120).aspx. Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator. URL: https://msdn.microsoft.com/en-us/library/dn673537.aspx

Родионов А. Н., Решетникова О. В. Некоторые аспекты исследования дисциплины встроенной транзакционной очереди серверной СУБД // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2016. Т. 14, № 2. С. 110–121.

© А. Н. Родионов, О. В. Решетникова, 2016

и на интернет-форумах [2]. Но всего этого может оказаться недостаточно, когда разработанное приложение начнет демонстрировать снижение производительности при увеличении рабочей нагрузки. И этому также есть разумное объяснение, которое сводится к тому, что СУБД и ее окружение в виде приложений баз данных – это большая и сложная система, для которой предсказать все поведенческие реакции – задача практически неосуществимая.

Единственное средство, которое остается, чтобы разобраться и понять, как в реальности будет протекать взаимодействие между базой данных и ее клиентами (для задуманной конфигурации и специально подобранных для этого инструментов, организованных в систему), – создать модель, имитирующую работу приложений. (Заметим, что набор инструментов, предоставляемых современными СУБД, весьма обширен, и для получения одного и того же результата можно создавать их комплексы множеством способов.)

Настоящая работа носит узконаправленный характер и нацелена на выявление возможности корректного (направленного на увеличение производительности) вмешательства в работу менеджера, управляющего порядком обслуживания транзакций в серверной СУБД. В качестве последней рассматривается MS SQL, но на ее месте может оказаться и сопоставимая с ней ORACLE.

Насколько важно регулировать порядок обслуживания транзакций, покажем на примере стандартной схемы взаимодействия клиентских приложений в режиме «отсоединенного» доступа с базой данных.

Обращения к базе данных представляют собой наборы SQL-инструкций, организованных в виде транзакций, имеющих форму хранимых процедур, размещенных, как и база данных, на сервере. Таким образом, стандартный сеанс взаимодействия приложения с базой данных (рис. 1) сводится к: активации соединения (1), предварительно открытого в момент старта приложения; передаче серверу команды на запуск нужной хранимой процедуры (2); постановке транзакции в очередь (3); обслуживанию транзакции и размещению полученных результатов во внутренней структуре хранения (4); передаче последних приложению (5).

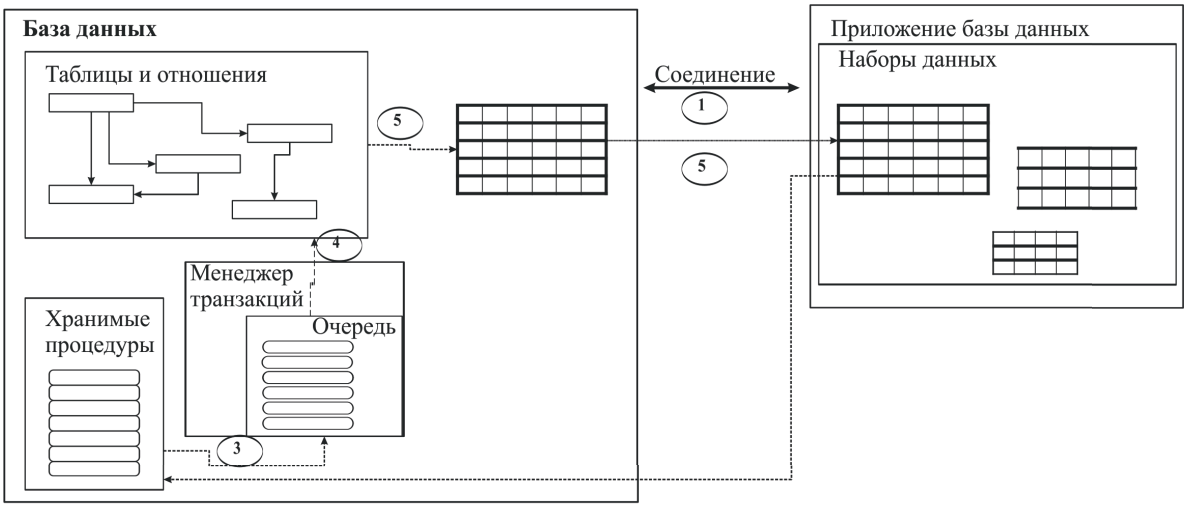

Рис. 1. Стандартный сеанс взаимодействия приложения с базой данных

[2] Разработка (компонент Service Broker). URL: https:// msdn.microsoft.com/ru-ru/library/bb522889(v=sql.120).aspx

Очевидно, что высокая интенсивность обмена в системе «сервер – приложения базы данных» предполагает существование в ресурсах сервера очереди, образуемой транзакциями (далее – транзакционная очередь, TQ).

Обрабатываемые сервером транзакции по своей природе неоднородны: операции чтения в них могут чередоваться с операциями на модификацию данных, адресатами транзакций могут выступать различные подмножества объектов базы данных и т.д. Кроме того, транзакциям желательно присваивать разные приоритеты исполнения, поскольку для разных классов транзакций время реакции (получения ответа) не должно быть одинаковым.

Но в случае с СУБД мы сталкиваемся с крайне ограниченным набором инструментов, которые можно было бы задействовать, чтобы вмешаться в организацию и обслуживание TQ. (Более того, несмотря на заявления производителей СУБД, касающиеся параллелизма в обработке транзакций, далеко не очевидно, вследствие особенностей организации вычислительного процесса, имеем ли мы дело с истинным или псевдо параллелизмом.)

Основная же сложность, которая возникает, как только мы приступаем к исследованию TQ, состоит в получении очереди фиксированной длины, поскольку сервер постоянно находится в состоянии обслуживания очереди.

Далее работа организована следующим образом. Ниже приводятся перечень и краткая характеристика диагностических процедур, отслеживающих состояние серверной TQ в произвольные моменты времени. Затем излагается решение, позволяющее исследовать дисциплину TQ, и дается описание работы программного модуля, созданного для получения характеристик TQ. Наконец, приводятся и интерпретируются экспериментальные данные, полученные с помощью разработанного приложения.

Краткий обзор возможностей системных диагностических средств контроля исполнения транзакций

Современный SQL-сервер содержит обширный набор средств для анализа своего текущего состояния. В частности, MS SQL Server (далее – сервер) предлагает к использованию динамические административные представления (Views). Эти представления, по сути, являются системными таблицами, в которые заносятся разнообразные сведения о состояниях и параметрах как самого сервера, так и обслуживаемых им объектах в характерные моменты времени. Для получения необходимой моментной информации из такой таблицы достаточно обратиться к ней с запросом соответствующего вида.

Нет смысла описывать возможности представлений, которые позволяют получить сведения о состояниях встроенных очередей сервера – TQ, и очередей брокера – Initiator Queue (IQ) и Target Queue (GQ). Все необходимое можно найти в on-line документации сервера [3]. Поэтому ограничимся только списком и назначением представлений.

Вся информация о транзакциях сосредоточена в системных представления с префиксом – sys.dm_tran_. Данные, касающиеся очередей, – в sys.dm_broker_queue_minitors, sys_dm_broker_ activated_tasks, sys_exec_background_job_queue и sys_exec_background_job_queue_stat.

Средства же, которые могли бы задать определенный приоритет обслуживания в MS SQL Server и Oracle, отсутствуют. Единственное, на что можно влиять, – на приоритет обслуживания в случае появления транзакций, взаимно блокирующих друг друга.

Организация и использование перехватывающей транзакционной очереди

Для того чтобы «взять на себя» управление транзакционной очередью, прежде всего, нужно определиться с дисциплиной и способом ее обслуживания, поскольку ни в документации по MS SQL, ни в каких-либо других источниках эти важнейшие характеристики не прописаны.

[3] Разработка (компонент Service Broker). URL: https://msdn.microsoft.com/ru-ru/library/bb522889(v=sql.120).aspx

Другими словами, необходимо провести небольшое исследование, направленное на получение искомых данных.

(Вероятнее всего, разработчики СУБД решили, что встроенные в MS SQL алгоритмы блокировки ресурсами и управления очередью – наилучшие. Также заметим, что декларируемый параллелизм отрицает стандартный для любой очереди порядок обслуживания: первым пришел – первым обслужен.)

Очевидное в этом случае решение сводится к последовательному выполнению следующих действий:

• формирование очереди фиксированной длины с заранее предопределенными требованиями (то, что принято называть смесью транзакций) и известным порядком элементов в очереди;

• инициализация обслуживания очереди;

• получение искомых характеристик обслуживания посредством использования диагностических средств MS SQL.

Правда, на практике реализовать такой алгоритм оказывается весьма проблематичным. Здесь мы сталкиваемся с тем, что поступающие на обработку требования сразу же анализируются сервером и направляются на обслуживание. Поэтому, в произвольные моменты времени, в общем случае, сервер будет работать с разными очередями. Таким образом, выполнение начального шага, направленного на синтез моделируемой очереди, представляет собой отдельную задачу.

Другой способ исследовать серверную TQ – сформировать «перехватывающую» очередь, которая будет переадресовывать все свои заявки в TQ по мере своего заполнения. В составе MS SQL присутствует компонент Server Broker (SB), который наилучшим образом, несмотря на свое иное назначение – создание асинхронных приложений в архитектуре SODA (сервисно-ориентированная база данных) [6], подходит для реализации предложенного плана. Покажем, каким образом можно использовать его функциональность.

Принципиальная схема взаимодействия между очередями SB и серверной TQ показана на рис. 2, Б. (Заметим, что это нетиповая конфигурация. В стандартной конфигурации поток сообщений направлен сначала от IQ к TQ, а затем – в обратном направлении. Но SB позволяет перенаправлять потоки нужным способом.)

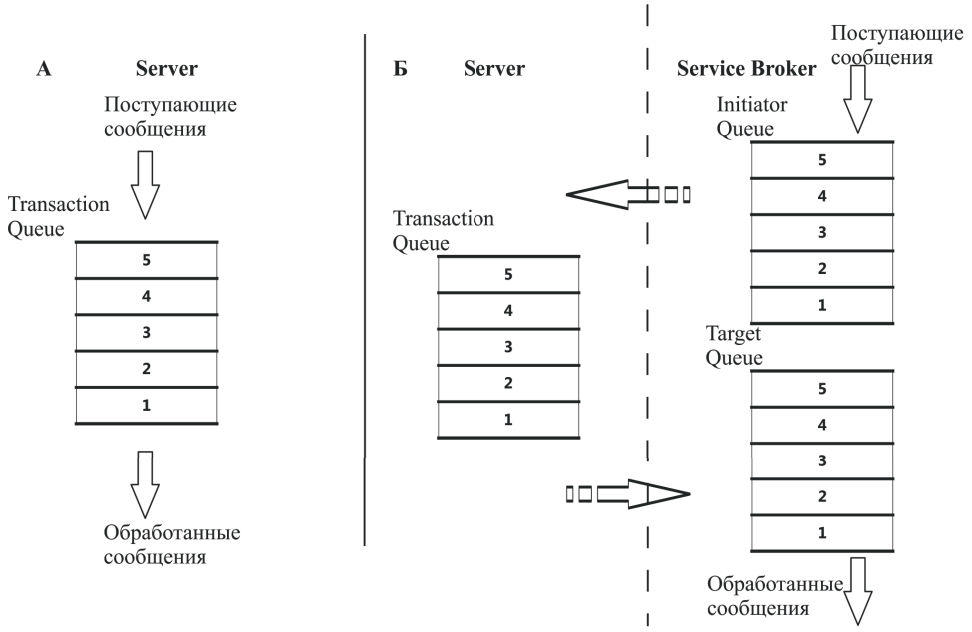

Рис. 2. Порядки поступления и обслуживания заявок в TQ:
А – стандартный; *Б* – с использованием брокера

SB содержит две очереди: входящих и исходящих сообщений. Сообщениями могут быть и инструкции к выполнению транзакции. Из первой очереди – IQ, сообщения передаются серверу, который их выполняет, а полученные результаты поступают во вторую очередь – GQ. Таким образом, разместив в IQ определенную смесь транзакций, на основании порядка, в котором результаты разместятся в GQ, можно установить дисциплину TQ. Входная очередь IQ в таком случае выполняет функцию моделируемой «перехватывающей» очереди.

Прежде чем воспользоваться предложенной схемой, требуется удостовериться в том, что поведение TQ по обслуживанию транзакций с привлечением SB (см. рис. 2, А) и без последнего идентичны (см. рис. 2, Б). Сформулируем два условия корректной эмуляции TQ с помощью «перехватывающей» очереди, роль которой играет IQ.

Первое состоит в том, что IQ и TQ должны быть одинаковыми, как по числу транзакций в них, так и по позициям, которые транзакции занимают в очереди. Второе условие будет касаться равенства двух функций вида $L = f(C)$, отражающих зависимость средней длины очереди L от номера шага обслуживания C. (Здесь под C подразумевается количество заявок, уже обслуженных сервером.) Одна функция строится для TQ, в которую заявки поступают «напрямую» (см. рис. 2, А), другая – когда заявки предварительно перехватываются и размещаются в IQ (см. рис. 2, Б). Детальное рассмотрение обоих условий отложим до изложения результатов, полученных в ходе проведения экспериментов.

Структура и интерфейс программного средства для исследования транзакционной очереди

Предполагается, что перед тем, как провести эксперименты, потенциальный исследователь самостоятельно создает базу данных и заполняет ее данными, строит и компилирует запросы, преобразуя их в хранимые процедуры, из которых предполагается формировать TQ.

Из показанного ниже главного окна программы (рис. 3) видно, каким образом организованы подготовка и проведение экспериментов. На первом шаге следует сформировать список хранимых процедур, которые будут образовывать TQ, и указать две характеристики для оценки времени обслуживания очереди: размер доверительного интервала и вероятность попадания суммарной продолжительности обслуживания TQ в этот интервал. Используя эти данные, встроенный в программу модуль вычисляет и уточняет необходимое число прогонов после завершения каждого очередного шага, реализуя тем самым стандартное правило автоматической остановки.

Программное средство реализовано на языке программирования C# и использует библиотеки классов ADO.NET и LINQ.NET.

На рис. 4 представлена диаграмма переходов при взаимодействии клиента с базой данных через механизм SB. Это наиболее общий вариант взаимодействия, поскольку используются два обработчика: для входящей и исходящей очередей сообщений.

С целью имитации единовременного поступления запросов от нескольких клиентов программное средство задействует механизм многопоточности.

Архитектура и функционирование приложения представлены в виде диаграммы последовательностей выполнения ключевых модулей программы (рис. 5), отражающей, в том числе, и характер взаимодействий между ними. Кратко опишем порядок запуска и основную функцию, которую выполняет каждый из представленных модулей.

На начальном этапе производится формирование коллекции участников (транзакций) **InstanceCollection** заданного размера. Каждый элемент коллекции **Instance** представлен тремя полями: **Type**, **Question**, **Answer**, назначение которых вытекает из их названия. Поле **Answer** при создании назначается пустым. Далее, для каждого участника генерируется отдельный программный поток **PleerInstance**, входом которого являются значения **Instance.Type** и **Instance.Question**. Поток открывает подключение к базе данных и выполняет к ней запрос

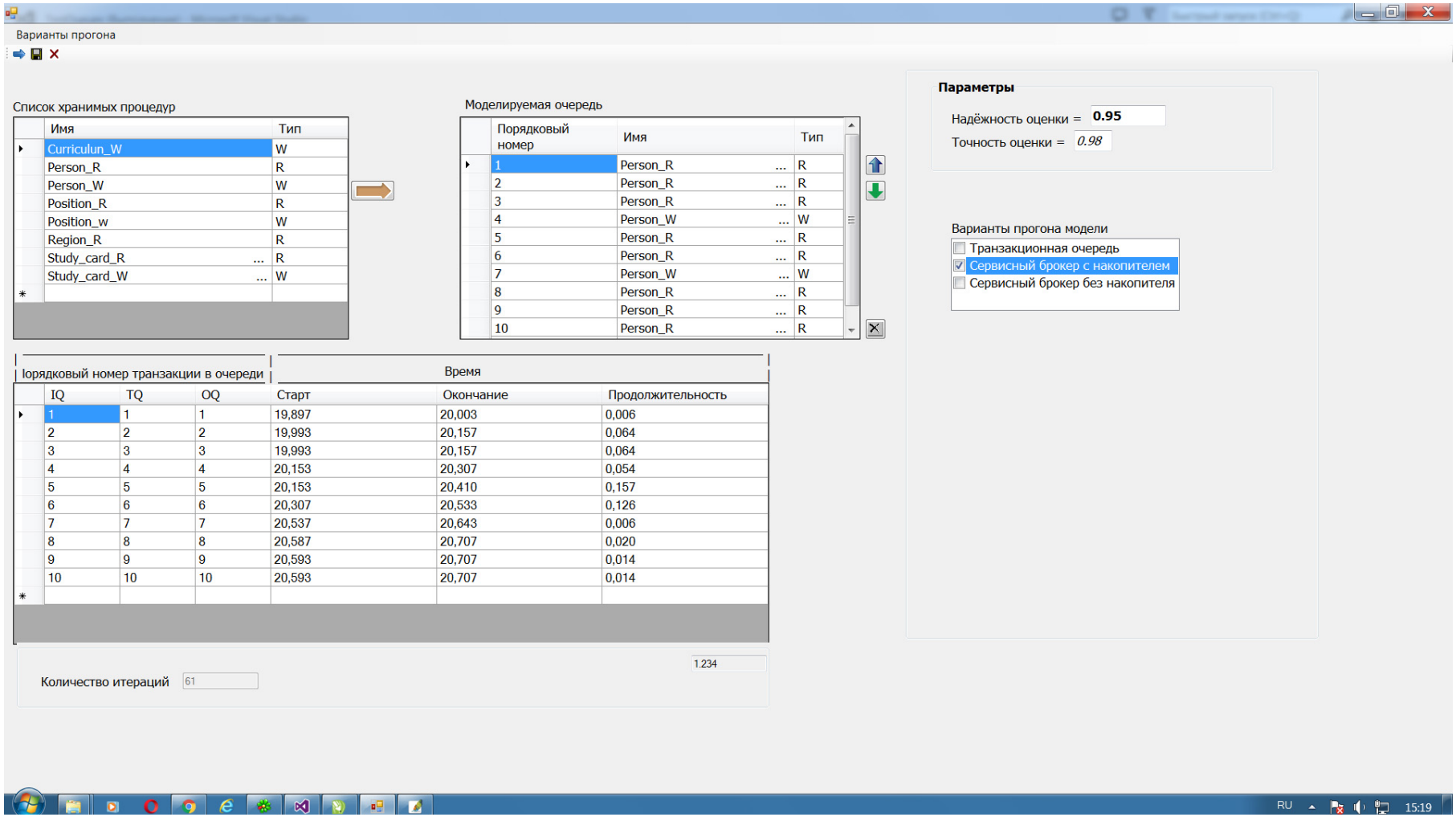

Рис. 3. Главное окно программы для исследования характеристик обслуживания очереди транзакций

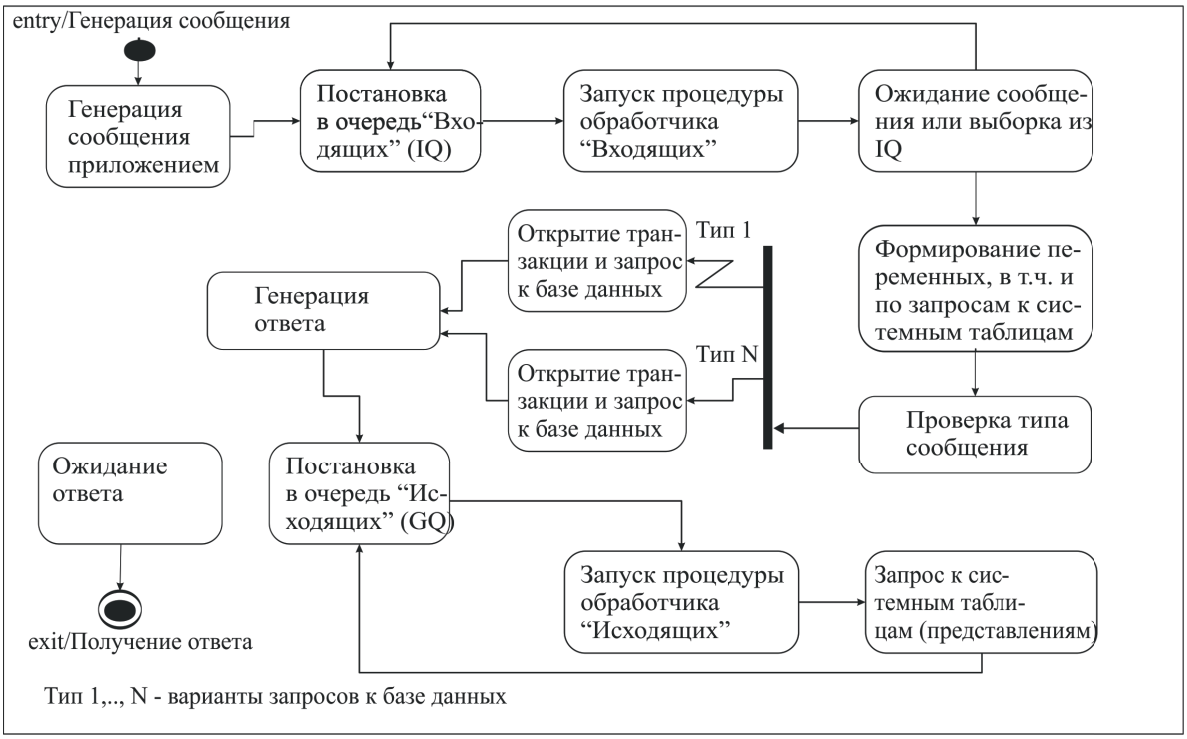

Рис. 4. Диаграмма переходов, ревлизующих диалог с SB

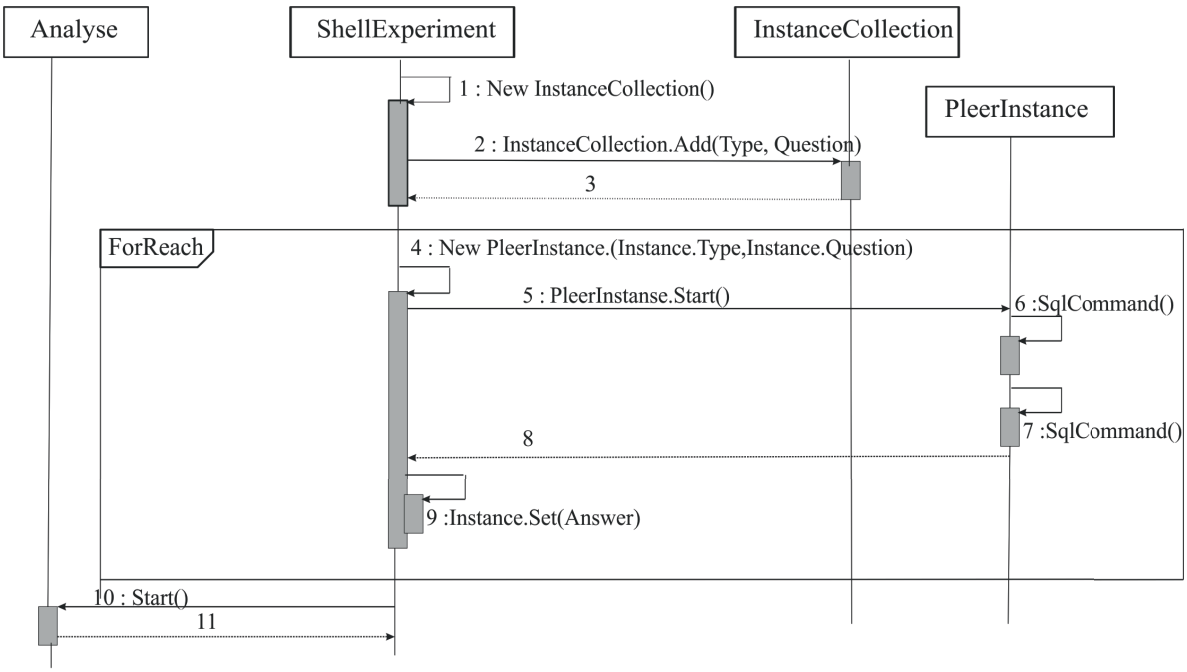

Рис. 5. Диаграмма последовательностей выполнения ключевых модулей программы

SqlCommand(), строка которого является инструкцией отправки сообщения «SEND» Transact-SQL. После этого поток переходит в состояние ожидания ответа от сервера.

Обработка отправленного клиентом сообщения производится в соответствии с вышеприведенной диаграммой переходов на уровне SQL Server. Все интересующие сведения о ходе процесса обслуживания сообщения фиксируются обработчиками очередей в специально предназначенной для этого временной таблице, являющейся частью SB.

После получения ответа от сервера, которое сохраняется в поле **Instance.Answer**, подключение к базе данных закрывается.

Эксперимент

Исследование дисциплины TQ и системы очередей IQ->TQ->GQ (см. рис. 2) выполнялось на компьютере со следующими характеристиками: тип – x64-based PC, процессор – Intel(R) Core (TM)2 Duo CPU E8400 @ 3.00 GHz, 3.003 GHz, ядер – 2, логических процессоров – 2, объем оперативной памяти – 4 Гб.

По сути, моделировались две одноканальные системы массового обслуживания: без использования SB с потоком требований, поступающих на обслуживание через равные промежутки времени (программное обеспечение использует цикл для отправки транзакций на сервер), и с использованием SB с накопителем заявок в лице IQ, отправляемых на обслуживание «одномоментно». В последнем случае проводилось две серии экспериментов: с IQ фиксированной длины (серверный брокер позволяет накапливать сообщения в своей IQ перед тем, как отправить их серверу) и с IQ, в которой, как и в TQ, поступающие заявки сразу же передаются на обслуживание.

Поскольку решалась заявленная ранее задача определения порядка обслуживания транзакций, которые уже сформировали очередь, то характеристики входного потока требований в расчет не принимались. Обратим внимание на то, что по совокупности таких свойств, как стационарность, ординарность и отсутствие последействий, которые могут быть установлены на основании анализа входной последовательности событий (поступающих транзакций), – это простейший (пуассоновский) поток требований. Это утверждение справедливо как в отношении отдельных однородных субпотоков, образованных требованиями (запросами) одного тира, так и общего потока, в котором сосредоточены требования, принадлежащие разным типам.

Заявки, участвующие в экспериментах, представляли собой обращение к единственной таблице базы данных либо на чтение (сообщение типа Select), либо на запись (сообщение типа Insert) и образовывали очередь, в которой заявки на чтение чередовались с заявками на запись. Листинг 1 содержит фрагмент программного кода, содержащего инструкции SELECT (чтение) и INSERT INTO (запись), реализующие обработку сообщения типа Insert.

Листинг 1

```
SET @s = 't'+CAST(@Msg as varchar(10));
    BEGIN TRANSACTION @s;
    SET @t1 = substring(CAST (CAST (GETDATE() as time(7)) as CHAR (17)), 7,10);
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    BEGIN TRANSACTION;
        SELECT Value FROM Test  WHERE Id = @Msg;
            IF (@@ROWCOUNT > 0) SEND ON CONVERSATION @DlgHandle
MESSAGE TYPE [Msg] (cast('0' as integer))
            ELSE BEGIN  INSERT INTO Test(Value, DtTime) VALUES ( cast ( @
Msg as char(10)), GETDATE());
        SEND ON CONVERSATION @DlgHandle  MESSAGE TYPE [Msg] (CAST('1' as
integer));
            END;
```

```
    COMMIT TRANSACTION;
    SET @t2 = substring(CAST (CAST (GETDATE() as time(7)) as CHAR (17)), 7,10);
COMMIT TRANSACTION @s.
```

В табл. 1 в качестве характерного примера приводятся результаты машинных имитационных экспериментов для одного произвольно взятого прогона модели, когда IQ используется в качестве накопителя.

Таблица 1

Результаты эксперимента по обработке транзакций
с использованием перехватывающей очереди *

Порядковый номер транзакции в очереди			Время		Продолжительность, сек
IQ	TQ	OQ	Старт	Окончание	
1	1	1	11,233	11,340	0,007
2	2	2	11,343	11,450	0,007
3	3	3	11,450	11,560	0,010
4	4	4	11,563	11,670	0,007
5	5	5	11,670	11,780	0,010
6	6	6	11,780	11,890	0,010
7	7	7	11,893	12,000	0,007
8	8	8	12,000	12,110	0,010
9	9	9	12,110	12,220	0,010
10	10	10	12,220	12,340	0,020

* Примечание: строки таблицы, выделенные темным цветом, соответствуют заявкам на модификацию данных, остальные – на чтение.

Нетрудно заметить, что если руководствоваться данными из табл. 1 (что подтверждается всеми прогонами модели), то порядок поступления и обслуживания транзакций в IQ и TQ один и тот же. Следовательно, первое условие корректности эмуляции TQ посредством средств SB, сформулированное выше, выполняется.

Второе условие более строгое. Покажем, как будет меняться средняя длина TQ на каждом шаге обслуживания в зависимости от того, передаются ли запросы в нее через SB или напрямую, без использования брокера. Чтобы создать идентичные условия для сравнения, достаточно ограничиться одинаковыми характеристиками входного потока требований.

Пусть заявки в TQ поступают через равные промежутки времени. В первом случае это достигается за счет отказа от использования специальной функции (метода) SB, делающей IQ активной только в тот момент, когда она достигнет фиксированной длины (По умолчанию предполагается, что заявки следуют транзитом через IQ, не накапливаясь в ней), а во втором – генерацией заявок внутри тривиального цикла. Обслуживание начинается в момент поступления заявок.

Два графика (рис. 6), построенные по результатам экспериментов, практически идентичны. Отсюда, в частности, следует, что SB не оказывает существенного влияния на длину очереди в различные моменты времени ее обслуживания, поскольку обе функции $L = f(C)$ одинаковы.

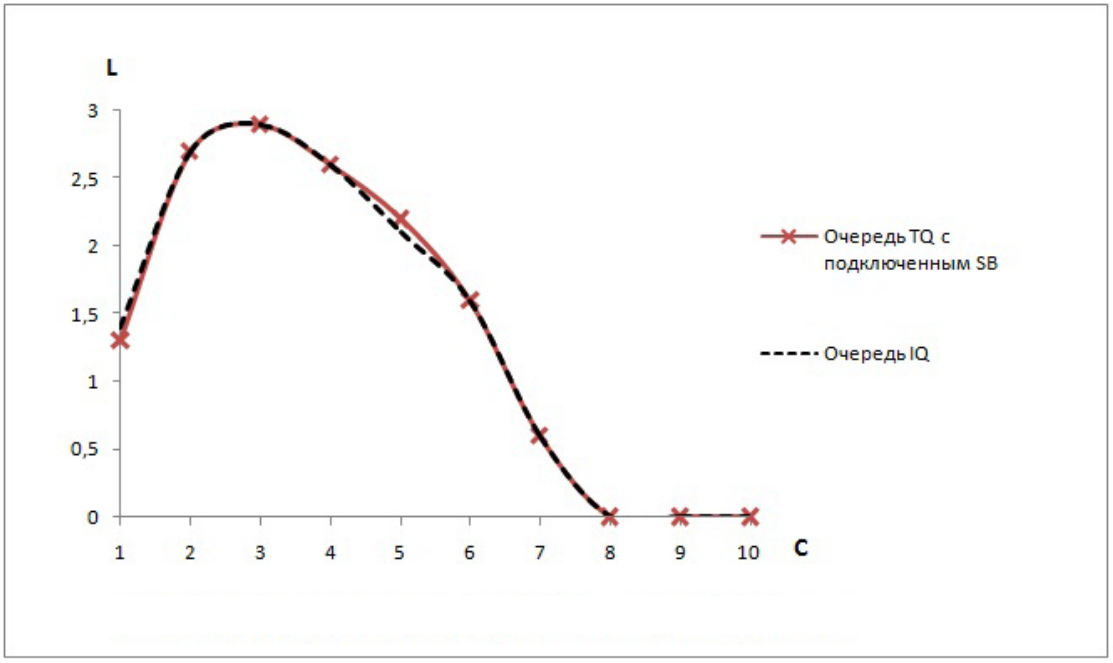

Рис. 6. Зависимости средней длины TQ от шага обслуживания

Представленная информация была бы неполной, если бы не были получены данные о дисциплине обслуживания TQ без подключения SB (см. рис. 2, А). Для наглядности данные одного из экспериментов приводятся в табл. 2.

Таблица 2

Экспериментальные данные обслуживания транзакций сервером

Момент времени постановки транзакции в TQ	Время		Продолжительность, сек
	старт	окончание	
19,893	19,897	20,003	0,006
19,893	19,993	20,157	0,064
19,893	19,993	20,157	0,064
19,893	20,153	20,307	0,054
19,893	20,153	20,41	0,157
20,15	20,307	20,533	0,126
19,893	20,537	20,643	0,006
20,533	20,587	20,707	0,020
19,893	20,593	20,707	0,014
20,15	20,593	20,707	0,014

Обратим внимание на два момента. Подключение SB увеличивает примерно в 5 раз общую продолжительность обслуживания очереди (что подтверждается данными, полученными в серии экспериментов). В то же время «скорость» выполнения одного-единственного запроса оказывается на два порядка меньше, если используется SB. Более того, как опять же показывают эксперименты, этот разрыв достигает одного порядка, если размеры очереди также увеличиваются в десять раз.

Из информации, содержащейся в табл. 2, видно, что транзакции действительно выполняются параллельно, что приводит к снижению общего времени обслуживания очереди, чего нельзя сказать для случая, когда подключен SB. В то же время, при подключении SB обработка транзакций, поступающих на обслуживание, производится строго последовательно. (Вообще говоря, при использовании одноядерного процессора, как в нашем случае, скорее надо говорить о квазипараллельности, поскольку команды процессора выполняются последовательно.)

Заключение

На основании полученных результатов можно сделать несколько выводов как частного, так и общего характера.

Во-первых, идея построения «перехватывающей» очереди как средства, с помощью которого можно влиять на порядок обслуживания транзакций и использование для этой цели входной и выходной очередей SB, состоятельна и может быть реализована на практике. Это подтверждается данными, полученными в ходе проведения экспериментов. Достаточно указать на то, что заявки обрабатываются строго в том порядке, в котором они поступают (см. табл. 1). Следовательно, их можно менять местами, задавая тем самым определенный приоритет для их обслуживания.

Во-вторых, существенное возрастание времени обслуживания транзакции, передающейся SB, вследствие наблюдаемого «отключения» параллелизма требует отдельного изучения и объяснения этого факта. При определенной доработке созданного программного средства соответствующие исследования могут быть выполнены.

В-третьих, конфигурирование прикладных программных систем на предпроектной и проектной стадиях предполагает наличие максимально полной информации о характеристиках серверной СУБД в тех ее конфигурациях, которые являются наиболее подходящим для организации обмена данными между приложениями и базой данных. Очевидно, что «недокументированные» возможности любой сложной системы, включая и собственно СУБД, намного шире, чем это обычно декларируется. Более того, далеко не всегда заявленные параметры и алгоритмы функционирования соответствуют фактическим.

В этой связи опираться исключительно на советы и рекомендации производителей СУБД и профессиональных разработчиков при создании архитектуры приложений было бы неправильно. Более действенный механизм – создание экспериментальных стендов, на которых можно воспроизвести промышленную среду и протестировать альтернативные решения, как это, например, предлагается в [7].

Список литературы

1. *Pang H., Carey M., Livny M.* Multiclass query scheduling in real-time database systems // IEEE Transactions on knowledge and data engineering. 1995. Vol. 7. No. 4. P. 533–551.

2. *Mentis A., Katsaros P., Angelis L., Kakarontzas G.* Quantification of interacting runtime qualities in software architectures: Insights from transaction processing in client – server architectures // Information and Software Technology. 2010. Vol. 52. No. 12. P. 1331–1345.

3. *Wang H., Xiao Y., Shu L.* Scheduling Periodic Continuous Queries in Real-Time Data Broadcast Environments // IEEE Transactions on computer. 2012. Vol. 61. No. 9. P. 1325–1340.

4. *Шаша Д., Бонне Ф.* Оптимизация баз данных: принципы, практика, решение проблем. М.: КУДИЦ-ОБРАЗ, 2004.

5. *Новиков Б. А., Домбровская Г. Р.* Настройка приложений баз данных. СПб.: БХВ–Петербург, 2006.

6. *Aschenbrenner K.* Pro SQL Server 2008 Service Broker. New York. Apress. 2008. 587 p.

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

Материал поступил в редколлегию 29.03.2016

A. N. Rodionov [1], **O. V. Reshetnikova** [2]

[1] *Computer Center of Far Eastern Branch of Russian Academy Sciences*
65 Kim Yu Chen Str., Khabarovsk, 680000, Russian Federation

[2] *Far Eastern State Transport University*
47 Serishev Str., Khabarovsk, 680021, Russian Federation

ran@newmail.ru, ov13r@yandex.ru

SOME ASPECTS OF EMBEDDED TRANSACTIONAL QUEUE DISCIPLINE INVESTIGATION FOR SERVER DBMS

A lot of factors effect productivity of client-server applications. Transactional queue management is one of them. Most widespread DBMS, as such MS SQL or ORACLE, forbid interfering into an order of transactional queue service. However there is a feasibility of order queue change. This article concentrates on decisions of some issues which establish a framework for subsequent interference into queue service. First of all it takes to unveil whether transactions really processed concurrency or not. Then for programmatically queue priority control it is proposed to use the interceptive queue which is configured by means of Service Broker.

Keywords: transactional and interceptive queues, service priority , concurrency processing.

References

1. Pang H., Carey M., Livny M. Multiclass query scheduling in real-time database systems. *IEEE Transactions on knowledge and data engineering*, 1995, vol. 7, no. 4, p. 533–551.

2. Mentis P. Katsaros, Angelis L., Kakarontzas G. Quantification of interacting runtime qualities in software architectures: Insights from transaction processing in client–server architectures. *Information and Software Technology*, 2010, vol. 52, no. 12, p. 1331–1345.

3. Wang H., XiaoY., Shu L. Scheduling Periodic Continuous Queries in Real-Time Data Broadcast Environments. *IEEE Transactions on computer*, 2012, vol. 61, no. 9, p. 1325–1340.

4. Shasha D., Bonnet P. Database Tuning. Principles, Experiments and Troubleshooting techniques Moscow, KUDIC-OBRAZ, 2004. (In Russ.)

5. Novikov B., Dombrovskaya G. Database Applications Tuning. St.-Petersburg, BHV-Petersburg, 2006. (In Russ.)

6. Aschenbrenner, Klaus. Pro SQL Server 2008 Service Broker. New York, Apress, 2008, 587 p.

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