УДК 004.724.4

*ГУСЕВ Е.И.*

# ИССЛЕДОВАНИЕ ОБЛАСТИ ПРИМЕНЕНИЯ НЕБЛОКИРУЮЩЕГО АЛГОРИТМА ФИКСАЦИИ РАСПРЕДЕЛЁННЫХ ТРАНЗАКЦИЙ

Предложена математическая модель, позволяющая сравнить традиционный алгоритм 2х-фазного блокирования с неблокирующим алгоритмом фиксации распределённых транзакций в СУБД. Определение параметров трафика к СУБД, при которых целесообразно использование предлагаемого алгоритма способно существенно повысить эффективность использования распределённых СУБД.

There are proposed a mathematical model that allows to compare the traditional algorithm for 2-phase locking with a non-blocking commit algorithm for distributed transactions in RDBMS. Defining the parameters of the traffic to the database in which it is expedient to use the proposed algorithm can significantly improve the efficiency of distributed DBMS.

## 1. Введение

Неблокирующий алгоритм фиксации распределённых транзакций [1] представляет собой реализацию принципов ACID [2] для распределённых СУБД [3,4,5]. В исследовании эффективности алгоритма можно выделить основные направления:

- исследование надёжности (устойчивости) алгоритма;
- анализ производительности;
- исследование накладных расходов алгоритма;
- исследование устойчивости системы на коллапс при превышении граничных значений интенсивностей.

Для моделирования работы алгоритма выделим параметры системы, влияющие на её эффективность. Это $t_{лIO}$ – среднее время локального ввода-вывода (с учётом индексаций, кэширования и других техник), $t_c$ – время пересылки сетевого пакета. Также необходимо учитывать граничные показатели работы системы:

$v_{I/O}$– максимальная интенсивность ввода-вывода (шины);

$v_{CPU}$ – максимальная интенсивность ввода-вывода;

$v_c$ – максимальная интенсивность сети.

После превышения какой-то из максимальных интенсивностей система приходит к коллапсу. Стоит правда указать на то, что $\nu_c$ может зависеть от интенсивностей $\nu_{I/O}$ и $\nu_{CPU}$. При моделировании стоимостной модели это необходимо учитывать, генерируя пропорционально интенсивности сетевого трафика служебный CPU и I/O трафик.

Можно выделить следующие виды коллапса по:

- блокировкам;
- времени сети;
- вводу-выводу;
- нехватке ресурсов процессора;
- реализации архитектуры СУБД.

Рассмотрев при каких параметрах трафика и применении какого алгоритма возникает какой тип коллапса, можно сравнивать эффективности алгоритмов на одной системе. Например, на заданной системе при входящей интенсивности возникновения конфликта 1000 заявок в секунду применение традиционных алгоритмов приводит к коллапсу по блокировкам, а применение неблокирующего алгоритма фиксации распределённых транзакций не приводит. Увеличение входящей интенсивности возникновения конфликта до 50000 заявок в секунду приведет к коллапсу и в предлагаемом алгоритме. Или перефразировав: граничная пропускная способность системы при традиционном алгоритме: 1000 взаимоблокирующих заявок, а при неблокирующем – 50000.

С другой стороны критерием эффективности может быть среднее время обработки заявки или средняя загрузка системы. Естественно надо понимать, что расчет средних показателей имеет смысл только если коллапс не наступает.

В данной статье мы проведем исследование коллапса блокировок с целью их исключения.

## 2. Постановка задачи

Предположим, мы имеем систему, на которую поступают заявки с интенсивностью *v*. Система имеет множество общих ресурсов. Обработка заявки занимает время *t*. Если заявка, поступившая в систему, обратившись к общему ресурсу, обнаруживает что общий ресурс занят, она ожидает его освобождения. Если более одной заявки ожидает общего ресурса, то доступ к ресурсу осуществляется в порядке очереди FIFO.

Можно разложить общий поток заявок на *n* непересекающихся потоков, каждый к своему ресурсу и со своей интенсивностью.

Обозначим интенсивность заявок к максимально востребованному ресурсу как $v_к$. Условимся, что вероятность конфликта к любому ресурсу подчиняется закону Пуассона в течение определённого промежутка времени $\Delta t$. Тогда $v_к=const$ в течение этого $\Delta t$.

Пусть время обработки первой заявки – *t*. Если образовалась очередь из 2 заявок, то время обработки второй заявки это *t* плюс время ожидания в очереди. (обозначим $t_x$, причем $t_x$ меньше *t*, т.к. вторая заявка поступила после первой) : Если очередь из трех заявок – то аналогично – время обработки 3-й заявки $t_x+2t$.

Если образовалась очередь из n заявок, то время обработки n –ой заявки равно $t_x+(n-1)t$, время блокирования ресурса – в таком случае – это $t_x+(n-1)t$, при условии что за время $t_x+(n-1)t$ заявок к этому ресурсу не поступит. Если за время $t_x+(n-1)t$ к ресурсу поступит менее чем *n* заявок – то очередь не увеличится, а если более – то увеличится. Если очередь увеличится – то время блокирования ресурса после прохождения времени $t_x+(n-1)t$ также.

Для удобства анализа выделим три состояния системы:

1. Стационарное – $v_к<1/t$ и вероятность превышения $1/t \sim 0$. Очереди не образуется.

2. Переходное – $v_к<1/t$ но вероятность превышения $v_к$ значения $1/t$ ненулевая, то есть отклонение $v_к$ от математического ожидания превышает $1/t$. В системе образуются очереди.

3. Коллапс – $v_к<1/t$. Очередь нарастает и время обработки каждой следующей заявки приходящей в очередь больше чем предыдущей.

Если очередь образовалась и $v_к<1/t$ – то очередь начинает сокращаться, но время обработки заявки в этом случае будет включать ожидание в очереди, то есть в переходном состоянии среднее время отклика увеличивается пропорционально средней длине очереди и зависит от дисперсии.

Таким образом, можно сделать следующие выводы:

1. Уменьшая время обработки заявки, мы при большей интенсивности конфликта останемся в рабочем состоянии.

2. В переходном состоянии возможно существенное (*n* кратное) увеличение времени отклика в случае образования очереди.

3. Пиковое кратковременное увеличение интенсивности способно создать очередь, что скажется на времени отклика последующих заявок в течение еще некоторого времени после завершения пика. Уменьшая время обработки заявки, мы снижаем вероятность увеличения времени отклика при возникновении пика. Т.е. Повышаем устойчивость системы.

4. Расчет $Q_к$ – удельной продолжительности задержек связанных с конфликтом за локальные ресурсы (данные) имеет смысл только в переходном состоянии. В стационарном состоянии она равна 0, а в коллапсе – бесконечность.

## 3. Расчет времени обработки заявки

Обозначим – все действия системы кроме ожиданий по причине простоя клиента за промежуток времени *dt* величиной *DB*. Также будем полагать, что интенсивность конфликта постоянна в течении этого времени *dt* (стационарный Пуассоновский поток), а значит вероятности событий конфликта *i*-заявок за ресурс ($p_0,p_1,p_2...p_i$) можно будет рассчитывать по закону Пуассона.

Основная формула для расчёта *DB* при асинхронной фиксации распределённых транзакций без блокирования общего ресурса равна:

$$DB = \int_0^t (v(p_0\ t_у + \sum_1^\infty p_{i.н}) + \frac{t_{фк}}{\tau} + \frac{t_{ск}}{\tau} + Q_{к.} + Q_{ф.п.} + Q_{а.})dt,$$

где: $\tau$ – время обхода круга;

*v* – интенсивность заявок;

$t_у$– время обработки в случае отсутствия конфликта;

$t_{iн}$ – время обработки в случае *i* конфликтов за время $\tau$;

$t_{фк}$ – время формирования курьерского пакета;

$t_{ск}$- время сверки пришедшего курьерского пакета с транзакциями, которые прошли с момента предыдущей сверки на локальной ноде;

$Q_к$ – удельная продолжительность задержек связанных с конфликтом за локальные ресурсы (данные);

$Q_a$ – удельные задержки вызванные конфликтом за внутренние ресурсы системы (например ожидание диска, шины ввода-вывода, обновления мета данных системы, конфликтов за доступ к общей оперативной памяти);

$Q_{ф.п}$ – удельная продолжительность задержек фоновых процессов определяется отношением времени работы остальных фоновых процессов, реализующих корректную работу СУБД к прошедшему времени.

Задержки $Q_{ф.п}\,dt$ – происходят в фоновых процессах СУБД и не включаются в суммарное время отклика. Этим показателем нельзя пренебрегать при расчёте коллапса по вводу-выводу или по CPU, однако, при стационарной работе он на время отклика не влияет.

Задержки $Q_a\,dt$ зависят от архитектурных особенностей СУБД. По этим задержкам также могут возникать коллапсы, однако мы не имеем возможности их моделировать, поскольку они очень сильно зависят от инженерных решений разработчика СУБД. Во многих СУБД также есть аппаратные средства для мониторинга и снижения такого рода задержек. Мы будем пренебрегать ими в идеальной модели, а при реальном моделировании рассчитаем погрешность, которую влечет данное допущение. Иными словами, область применения данных алгоритмов будет такой, при которых этих задержек нет или они существенно меньше тех, которые мы учитываем. Таким образом:

$$SE = \sum \int_0^t (v(p_0\ t_у + \sum_1^\infty p_{i.н}) + \frac{t_{фк}}{\tau} + \frac{t_{ск}}{\tau} + Q_{к.})dt.$$

Если система находится в стационарном состоянии – то величиной $Q_к$ можно пренебречь. В этом случае получаем:

$$SE = \int_0^t (v(p_0\ t_у + \sum_1^\infty p_{i.н}) + \frac{t_{фк}}{\tau} + \frac{t_{ск}}{\tau})dt.$$

Главным преимуществом предлагаемого алгоритма является то, что в случае его неэффективности он в неэффективной части легко может быть заменён на традиционные блокирующие алгоритмы [6]. Учитывая это, основной задачей становится определение области применения. То есть разбиение реального трафика на классы, определение классов, для которых применение алгоритма неэффективно и получение для реального трафика результирующего алгоритма работы менеджера транзакций включающего в себя обработку как традиционными методами

Следует заметить что $v$ – это суммарная интенсивность к каждому общему ресурсу. Обозначим интенсивность к каждому общему ресурсу как $\lambda_к$. Тогда:

$$SE = \int_0^t (\sum_{k=1}^\infty (\lambda_k \left( p_0 t_у + \sum_1^\infty p_{i.н} \right) + \frac{t_{фк}}{\tau} + \frac{t_{ск}}{\tau})dt$$

или

$$\frac{d(SE)}{dt} = \left( \sum_{k=1}^\infty \lambda_k \left( p_0 t_у + \sum_1^\infty p_{i.н} \right) + \frac{t_{фк}}{\tau} + \frac{t_{ск}}{\tau} \right),$$

где: $p_0 = e^{-\lambda_k \tau},\ p_i = \frac{(\lambda_k \tau)^i e^{-\lambda_k \tau}}{i!}$.

Можно расписать для каждого общего ресурса:

$$\frac{d(SE_k)}{dt} = \lambda_k (p_0 t_у + \sum_1^\infty p_{i.н} + \frac{t_{фк}}{\tau} + \frac{t_{ск}}{\tau});$$

где: $\frac{d(SE)}{dt} = \sum_{k=1}^\infty \frac{d(SE_k)}{dt}$.

### 4. Расчёт времени обработки заявки с учётом трафика

Наиболее простой способ классификации – это следовать тем типам команд, которые существуют в языке SQL, а именно: чтение данных (SELECT),чтение предполагающее блокирование (SELECT FOR UPDATE), добавление данных (INSERT),изменение существующих данных (UPDATE), удаление данных (DELETE). Для каждого класса команд распишем ($p_0$,$p_1$,$p_2$…$p_i$) и времена $t_y$ и $t_{iн}$. Следует указать что расчёт $t_y$ и $t_{iн}$. Для каждого случая следует проводить эмпирическим путём, поскольку разброс

значений, вызванный эффективностью индексации данных и процентом кэширования, может, быть отличаться на порядок и более. Стоит скорее ориентироваться на $t_{лIO}$ предполагая доступ к данным через индекс. Для OLTP систем такое допущение верно. В хранилищах при фул-сканах в грид-системах подход в принципе другой – предполагающий распараллеливание в каждой SQL команде. Имеем:

$$\frac{d(SE_k)}{dt} = \lambda_k(p_0 t_{\text{у}} + \sum_1^\infty p_i t_{i.\text{н}} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau});$$

где: $p_0$=1, $p_i$=0 ($i$=1,2,3,4…).

Для чтений (sel), поскольку они неблокирующие $t_{усп}=t_{\text{лIO}}$ тогда,

$$\frac{d(SE_{k.sel})}{dt} = \lambda_{k.sel}(t_{лIO} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau});$$

Для вставок (ins), поскольку они также неблокирующие – аналогично:

$$\frac{d(SE_{k.ins})}{dt} = \lambda_{k.ins}(t_{лIO} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau});$$

Для операций UPDATE в случае неуспеха необходимо воспроизвести и применить столько изменений, сколько конфликтов произошло. Воспроизведение изменений на момент начала транзакции займёт $t_{лIO}$. Каждое изменение на прошедший конфликт осуществляется в памяти, а значит $i$-кратное разрешение конфликта менее продолжительно чем $t_{лIO}$. Таким образом можно считать что $t_у=t_{лIO}$, а $t_н=2t_{лIO}$. Также распишем вероятности возникновения 0 конфликтов, 1 конфликта, 2х и т. д.

$$p_0 = e^{-\lambda_{k.upd}\tau},$$

$$p_i = \frac{(\lambda_k\tau)^i e^{-\lambda_k\tau}}{i!} \quad (i = 1,2,3,4,\dots)$$

Имеем:

$$\frac{d(SE_k)}{dt} = \lambda_k(p_0 t_{\text{у}} + \sum_1^\infty p_i t_{i.\text{н}} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau} =$$

$$\lambda_k(p_0 t_{лIO} + 2t_{лIO}\sum_1^\infty p_i \quad + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau} =$$

$$\lambda_k(p_0 t_{лIO} + 2t_{лIO}\sum_1^\infty p_i \quad + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau} =$$

$$\lambda_k(e^{-\lambda_{k.upd}\tau} t_{лIO} + 2t_{лIO}(1 - e^{-\lambda_{k.upd}\tau}) + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau}) =$$

$$\lambda_k(2t_{лIO} - t_{лIO}e^{-\lambda_{k.upd}\tau} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau}) \approx$$

$$\lambda_k\left(1{,}5t_{лIO} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau}\right).$$

Коэффициент 1.5 в данном случае условный, поскольку принимается пятидесятипроцентная вероятность возникновения конфликта. В принципе такое упрощение снимает зависимость от того что поток должен быть пуассоновским (Что существенно расширяет сферу применения алгоритма). Также, если провести более точные измерения этих времён обработки успешной и неуспешной заявки, мы получим то, что экспоненциальная компонента будет складываться с постоянными временами обработки, внося погрешность тем большую, чем больше соотношение интенсивности потока к общему k-тому ресурсу ко времени обхода круга. Аналогично выглядит и ситуация с DELETE и SELECT FOR UPDATE.

Рассчитаем для 2PL(2 Phase Locking Algorithm):

$$\frac{DB}{dt} = 2\nu t_{\text{с}} + Q_{\text{к}} + Q_{\text{ф.п}} + Q_{\text{а}};$$

$$\frac{SE}{dt} = 2\nu t_{\text{с}}.$$

Сравним производные $SE$ для обоих алгоритмов. Поскольку в точке t=0 значения $SE$ в обоих случаях =0 и совпадают, и обе производные всегда больше 0 – то сравнение производных можно применять вместо сравнения $SE$. Для случаев SELECT и INSERT:

$$\lambda_k\left(t_{лIO} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau}\right) < 2\nu t_{\text{с}}$$

$$\lambda_k t_{\text{лок}IO}\left(\quad 1 + \frac{t_{\text{фк}}}{\nu\tau t_{лIO}} + \frac{t_{\text{ск}}}{\nu\tau t_{лIO}}\right) < 2\nu t_{\text{с}};$$

$$1 + \frac{t_{\text{фк}}}{\nu\tau t_{лIO}} + \frac{t_{\text{ск}}}{\nu\tau t_{лIO}} \approx 1;$$

$$\lambda_k t_{лIO} < 2\nu t_{\text{с}}.$$

В данном случае мы пренебрегли выражением в скобках, поскольку $\tau$ мы можем увеличивать в широких пределах. Распишем для случая UPDATE:

$$\lambda_k(2t_{лIO} - t_{лIO}e^{-\lambda_{k.upd}\tau} + \frac{t_{\text{фк}}}{\nu\tau} + \frac{t_{\text{ск}}}{\nu\tau}) < 2\nu t_{\text{с}};$$

$$2\lambda_k t_{лIO}\left(1 - \frac{e^{-\lambda_{k.upd}\tau}}{2} + \frac{t_{\text{фк}}}{2\nu\tau} + \frac{t_{\text{ск}}}{2\nu\tau}\right) < 2\nu t_{\text{с}};$$

$$\frac{t_{\text{фк}}}{2\nu\tau} + \frac{t_{\text{ск}}}{2\nu\tau} - \frac{e^{-\lambda_{k.upd}\tau}}{2} \approx 1;$$

$$2\lambda_k t_{лIO} < 2\nu t_{\text{с}}.$$

С учетом того, что увеличивая время обхода круга $\tau$ отношение $1/\nu\tau$ можно сделать сколь угодно малым, приравниваем выражение в скобках к 1. Поскольку $\lambda_k$ всегда меньше *v,* неравенство считаем доказанным. Аналогично выглядит и ситуация с DELETE и SELECT FOR UPDATE.

Таким образом, мы показали, что среднее время обработки заявки (а это *SE* разделённое на количество сессий) при использовании неблокирующего алгоритма меньше. Причём учитывая то, что $t_c$ всегда больше $t_{лок.IO}$ на порядок (а то и два) эффективность неблокирующего алгоритма фиксации распределённых транзакций на порядок эффективнее традиционного алгоритма 2PL[6] при сравнении возможности коллапса по блокировкам.

Дальнейшее усовершенствование алгоритма лежит в минимизации задержек на выполнении tфк и tск.

## Список литературы


1. Гусев Е.И. Исключение блокирования общего ресурса в распределённых системах / Гусев Е.И.; Кулаков А.Ю. // Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка. – 2011. Випуск 54. – С.162 – 166.
2. Härder, Theo; Reuter, Andreas (December 1983). "Principles of Transaction-Oriented Database Recovery" (PDF). ACM Computing Surveys 15 (4): 287–317. doi:10.1145/289.291.
3. Open Group Standard DRDA, Version 5, Volume 1: Distributed Relational Database Architecture (DRDA) ISBN: 1-931624-91-7 Document Number: C112
4. Open Group Standard DRDA, Version 5, Volume 2: Formatted Data Object Content Architecture (FD:OCA) ISBN: 1-931624-92-5 Document Number: C113
5. Open Group Standard DRDA, Version 5, Volume 3: Distributed Data Management (DDM) Architecture ISBN: 1-931624-93-3 Document Number: C114
6. Philip A. Concurrency Control and Recovery in Database Systems/ Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman// Addison Wesley Publishing Company, 1987, ISBN 0-201-10715-5