Алгоритм замены страниц LRU-K для дисковой буферизации баз данных

Элизабет Дж. О'Нил [1] **Патрик Э. О'Ней** [1] **Эрхард У** [2]

1 Факультет математики и компьютерных наук
Массачусетский университет в Бостоне
Кампус Харбор
Бостон. MA **02125-3393**

2 Факультет компьютерных наук
ETH Zurich
CH-8092 Цюрих
Переключено и

E-mai1: eoneilG'cs.umb.edu, poneilHcs.umb.edu, weikumHinf.ethz.ch

АБСТРАКТ

В данной статье представлен новый подход к буферизации дисков баз данных, названный методом LRU-K. Основная идея LRU-K заключается в отслеживании времени последних *K* обращений к популярным страницам базы данных и использовании этой информации для статистической оценки времени между обращениями на основе каждой страницы. Хотя подход LRU-K обеспечивает оптимальный статистический вывод при относительно стандартных предположениях, он довольно прост и не требует больших затрат на хранение данных. Как мы продемонстрировали в ходе имитационных экспериментов, алгоритм LRU-K превосходит обычные алгоритмы буферизации в различении часто и редко ссылающихся страниц. Фактически, LRU-K может приблизиться к поведению алгоритмов буферизации, в которых наборы страниц с известной частотой доступа вручную назначаются различным буферным пулам специально подобранных размеров. Однако, в отличие от таких буферных алгоритмов, метод LRU-K является самонастраивающимся и не зависит от внешних подсказок о характеристиках рабочей нагрузки. Более того, алгоритм LRU-K адаптируется в реальном времени к изменяющимся частотам доступа.

1. Введение

1.1 Постановка проблемы

Все системы баз данных сохраняют дисковые страницы в буферах памяти в течение некоторого времени после того, как они были считаны с диска и к ним обратилось определенное приложение. Это делается для того, чтобы сохранить популярные страницы в памяти и сократить дисковый ввод-вывод. В своем "Правиле пяти минут" Грей и Путцолу предлагают следующий компромисс: мы готовы платить больше за буферы памяти до определенного момента, чтобы снизить стоимость дисковых манипуляторов для системы ([GRAYPUT], см. также [CKS]). Критическое решение о буферизации возникает, когда требуется новый слот буфера для страницы, которая должна быть считана с диска, а все текущие буферы уже заняты: Какая текущая страница должна быть удалена из буфера? Это называется политикой замены страниц, и различные алгоритмы буферизации получили свои названия от типа политики замены, которую они предполагают (см., например, [COFFDENN], [EFFEHAER]).

SIGMOD /5/93/Washington, DC.JJSA
• 1 993 ACM 0-89791-592-5/93/0005/0297...4 J .50

Алгоритм, используемый почти всеми коммерческими системами, известен как LRU, то есть Least Recently Used. Когда требуется новый буфер, политика LRU удаляет из буфера страницу, к которой не обращались дольше всего. Буферизация LRU изначально разрабатывалась для моделей использования в логике структурирования (например, [DENNING], [COFFDENN]), и не всегда хорошо вписывается в среду баз данных, что также отмечалось в [REITER], [STONj, [SACSCH] и [CHOUDEW]. На самом деле алгоритм буферизации LRU имеет проблему, которая решается в текущем ра pe r: он решает, какую страницу удалить из буфера, на основе слишком малого количества информации, ограничиваясь только временем последнего обращения. s rcnly, LRU не в состоянии отличить страницы с относительно частыми обращениями от страниц с очень редкими обращениями, пока система не потратит много ресурсов, удерживая страницы с редкими обращениями в буфере в течение длительного периода времени.

Пример 1.1. Рассмотрим многопользовательскую базу данных, которая обращается к случайно выбранным записям клиентов через кластеризованное B-дерево с индексированным ключом CUST-ID, чтобы получить необходимую информацию (см. [TPC-A]). Предположим упрощенно, что существует 20 000 клиентов, что запись о клиенте имеет длину 2000 байт, и что пространство, необходимое для индекса B-дерева на уровне листьев, включая свободное пространство, составляет 20 байт для каждой записи ключа. Тогда, если страницы диска содержат 4000 байт полезного пространства и могут быть заполнены, нам потребуется 100 страниц для хранения узлов листового уровня B-дерева индекса (существует один корневой узел B-дерева) и 10 000 страниц для хранения записей. Схема обращения к этим страницам (без учета корневого узла B-дерева) выглядит следующим образом: 11, R1, 12, R2, 13, R3, …, чередующиеся ссылки на страницы листьев случайного индекса и страницы записей. Если для данного приложения мы можем позволить себе буферизировать в памяти только 101 страницу, то корневой узел B-дерева является автоматическим; мы должны буферизировать *все* листовые страницы B-дерева, поскольку каждая из них ссылается с вероятностью .005 (один раз на каждые 200 обращений к общей странице), в то время как явно расточительно вытеснять одну из этих листовых страниц страницей данных, поскольку страницы данных имеют вероятность обращения только .00005 (один раз на каждые 20 000 обращений к общей странице). При использовании алгоритма LRU страницы, хранящиеся в буферах памяти, будут сотней последних обращений. В первом приближении это означает 50 страниц листьев B-дерева и 50 страниц записей. Учитывая, что страница не получает дополнительных заслуг за то, что на нее дважды ссылались в недавнем прошлом, и что lbis с большей вероятностью произойдет со страницами листьев B-дерева, данных будет даже немного *больше.*

страниц в памяти, чем листовых страниц. Такое поведение явно не подходит для очень распространенной парадигмы доступа к диску.D

Пример 1.2. В качестве второго сценария, когда LRU сохраняет в кэше неподходящие страницы, рассмотрим многопроцессное приложение базы данных с хорошей "локальностью" ссылок на общие страницы, так что на 5000 буферизованных страниц из 1 миллиона дисковых страниц приходится 95 % обращений со стороны параллельных процессов. Теперь, если несколько пакетных процессов начнут "последовательное сканирование" всех страниц базы данных, страницы, считанные последовательными fiCdHS, заменят часто ссылающиеся страницы в буфере wia страницами, на которые вряд ли будут ссылаться снова. Это распространенная жалоба во многих коммерческих ситуациях: захламление кэша последовательным сканированием приводит к заметному ухудшению времени интерактивного отклика. Время отклика увеличивается потому, что страницы, считываемые при последовательном сканировании, используют дисковые манипуляторы для замены страниц, обычно хранящихся в буфере, что приводит к увеличению объема ввода-вывода для страниц, которые обычно остаются резидентными, и поэтому образуются длинные очереди ввода-вывода.O

Повторим проблему, которую мы видим в этих двух примерах: LRU не может отличить страницы, на которые ссылаются относительно часто, от страниц, на которые ссылаются очень редко. После того как страница была считана с диска, алгоритм LRU гарантирует ей долгую жизнь в буфере, даже если к ней никогда не обращались раньше. В литературе были предложены решения этой проблемы. Предыдущие подходы делятся на следующие две основные категории.

- *Страница Пул Тантинг:*
  Рейтер в своем алгоритме Domain Separation [REITER] предложил, чтобы DBA давал более точные подсказки о пулах страниц, к которым осуществляется доступ, разделяя их по существу на различные пулы буферов. Таким образом, страницы узлов B-дерева могли бы конкурировать за буферы только с другими страницами узлов, страницы данных - только с другими страницами данных, а DBA мог бы ограничить объем буферного пространства, доступного для страниц данных, если повторное обращение к ним кажется маловероятным. Такие возможности "настройки пула" поддерживаются некоторыми коммерческими системами баз данных и используются в приложениях с высокими требованиями к производительности (см., например, [TENGGUM, DANTOWS, SHASHA]). Проблема
  Этот подход требует больших человеческих усилий и не позволяет должным образом решить проблему эволюционирующих шаблонов доступа к горячим точкам (локальность в пуле страниц данных, меняющаяся с течением времени).

- *Анализ плана выполнения **запросов:***
  Еще одно предложение заключалось в том, что оптимизатор запросов должен предоставлять больше информации о типе использования плана выполнения запроса, чтобы система знала о вероятности повторного обращения к плану и могла действовать соответствующим образом (см. модель Hot Set Model из [SACSCH], алгоритм DBMIN из [CHOUDEW]) и его расширения [FNS, NFS, YUCORN], подходы hint- passing из [CHAKA, HAAS, ABG, JCL, COL] и предиктивный подход из [PAZDO]). Этот подход может хорошо работать в условиях, когда повторное выполнение одного и того же плана является основным фактором буферизации.

В примере 1.2, приведенном выше, мы, предположительно, знаем достаточно, чтобы отбросить страницы, прочитанные последовательными `sts`. Сайт
Алгоритм DBMIN также хорошо справляется с референтными задачами.
если бы вся строка ссылок была получена одним запросом. Однако планы запросов для простых многопользовательских транзакций из примера 1.1 не отдают предпочтения сохранению страниц B-дерева или страниц данных в буфере, поскольку каждая страница ссылается ровно один раз в течение плана. В многопользовательских симуляторах планы оптимизации запросов могут пересекаться сложным образом, а рекомендательные алгоритмы оптимизаторов запросов не говорят нам, как учесть такое пересечение. Для принятия такого решения должна существовать более глобальная политика замены страниц.

1.2 Вклад газеты

Обе вышеперечисленные категории решений исходят из того, что, поскольку LRU плохо различает часто и редко ссылающиеся страницы, необходимо, чтобы какой-то другой агент предоставлял подсказки того или иного рода. Вклад данной статьи заключается в разработке нового самодостаточного алгоритма замещения страниц, который учитывает больше истории доступа к каждой странице, чтобы лучше различать страницы, которые следует хранить в буфере. Такой подход представляется разумным, поскольку история страниц, используемая алгоритмом LRU, весьма ограничена: просто время последнего обращения.

В этой статье мы тщательно исследуем идею учета истории двух последних ссылок, или, более точно, последних K ссылок, K > 2. Конкретный алгоритм, разработанный в этой статье, который учитывает знание двух последних ссылок на страницу, называется LRU-2, а естественное обобщение - алгоритм LRU-K; мы называем классический алгоритм LRU в этой таксономии как LRU-.
1. Оказалось, что при K > 2 алгоритм LRU-K обеспечивает несколько лучшую производительность по сравнению с LRU-2 при постоянных шаблонах доступа, но менее чувствителен к изменениям в шаблонах доступа, что важно для некоторых приложений.

Несмотря на то, что алгоритм LRU-K извлекает пользу из дополнительной информации о частоте обращения к странице, LRU-K принципиально отличается от алгоритма замены на наименее часто используемую (LFU). Принципиальное отличие заключается в том, что LRU-K имеет встроенное понятие "старения", учитывая только последние K обращений к странице, в то время как алгоритм LFU не имеет средств для различения недавней и прошлой частоты обращений к странице. и поэтому не может справиться с изменяющимися шаблонами доступа. LRU-K также сильно отличается от более сложных алгоритмов буферизации на основе LFU, в которых используются схемы старения, основанные на частоте обращений. Эта категория алгоритмов, к которой относятся, например, GCLOCK и варианты LRD [EFFEHAER], в значительной степени зависит от тщательного выбора различных параметров, зависящих от рабочей нагрузки, которые управляют процессом старения. Алгоритм LRU- K, с другой стороны, не требует подобной ручной настройки.

Алгоритм LRU-K обладает следующими важными свойствами:

- Он хорошо различает наборы страниц с разными уровнями частоты обращения (например, индексные страницы и **страницы** данных**).** Таким образом, он приближается к эффекту **назначения** наборов страниц разным буферным пулам специально подобранных размеров. Кроме того, она адаптируется к изменяющимся шаблонам доступа.
- Он **обнаруживает** локальность ссылок в рамках выполнения запросов, между несколькими запросами в одной транзакции, а также локальность между несколькими транзакциями в многопользовательской среде.
- Он самодостаточен, так как не нуждается в посторонней помощи.
  подсказки.
- Она довольно проста и не требует больших затрат на ведение бухгалтерского учета.

Оставшаяся часть данной работы построена следующим образом. В разделе 2 мы представляем основные концепции подхода LRU-K к буферизации дисковых страниц. В разделе 3 мы приводим неформальные аргументы в пользу того, что алгоритм LRU-K является оптимальным в некотором четко определенном смысле, учитывая знание последних K ссылок на каждую страницу. В разделе 4 представлены результаты моделирования производительности LRU-2 и LRU-K в сравнении с LRU-1. В разделе 5 приведены заключительные замечания.

2. Концепции буферизации LRU-K

В данной работе мы рассматриваем поведение страничных ссылок со статистической точки зрения, основываясь на ряде предположений из *модели независимых ссылок* для пейджинга, приведенной в разделе 6.6 [COFFDEN1'sf]. Мы начнем с интуитивной формулировки; более полное математическое развитие описано в [OOW]. Предположим, что нам дано множество N = {1, 2, …, nJ дисковых страниц, обозначаемых целыми положительными числами, и что исследуемая система баз данных делает последовательность ссылок на эти страницы, задаваемых строкой *ссылок*: ri.
r2. . -i. ..., где rt = p (p C N) означает, что термин
Номер t в строке ссылок относится к дисковой странице p. Заметим, что в оригинальной модели [COFFDENNj строка ссылок представляла обращения к странице одного пользовательского процесса, поэтому предположение о том, что строка отражает все обращения системы, является отклонением. В последующих рассуждениях, если не указано иное, мы будем измерять все временные интервалы в терминах подсчета последовательных обращений к странице в строке ссылок, поэтому подстрочный индекс общего термина обозначается 't'. В любой заданный момент времени t мы предполагаем, что каждый
Страница диска p имеет вполне определенную вероятность, bp, быть
следующая страница, на которую ссылается система: Pr(**rt+l = p) = bp,** для всех p K N. Это подразумевает, что строка ссылок является вероятностно-биллистической, последовательностью случайных величин. Изменение шаблонов доступа может изменить эти вероятности ссылок на страницы, но мы предполагаем, что вероятности bp имеют относительно долгий пери-.
оды стабильных значений, и начинаем с предположения, что вероятности не меняются для длины эталонной строки; таким образом, мы предполагаем, что bp не зависит от t.

Gearly, каждая дисковая страница p имеет ожидаемое ссылочное межзвенное время, Ipt**the** время между последовательными вхождениями p в ссылочную строку, и мы имеем: Ip - bp ^{1} Мы намерены, чтобы наша система баз данных использовала подход, основанный на

Байесовская статистика позволяет оценить это время доступа по наблюдаемым обращениям. Затем система пытается сохранить в буферах памяти только те страницы, время прибытия которых оправдывает их размещение, т. е. страницы с наименьшим временем прибытия или, эквивалентно, наибольшей вероятностью обращения. Это статистическое приближение к алгоритму Atj из [COFFDENN], который, как было показано, является оптимальным. Алгоритм LRU-1 (классический LRU) можно представить как использующий такой статистический подход, сохраняя в памяти только те страницы, которые, как кажется, имеют наименьшее время между обращениями; учитывая ограниченность информации LRU-1 о каждой странице, наилучшей оценкой времени между обращениями является интервал времени до предыдущего обращения, и страницы с наименьшими такими интервалами сохраняются в буфере.

Определение 2.1. Обратное K-расстояние bt(p,K). Если известна эталонная строка до момента времени t, то rJ, r2. . rt. обратное K-расстояние bt(p,K) - это расстояние назад
на K последнюю ссылку на страницу p:

bt(pt) = x, если rt- имеет значение p и было ровно K-1 других значений i с
t - x < is t , где r-i P.

= >, если p не встречается по крайней мере K раз в
ri. r2. . , rt

Определение.2. **Алгоритм LRU-K.** Алгоритм LRU-K определяет политику замены страниц, когда буферный слот необходим для новой страницы, считываемой с диска: страница p, которая будет отброшена (т. е. выбрана в качестве замены), - это та, чье обратное K-расстояние, bt(p,K), является максимальным из всех страниц в буфере. Единственный случай, когда выбор неоднозначен, - это когда более одной страницы имеют bt(p,K) = < B этом случае для выбора жертвы повторного размещения среди страниц с бесконечным Backward K-distance можно использовать вспомогательную политику; например, в качестве вспомогательной политики можно использовать классическую LRU. Заметим, что LRU-1 соответствует классическому алгоритму LRU.O

Алгоритм LRU-2 значительно превосходит LRU-1, поскольку, принимая во внимание два последних обращения к странице, мы впервые можем оценить Ip по фактическому времени между обращениями, а не просто по нижней границе времени до самого последнего обращения. Мы используем больше информации, и в результате наши оценки значительно улучшились, особенно в отношении страниц, которые имеют большое время между ссылками и поэтому должны быть быстро удалены из буфера. Обратите внимание, что мы не делаем никаких общих предположений о вероятностном распределении ip. В полной версии статьи [OOW] мы предполагаем экспоненциальное распределение для Ip, чтобы продемонстрировать оптимальность алгоритма LRU-K. Как уже упоминалось, мы моделируем bp и, следовательно, ip US с возможностью случайных изменений с течением времени, предполагая лишь, что изменения происходят достаточно редко, чтобы статистический подход к определению доступа к странице на основе прошлой истории был обычно правомерен. Эти предположения кажутся оправданными для большинства ситуаций, возникающих при использовании баз данных.

2.1. Реалистичные предположения при буферизации БД

Общий алгоритм LRU-K имеет две особенности, характерные для случаев, когда K z 2, которые требуют тщательного рассмотрения для обеспечения правильного поведения в реалистичных ситуациях. Первая, **известная как *Earl y Page Replacement,* возникает в ситуациях**, когда страница, недавно считанная в буфер памяти, не заслуживает сохранения в буфере по стандартным критериям LRU-K, например, потому что **страница имеет значение** bttrfi), равное бесконечности. Очевидно, что мы хотим относительно быстро удалить эту страницу из буфера, чтобы сэкономить ресурсы памяти для более достойных дисковых страниц. Однако мы должны учитывать тот факт, что страница, которая в целом не пользуется популярностью, может все же испытать всплеск корреляционных обращений вскоре после того, как на нее впервые обратились. Мы рассмотрим **эту проблему** в разделе 2.1.1. Вторая особенность, с которой мы должны справиться в случаях, когда K > 2, заключается в том, что необходимо сохранять историю ссылок для страниц, которые в данный момент не присутствуют в буфере. Это является отклонением от существующих алгоритмов замены страниц и будет называться *проблемой сохранения информации* о *ссылках на страницы, которая* рассматривается ниже в разделе
2.1.2. В разделе 2.1.3 при**водится** псевдокод алгоритма буферизации LRU-K, который решает **вышеупомянутые** проблемы.

2.1.1. Ранняя замена страниц и проблема коррелированных ссылок

Чтобы избежать расточительного использования буферов памяти, как показано в примерах 1.1 и 1.2, LRU-K принимает решение о том, следует ли исключить страницу p из резидентного списка после короткого периода времени, прошедшего с момента ее последнего обращения. Канонический период может составлять 5 секунд. Чтобы продемонстрировать необходимость такого тайм-аута, мы зададим следующий вопрос: Что мы подразумеваем под последними nro ссылками на страницу? Ниже мы перечислим четыре способа, которыми может произойти пара ссылок на одну и ту же страницу диска; первые три из них называются коррелированными парами ссылок и, скорее всего, произойдут за короткий промежуток времени.

(1) **Внутритранзакционные.** Транзакция обращается к странице, а затем снова обращается к той же странице до фиксации. Это может произойти с некоторыми транзакциями обновления, сначала считывающими строку, а затем обновляющими значение в ней.

(2) Повторная транзакция. Транзакция получает доступ к странице, затем прерывается и выбывает, а повторная транзакция снова получает доступ к странице с той же целью.

(3) **Внутрипроцессный.** Транзакция ссылается на страницу, затем фиксируется, и следующая транзакция того же процесса снова получает доступ к странице. Такая схема доступа обычно возникает в приложениях пакетного обновления, которые последовательно обновляют 10 записей, фиксируют их, а затем снова обращаются к следующей записи на той же странице.

(4) **Межпроцессный.** Транзакция обращается к странице, а затем (часто другой) процесс обращается к той же странице по независимым причинам. (По крайней мере, пока у нас нет большого количества коммуникаций между процессами, когда информация передается от одного процесса к другому в базе данных

мы можем предположить, что ссылки разных процессов

Напомним, что наша цель буферизации дисковых страниц в памяти - сохранить страницы с относительной долгосрочной популярностью для экономии дискового ввода-вывода. Пример такой долгосрочной популярности приведен в примере 1.1, где на 100 страниц листьев B-дерева часто ссылаются параллельно выполняющиеся транзакции. Смысл в том, что коррелированные типы пар ссылок (1) через
(3) заключается в том, что если мы будем учитывать эти пары ссылок при оценке времени интервального доступа Ip, мы часто будем приходить к неверным выводам. Например, пара ссылок типа (1) может быть обычным шаблоном доступа, так что если после первого обращения типа (1) мы сразу же выкинем страницу из буфера, потому что не видели ее раньше, то для второго обращения нам, вероятно, придется прочитать ее снова. С другой стороны, после второго обращения, когда транзакция завершена, если мы скажем, что у этой страницы короткое время промежуточного прибытия, и оставим ее в буфере на сто секунд или около того, это, скорее всего, будет ошибкой; два коррелированных обращения не являются достаточным основанием для вывода о том, что появятся независимые обращения. Существует несколько очевидных способов решения проблемы коррелированных ссылок, самый простой из которых заключается в следующем: система *не* должна сбрасывать страницу сразу после ее первого обращения, а должна удерживать ее в течение короткого периода времени, пока вероятность зависимого последующего обращения не станет минимальной; тогда страницу можно сбрасывать. В то же время время между обращениями должно рассчитываться на основе некоррелированных пар обращений, где предполагается, что каждое последующее обращение одного и того же процесса в течение периода тайм-аута коррелировано: связь является транзитивной. Мы называем этот подход, который ассоциирует коррелированные ссылки, методом *Time-Out Correlatton*; а *период* тайм-аута - *периодом коррелированных ссылок* Идея не нова; в [ROBDEVj эквивалентное предложение сделано в разделе 2.1, под заголовком: Факторинг локальности.

Математическая формулировка такого коррелированного периода отсчета выглядит следующим образом. Строка отсчета, r i. r2. . It, is переопределяется каждый раз, когда самая последняя ссылка rt проходит через период тайм-аута, чтобы свернуть любую последовательность коррелированных ссылок до временного интервала, равного нулю. Если ссылка на страницу p делается семь раз в течение периода коррелированных ссылок, мы не хотим наказывать или поощрять страницу за это. По сути, мы оцениваем время между прибытиями Ip по временному интервалу от конца одного Коррелированного ссылочного периода до начала следующего. Очевидно, что можно различать процессы, делающие ссылки на страницу; для простоты, однако, в дальнейшем мы будем считать, что ссылки не различаются по процессам, поэтому любые пары ссылок в пределах коррелированного ссылочного периода считаются коррелированными.

Другой вариант - варьировать подход Time-Out Correlation, основываясь на более глубоких знаниях о событиях в системе. Например, мы можем сказать, что период тайм-аута заканчивается после успешной фиксации транзакции, получившей доступ к нему, и следующей транзакции из того же процесса (чтобы исключить случаи (1) и (3) выше), или после того, как повторная попытка первой транзакции была отклонена (чтобы исключить случай (2)); как...

Возможно, существуют и другие сценарии с коррелированными эталонными парами, не охваченные этими тремя случаями. Другая идея заключается в том, чтобы позволить DBA переопределять установленный по умолчанию период корреляционной ссылки, задавая параметр для конкретной обрабатываемой таблицы.

2.1.2. Сайт Страница Ссылка Сохранено Проблема с информацией

Мы утверждаем, что в алгоритме LRU-K, где K > 2, необходимо сохранять в памяти историю ссылок на страницы, которые сами не присутствуют в буфере, что является отходом от большинства алгоритмов замены буфера прошлого. Чтобы понять, почему это так, рассмотрим следующий сценарий в алгоритме LRU-2. Каждый раз, когда на страницу p ссылаются, она становится резидентом буфера (возможно, она уже является резидентом буфера), и у нас есть история по крайней мере одной ссылки. Если предыдущее обращение к странице p было настолько давним, что у нас нет записи о
это, a после Корреляционного периода отсчета мы говорим, что наш
оценка bttr,2) равна бесконечности, и сделать содержащий буфер слот доступным по требованию. Однако, хотя мы можем выкинуть страницу p из памяти, нам необходимо сохранить информацию о ней в истории на некоторое время; в противном случае мы можем сравнительно быстро снова обратиться к странице p и снова не иметь записей о предыдущем обращении, снова выкинуть ее, снова обратиться к ней и т. д. Хотя на страницу часто ссылаются, у нас не будет истории о ней, чтобы распознать этот факт. По этой причине мы предполагаем, что система будет хранить информацию о любой странице в течение некоторого периода времени после последнего обращения к ней. Этот период мы называем ***периодом*** *сохраненной* ***информации.***

Если дисковая страница p, на которую раньше никто не ссылался, со временем становится достаточно популярной, чтобы оставаться в буфере, мы должны распознать этот факт, если два обращения к странице происходят с интервалом не более периода сохраненной информации. Несмотря на то, что мы отбрасываем страницу после первого обращения, мы сохраняем информацию в памяти, чтобы распознать, когда второе обращение дает значение bt(p,2), которое удовлетворяет нашему критерию LRU-2 для сохранения в буфере. Информация об истории страницы, хранящаяся в резидентной структуре данных памяти, обозначается HIST(p) и содержит два последних подстрочных индекса i и j, где $_{ri}$ = £j - p, или jllst последней ссылки, если известна только одна. Предположение о резидентной информационной структуре памяти может потребовать некоторого обоснования. Например, почему бы не хранить информацию о последних ссылках в заголовке самой страницы? Очевидно, что в любой момент, когда эта информация понадобится, страница будет резидентной в буфере. Ответ заключается в том, что такое решение потребует, чтобы страница всегда записывалась обратно на диск, когда
выкидываются из буфера из-за обновлений HIST(p); в задачах с большим числом обращений к часто ссылающимся страницам, которые в противном случае можно было бы просто выкинуть из буфера без записи на диск, это добавит большое количество накладных расходов на ввод-вывод.

Для определения размера периода сохранения информации мы предлагаем использовать правило пяти минут из [GRAP I как руководство к действию. Компромисс между затратами и выгодами при хранении страницы p объемом 4 Кбайт в буферах памяти составляет время между поступлениями ip около 100 секунд.
Возвращаясь к обсуждению LRU-2, следует немного подумать.

что период сохранения информации должен быть примерно в два раза больше, поскольку мы измеряем, как далеко назад нам нужно вернуться, *чтобы увидеть две* ссылки, прежде чем мы бросим страницу. Таким образом, каноническое значение периода сохранения информации может составлять около 200 секунд. Мы считаем, что это разумное эмпирическое правило для большинства приложений баз данных. Однако приложения с высокой производительностью могут решить увеличить буферный пул сверх экономически обоснованного размера, который вытекает из правила пяти минут. В таких приложениях период сохранения информации должен быть установлен соответственно выше. Чтобы определить разумное значение, рассмотрим максимальное обратное K-расстояние всех страниц, которые мы хотим гарантировать, что они будут находиться в памяти. Это значение является верхней границей для периода сохранения информации, поскольку ни одна возможная строка новых обращений к странице после этого периода не позволит ей пройти критерий сохранения в буфере.

2.1.3. Схема алгоритма буферизации LRU-K

Алгоритм LRU-K на рисунке 2.1 основан на следующих структурах данных:

- HIST(p) обозначает блок управления историей страницы p; он содержит время K последних обращений к странице p, без учета коррелирующих обращений: HIST(p,1) обозначает время последнего обращения, HIST(p,2) - время второго по счету обращения и т. д.
- LAST(p) обозначает время последнего обращения к странице p, независимо от того, является ли это корреляционной ссылкой или нет.

Эти две структуры данных поддерживаются для всех страниц с обратным K-расстоянием, которое меньше периода сохранения информации. Асинхронный демонический процесс должен очищать контрольные блоки истории, которые больше не оправдывают себя, если они не соответствуют критерию сохраненной информации.

Основываясь на этих структурах данных, концептуальная схема алгоритма LRU-K в виде псевдокода приведена на рисунке
2.1. Обратите внимание, что в этой схеме не учтены задержки ввода-вывода; в реальной реализации потребуется больше асинхронных единиц работы. Кроме того, для упрощения изложения в наброске не учтены дополнительные структуры данных, необходимые для ускорения циклов поиска; например, поиск страницы с максимальным обратным K-расстоянием на самом деле будет основан на дереве поиска. Несмотря на упущение таких деталей, очевидно, что алгоритм LRU-K довольно прост и не требует больших затрат на ведение учета.

Алгоритм работает следующим образом. Когда происходит обращение к странице, уже находящейся в буфере, нам нужно только обновить HIST и LAST для этой страницы. Фактически, если страница ссылалась последней в течение коррелированного периода ссылок, нам нужно только обновить ее LAST, чтобы продлить текущий коррелированный период ссылок; в противном случае произошел значительный разрыв в ссылках, и нам нужно закрыть старый коррелированный период ссылок и **начать** новый. Чтобы закрыть старый период, мы вычисляем его длину во времени, LAST(p) - HIST(p,1), период времени корреляционной ссылки, который нам нужно сократить до точки. Это сокращение тянет за собой более ранние значения HIST(pj) (i
= 2,...,K) вперед во времени на эту величину. В тот же

цикл, они сдвигаются вниз на один слот в массиве HIST, чтобы разместить новое значение HIST, связанное с началом нового коррелированного периода отсчета. Наконец, LAST обновляется до текущего времени.

Когда ссылаются на страницу, не находящуюся в буфере, необходимо найти жертву для замены, чтобы освободить слот в буфере. Страницы буфера, находящиеся в данный момент в пределах периода корреляции ссылок, не подлежат замене, и среди остальных мы выбираем ту, которая имеет максимальное обратное K-расстояние, **bt(q,K),** или, в текущей нотации, минимальное HIST(q,K). Эта жертва удаляется из буфера, возможно, требуя обратной записи. Затем инициализируется или обновляется блок HIST для новой страницы, на которую ссылаются.

```
Процедура, которая будет вызвана при обращении к
странице p в момент времени t:
-- p .
then и p уже находится в буфере
        /* обновить информацию об истории p */
   если t - LAST(p) > Correlated
   Reference_Period then/*          новая,
   некоррелированная ссылка */
      Период_корреляции_от_рефд_страницы:=
      LAST(p) - HIST(p,l) для i :- 2 to K do
      odHIST(pJ) := HIST(pa-1) +
                          коррел_период_от_рефд_страни
                          цы

      HIST (p,1) := t
   fi LAST(p) := t
   else    /- корреляционная
      ссылка */ LAST(p) := t

else    /* выберите жертву для замены */
   min := t
   для всех страниц q в буфере сделайте
      если t - LAST(q) > Corre1ated_Reference_Period
                              /* если подходит для замены
         */ и HIST(q,K) < min
                  /* и человек Обратное K-расстояние до
                  сих пор */
   odзатем
         жертва := q
         min := HIST(qK)
      fi

   Если жертва грязная, то
                  записать жертву обратно в базу данных fi
   /* теперь извлекаем страницу, на которую ссылаемся
   */
   извлекает p в буферный кадр, ранее находившийся
   у жертвы, если HIST(p) не существует
   тогда /* инициализация блока управления историей
   */
      выделить HIST(p)
      for i := 2 to K do HIST(pJ) := 0 od
   else
      for i := 2 to K do HIST(pJ) := HIST(pa-1) od
   fi
   HIST(p,l) := t
   LAST(p) := t
fi
```

Рисунок 2.1. Псевдокод алгоритма буферизации LRU-K, описанный в разделе 2.1.3

3 . Оптимальность алгоритма LRU-K в рамках модели **независимых ссылок**

В этом разделе мы приводим неформальные аргументы в пользу того, что алгоритм LRU-2 обеспечивает по сути оптимальное поведение буферизации на основе предоставленной информации; этот результат легко обобщается на LRU-K. Математический анализ поведения алгоритма LRU-K, включая доказательство оптимальности, приведен в [OOW]. В дальнейшем мы для простоты будем считать, что период коррелированных ссылок равен нулю и что это не вызывает никаких отрицательных эффектов; по сути, мы предполагаем, что коррелированные ссылки были исключены.

Как и прежде, в качестве отправной точки для обсуждения мы берем *независимую эталонную* ***модель*** для пейджинга, представленную в разделе 6.6 [COFFDENN]. Мы берем наши обозначения из этой ссылки и делаем несколько отступлений от представленной там модели. Начнем с набора N = {1, 2, …, n} дисковых страниц и набора M = {1, 2, , m} буферов памяти, 1s ms n. Поведение **системы при** пейджинге во времени описывается ее (страничной) строкой ссылок: r1, r2, ... , It, .
., где -t - означает, что дисковая страница p, p C N, ссылается системой в момент времени t. Согласно предположению о независимых ссылках, строка ссылок представляет собой последовательность независимых случайных величин с общим
стационарное распределение {§ i P2. . §p), по одной вероятности для каждой из n дисковых страниц. где Pr(rt = p) = p, для всех p E N и всех подскриптов t. Пусть случайная переменная $_{dt}$(p) обозначает интервал времени до следующего появления p в опорной строке через n: из предположений выше, dttr) имеет стационарное геометрическое распределение:

(3.1) Pr(dt(p) - k) - p (1 -) $^{k\ 1}$ k = 1, 2, ...

со средним значением Ip - петля. Заметим, что в силу предположения о стационарности Pr(dt(p) = k) не зависит от t.

Определение 3.1. Алгоритм буферизации At. Пусть Ao обозначает алгоритм буферизации, который заменяет в памяти буферизованную страницу p, ожидаемое значение Ip которой максимально, т. е. страницу, для которой p наименьшее. 8$

Теорема 3.2. [COFFDENN] [ADU)
Алгоритм Ao оптимален при предположении о независимости ссылок. 8

Теперь **рассмотрим** эталонную строку '" - r1, r2... . , It, ..., с некоторым вектором вероятностей ссылок g = ifi i. P2. . Int для n дисковых страниц N. Чтобы отразить нормальное состояние незнания относительно ссылочных вероятностей, с которыми запускается алгоритм буферизации, мы не можем предположить, что нам известно §i. В этой ситуации лучшее, что мы можем сделать, - это **статистически оценить §i для** каждой страницы i, основываясь на истории обращений к этой странице. Оказывается, анализ для получения статистической оценки §i позволяет нам вывести определенные порядковые свойства этих величин. В частности, учитывая любой вектор вероятности ссылок, имеющий по крайней мере два различных значения, мы можем заключить, что для любых двух дисковых страниц

x и y, если bt(x,K) < bt(y,K). то страница x имеет более высокую оценку вероятности обращения.

Этот результат не позволяет оценить абсолютные значения переменных°*° &i-, но его достаточно, чтобы упорядочить значения β_i для разных страниц и определить страницу с наименьшим значением fi. Среди всех страниц, находящихся в буфере в данный момент, страница x с наибольшим обратным K-расстоянием имеет наименьшую расчетную вероятность обращения. Заметим, что это наилучшая возможная статистическая оценка порядковых свойств 8i * ues, учитывая только знание последних K обращений к странице.

Из теоремы 3.2 следует, что оптимально заменять страницу с наименьшей ссылочной вероятностью всякий раз, когда нам нужно отбросить страницу, чтобы освободить буферный кадр для новой страницы; это и есть Ao g *thm. Кажется разумным, что наилучшая аппроксимация Ao будет состоять в том, чтобы мы бросали страницу x с наименьшей оцененной вероятностью ссылки, которая является страницей с наибольшим обратным K-расстоянием в соответствии с вышеприведенным аргументом; это алгоритм LRU- K. На самом деле, основной результат работы [OOW] очень близок к этому разумно звучащему утверждению:

Теорема 3.3. (OOW]
При предположении о независимых ссылках на страницы и знании последних K ссылок на страницы в буфере ожидаемая стоимость, полученная в результате работы LRU-K**algorithm** с m буферами памяти, меньше, чем стоимость, полученная от любого другого алгоритма, работающего с m-1 буферами. %

Таким образом, LRU-K действует оптимально во всех слотах буфера, кроме (возможно) одного из m, что является незначительным приращением побережья при большом m. Заметим, что этот результат для LRU-K в равной степени применим и к **particular,** LRU-1.
Считается, что алгоритм LRU действует оптимально (в соответствии с предположением о независимой ссылке на страницу), учитывая ограниченные знания о времени последней ссылки.

4. Эксплуатационные характеристики.

Прототип реализации алгоритма LRU-2 был профинансирован корпорацией Amdahl для исследования оптимальных альтернатив эффективного поведения буфера в продукте базы данных Huron. Небольшие изменения в прототипе позволили нам смоделировать поведение LRU-K, K u 1, в нескольких ситуациях, представляющих интерес. Мы исследовали три типа ситуаций с рабочей нагрузкой:

Эти три эксперимента рассматриваются в следующих трех подразделах.

4.1 Эксперимент с двумя бассейнами

Мы рассмотрели два пула дисковых страниц, пул 1 с Ni страницами и пул 2 с N2 страницами, причем Ni - N2 В этом *эксперименте с двумя пулами* поочередно делаются ссылки на пул 1 и пул 2, а затем случайным образом выбирается страница из этого пула в качестве элемента последовательности. Таким образом, каждая страница пула 1 с вероятностью bi = 1/(2Nt) встречается как любой элемент строки ссылок w, а каждая страница пула 2 с вероятностью Q = 1/(2N2) Этот эксперимент призван смоделировать чередование ссылок на индексные и рекордные страницы в примере 1.1: 11, R1, 12, R2, 13, R3, We хотим продемонстрировать, как алгоритмы LRU-K с различными
K различают страницы двух пулов, и насколько хорошо они справляются с удержанием в буфере страниц пула с более частыми обращениями (*более горячие* страницы *пула*). Коэффициент попадания в буфер для различных алгоритмов в идентичных условиях дает нам хорошую оценку эффективности алгоритма LRU-K при различных значениях K. Также был измерен оптимальный алгоритм Ao, который автоматически сохраняет в буфере максимально возможный набор страниц пула 1.

Коэффициент попадания в буфер для каждого алгоритма оценивался следующим образом: сначала алгоритм достигал квазистабильного состояния, отбрасывал начальный набор из 10 N} ссылок, а затем измерял...

при выполнении следующих T = 30 N i обращений. Если количество
такие ссылки, находящие запрашиваемую страницу в буфере, задаются h, тогда коэффициент попадания в кэш C задается:

- синтетическая рабочая нагрузка со ссылками на два пула страниц, которые имеют разные частоты ссылок, моделирующая пример 1.1,
- синтетическая рабочая нагрузка со случайными ссылками на набор **страниц с зипфианским распределением частот ссылок**. и
- реальная выборка OLTP-нагрузки со случайными, се-квенциальными и навигационными ссылками на базу данных CODASYL.

$$C = h / T$$

Помимо измерения коэффициента попадания в кэш, два алгоритма LRU-1 и LRU-2 также сравнивались по соотношению стоимость/производительность, как показано ниже. Для заданного NJ, N2 и размера буфера B(2), если LRU-2 достигает коэффициента попадания в кэш C(2), мы ожидаем, что LRU-I достигнет меньшего коэффициента попадания в кэш. Но увеличивая количество доступных буферных страниц, LRU-1 в конечном итоге достигнет эквивалентного коэффициента попадания в кэш, и мы говорим, что это произойдет, когда количество буферных страниц будет равно B(1). Тогда отношение B(1)/B(2) между размерами буферов, которые дают одинаковый эффективный коэффициент попадания, является мерой сопоставимой эффективности буферизации двух алгоритмов. Мы ожидаем, что B(1)/B(2) > 1,0, а значение 2,0, например, указывает на то, что если LRU-2 достигает определенного коэффициента попадания в кэш с помощью B(2) буферных страниц, то LRU-1 должен использовать вдвое больше буферных страниц для достижения того же коэффициента попадания.

Результаты этого имитационного исследования приведены в таблице 4.1, где N1 = 100 и N2 = 10 000.

B	LRU-I	LRU-2	LRU-3	Ao	B(1)/B(2)
60	0.14	0.291	0.300	0.300	2.3
80	0.18	0.382	0.400	0.400	2.6
100	0.22	0.4J9	0.49J	0.TOO	3.0
120	0.26	0.496	0.501	0.501	3.3
140	0.29	0.502	0.502	0.502	3.2
160	0.32	0.503	0.503	0.503	2.8
180	0.34	0.504	0.504	0.504	2.5
200	0.37	0.505	0.505	**0.505**	**2.3**
250	0.42	0.508	0.508	0.508	2.2
300	0.45	0.510	0.510	0.510	2.0
350	0.48	0.513	0.513	0.513	1.9
400	0.49	0.515	0.515	0.515	1.9
450	0.50	**0.517**	0.518	0.518	1.8

Таблица 4.1. Результаты моделирования эксперимента с двумя пулами, с пулами дисковых страниц Ni - llXl страниц и N2 - 10 000 страниц. В первом столбце указан размер буфера В. Со второго по пятый столбцы показаны коэффициенты попадания LRU-1, LRU-2, LRU-3. и Ao. В последнем столбце показано равное отношение эффективного размера буфера B(1)/B(2) для LRU-1 и LRU-2.

Рассмотрим B(1)/B(2) в верхней строке таблицы 4.1. Значение B(2) соответствует В этой строки, 60, где мы уверены, что LRU-2 имеет коэффициент попадания в кэш 0,291; для достижения такого же коэффициента попадания в кэш с помощью LRU-1 требуется примерно 140 страниц (поэтому B(1) = 140), а значит, 2,3 = 140/60. LRU-2 превосходит LRU-1 более чем в 2 раза по этому показателю стоимости/производительности. Из этого эксперимента также следует, что результаты LRU-3 еще ближе к результатам оптимальной политики Ao - ed to результатов LRU-2. Фактически, можно доказать, что при стабильных шаблонах доступа к страницам LRU-K приближается к At с увеличением Однако для развивающихся моделей доступа LRU-3 менее отзывчиваthan LRU-2 в том смысле, что ей требуется больше ссылок, чтобы адаптироваться к динамическим изменениям частоты обращений. По этой причине мы поддерживаем LRU-2 как в целом эффективную политику. Общая политика LRU-K с K > 2 может быть полезной для специальных приложений, но это требует дальнейшего изучения.

Для читателей, которые считают, что пулы из 100 и 10 000 страниц, а также количество буферов В в диапазоне 100 нереально малы для модемных приложений, отметим, что те же **результаты сохраняются,** если все номера страниц. N1, N2 и В умножаются на 1000. Меньшие числа были использованы в моделировании для экономии усилий.

4.2 Эксперимент со случайным доступом Zipfian

398]. Смысл констант a и b заключается в том, что фракция a ссылок обращается к фракции b N страниц (и то же самое соотношение рекурсивно выполняется в рамках фракции b более горячих страниц и фракции l-b более холодных страниц). В табл. 4.2 сравниваются коэффициенты попадания в буфер для LRU-I, LRU-2 и Ao при различных размерах буфера, а также эквиэффективное отношение размеров буфера B(1)/B(2) для LRU-1 и LRU-2 для a = 0,8 и b = 0,2 (т.е. перекос 80-20).

Во втором эксперименте исследовалась эффективность LRU-K для одного пула страниц с перекошенным случайным доступом. Мы генерировали ссылки на N = 1000 страниц (пронумерованных cl поN) with с зипфианским распределением частот ссылок; то есть вероятность ссылки на страницу с номером страницы меньше или равным i равна (i /]"}) log a / log b с константами a и b от 0 до 1 [CKS, KNUTH, p.

40	0.53	0.61	0.640	2.0
60	O.57	0.65	0.677	2.2
80	0.61	67	0.705	2.1
1€Ø	0.63	0.68	0.727	1.6
120	0.64	0.71	0.745	1.5
140	0.67	0.72	0.761	1.4
160	0.70	0.74	0.776	1.5
180	0.71	0.73	0.788	1.2
200	0.72	0.76	0.825	1.3
	o.7s	o.80	0.846	1.1
300	0.87	0.87	0.908	1.0

Таблица 4.2. Результаты моделирования коэффициентов попадания в буферный кэш для случайного доступа с распределением Zipfian 80-20 к пулу дисковых страниц N=l000 страниц.

Как и в эксперименте с двумя пулами в разделе 4.1, LRU-2 достиг значительных улучшений в отношении коэффициента попаданий при фиксированном размере буфера, а также в отношении стоимость/производительность. По сравнению с результатами раздела 4.1, выигрыш LRU-2 немного ниже, потому что перекос в этом эксперименте со случайным доступом Zipfian на самом деле мягче, чем перекос в эксперименте с двумя пулами. (Рабочая нагрузка с двумя пулами в разделе 4.1 примерно соответствует a = 0,5 и b = 0,01; однако в пределах долей страниц b и 1-b ссылки распределены равномерно).

4.3 Эксперимент с трассировкой OLTP

Третий эксперимент был основан на часовой трассировке страничных ссылок в производственной OLTP-системе крупного банка. Эта трассировка содержала около 470 000 страничных ссылок на базу данных CODASYL общим размером 20 гигабайт. Трасса была введена в нашу имитационную модель, и мы сравнили производительность LRU-2, классического LRU-1, а также LfU. **Результаты этого эксперимента, коэффициенты попаданий для различных размеров** буфера B и эквиэффективное соотношение размеров буфера B(1)/B(2) для LRU-1 и LRU-2, приведены в таблице 4.3.

LRU-2 превосходила и LRU, и LFU во всем спектре размеров буферов. При малых размерах буфера (n 600) LRU-2 улучшила коэффициент попадания в буфер более чем в 2 раза по сравнению с LRU-1. Более того, соотношение B(1)/B(2) в этом диапазоне размеров буферов показывает, что LRU-1 пришлось бы увеличить размер буфера более чем в 2 раза, чтобы достичь такого же коэффициента попадания, как у LRU-2.

B	LRU-1	LRU-2	LFU	B(1)/B(2)
lt3€i	0.005	0.07	0.07	4.5
2£D	0.01	**0.15**	0.11	**3.25**
30fi	0.02	0.20	0.15	3.0
40£1	0.06	0.23	0.17	2.75
5tX1	0.09	0.24	0.19	2.4
6fO	**0.13**	**0.25**	**0.20**	**2.16**
800	**0.18**	**0.28**	**0.23**	1.9
10£D	**0.22**	**0.29**	**0.25**	1.6
12€D	**0.24**	**0.31**	**0.27**	**1.66**
14tX1	0.26	0.33	0.30	1.5
l6tXl	0.29	0.34	0.31	1.5
21)fD	0.31	0.36	0.33	1.3
31XD	0.38	0.40	0.39	1.1
50€D	0.46	0.47	0.44	1.05

Таблица 4.3. Результаты моделирования коэффициента попадания в буферный кэш на примере трассировки OLTP.

Производительность LFU оказалась на удивление хорошей. Политика LFU по сохранению страниц с наибольшей частотой обращений действительно является правильным критерием для стабильного доступа. Однако недостатком LFU является то, что он никогда не забывает о предыдущих ссылках при сравнении приоритетов страниц: таким образом, он не адаптируется к изменяющимся шаблонам доступа. По этой причине LFU показал себя значительно хуже, чем алгоритм LRU-2, который отслеживает частоту *последних* обращений к страницам. Отметим, однако, что рабочая нагрузка OLTP в этом эксперименте демонстрировала довольно стабильные шаблоны доступа. В приложениях с динамически перемещающимися "горячими точками" алгоритм LRU-2 превзошел бы LFU еще более значительно.

При больших размерах буфера (> 3lXl0) разница в показателях попаданий трех политик становится незначительной. Поэтому можно задаться вопросом, действительно ли превосходство LRU-2 при малых размерах буфера имеет значение. Ответ на этот вопрос кроется в характеристиках трассы OLTP (которая, вероятно, весьма типична для большого класса рабочих нагрузок приложений). Трасса демонстрирует чрезвычайно высокий перекос в доступе к самым "горячим" страницам: например, 40 % обращений обращаются только к 3" страниц базы данных, к которым были обращения в трассе. Для более высоких долей ссылок этот перекос доступа сглаживается: например, 909с ссылок обращаются к 65% страниц, что уже нельзя считать сильным перекосом. Анализ трассировки показал, что только около 1400 страниц удовлетворяют критерию правила пяти минут для хранения в памяти (т. е. повторные обращения к ним происходят в течение 100 секунд, см. раздел 2.1.2). Таким образом, размер буфера в 1400 страниц **является экономически** оптимальной конфигурацией. Нет смысла увеличивать размер буфера для хранения дополнительных страниц, когда локальность выровняется. Алгоритм LRU-2 сохраняет этот пул из 1400 горячих страниц в памяти, затрачивая всего две трети памяти классического алгоритма LRU (т. е. B(1)/B(2) = 1,5 для B=1400).

5. Заключительные замечания

В этой статье мы представили новый алгоритм буферизации баз данных, названный LRU-K. Результаты моделирования показывают, что алгоритм LRU-K имеет значительные преимущества по стоимости/производительности по сравнению с традиционными алгоритмами типа LRU, поскольку LRU-K лучше различает страницы, на которые часто ссылаются, и страницы, на которые ссылаются нечасто. В отличие от подхода, предполагающего ручную настройку назначения пулов страниц нескольким буферным пулам, наш алгоритм является самодостаточным, поскольку он не зависит от каких-либо внешних подсказок. Подобно подходам, направленным на автоматическое извлечение подсказок для менеджера буферов из анализа планов выполнения запросов, наш алгоритм учитывает также межтранзакционную локальность в многопользовательских системах. Наконец, в отличие от LFU и его разновидностей, наш алгоритм хорошо справляется с изменяющимися шаблонами доступа, такими как перемещение хот-спотов.

Одна из новых концепций нашего подхода заключается в том, что информация о истории страницы сохраняется после ее пребывания. Но очевидно, что только так мы можем гарантировать, что страница, на которую ссылаются с метрономной регулярностью в интервалах, чуть превышающих период ее пребывания, никогда не будет замечена как дважды ссылающаяся. Вопрос о том, сколько места мы должны отвести под блоки управления историей нерезидентных страниц, остается открытым. Хотя верхняя граница может быть определена на основе свойств рабочей нагрузки и заданного периода сохранения информации, лучшим подходом было бы динамическое преобразование буферных кадров в блоки управления историей и наоборот.

Разработка алгоритма LRU-K была в основном мотивирована приложениями OLTP, приложениями поддержки принятия решений в больших реляционных базах данных и особенно комбинациями этих двух категорий рабочих нагрузок. Однако мы считаем, что потенциальная отдача от нашего алгоритма может быть еще выше для нетрадиционных инженерных и научных баз данных. Причина в том, что управление буфером для таких приложений по своей сути сложнее из-за большего разнообразия шаблонов доступа. Подход к настройке пула страниц, описанный в разделе 1, явно не подходит для этой цели. Подходы, в которых подсказки по управлению буфером извлекаются из анализа планов выполнения запросов, также сомнительны по следующей причине. Нетрадиционные приложения баз данных указанного типа, вероятно, будут активно использовать функции, определяемые пользователем, как это поддерживается объектно-ориентированными и расширяемыми системами баз данных. В отличие от реляционных запросов, шаблоны доступа к этим определяемым пользователем функциям не могут быть предварительно проанализированы, если функции написаны на языке программирования общего назначения (обычно на C++). Таким образом, перспективные приложения пост-реляционных систем баз данных нуждаются в действительно самодостаточном **алгоритме** управления буфером**.** Алгоритм LRU-K является таким самонастраивающимся и адаптивным алгоритмом буферизации даже при наличии изменяющихся моделей доступа. Мы считаем, что LRU-K - хороший кандидат для решения задач управления буферами нового поколения.

Ссылки

[ABG] Rafael Alonso, Daniel Barbara, Hector Garcia-Molina, Data Caching Issues in an Information Retrieval System, ACM Transactions on Database Systems, v. 15, no. 3, pp. 359-384, September 1990.

[ADU] Альфред В. Ахо, Питер Дж. Деннинг и Джеффри Д. Ульман: Принципы оптимальной замены страниц. J. ACM, v. 18, no. t, pp. 80-93, 1971.

[CHAKA] Эллис Э. Чанг, Рэнди Х. Katz. Использование наследования и семантики структуры для эффективной кластеризации и буферизации в объектно-ориентированной СУБД, Труды конференции ACM SIGMOD 1989, с. 348-357.

[CHOUDEW] Hong-Tai Chou и David J. DeWitt: An Evaluation of Buffer Management Strategies for Relational Database Systems. Труды седьмой Международной конференции по очень большим базам данных, стр. 127- 141, август 1985 г.

[CKS] Джордж Коупленд, Том Келлер и Марк Смит: Конфигурирование буфера и диака базы данных и борьба с узкими местами. Труды четвертого международного семинара по высокопроизводительным транзакционным системам, сентябрь 1991 г.

[COL] C.Y. Chan, B.C. Ooi, H. Lu, Extensible Buffer Management of Indexes, Proceedings of the Eighteenth International Conference on Very Large Data Bases, pp. 444-454, August 1992.

tCOFFDENN] Эдвард Г. Коффман-младший и Питер Дж. Деннинг: Теория операционных систем. Prentice-Hall, 1973.

[DANTOWS] Асит Дан и Дон Таусли: Приближенный анализ схем замены буферов LRU и FIFO. Труды конференции ACM Sigmetrics 1990, v. 18, No. 1, pp. 143-149.

[DENNING] П. Дж. Деннинг: Модель рабочего набора для поведения программ. Communications of the ACM, v. 11, no. 5, pp. **323-333,** 1968.

[EFFEHAER] Вольфганг Эффельсберг и Тео Хаердер: Принципы управления буфером базы данных. ACM Transactions on Database Systems, v. 9, no. 4, pp. 560- 595, December 1984.

[[FNS]] Christos Faloutsos, Raymond Ng, and Timos Sellis, Predictive Load Control for Flexible Buffer Allocation, Proceedings of the Seventeenth International Ccinference on Very **Large Data** Bases, pp. 265-274, September 1991.

[GRAYP 1 Джим Грей и Франко Путцолу: Правило пяти минут для торговли памятью для доступа к диску и правило 10 байт для торговли памятью для процессорного времени. Труды конференции ACM SIGMOD 1987 года, стр. **395-398**

[HAAS] Laura M. Haas et al., Starburst Mid-Flight: As the Dust Clears. IEEE Transactions on Knowledge and Data Engineering, v. 2, no. 1, pp. 143-160, March 1990.

[JCL] P. Джаухари, М. Кери, М. Ливни, Priority-Hints: An

[KNUTH] D.E. Knuth, The Art of C om p u t e r Programming, Volume 3: Sorting and Searching, Addison- Wesley, 1973.

[NFS] Raymond Ng, Christos Faloutsos, T. Sellis, Flexible Buffer Allocation Based on Marginal Gains, Proceedings of the **1991 ACM** SIGMOD Conference, pp. 387-396.

(OOW] Elizabeth O'Neil, Patrick O'Neil, and Gerhard Weikum, The LRU-K Page Replacement Algorithm for Database Disk Buffering, Tech. Report 92-4, Department of Mathematics and Computer Science, University of Massachusetts at Boston, December 1, 1992.

{PAZDO} Марк Палмер, Стэнли Б. Здоник, Fido: A Cache That Learns to Fetch, Proceedings of **the Seventeenth** International Conference on Very Large Databases, pp. 255- 264, September 1991.

[REITER] Аллен Райтер: Smdy of Buffer Management Policies for Data **Management** Systems. Tech. Summary Rep. No. 1619, Центр математических исследований, Висконсинский университет, Мэдисон, март 1976 г.

[ROBDEV] Джон Т. Робинсон и Мурта В. **Девараконда:** Управление кэшем **данных с** помощью замены на основе частоты. Труды конференции ACM Sigmetrics 1990, v. 18, No. 1, pp. 134-142.

[SACSCH] Giovanni Mario Sacco and Mario Schkolnick: Buffer Management in Relational Database Systems, ACM Transactions on Database Systems, v. 11, no. 4, pp. 473- 498, December 1986.

[SHASHA] Dennis E. Shasha, Database **Tuning:** A Principled Approach, Prentice Hall, 1992.

[STONj Майкл Стоунбрейкер: Поддержка операционной системы для управления базами данных. Communications of the ACM, v. 24, no. 7, pp. 412-418, July 1981.

{TENGGUM) J.Z. Teng, R.A. Gumaer, Managing IBM Database 2 Buffers to Maximize Performance, I BM Systems Journal, v. 23, n. 2, pp. 211-218, 1984.

[TPC-A] Совет по производительности обработки транзакций (TPC): TPH BENCHMARK A Standard Specification. The Performance Handbook: for Database and Transaction Processing Systems, Morgan Kaufmann 1991.

{YUCORN} P.S. Yu, D.W. Cornell. Оптимальное распределение буферов в среде с несколькими запросами, Труды седьмой Международной конференции по инженерии данных, pp. 622-631, апрель 1991.

Алгоритм управления буфером на основе приоритетов, Труды Шестнадцатой международной конференции по базам данных очень большого объема, август 1990 г.