УДК 004.67

Гусев Е.И.

ОПТИМИЗАЦИЯ ДОСТУПА К РАСПРЕДЕЛЁННЫМ СТРАНИЦАМ ПАМЯТИ В CLOUD COMPUTING СИСТЕМАХ ОСНОВАННЫХ НА SHARED EVERYTHING АРХИТЕКТУРЕ ИСПОЛЬЗУЯ МЕТОД РАЗГРУЗКИ ОЧЕРЕДЕЙ

Национальный технический университет Украины
"Киевский политехнический институт"

eugene@levelextra.ru

Реализованная в Oracle RAC shared everything архитектура использует наиболее перспективный на сегодня способ обработки общего ресурса в Cloud Computing СУРДБ при обслуживании OLTP трафика. В данной работе рассмотрен вариант оптимизации этого подхода, повышающий устойчивость на коллапс традиционных алгоритмов обработки общих страниц

Ключевые слова: Cloud Computing, метод разгрузки очередей, обработка общего ресурса, распределённые страницы памяти

Постановка задачи

В работе [1] была рассмотрена проблема коллапса по *блокировкам Cloud Computing* системах. Применение алгоритмов 2 фазного блокирования [2] и 2х фазной фиксации транзакции [3] часто становится узким местом в *Cloud Computing* системах. Именно поэтому технология *Oracle RAC* [4] не применяет озвученные алгоритмы, а использует собственный подход, известный как *shared everything*. Более подробно модель доступа к странице данных в *Oracle RAC* рассмотрена в [5]. Однако стоит заметить, что классическая схема работы с общим ресурсом глобального кэша, страдает существенным недостатком – низкой устойчивостью на коллапс по блокировкам [1] и как следствие ограниченными возможностями масштабирования. Ограничения в первую очередь касаются высоких требований ко времени отклика сети [6]. Это не позволяет использовать глобальные сети в качестве транспорта распределённого облака и фактически сводит на нет попытки реализовать с помощью *shared everything* подхода дистанционно разнесённое облако. Выявление и устранение недостатков *shared everything* подхода являются целью данной работы. В качестве альтернативы традиционному способу доступа к странице во время пиковых нагрузок, когда резко увеличиваются длины очередей к «горячим» страницам предлагается метод разгрузки очередей.

Выбор математической модели задержек в Oracle RAC учитывающей возникновение конфликта

Рассмотрим общую ситуацию с доступом к ресурсам, ожидающим очередь. В работе [5] было показано что в *shared everything* системах время доступа к странице без учёта ожидания очереди $2t_{net}+t_{send}$, а время ожидания очереди t_{queue}. Очевидно, что общее время t_{acc}, – это сумма $2t_{net}+t_{send}$ и t_{queue}.

Моделируя входящий поток доступа к страницам стационарным пуассоновским потоком, получаем отражённое в таблице 1 распределение ряда времён обработки (верхняя строка) и соответствующих им вероятностей (нижняя строка) в зависимости от количества конфликтов:

Табл. 1. Времена обработки и вероятности

$t_{send}/2$	$2t_{send}/2$	...	$nt_{send}/2$
$(\nu_{pl} t_{wpl}) e^{-\nu_{pl} t_{wpl}}$	$\frac{(\nu_{pl} t_{wpl})^2}{2!} e^{-\nu_{pl} t_{wpl}}$	...	$\frac{(\nu_{pl} t_{wpl})^n}{n!} e^{-\nu_{pl} t_{wpl}}$

Время t_{wpl} – это время в течении, которого проводится наблюдение. Не сложно вычислить среднее время

дополнительного ожидания как сумму ряда:

$$t_{queue}=\sum_{n=1}^{\infty}\frac{t_{send}}{2}\frac{\left|v_{pl}t_{wtl}\right|^{n}}{n!}e^{-v_{pl}t_{wpl}}=\frac{v_{pl}t_{wpl}t_{send}}{2} \quad (1)$$

Нас в первую очередь интересует время в течении которого ресурсом может пользоваться одна заявка – т.е. время блокирования страницы. Это сумма t_{send} и t_{queue}. Получаем уравнение и решаем его:

$$t_{wpl}=t_{send}+\frac{v_{pl}t_{wpl}t_{send}}{2} \quad => \quad t_{wpl}=\frac{2t_{send}}{2-v_{pl}t_{send}} \quad (2)$$

Таким образом, общее время доступа к странице традиционным для *shared everything* способом равно:

Необходимо заметить, что данная

$$t_{acc}=2t_{net}+t_{send}+\frac{v_{pl}t_{send}^{2}}{2-v_{pl}t_{send}}=2t_{net}+\frac{2t_{send}}{2-v_{pl}t_{send}} \quad (3)$$

формула имеет смысл лишь в том случае если количество узлов облачной СУРБД достаточно большое. В противном случае необходимо учитывать эффект т. н. естественной оптимизации – т. е. такой оптимизации когда страница уже находилась на выполняющем обработку узле либо узел, выполняющий обработку является мастер узлом. В указанных случаях узлу не имеет смысла по сети обращаться к самому себе и время доступа сокращается. Такой подход уже был отражён в статье [5], однако приведём таблицу с временами доступа без учета ожиданий в зависимости от закешированности запрашиваемой страницы локально и того, является ли обрабатывающий узел мастером или нет:

l и *m* в табл. 2 обозначают события закешированности на локальном узле (l) и то, что обрабатывающий запрос узел оказался мастер-узлом. l и *m* с верхней чертой – соответственно – обратные события.

Рассмотрим теперь влияние фактора естественной оптимизации на время ожидания страницы (обозначим t_{pw}). Если длина очереди *n*, то при обработке каждых двух соседних запросов вероятна ситуация, что запросы с одного узла. В случае если все узлы имеют одинаковую ёмкость и, учитывая, что поток стационарный, для количества узлов *k* вероятность такого события 1/k.

Табл.2. Время доступа без ожиданий очереди

	l m	l̄ m	l m̄	l̄ m̄	Среднее время доступа
xx	0	$t_{net}+t_{send}$	0	$2t_{net}+t_{send}$	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}-\frac{t_{net}}{n}\right)$
sx	0	$t_{net}+t_{send}$	$2t_{net}$	$2t_{net}+t_{send}$	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}+\frac{t_{net}}{n}\right)$
xs	0	$t_{net}+t_{send}$	0	$2t_{net}+t_{send}$	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}-\frac{t_{net}}{n}\right)$
ss	0	$t_{net}+t_{send}$	0	$2t_{net}+t_{send}$	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}-\frac{t_{net}}{n}\right)$

Количество «соседств» между запросами в очереди на 1 меньше длины очереди, но есть еще одно «соседство» между первой пересылкой и началом очереди во время которого вероятно, что заявки будут с одного узла. Таким

$$t_{pw.n}=n\frac{t_{send}}{2}-\frac{1}{k}\left(n\frac{t_{send}}{2}\right)=n\frac{t_{send}}{2}\left(1-\frac{1}{k}\right) \quad (4)$$

образом, для очереди длиной *n* получаем:

Поскольку распределение вероятностей количества событий не поменялось, при вычислении среднего времени получаем:

Преобразовав аналогично (1) получаем:

$$t_{pw}=\sum_{n=1}^{\infty}\left(\left(n\frac{t_{send}}{2}\left(1-\frac{1}{k}\right)\right)\frac{\left(v_{pl}t_{wpl}\right)^{n}}{n!}e^{-vt_{wpl}}\right) \quad (5)$$

$$t_{pw}=\frac{t_{send}}{2}\left(\left(1-\frac{1}{k}\right)v_{pl}t_{wpl}\right) \quad (6)$$

Время t_{wpl} – это время в течении которого ожидается конфликт. В нашем случае это

$$t_{wpl}=t_{send}+\frac{t_{send}}{2}\left(\left(1-\frac{1}{k}\right)v_{pl}t_{wpl}\right) \quad (7)$$

сумма t_{send} и t_{pw}.

Решение полученного уравнения:

$$t_{wpl}=\frac{2t_{send}}{2-v_{pl}t_{send}\left(1-\frac{1}{k}\right)}=\frac{2kt_{send}}{2k-v_{pl}t_{send}(k-1)} \quad (8)$$

откуда:

$$t_{pw}=\frac{(k-1)v_{pl}t_{send}^{2}}{2k-v_{pl}t_{send}(k-1)} \quad (9)$$

Приведём в таблице 3 задержки с учетом ожидания в очереди во всех базовых сценариях, поскольку в сценарии нет очередей за общий ресурс, то и время ожидания не считаем, символ количества узлов снова обозначим как n, а не *k*:

Табл.3. Задержки с учетом ожидания очереди

xx	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}-\frac{t_{net}}{n}\right)+\frac{(n-1)v_{pl}t_{send}^2}{2n-v_{pl}t_{send}(n-1)}$
sx	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}+\frac{t_{net}}{n}\right)+\frac{(n-1)v_{pl}t_{send}^2}{2n-v_{pl}t_{send}(n-1)}$
xs	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}-\frac{t_{net}}{n}\right)+\frac{(n-1)v_{pl}t_{send}^2}{2n-v_{pl}t_{send}(n-1)}$
ss	$\left(1-\frac{1}{n}\right)\left(2t_{net}+t_{send}-\frac{t_{net}}{n}\right)$

В процессе моделирования трафика необходимо определить вероятности *xx, sx, xs* и *ss*. При определении эффективности алгоритма разгрузки очереди лучше использовать модель трафика, удовлетворяющую интерфейсу модели. Это даст возможность сопоставить и среднее время отклика, и среднюю длину очереди для различных подходов к доступу при изменении интенсивности как основного параметра трафика. Примером моделирования такого трафика может быть элементарный трафик с обратной связью, подробно описанный в работе [7], где он упоминается под названием «простейший». Однако для оценки эффективности алгоритма достаточно привести зависимость времени отклика, с учётом времени ожидания очереди от интенсивности конфликта, что и будет сделано в данной работе.

Описание метода разгрузки очереди.

Идея метода достаточно проста: если нарастает очередь – то необходимо снизить количество пересылок между узлами. Для этого выбирается узел, который эксклюзивно будет вносить изменения в «горячую» страницу. Это позволит избежать очередей, которые образуются из-за увеличения времени отклика вследствие добавления ко времени обработки времён пересылок между узлами. Принципиально также заложить в алгоритм возможность мигрировать с узла, вносящего изменения на другой узел, для обеспечения динамического балансирования нагрузки.

Распишем возможную структуру обмена сообщениями для реализации вышеперечисленных требований.

1. При возникновении длинной очереди мастер узел должен перевести страницу в режим разгрузки, т. е. узел выполняющей обработку в данный момент времени должен быть уведомлен о переводе страницы в режим разгрузки и назначен узлом выполняющим разгрузку (в дальнейшем узел-хостер). Это выдвигает требования к мастер узлу отслеживать историю пересылаемых сообщений, в противном случае очередь начнёт разгружаться только после обслуживания запросов страниц, пришедших после момента перевода в режим разгрузки.

2. На запросы страниц мастер-узел даёт ответ, с указанием узла-хостера, которому надо пересылать пакет с действиями, выполняемыми над страницей.

3. Обрабатывающий узел формирует и пересылает хостеру своё задание.

4. Хостер вносит изменения в страницу и пересылает текущую версию страницы обрабатывающему узлу. На этом этапе возможна оптимизация, которую мы при моделировании учитывать не будем: вместо пересылки всей страницы пересылаются изменения с последнего кэширования на узле обработчике.

5. Обрабатывающий узел получает страницу в состоянии, как будто он сам внёс в неё изменения и сразу же отправил дальше по запросам других страниц.

Важным моментом в этом алгоритме является минимизация ресурсов на мониторинг возникновения очереди. В данной работе мы отметим важность этих вопросов но при моделировании будем предполагать, что очередь начнёт разгружаться только начиная с обслуживания запросов страниц, пришедших после момента перевода в режим разгрузки.

Математическая модель метода разгрузки очередей

Рассмотрим описанный выше алгоритм в задержках. На пересылка мастер узлу запроса и получение от него ответа это 2 t_{net}. Отправка «задания на изменение страницы» это также t_{net}, *а* ответ хостера с пересылкой актуальной версией страницы со сделанными изменениями – это t_{send}. Таким образом общее время:

$$t_{acc} = 3t_{net} + t_{send} \quad (10)$$

С учётом естественной оптимизации, поскольку в SCUR состояние блок не переводится, а временем стоянием в очереди, поскольку оно не включает в себя времена пересылок, мы пренебрегаем, получаем зависимость времени доступа от того является ли узел мастером или хостером отражённую в таблице 4:

Табл.4. Время доступа при разгрузке очередей

	m	h	$\overline{\text{h}}\,\overline{\text{m}}$	Среднее время доступа
xx	$t_{net}+t_{send}$	0	$3t_{net}+t_{send}$	$3t_{net}+t_{send}-\frac{1}{n}(5t_{net}+t_{send})$
sx	-	-	-	-
xs	$t_{net}+t_{send}$	0	$3t_{net}+t_{send}$	$3t_{net}+t_{send}-\frac{1}{n}(5t_{net}+t_{send})$
ss	-	-	-	-

h в данном случае обозначает что обращение было к узлу-хостеру, а m что обращение к узлу мастеру. Очевидно, что разнесение мастера и хостера на разные узлы даёт уменьшение времени отклика. Как предельный случай для 2х узлов:

$$t_{acc} = \frac{1}{2}(t_{net} + t_{send}) \quad (11)$$

Но самым главным плюсом является устойчивость на коллапс по страницам памяти, который при таком подходе не возникнет при соотношении:

$$\nu_{pi} = \frac{1}{t_{send}} \quad (12)$$

Отобразим полученные результаты графически. На приведенных 4х графиках. Анализируя результаты видно, что метод разгрузки очередей неэффективен при низких интенсивностях конфликта за общую страницу, однако при приближении к критической интенсивности перехода в коллапс становится эффективным. Более того, метод разгрузки очередей вообще не приводит к коллапсу по доступу к общим страницам при интенсивностях больших чем $1/t_{send}$.

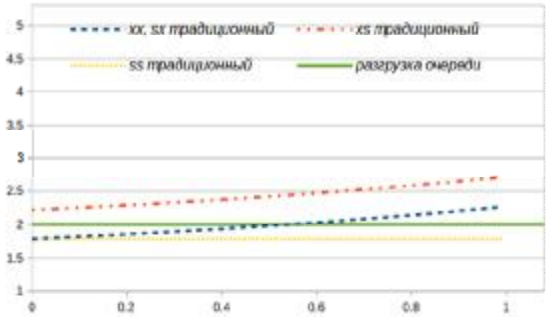

Рис.1. Результаты сравнения при n=3 и $t_{send}=t_{net}$

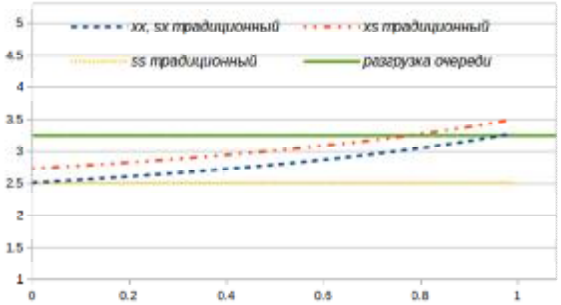

Рис.2. Результаты сравнения при n=8 и $t_{send}=t_{net}$

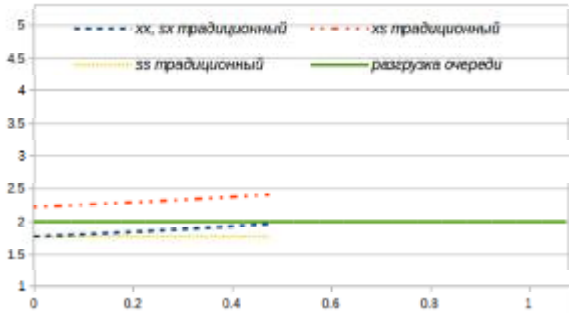

Рис.3. Результаты сравнения при n=3 и $t_{send}=2t_{net}$

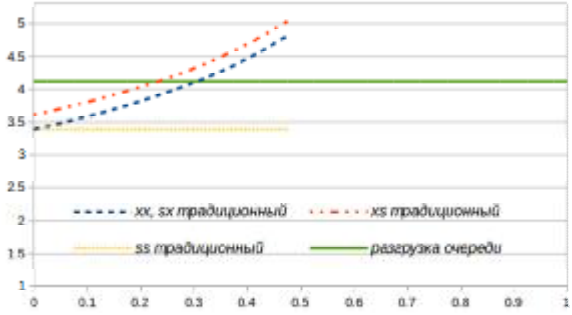

Рис.4. Результаты сравнения при n=8 и $t_{send}=2t_{net}$

Выводы

Предложенный метод разгрузки очередей позволяет решить проблему коллапса за общие страницы в облачных СУРБД использующих shared everything архитектуру. При этом остаётся возможность использовать его совместно с традиционными алгоритмами доступа к странице. Заметим, что применение алгоритма всегда увеличивает количество сетевых пакетов и как следствие – объём пересылаемых данных, что является платой за уход от коллапса по общим страницам.

Сравнивая предложенный метод с традиционными техниками ухода от коллапса по общим страницам через снижение вероятности запроса страницы стоит отметить дополняющий эффект к традиционным техникам, а также возможность снижать вероятность коллапса в тех случаях когда коллапс за страницу связан с единственным горячим ресурсом страницы (например горячей строкой). Детальное сравнение и комбинирование метода разгрузки очередей с традиционными техниками ухода от коллапса за страницу (среди которых стоит отметить наиболее популярные: секционирование и реверсивный индекс) представляет интерес однако выходит за рамки задач очерченных данной работой. Обозначая рамки последующего исследования предложенного метода необходимо обратить внимание на процесс активации метода разгрузки очередей при их возникновении и исследовать различные варианты как прогнозирования возникновения очереди так и реактивной разгрузки очереди при её возникновении в контексте исследования задержек переходного процесса и потребления ресурсов.

Список литературы

1. Гусев Е.И. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций – Вісник НТУУ "КПІ".Сер. Інформатика, управління та обчислювальна техніка. – 2012. Випуск 57. – С.76-80

2. Menasce D. A., Popek G. J., Muntz R. R. "A locking protocol for resource coordination in distributed databases," ACM Trans. Database Syst. 5, 2 (June 1980), C. 103-138.

3. Lampson B., Sturgis H. "Crash recovery in a distributed data storage system," Tech. Rep., Computer Science Lab., Xerox Palo Alto Research Center, Palo Alto, Calif., 1976

4. Markus Michalewicz, Burt Clouse, John McHugh Oracle Real Application Clusters (RAC). – Oracle White Paper . June 2013

5. Гусев Е.И. Математическое моделирование распределённой кластерной системы использующей shared everything подход (Oracle RAC). – Вісник НТУУ "КПІ".Сер. Інформатика, управління та обчислювальна техніка. – 2014. Випуск 60. – С.106-113

6. Гусев Е.И, Кулаков А.Ю. "Исключение блокирования общего ресурса в распределённых системах". - Вісник НТУУ "КПІ".Сер. Інформатика, управління та обчислювальна техніка. – 2011. Випуск 54. – С.162-166.

7. Гусев Е.И. Моделирование трафиков и оценка скорости распределённого доступа в системах облачных вычислений с общим ресурсом на примере Oracle RAC – Вісник НТ УУ "КПІ".Сер. Інформатика, управління та обчислювальна техніка. – 2015. Випуск 62. – С. 32-42.

Статью представлено в редакцию 18.12.2015