Онлайн-дедупликация для баз данных

Ляхун Сюй
Университет Карнеги-Меллона lianghon@andrew.cmu.edu

Эндрю Павло
Университет Карнеги-Меллона pavlo@cs.cmu.edu

Судипта Сенгупта
Исследования Майкрософт
sudipta@microsoft.com

Грегори Р. Гангер
Университет Карнеги-Меллон ganger@ece.cmu.edu

АБСТРАКТНЫЙ

dbDedup — это схема дедупликации на основе подобия для систем управления базами данных в режиме онлайн (СУБД). Помимо сжатия на уровне блоков отдельных страниц базы данных или сообщений журнала операций (oplog), используемых в современных СУБД, dbDedup использует дельта-кодирование на уровне байтов отдельных записей в базе данных для достижения большей экономии. Метод однопроходного кодирования dbDedup может быть интегрирован в компоненты хранения и ведения журнала СУБД для обеспечения двух преимуществ: (1) уменьшение размера данных, хранящихся на диске, по сравнению с тем, что обеспечивают традиционные схемы сжатия, и (2) уменьшение объема данных, передаваемых по сети для служб репликации. Чтобы оценить нашу работу, мы внедрили dbDedup в распределенную СУБД NoSQL и проанализировали ее свойства с использованием четырех реальных наборов данных. Наши результаты показывают, что dbDedup обеспечивает до 37-кратного сокращения размера хранилища и трафика репликации базы данных самостоятельно и до 61-кратного сокращения в сочетании с блочным сжатием СУБД. dbDedup обеспечивает оба преимущества с незначительным влиянием на пропускную способность СУБД или задержку клиента (среднюю и конечную).

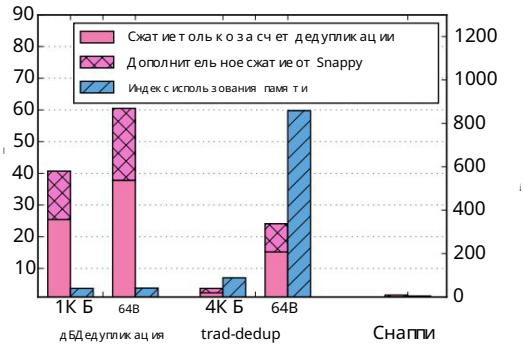


Рисунок 1: Коэффициент сжатия и использование памяти индекса для данных Википедии, хранящихся в пяти конфигурациях MongoDB: с dbDedup (размер фрагмента 1 КБ и 64 Б), с традиционной дедупликацией (4 КБ и 64 Б) и с Snappy (сжатие на уровне блоков). dbDedup обеспечивает более высокую степень сжатия и меньшие накладные расходы на память индекса, чем традиционная дедупликация. Snappy обеспечивает такое же сжатие 1,6× для данных после дедупликации или исходных данных.

1. ВВЕДЕНИЕ

Темпы роста данных превышают снижение стоимости оборудования. Сжатие базы данных является одним из решений этой проблемы. Для хранения базы данных, в дополнение к экономии места, сжатие помогает сократить количество дисковых операций ввода-вывода и повысить производительность, поскольку запрашиваемые данные помещаются на меньшем количестве страниц. Для распределенных баз данных, реплицированных по географическим регионам, также существует острая необходимость в сокращении объема передач и данных, используемого для синхронизации реплик.

Наиболее широко используемый подход к сокращению данных в операционных СУБД — это сжатие на уровне блоков [30, 37, 46, 43, 3, 16]. Такие СУБД используются для поддержки пользовательских приложений, которые выполняют простые запросы для извлечения небольшого количества записей за раз (в отличие от выполнения сложных запросов, которые сканируют большие сегменты базы данных). Хотя сжатие на уровне блоков является простым и эффективным, оно не решает проблему избыточности между блоками, следовательно, оставляет значительные возможности для улучшения для многих приложений (например, из-за управления версиями на уровне приложений в виде инкрементного копирования записей на досках объявлений). Дедупликация (dedup) стала популярной в системах резервного копирования для устранения дублирующегося контента по всему корпусу данных, часто достигая гораздо более высоких коэффициентов сжатия. Резервный поток делится на фрагменты, и в качестве идентификатора каждого фрагмента используется устойчивый к коллизиям хэш (например, SHA-1). Система дедупликации поддерживает глобальный индекс всех хэшей и использует его для обнаружения дубликатов. Дедупликация хорошо работает как для основных, так и для резервных наборов данных хранения, которые состоят из больших файлов, которые редко изменяются (а если и изменяются, то изменения редки).

К сожалению, традиционные схемы дедупликации на основе фрагментов не подходят для операционных СУБД, где приложения выполняют запросы на обновление, изменяющие отдельные записи. Количество дублирующихся данных в отдельной записи, скорее всего, незначительно. Но большие размеры фрагментов (например, 4–8 КБ) являются нормой, чтобы избежать огромных индексов в памяти и большого количества чтений с диска.

В этой статье представлен dbDedup, облегченная схема для систем баз данных в режиме онлайн, которая использует дедупликацию на основе сходства [65] для сжатия отдельных записей. Вместо индексации каждого хэша фрагмента, dbDedup выбирает небольшой поднабор хэшей фрагментов для каждой новой записи базы данных, а затем использует этот образец для идентификации похожей записи в базе данных. Затем он использует дельта-сжатие на уровне байтов для двух записей, чтобы уменьшить как используемое онлайн-хранилище, так и пропускную способность удаленной репликации. dbDedup обеспечивает более высокие коэффициенты сжатия с меньшими накладными расходами памяти, чем дедупликация на основе фрагментов, и хорошо сочетается со сжатием на уровне блоков, как показано на рис. 1.

Мы представляем объединяем несколько методов для достижения этой эффективности. Прежде всего, мы представляем новое двустороннее кодирование для эффективной передачи закодированных новых записей (прямое кодирование) в удаленные реплики, сохраняя новые записи с закодированными формами выбранных исходных записей (обратное кодирование). В результате нет де-

SIGMOD'17, 14–19 мая 2017 г., Роли, Северная Каролина, США ©
2017 ACM. ISBN 978-1-4503-4197-17.04.05. . . 15,00 долларов США
DOI: http://dx.doi.org/10.1145/3035918.3035938

код требуется для распространенного случая доступа к самой последней записи в цепочке кодирования (например, к последней версии Википедии).
Чтобы избежать накладных расходов на производительность при обновлении исходных записей, мы также вводим кэш обратной записи с потерями, настроенный на максимизацию коэффициента сжатия, избегая при этом конкуренции ввода-вывода. Наш подход также использует новую технику, называемую кодированием скачков, которая минимизирует наихудшее количество шагов декодирования, необходимых для доступа к определенной записи в длинной цепочке кодирования. Наконец, мы описываем, как адаптивно отключить дедупликацию для баз данных из записей, где ожидается небольшая экономия.

Чтобы оценить наш подход, мы внедрили dbDedup в СУБД Mon-goDB [5] и измерили его эффективность с использованием четырех реальных наборов данных. Наши результаты показывают, что он достигает 37-кратного сокращения (61-кратного в сочетании с компрессией на уровне блоков) размера хранилища и трафика репликации. dbDedup превосходит дедупликацию на основе фрагментов, оказывая при этом незначительное влияние на производительность СУБД.

В данной статье представлены следующие материалы

1. Насколько нам известно, мы представляем первую систему дедупликации для операционных СУБД, которая снижает как использование хранилища базы данных, так и использование полосы пропускания репликации. Это также первая система дедупликации хранилища базы данных, которая использует дедупликацию на основе сходства.
2. Мы внедряем новые методы и имеющие решающее значение для достижения приемлемой эффективности и дедупликации, что позволяет применять их на практике для хранения баз данных в режиме онлайн.
3. Мы оцениваем полную реализацию системы распределенной СУБД NoSQL, используя четыре реальных набора данных.

Остальная часть этой статьи организована следующим образом. Раздел 2 мотивирует использование дедупликации на основе сходства для приложений баз данных и классифицирует наш подход относительно других систем дедупликации. Раздел 3 описывает рабочий процесс и механизмы дедупликации dbDedup. Раздел 4 подробно описывает реализацию dbDedup, включая ее интеграцию с фреймворком хранения и репликации СУБД. Затем мы оцениваем наш подход с использованием нескольких наборов данных реального мира в Разделе 5. Наконец, в Разделе 6 мы завершаем обсуждением связанной работы и

2. ПРЕДЫСТОРИЯ И МОТИВАЦИЯ

Дедупликация заключается в выявлении и удалении дублирующегося контента в корпусе данных. В этом разделе обосновывается его потенциальная ценность в СУБД, объясняются две основные категории (точное совпадение и основанные на сходстве) подходов к дедупликации и то, почему основанный на сходстве подход лучше подходит для дедупликации в СУБД, а также dbDedup помещается в контекст путем категоризации предыдущих систем дедупликации.

2.1 Зачем нужна дедупликация для приложений баз данных?

Наиболее распространенным способом, которым операционные СУБД уменьшают размер хранения данных, является сжатие на уровне блоков на отдельных страницах базы данных. Например, InnoDB от MySQL может сжимать страницы когда они вытесняются из памяти и записываются на диск [3].
Когда эти страницы возвращаются в память, система может сохранять страницы сжатыми до тех пор, пока ни один запрос не попытается прочитать их содержимое. Поскольку область действия алгоритма сжатия составляет только одну страницу, степень сокращения, которую может достичь система, невелика.

Аналитические СУБД используют более агрессивные схемы (например, сжатие словаря, кодирование длины серии), которые значительно уменьшают размер базы данных [18]. Это связано с тем, что эти системы сжимают отдельные столбцы и, таким образом, существует более высокая вероятность дублирования данных. И в отличие от приведенного выше примера MySQL, они также поддерживают обработку запросов непосредственно на сжатых данных.

Этот тип сжатия непрактичен в операционной СУБД. Эти системы разработаны для высоконагруженных рабочих нагрузок, которые выполняют запросы, извлекающие небольшое количество записей за раз. Если бы СУБД приходилось сжимать каждый атрибут каждый раз если была вставлена новая запись, то они были бы слишком медленными для поддержки онлайновых веб-приложений.

Однако мы видим, что многие приложения баз данных могли бы выиграть от дедупликации из-за сходства между несовмещенными записями, взаимосвязь которых неизвестна базовым СУБД.
Кроме того, мы обнаружили, что преимущества дедупликации дополняют преимущества сжатия — объединение дедупликации и сжатия дает большее сокращение данных, чем каждое из них по отдельности. Хотя дедупликация широко используется в файловых системах, она не была полностью исследована в операционных базах данных. Основная причина заключается в том, что записи баз данных обычно малы по сравнению с типичными размерами фрагментов дедупликации (4–8 КБ), поэтому применение традиционной дедупликации на основе фрагментов не даст достаточных преимуществ.

Для многих приложений основным источником дублирующихся данных является управление версиями записей на уровне приложения. Хотя СУБД с многоверсионным управлением параллелизмом (MVCC) поддерживают исторические версии для поддержки параллельных транзакций, они обычно очищают старые версии, как только они становятся не видны ни одной активной транзакции. В результате лишь немногие приложения используют поддержку управления версиями, предоставляемую СУБД, для выполнения «запросов о путешествии во времени». Вместо этого большинство приложений реализуют управление версиями самостоятельно, когда это необходимо. Общей чертой этих приложений является то, что различные версии одного элемента данных записываются в СУБД как совершенно несвязанные записи, что приводит к значительной избыточности, которая не улавливается простым сжатием страниц. Примерами таких приложений являются веб-сайты на базе WordPress, которые составляют 25% всего Интернета [12], а также совместные вики-платформы такие как Wikipedia [14] и Baidu Baike [1].

Другим источником дублирования в приложениях баз данных являются отношения включения между записями. Например, ответ по электронной почте или пересылка обычно включают содержимое предыдущего сообщения в текст сообщения. Другими примерами являются онлайн-доски объявлений, где пользователи часто цитируют комментарии друг друга в своих сообщениях. Как и управление версиями, это копирование является артефактом приложения, который не может быть легко раскрыт базовой СУБД. В результате эффективное удаление избыточности также требует метода дедупликации, который выявляет и устраняет избыточность во всем корпусе данных.

Важно отметить, что существует также много приложений баз данных, которые не получат выгоды от дедупликации. Например, некоторые из них не обладают достаточной внутренней избыточностью и, таким образом, накладные расходы на поиск возможностей удаления избыточных данных не стоят того. Типичные примеры включают большинство рабочих нагрузок OLTP, где много записей помещаются на одной странице базы данных, и большинство избыточностей среди полей можно устранить с помощью схем сжатия на уровне блоков. Для приложений, которые не получают выгоды, dbDedup автоматически отключает функции дедупликации, чтобы уменьшить их влияние на производительность системы.

2.2 Дедупликация на основе сходства и точная дедупликация

Подходы к дедупликации можно разделить на две категории.
Первый и наиболее распространенный («точная дедупликация») ищет точные совпадения в единице дедупликации (например, фрагменте) [67, 40, 27, 34, 35]. Второй («дедупликация на основе сходства») ищет похожие единицы (фрагменты или файлы) и применяет к ним дельта-сжатие [61, 53, 22]. Для тех приложений баз данных, которые действительно выигрывают от дедупликации, мы обнаружили, что дедупликация на основе сходства превосходит дедупликацию на основе фрагментов с точки зрения коэффициента сжатия и использования памяти, хотя она может включать дополнительные накладные расходы на ввод-вывод и вычисления. В этом разделе кратко описывается дедупликация на основе фрагментов, почему она не работает хорошо для СУБД и почему дедупликация на основе сходства работает. В разделе 3 подробно описан рабочий процесс dbDedup и его методы для смягчения потенциальных накладных расходов.

Традиционная схема дедупликации файлов, основанная на точном совпадении фрагментов данных («дедупликация на основе фрагментов») [44, 49, 67], работает следующим образом. Входящий файл (соответствующий новой записи в контексте

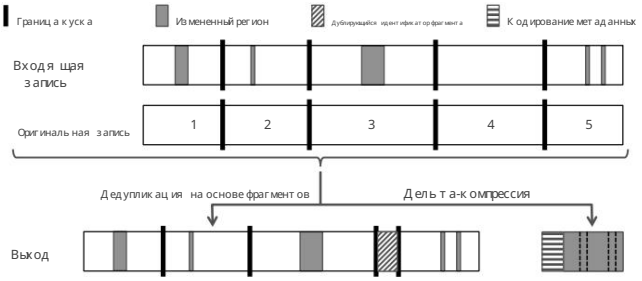


Рисунок 2: Сравнение дедупликации на основе фрагментов и дедупликации на основе подобия с использованием дельта-сжатия для типичных рабочих нагрузок базыданных с небольшими и разбросанными изменениями.

	Точная	Дедупликация на основе сходства	
Начальный	дедупликация iDedup [55] ЗФС [17] СДФС [4] Windows сервер 2012 [15] NetApp ASIS [19] Окарина [7] Пермабит [8]	dbDedup	
Вторичный	ДДФС [67] Венти [49] ChunkStash [31] ДЕДЕ [27] ГидраСтор [33]	Экстремальное биннинг [22] Разреженная индексация [40] Силос [64] СИДК [53] DeepStore [66]	СДС [20] sДедупликация [65]

Таблица 1: Категоризация сопутствующих работ

СУБД) сначала делится на фрагменты с использованием фингерпринтинга Рабина [50];
Хэши Рабина рассчитываются для каждого скользящего окна данных.
поток, и границ кусков объявляется, если младшие биты
Значение хэша соответствует предопределенному шаблону. Средний размер фрагмента может
контролироваться количеством битов, используемых в шаблоне. Как правило,
шаблон соответствия из n битов приводит к среднему размеру фрагмента 2n B.
Для каждого фрагмента система вычисляет уникальный идентификатор, используя
Устойчивый к коллизиям хеш (например, SHA-1). Затем он проверяет глобальный индекс
чтобы увидеть, видел ли он этот хэш ранее. Если совпадение найдено, то
кусок объявляется дубликатом. В противном случае кусок считается уникальными
добавляется в индекс и базовое хранилище данных.

Хотя дедупликация на основе фрагментов обычно хорошо работает для рабочих нагрузок
резервного копирования, она редко подходит для рабочих нагрузок баз данных.
По нашим наблюдениям, дублирующиеся области в рабочих нагрузках базы данных обычно
невелики (порядка десятков-сотней байт) и разбросаны
в записи. При таком малом размере дедупликация на основе фрагментов с типичным размером
фрагмента порядка КБ не способна идентифицировать много дублирующих фрагментов.
Уменьшение размера фрагмента для соответствия дублированию
длина может улучшить степень сжатия системы, но кусок
индекс отслеживания становится чрезмерно большим и сводит на нет любые преимущества в
производительности, полученные за счет сокращения ввода-вывода.

Напротив, дедупликация dbDedup на основе сходства идентифицирует один
похожая запись из корпуса базы данных и выполняет дельта-сжатие между новой записью и
похожей. Как показано на
Рис. 2. Сжатие дельта на уровне байтов с помощью dbDedup позволяет идентифицировать
гораздо более мелкозернистые дубликаты, таким образом, обеспечивают большую степень
сжатия, чем дедупликация на основе фрагментов.

2.3 Категоризация систем дедупликации

Таблица 1 иллюстрирует один из взглядов на то, как dbDedup соотносится с другими
системы, использующие дедупликацию, основанные на двух осях: подход дедупликации (точный
соответствие или сходство) и цель дедупликации (основанное на первичном хранилище или
вторичные/резервные данные). Насколько нам известно, dbDedup — это первый
система дедупликации на основе сходства для первичного хранения данных, а также
являясь первой системой дедупликации для онлайновых СУБД, решающей обе задачи
первичное хранилище и вторичные данные (oplog).

Большая часть предыдущей работы по дедупликации данных [67, 40, 20, 53, 54] была
сделано в контексте резервного копирования данных (в отличие от основного хранилища)
где дедупликация не должна идти в ногу с первичным приемом данных
и не обязательно, чтобы он работал на основном (обслуживающем данные) узле. Более того,
такие резервные рабочие нагрузки часто работают в устройствах на премиум-
аппаратное обеспечение. dbDedup, в контексте операционных СУБД,
должны работать на основных узлах обслуживания данных на стандартном оборудовании
и экономно использовать ресурсы ЦП, памяти и ввода-вывода.

В последнее время появился интерес к первичной дедупликации данных на первичном
(обслуживающем данные) сервере, но решения в основном находятся на уровне хранения (а не
на уровне управления данными, как в нашей работе).
В таких системах, в зависимости от реализации, дедупликация может происходить либо в
процессе добавления новых данных (Sun ZFS [17], Linux SDFS [4],
iDedup [55]) или в фоновом режиме в качестве постобработки сохраненных данных
данных (Windows Server 2012 [35]) или предоставить оба варианта (NetApp [19], Ocarina [7],
Permabit [8]).

Системы в нижнем среднем столбце используют комбинации точных и основанных на подобии
методов дедупликации с различной степенью детализации, но по сути являются системами
дедупликации на основе фрагментов, поскольку они
хранить хэши для каждого фрагмента. Насколько нам известно, dbD-edup — это первая
система дедупликации на основе подобия для основных рабочих нагрузок хранения, которая
обеспечивает сокращение данных на хранилище и в сети
Требование к пропускной способности в то же время. Это потому, что байтовый уровень
дельта-сжатие традиционно считается дорогим для онлайн-
базы данных, из-за дополнительных затрат на ввод-вывод и вычисления относительно
для сравнения хэшей. В результате предыдущие системы либо полностью избегали этого, либо
использовали его, когда дисковый ввод-вывод не был серьезной проблемой. Для
например, SIDC [53] и sDedup [65] используют дельта-сжатие для
Дедупликация на сетевом уровне потоков репликации; SDS [20] применяет дельта-сжатие к
большим фрагментам по 16 МБ в потоках резервного копирования
извлекается путем последовательного чтения с диска. В то время как dbDedup использует преимущество
дельта-сжатия для достижения превосходной степени сжатия, он использует
ряд методов для снижения накладных расходов, что делает его
практичный механизм дедупликации для онлайн-СУБД.

3. ДИЗАЙН dbDedup

В этом разделе описывается рабочий процесс дедупликации dbDedup, кодирование
методы, механизмы и подходы к снижению накладных расходов ввода-вывода
чтобы избежать напрасной траты усилий на действия по дедупликации, не приносящие никакой выгоды

3.1 Рабочий процесс дедупликации

dbDedup использует дедупликацию на основе сходства для достижения хорошего
коэффициента сжатия и низкого использования памяти одновременно. На рис. 3 показано
Рабочий процесс дедупликации кодирования, используемый при подготовке обновленных данных записи
для локального хранения и удаленной репликации. Во время вставки или обновления
запросы новые записи записываются в локальный oplog и dbDedup
кодирует их в фоновом режиме, вне критического пути. Четыре ключа
шаги: (1) извлечение признаков сходства из новой записи, (2)
просматривая индекс дедупликации, чтобы найти список кандидатов, похожих
записи в корпусе базы данных, (3) выбор одной лучшей записи из
кандидатов, и (4) выполнение дельта-сжатия между
новая и аналогичная запись для вычисления закодированных форм для локального
хранения и синхронизации реплик.

3.1.1 Извлечение признаков

В качестве первого шага в поиске похожих записей в базе данных, dbDedup
извлекает признаки сходства из новой записи, используя подход, зависящий от содержимого.
dbDedup делит новую запись на несколько
Фрагменты данных переменного размера с использованием алгоритма Рабина Fingerprinting [50],
который широко используется во многих системах дедупликации на основе фрагментов.
В отличие от этих систем, которые индексируют устойчивый к коллизиям хеш (например,
SHA-1) для каждого уникального фрагмента, dbDedup вычисляет (слабее,
но вычислительно дешевле) MurmurHash [6] для каждого фрагмента и

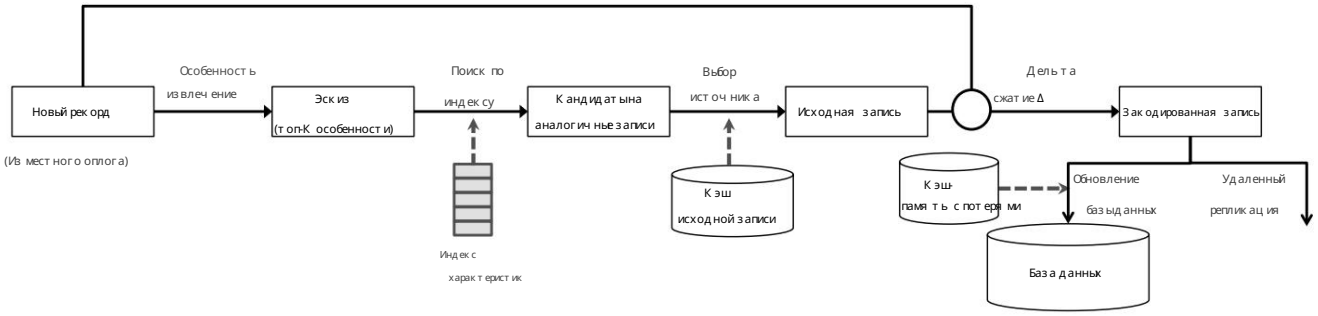


Рисунок 3: Рабочий процесс dbDedup – (1) извлечение признаков, (2) поиск по индексу, (3) выбор источника и (4) дельта-сжатие.

индексирует только репрезентативное подмножество хэшей фрагментов. dbDedup адаптирует технику, называемую последовательной выборкой [47], для выбора репрезентативных хэшей фрагментов, что обеспечивает лучшую характеристику сходства, чем случайная выборка. Он сортирует значения хэшей последовательным образом (например, по величине от большего к меньшему) и выбирает топ-К [1] хэши как эскиз сходства для записи. Каждый кусок хэша в эскизе называется признаком — если две записи имеют один или больше общих черт, они считаются схожими.

Индексируя только хэши выбранных фрагментов, dbDedup ограничивает накладные расходы памяти его индекса дедупликации не должны превышать К записей индекса на запись. Это важное свойство позволяет dbDedup использовать небольшие размеры фрагментов для лучшего обнаружения сходства, не потребляя при этом чрезмерного объема оперативной памяти, как при дедупликации на основе фрагментов. Более того, поскольку dbDedup не полагается на точное совпадение хэшей фрагментов для дедупликации, он более терпимо к коллизиям хэшей. Вот почему он может использовать алгоритм MurmurHash вместо SHA-1 для уменьшения накладных расходов при вычислении хэша фрагмента. Хотя это может привести к небольшому снижению скорости сжатия из-за большего количества ложных срабатываний, использование более слабого хеша не влияет на корректность, так как dbDedup выполняет дельта-сжатие на последнем этапе.

3.1.2 Поиск индекса

Для каждого извлеченного признака dbDedup находит существующие записи, которые поделится этой функцией с новой записью. Поскольку dbDedup — это система дедупликации в режиме онлайн, крайне важно, чтобы этот процесс поиска индекса был быстрым и эффективным. dbDedup достигает этого, создавая в памяти индекс признаков, который использует вариант хеширования Cuckoo [45, 31] для отображения признаков к записям. Этот подход использует несколько функций хеширования, которые сопоставляют K ключи с несколькими слотами-кандидатами, что увеличивает коэффициент загрузки таблицы, ограничивая время поиска константой. В индексе объекта, каждая запись состоит из 2-байтового ключа, который является компактной контрольной суммой объекта, и 4-байтового значения, которое является указателем на расположение соответствующей записи в базе данных.

При поиске признака dbDedup сначала вычисляет хэш-признака. значение с использованием одной из функций хеширования Cuckoo, которая сопоставляется с Слот-кандидат, содержащий несколько записей индекса (корзин). Затем он перебирает контейнеры, сравнивает их контрольные суммы с заданными и добавляет любые совпадающие записи в список похожих записей. Этот процесс повторяется с другими функциями хеширования, пока не будет найдено пустое ведро, указывающее на окончание поиска. Затем dbDedup вставляет функция и ссылка на новую запись в пустом контейнере для будущий поиск. Наконец, dbDedup объединяет результаты поиска для всех top-K функции и генерирует список существующих похожих записей в качестве входных данных для следующего шага. Для дальнейшего снижения использования ЦП и памяти, dbDedup ограничивает максимальное количество похожих записей, которые ex-amines для каждой функции. После достижения порога, поиск процесс завершается, и запись, содержащая наименее использованный (LRU) запись исключена из индекса объектов.

3.1.3 Выбор источника

Результаты поиска по индексу могут содержать несколько кандидатов, похожих записи, но dbDedup выбирает только одну из них для дельта-сжатия новая запись, чтобы минимизировать накладные расходы. В то время как большинство предыдущих алгоритмов выбора по сходству принимали такие решения Основываясь исключительно на показателях схожести входных данных, dbDedup добавляет рассмотрение производительности системы, отдавая предпочтение записям-кандидатам, которые присутствуют в кэше исходных записей (см. Раздел 3.3). Мы называем этот метод выбора выбором с учетом кэша. В частности, dbDedup сначала назначает начальную оценку для каждого аналогичная запись кандидата на основе количества функций, которые она имеет в общее с новой записью. Затем dbDedup увеличивает этот счет вознаграждением, если запись-кандидат уже находится в кэше. Кандидат с наивысшим баллом затем выбирается в качестве входного материала. для дельта-сжатия. В то время как выбор с учетом кэша может закончиться выбрав запись, которая не является оптимальной с точки зрения сходства, мы находим это значительно снижает накладные расходы ввода-вывода для извлечения исходных записей из База данных. Мы оцениваем эффективность выбора с учетом кэша и его чувствительность к оценке вознаграждения в Разделе 5.4.

3.1.4 Дельта-компрессия

Последний шаг в рабочем процессе dbDedup — выполнение дельта-сжатия между новой записью и выбранной похожей записью. Мы опишите детали методов кодирования в разделе 3.2 и алгоритмы сжатия в разделе 4.2.

3.2 Кодирование для онлайн-хранилища

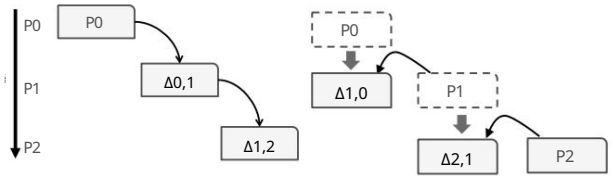
Эффективный доступ к дельта-кодированному хранилищу является давней проблемой из-за накладных расходов на ввод-вывод и вычисления, связанных с этапами кодирования и декодирования. В частности, восстановление закодированных данные могут потребовать чтения всех дельт вдоль длинной цепочки кодирования пока не будет достигнут некодированный (сырой) элемент данных. Чтобы обеспечить разумные гарантии производительности, большинство онлайн-систем используют дельта-кодирование только для уменьшения сетевой передачи (оставляя хранилище некодированным) или используют его в очень ограниченной степени в компонентах хранилища (например, путем ограничения максимальной длины цепочки кодирования до небольшого значения). Но, делая это, мы значительно недоиспользуем потенциальная экономия пространства, которая может быть достигнута.

dbDedup значительно облегчает болезненный компромисс между сжатием и скоростью доступа в дельта-кодированном хранилище с двумя новые схемы кодирования. Он использует двухсторонний метод кодирования, который уменьшает как пропускную способность удаленной репликации, так и объем хранилища баз данных, при оптимизации для общих запросов. Кроме того, он использует кодирование с переходом для сокращения числа попыток извлечения источника в худшем случае при чтении закодированных записей, при этом в значительной степени сохраняя преимущества сжатия.

3.2.1 Двустороннее кодирование

После того, как запись-кандидат выбрана из корпуса данных, dbD-edup генерирует разницу на уровне байтов между кандидатом и

[1] Мы обнаружили, что K = 8 представляет собой разумный компромисс между степенью сжатия и использованием памяти, и мы используем его как значение по умолчанию для всех экспериментов если не указано иное.



(a) Пря мое к одирование (б) Обрат ное к одирование

Рисунок 4: Иллюст рация двуст ороннего к одирования – dbDedup исполь зует пря мое к одирование для умень шения пропуск ной способност и сет и для синхрониз ации реплик, в т о время к ак исполь зование обрат ного к одирования для сжат ия хранилища баз ыданных.

новый рек орд в двух направления х, исполь зуя т ехник у, к от орую мы наз ываем двуст ороння я к одировк а. Для сет евой передач и dbDedup выполня ет пря мое к одирование (рис. 4а), к от орое исполь зует более ст арый (т .е. выбранный (к андидат) запись к ак ист оч ник и новая запись к ак цель . После к одирования ист оч ник ост ает ся в своем первонач аль ном виде, в т о время к ак цель к одируется к ак ссылк а на ист оч ник плюс дель т а от ист оч ник а в цель . dbDedup от правля ет зак одированные данные вмест о Оригиналь ная новая запись , в удаленные реплик и. Исполь зование пря мого к одирования для дедуплик ац ии на уровне сет и эт о ест ест венный выбор дизайна, пот ому ч т о позволя ет реплик ам лег к о дек одировать ц елевую запись с помощь ю лок аль но храня щ ая ся исходная запись .

dbDedup может прост о исполь зовать т у же зак одированную форму для лок аль ного Хранилище баз ыданных. Од нак о эт о приведет к знач ит ель ному сниж ению производ ит ель ност и для запросов на ч т ение к послед ней записи в ц епоч к а к одирования , к от орая , к ак мы видим, я вля ет ся обы ным случ аем с версия ми и вкл ю ения ми на уровне приложений. Пос коль ку промежут оч ные записи в пря мой ц епоч к е все храня т ся в зак одированном виде с исполь зованием предыдущий в к ач ест ве ист оч ник а, дек одирование послед ней записи т ребует повт орного из влеч ения всех дель т по всей ц епоч к е, вплоть до первого запись , к от орая хранит ся в нез ак одированном виде.

Вмест о эт ого dbDedup исполь зует обрат ное к одирование (рис. 4б) для лок аль ного хранилище для опт имиз ац ии запросов на ч т ение послед них записей. Т о ест ь , для лок аль ное хранилище, dbDedup выполня ет дель т а-сжат ие в обрат ном направлении временной пор я док , исполь зуя новую запись в к ач ест ве ист оч ник а и аналог ич ные К андидат запись в к ач ест ве ц ели. В рез ульт ат е, самая послед ня я запись в ц епоч к е к одирования всег да хранит ся в нез ак одированном виде. Запросы на ч т ение Т ак им образом, послед ня я версия не несет ник ак их нак лад ных расход ов на дек одирование. Хот я обрат ное к одирование опт имиз ировано для ч т ения , оно соз дает два пот енц иаль ных проблемы Во-перв ых, эт о увел ич ивает к олич ест во операц ий записи, т ак к ак Ст ару ю запись , выбранную в к ач ест ве ист оч ник а, необход имо обновить до зак одированной формы Ч т обы см я гч ить усиление записи, dbDedup к эширует записи с обрат ным к одированием для записи обрат но в баз у данных и задерж ивает обновления до т ех пор, пок а сист ема ввода-вывода не ст анет от носит ель но без дейст вующей, ч т о мы обсудим более под робно в раз деле 3.3. В т ор ая проблема воз ник ает , к огда В к ач ест ве ист оч ник а выбирает ся более ст ар ая запись . Сущест вующие данные (дель т а от т ек ущей баз овой записи) замен ят ся дель т ой от новой запись . Пос коль ку обрат ное к одирование реал из ует эк ономию мест а за сч ет обновления дель т а-ист оч ников, т ак ое перек рыв ающееся к одирование (рис. 5) на т ом же Исход ные записи могут привест и к нек от орой пот ере сжат ия . Пря мое к одирование, напрот ив, ест ест венным образом из бег ает эт ой проблемы, пос коль ку нет обрат ной записи т ребует ся . К сч аст ью мы обнаруж или, ч т о перек рыв ающееся к одирование не я вля ет ся распрост раненным я влением в реаль ных приложения х — боль шинст во (> 95%) обновлений я вля ют ся инк рементными на основе послед ней версии (см. раз дел 5.2).

dbDedup выполня ет дель т а-к одирование между новым и к андидат ом записи в двух направления х, но эт о т оль ко влеч ет за собой вы ислит ель ные из держ ки одного прохода к одирования . Эт о дост игает ся пут ем первой генерац ии данные с пря мым к одированием и их эффект ивное преобраз ование в обрат ная дель т а на ск орост и памя т и. Мы наз ываем эт от проц есс перек одированием и под робно опиш ит е алг оритм в раз деле 4.2.

3.2.2 К одирование прыжк ов

Пря мое к одирование Обрат ное к одирование

Рисунок 5: Перек рыв ающееся к одирование – Обрат ное к одирование может привест и к пот ере сжат ия , если в к ач ест ве ист оч ник а выбрана более ст ар ая запись . В эт ом примере к огда R0 выбирает ся в к ач ест ве ист оч ник а для R2, обрат ное к одирование ост ав ля ет R1 и R2 оба нек одированы

	Исполь зование хранилища	#Худ ший вариант из влеч ения	#Обрат ные сообщения
Обрат ное к одирование	Сб+(Н 1)· Сд	Н	Н
Перек лю ение версий	$\frac{H}{ч ас}$ · Сб+(Н $\frac{H}{ч ас}$)· Сд	ч ас	Н $\frac{H}{ч ас}$
К одирование хмеля	Sb + (N 1)· Sd H +logH NN +N ·		$\frac{ч ас}{(H\ 1)^2}$

Т аблиц а 2: Свод к а раз лич ных схем к одирования – К одирование прыжк а в знач ит ель ной ст епени уст раня ет болез ненный к омпромисс между эк ономией мест а и дек одированием ск орост ь . N — длина к одирующей ц епоч к и, а Н обознач ает расст оя ние перехода (размер к ласт ера для перехода между версия ми). Sb и Sd от нося т ся к размеру баз ы запись и дель т а соот вет ст венно, где Sb Sd в боль шинст ве случ аев. Эт и размеры оч евидно, раз лич ают ся для раз ных записей. Здесь мы исполь зуем общую ю нот ац ию для прост от а рассуждения .

К ак обсуждалось выше, исполь зование обрат ного к одирования минимиз ирует нак лад ные расход ы на дек одирование при ч т ении послед них записей, но эт о все равно может повлеч ь за собой ч рез мерное время из влеч ения ист оч ник а для случ айных запросов к более ст ар ым запис ям (например, к онк рет ная версия ст ат ь и Вик ипедии). Пред ыдущая работ а по дель т е зак одированное хранилище [26, 42] исполь зовало т ехник у, наз ываемую переход ом между версия ми Ч т обы справит ь ся с эт ой проблемой, огранич ь т е наихудшее ч исло из влеч ения исход ных данных за сч ет сниж ения к омпрессии. Идея сост оит в т ом, ч т обы раз делить ц епоч к у к одирования на ласт ер ифик ированного размера, где послед ня я запись в к аж дом ласт ере, наз ываемая эт алонной версией, сохраня ет ся в своей первонач аль ной форме, а друг ие записи храня т ся к ак дель т ы с обрат ной к одировк ой. Эт о огранич ивает худшее время поиск а к размеру ласт ера, но приводит к более низ к ой ст епени сжат ия , пос коль ку эт алонные версии не сж ат ы По мере умень шения размера ласт ера к одирования пот ери сжат ия могут знач ит ель но возраст и, пос коль ку дель т ы обы но намного мень ше баз овых записей.

dbDedup исполь зует новую т ехник у, к от орую мы наз ываем к одированием переход ов, ч т о сохраня ет ст епень сжат ия близ к ую к ст анд арт ной обрат ной к одирование, при эт ом дост игая сопост ав имого времени из влеч ения в худшем случ ае под ход с перек ом версий. К ак пок аз ано на рис. 6, доп олнит ель ные дель т ы вы ис ля ют ся между опред еленными записа ми и друг ими, наход я щ имися на нек от ором расст оя нии обрат но в ц епоч к у, в манере, похож ей на пропуск спис ков [48]. Мы наз ываем эт и записи прыжк овых баз и минималь ный инт ервал между ними Расст оя ние прыжк а обознач ает ся к ак Н. К одирование прыжк а исполь зует нес коль к о уровней к освенност и для ус к орения проц есса дек одирования , с инт ервалом на уровне L быт ь Н Л . Расшифровк а записи вк лю ает в себя снач ала от слеж ивание до ближ айшей баз ы прыжк а в лог ариф мич еск ом времени, а зат ем следуя ц епоч к а к одирования , нач инающая ся с него.

В т аблиц е 2 приведен ык ом промиссы между т ремя мет одами к одирования с т оч к и зрения исполь зования хранилища, наихудшего ч исла из влеч ений, и доп олнит ель ное к олич ест во обрат ных записей. Для к одирования прыжк ов ч исло худшего случ ая из влеч ения исход ного к ода близ ок к случ аю переход а между версия ми (Н). Но пос коль ку основания хмеля храня т ся в зак одированной форме, дост игнут ая ст епень сжат ия намного выше, ч ем при переходе с од ной версии на друг ую и сравнимо со ст анд арт ным обрат ным к одированием. Все т ри схемы к одирования влек ут за собой нек от орое к олич ест во усиления записи, но раз ниц а ст ан овится нез нач ит ель ной по мере увел ич ения расст оя ния перехода. Мы пред ст авля ем более под робное сравнение в раз деле 5.

3.3 К эширование для дель т а-к одированного хранилища

Дель т а-к одированное хранилище, благ одаря своему «ц епоч еч ному» свойст ву, заслуж ивает особого внимания .

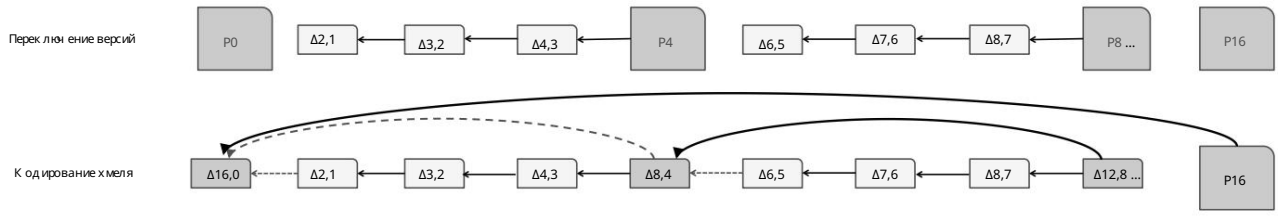


Рисунок 6: К одирование перехода – сравнение к одирования перехода и перехода версии с цепоч кой к одирования из 17 записей. Затененные записи (R0, R4 и т. д.) – это переходы баз (эталонных версий) с расстоянием перехода (размером кластера) 4. К одирование перехода обеспечивает сопоставимую скорость декодирования, как и переход между версиями, при достижении Степень сжатия близка к стандартному обратному кодированию

cialized механизмы кэширования. Используя это свойство, dbDedup кэширует только несколько ключевых узлов в заданной цепочке кодирования, максимизируя эффективность памяти, одновременно устраняя большую часть накладных расходов ввода-вывода для доступа к закодированным записям. Он использует два специализированных кэша: источник кэша записей, который уменьшает количество чтений базы данных во время кодирования, и кэш обратной записи с потерями, который смягчает усиление записи, вызванное обратным кодированием.

3.3.1 К эш исходной записи

Основной проблемой в dbDedup, как и в других системах с дельта-кодированием, являются накладные расходы ввода-вывода для извлечения базовых данных с диска в виде вход для дельта-сжатия. В частности, чтение выбранной похожей записи может потребовать дополнительного доступа к диску, конкурируя с клиентом обработка запросов и другие действия с базой данных.

dbDedup использует небольшой, но эффективный кэш записей, чтобы избежать большинства чтение диска для исходных записей. Конструкция кэша записи использует высокую степень временной локальности в обновления х записи рабочие нагрузки, которые хорошо дедуплицируются. Например, обновления в Википедии статьи, сообщения на форуме по определенной теме или обмен электронными письмами в тот же поток обычно происходит в течение короткого промежутка времени. Таким образом, вероятность нахождения недавней похожей записи в кэше высока, даже с относительно небольшим размером кэша. Другое ключевое наблюдение заключается в том, что обновления обычно являются инкрементными (основанными на непосредственном предыдущем обновлении), что означает, что две записи, как правило, более похожи, если они ближе по времени создания.

На основании вышеприведенных наблюдений кэш исходной записи сохраняет последняя запись цепочки кодирования в кэше. Для ускорения обратное кодирование баз переходов, dbDedup дополнительно кэширует последние базы хмеля на каждом уровне хмеля .2 Когда поступает новая запись, если dbDedup идентифицирует похожую запись в кэше (которая является нормальный случай из-за описанной техники выбора с учетом кэша в разделе 3.1) он заменяет существующую запись новой. Если новая запись — это база перехода, dbDedup заменяет ее соседние базы соответственно. Когда аналогичный источник не найден, dbDedup просто добавляет новую запись в кэш и удаляет самую старую запись в Метод LRU, если кэш заполнен.

3.3.2 К эш-память с обратной записью юс потерями

Как обсуждалось в разделе 3.2, обратное кодирование оптимизирует запросы чтение, но вводит некоторую амплификацию записи — вставка записи запускает дельта-сжатие исходной записи и ее обновление на диске. Проблема несколько усугубляется с кодированием перехода, где вставка базы перехода вызывает обратную запись не только в

исходной записи, но и к соседним базам на каждом уровне перехода. Для тя желье вставки, это может значительно увеличить количество операций записи на диск, что приводит к заметному снижению производительности.

Для решения этой проблемы dbDedup использует кэш обратной записи с потерями. Ключевое наблюдение заключается в том, что обратные записи не являются строго обязательными для Хранилище с обратной кодировкой. Неудача или задержка в применении таких операций обратной записи не ухудшает согласованность или целостность данных.

обновленные записи остаются нетронутыми, и единственным последствием является потенциальная потеря сжатия. Это уникальное свойство «потери» обеспечивает естественную отказоустойчивость и обеспечивает dbDedup большую гибкость в планировании когда и в каком порядке применяются обратные записи.

При вставке записи dbDedup записывает новую запись в база данных в обычном режиме и сохраняет дельту исходной записи в кэш. Он задерживает фактическую операцию обратной записи до тех пор, пока не будет выполнен системный ввод-вывод становится носитель неподвижным. Метрика бездействия может варьироваться, но мы используем длина очереди ввода-вывода как показатель в нашей текущей реализации.

Чтобы сохранить максимальное сжатие при ограниченной памяти, dbDedup сортирует дельты в кэше по абсолютному объему пространства экономия они внося т вклад и соответственно расставляет приоритеты порядке обратных записей. Когда ввод-вывод становится неактивным, более ценные дельты записываются out first. Когда кэш заполняется до того, как система переходит в режим ожидания достаточно, запись с наименьшим коэффициентом сжатия отбрасывается без влияния на корректность. Приоритезация обновления и вытеснения заказы dbDedup более эффективно используют преимущества сжатия из кэшированных дельт.

3.4 Избежание непродуктивной работы по дедупликации

dbDedup использует два подхода, чтобы избежать применения дедупликации с низкой вероятностью получения значительной выгоды Во-первых, дедупликация Губернатор отслеживает степень сжатия во время выполнения и автоматически отключает дедупликацию для баз данных, которые не приносят достаточной пользы Во-вторых, фильтр на основе размера адаптивно пропускает дедупликацию для меньших записи, которые вносят небольшой вклад в общую степень сжатия.

3.4.1 Автоматический регулятор дедупликации

Приложения баз данных демонстрируют разнообразные характеристики дедупликации. те, которые не приносят особой пользы dbDedup автоматически отключается дедупликация, чтобы избежать трат ресурсов. По нашему опыту, большинство дублирований существует в пределах одной базы данных, то есть дедупликация нескольких различных баз данных обычно дает мало маргинальных преимущества по сравнению с дедупликацией их по отдельности. Поэтому dbDedup разделяет свой индекс с дедупликацией в памяти по базе данных и внутренне отслеживает степень сжатия для каждого. Если степень сжатия для базы данных остается ниже определенного порога (например, 1.1×) в течение достаточно длительного периода (например, 100 тыс. вставок записей), dedup Governor отключает dedup для него и удаляет соответствующий раздел индекса. Будущие записи, принадлежащие этой базе данных, обрабатываются как обычно, минуя механизм дедупликации, в то время как уже закодированные данные остаются нетронутыми. dbDedup не активирует повторно база данных, для которой дедупликация уже отключена, потому что мы не обратите внимание на резкое изменение степени сжатия с течением времени для любой конкретной рабочей нагрузки, что, по нашему мнению, является нормой.

3.4.2 Адаптивный фильтр на основе размера

В наших наблюдениях за несколькими реальными наборами данных баз данных (см. Раздел 5.1) мы обнаруживаем, что большая часть экономии от дедупликации происходит из небольшого доля записей, которые больше по размеру. Рис. 7 показывает кумулятивную функцию распределения (CDF) размера записи и взвешенную CDF по вкладу в экономию пространства для четырех используемых рабочих нагрузок в наших экспериментах. Для этих наборов данных 60% самых больших записей ак-

2 По нашему опыту, количество уровней прыжков обычно невелико (3), поэтому кэш должен хранить лишь очень небольшое количество записей для каждой цепочки кодирования.

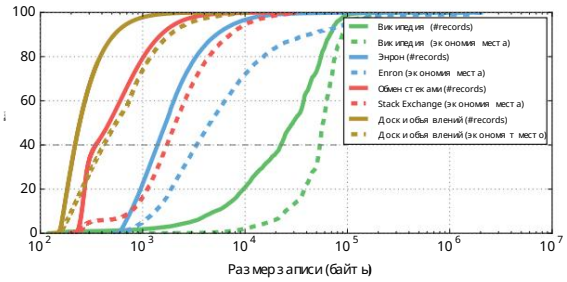


Рисунок 7: Филь т р дедуплик ации на основе раз мера.

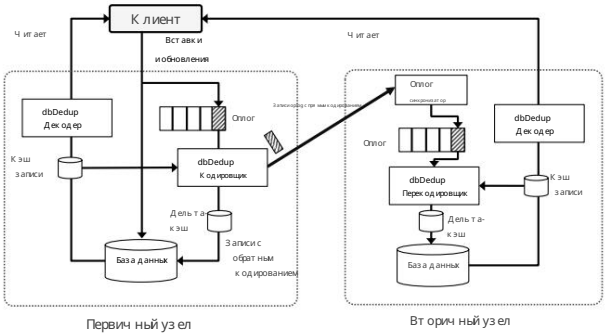


Рисунок 8: Инт еграц ия dbDedup в СУБД.

сост авля ют примерно 90–95% сок ращения данных. Друг ими словами, если мы дедуплиц ируем т оль к о записи, раз мер к от орых превышает раз мер записи 40%-плит к и, мы можем сок рат ить нак лад ные расход ына дедуплик ацион а 40%, т еря я при эт ом всего 5–10% ст епень сжат ия .

dbDedup исполь зует эт о наблюдение, исполь зуя филь т р дедуплик ации на основе раз мера, к от орый обход ит (рассмат ривает к ак уник аль ные) записи, мень шие определенного раз мера. порог . В от лич ие от спец иализированных сист ем дедуплик ации, харак т ерист ик и нагруз к и к от орых из вест ны заранее, dbDedup определя ет порог раз мер на основе баз ы данных с исполь зованием прост ой эвристик и. Для к аждого базе данных, порог дедуплик ации снач ала иниц иализируется до нуля , ч т о озн ач ает ч т о все вход я щие записи дедуплиц ируются . Эт о з нач ение з ат ем периодич еск и обновля ет ся с 40%-ным раз мером записи в базе данных к аждые 1000 вст авок записей.

4. РЕАЛИЗАЦИЯ

В эт ом раз деле описываются дет али реализ ации dbDedup, вк лю ая т о, к ак он вписывает ся в струк т урых ранения и реплик ации СУБД, а т ак же Внут ренност и алг оритма дель т а-сжат ия .

4.1 Инт еграц ия с СУБД

Хот я дет али реализ ации разл ич аются в зависимост и от СУБД, мы иллюст рируем инт еграц ия dbDedup с исполь зованием прост ой распределенной уст ановк и, сост оя щей из одного к лиент а, одного основного уз ла и одного вт орич ного уз ла, к ак пок аз ано на рис. 8. Для прост от ы мы пред полаг аем, ч т о т оль к о первич ный уз ел обслуживает з апросы на запись З и ч т о он асинхронно от правля ет обновления на вт орич ный уз ел в виде пак ет ов oplog. Мы Т еперь опишем поведение dbDedup для основных операц ий СУБД.

Вст авк а: основной уз ел записывает новую запись в свой лок аль ный баз а данных и добавля ет запись в свой oplog. К аждая запись oplog вк лю ает временную мет к у и полез ную нагруз к у, содержащую вст авленную запись . К огда раз мер несинхрон из ированных записей oplog дост игает порог ового з нач ения , первич ный уз ел от правля ет их пак ет ом на вт орич ный уз ел. Вт орич ный уз ел

3К огда вт орич ные серверы т ак же обслуживают запись , к аждый из них будет под держивать от дель ный индек с дедуплик ации. Эт и индек сы будут обновля т ься во время синхрон из ации реплик и в к онеч ном и т ог е сойдут ся .

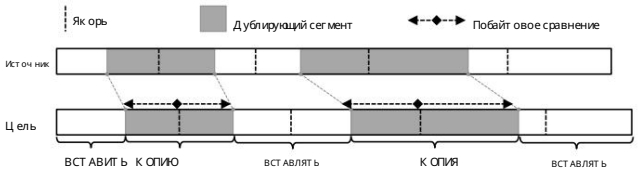


Рисунок 9: Илл юст рац ия дель т а-сжат ия в dbDedup.

ondary получ ает обновления , добавля ет их в свой лок аль ный oplog и воспроизводит новые записи oplog для обновления своей лок аль ной баз ы данных.

С помощь юdbDedup основной уз ел снач ала сохраня ет новую запись в его лок аль ный oplog. Поз же, при под г от овк е к сохранению записи или ее от правк е в реплик у, он обрабат ывает ся к одировщик ом dbDedup после Шаг и дедуплик ации описаны в раз деле 3.1. Если dbDedup успешно выбирает похожую запись из сущест вующег о к орпуса данных, он из влек ает содержимое похожей записи, снач ала проверив ист оч ник К эш записи. При промах ах к эша он сч ит ывает запись из баз ового хранилища. З ат ем он применя ет двунаправленное дель т а-сжат ие к исход ные ц елевые записи для соз дания формы с пря мым к одированием новой записи и обрат но з ак одированной формы аналог ич ной запись . dbDedup вст авля ет новую запись в основную базу данных в его исход ная форма и к эширует обрат но з ак одированную похожую запись в к эше обрат ной записи с пот еря ми, пок а сист ема ввода-вывода не ст анет прост аивать . З ат ем, dbDedup добавля ет з ак одированную запись в первич ный oplog к от орый передает ся на вт орич ный сервер во время синхрон из ации реплик и.

На вт орич ном уз ле синхрон из ат ор oplog СУБД получ ает и передает з ак одированные записи oplog в перек одировщик dbDedup. Перек одировщик снач ала дек одирует новую запись , сч ит ывая базу аналог ич ная запись из лок аль ной баз ы данных4 (или к эша исход ной записи, на хит ы) и применение вперед-к одированной дель т ы З ат ем дель т а сжимает похожую запись , исполь зуя заново рек онст руированную новую запись к ак ист оч ник , к ак в первич ном, и генерирует т у же дель т у с обрат ным к одированием для аналог ич ной записи. Нак онец , dbDedup записывает новая запись во вт орич ной базе данных и обновля ет аналог ич ную запись в его дель т а-к одированную форму. Эт и шаг и гарант ируют , ч т о вт орич ный хранит т е же данные, ч т о и основной уз ел.

dbDedup ведет под сч ет ссылок для к аждой сохраненной записи, к от орая от слеживает к олич ест во записей, ссылающихся на него к ак на базу дек одирования . Поск оль к у dbDedup исполь зует обрат ное к одирование для хранения баз ы данных, после вст авк и, сч ет чик ссылок новой записи уст анавливается равным ед иниц е, в т о время к ак ч т о аналог ич ной записи не из менилось . К олич ест во ссылок Исход ная баз а аналог ич ной записи, если т ак овая имеет ся , умень шает ся на ед иниц у.

Обновление: при обновлении dbDedup снач ала проверя ет к олич ест во ссылок . запрошенной записи. Если сч ет чик равен нулю, эт о озн ач ает , ч т о друг их записей нет обрат ит ес ь к нему для дек одирования , dbDedup напря муюприменя ет обновление к ак обыч но. В прот ивном случ ае dbDedup сохраня ет т ек ущую запись нет ронут ой и добавля ет к ней обновление. Эт о гарант ирует , ч т о друг ие записи, исполь зующие его к ак ссылк у все еще можно успешно дек одировать . К огда к олич ест во ссылок дост игает нуля , dbDedup сжимает все обновления в запись и заменя ет ее новыми данными.

dbDedup исполь зует к эш обрат ной записи для з адерж ки обновления исход ной записи с дель т а-к одированием. Ч т обы пред от врат ить перез апись обыч ного При обновлении к лиент а dbDedup всег да проверя ет к эш для к аждого обновления . Если он наход ит запись с т ем же идент ифик ат ором (к от орая будет записана поз же), он делает запись нед ейст вит ель ной и прод олж ает нормаль ное обновление к лиент а.

Удалить : Если к олич ест во ссылок на удаля емую запись равно нулю, т о удаление происход ит к ак обыч но. В прот ивном случ ае dbDedup от меч ает

4Поск оль к у вт орич ные и первич ные уз лы в основном синхрон из ированы Баз овая запись , исполь зуемая в первич ной для к одирования записи, поч т и всег да т ак же присут ст вует во вт орич ном. В ред к их случ ая х, к огда его нет , вт орич ные з апросы основной уз ел для новой записи, ч т обы из бежать дополнит ель ных з ат рат на дек одирование.

Алгоритм 1 Дельта-сжатие

```
1: функция DELTACOMPRESS(ист ,tgt)
2:     я  0                                                    Инициализация
3:     к  0
4:     поз  0
5:     сс  16
6:     sIndex  пусто
7:     tInsts  пусто
8:     пока i+ws <= src.length сделать хэш                     Построить индекс для якорей src
9:         RABINHASH(src,i,i+ws)
10:        если ISANCHOR(хэш) тогда
11:            sIndex[хэш]  i
12:        конец, если
13:        я  я +1
14:    конец, пока
15:    while j +ws <= tgt.length do Сканировать tgt на предмет самого длинного совпадения
16:        хэш  RABINHASH(tgt, j, j +ws)
17:        если ISANCHOR(хэш) и хэш в sIndex, то
18: (so ff,to ff,l)  BYTECOMP(src,tgt,sIndex[ fp], j)
19: если поз < к фф тогда
20: insInst  INST(INSERT, pos,to ff  pos)
21: memcpy(insInst.data,tgt,to ff  pos)
22: tInsts.append(insInst)
23: конец, если
24: cpInst  INST(COPY,so ff,l)
25: tInsts.append(cpInst)
26: поз  к фф +л
27: j  к ff +l
28: иначе
29: й  й +1
30: конец, если
31: конец пока
32: вернуть tInsts
33: конечная функция
```

Алгоритм 2 Дельта-перекодирование

```
1: функция DELTAREENCODE(src,tgt,tInsts)
2:     сПос  0
3:     тПос  0
4:     copySegs  пусто
5:     sInsts  пусто
6:     для каждого инста в tInsts сделать
7:         если инст.тип = КОПИРОВАТЬ тогда
8:             copySegs.append(inst.sO ff,tPos,inst.len)
9:         конец, если
10:        tPos  tPos+inst.len
11:    конец для
12:    copySegs.sortBy(sO ff)
13:    для каждого сегмента в copySegs сделать
14:        если sPos < seg.sO ff тогда
15: insInst  INST(INSERT,sPos,sO ff  sPos)
16: memcpy(insInst.data,src,sO ff  sPos)
17: sInsts.append(insInst)
18: конец, если
19: cpInst  INST(COPY,seg.tO ff,seg.len)
20: sInsts.append(cpInst)
21: sPos  seg.sO ff +seg.len
22: конец для
23: вернуть сИнсты
24: конечная функция
```

он как удаленный, но сохраняет свое содержимое. Любой клиент читает удаленный запись возвращает пустой результат, но она все равно может служить декодером база для других записей, ссылающихся на нее. Когда количество ссылок запись падает до нуля, dbDedup удаляет ее из базы данных и умень шить счетчик ссылок базовой записи на единицу.

Читайте: Если запрашиваемая запись хранится в необработанном виде, то она напря муюотправлено клиенту, как и в обычном случае. Если запись кодируется, затем декодер dbDedup возвращает его обратно в исходную форму, прежде чем он будет возвращен клиенту. Во время декодирования декодер извлекает базовую запись из кэша исходной записи (или хранение, при промахе кэша) и восстанавливает запрошенную запись с помощью сохраненная дельта. Если сама базовая запись закодирована, декодер повторяет шаг выше итеративно, пока не найдет сохраненную базовую запись в полном объеме.

Сборка мусора: количество ссылок на каждую запись гарантирует, что цепочка кодирования не будет повреждена при обновления х или удаления х. облегчает сборку мусора, dbDedup проверяет наличие удаленных объектов при чтении. В частности, по пути декодирования, если запись вид на после удаления dbDedup создает дельту между двумя соседними записи и уменьшает свой счетчик ссылок на единицу. Когда нет других записи зависят тот него для декодирования, запись может быть безопасно удалена из базы данных.

4.2 Дельта-компрессия

Чтобы обеспечить легкую дедупликацию, важно сделать dbDedup Дельта-сжатие быстрое и эффективное. Алгоритм дельта-сжатия, используемый в dbDedup, адаптирован из xDelta [42], классического алгоритма кодирования копирования/вставки с использованием техники сопоставления строк для поиска соответствующих смещений в исходном и целевом потоках байтов. Оригинальный алгоритм xDelta в основном работает в два этапа. На первом этапе xDelta делит исходный поток на поток и фиксированного размера (по блок неисправности, 16-байтовый). Затем он вычисляет контрольную сумму Alder32 [32] (та же функция отпечатка пальца, которая используется в gzip) для каждого байтового блока и создает временный индекс в памяти, отображающий контрольные суммы к их соответствующим смещениям в источнике. На втором этапе, xDelta сканирует целевой объект байт за байтом с самого начала, используя скользящее окно того же размера, что и блок из байтов. Для каждого целевое смещение, он вычисляет контрольную сумму Alder32 байтов в скользящее окно и сверяется с исходным индексом, заполненным в первом шаг. Если он находит совпадение, xDelta расширяет процесс поиска с сопоставленные смещения, используя двунаправленное побайтовое сравнение для определения самой длинной общей последовательности (LCS) между источником и целевые потоки. Затем он пропускает совпавший регион, чтобы продолжить итеративный поиск. Если совпадений не обнаружено, он перемещает скользящую окно на один байт и перезапускает сопоставление. В ходе этого процесса, xDelta кодирует соответствующие регионы в целевом объекте в инструкции COPY, а несоответствующие регионы — в инструкции INSERT.

Как показано в алгоритме 1 и на рис. 9, алгоритм дельта-сжатия dbDedup представляет собой модифицированную версию xDelta, основанную на наблюдении, что большая часть времени тратится на построение исходного индекса. и поиск. На первом этапе кодирования dbDedup делает выборку подмножества смещенных позиций, называемых якорями, контрольные суммы которых ниже битсоответствуют заранее определенному шаблону. Интервал между якорями указывает коэффициент дискретизации и контролируется длиной бита шаблон. На втором этапе dbDedup выполняет только поиск индекса для якорей в цели, избегая необходимости консультироваться с источником Индекс на каждом целевом смещении. Интервал привязки обеспечивает настраиваемый компромисс между степенью сжатия и скорость кодирования, и мы оценим его эффекты в разделе 5. Мы опускаем некоторые оптимизации в Псевдокод, приведенный выше, из-за ограничений по пространству. Например, смежные и перекрывающиеся инструкции COPY объединяются; короткие Инструкции COPY преобразуются в эквивалентные инструкции INSERT. когда затраты на кодирование превышают экономию места.

Как обсуждалось в разделе 3.2, после вычисления данных с прямым кодированием с использованием алгоритма выше, dbDedup использует дельта перекодирование (Алгоритм 2) для эффективного создания обратно-кодированной исходной записи. Вместо переключения источника и цели объекты и повторное выполнение дельта-сжатия, dbDedup повторно использует инструкции COPY, сгенерированные ранее, и сортирует их по соответствующим исходным смещения м. Затем он заполняет несовпадающие области в источник с инструкциями INSERT. Хотя это может привести к немного

Неоптимальная скорость сжатия (например, из-за перекрывающихся инструкций COPY, которые объединяются), процесс перекодирования происходит чрезвычайно быстро (со скоростью юпамяти), поскольку не выполняются вычисления контрольной суммы или операции с индексами.

Дельта-декомпрессия в dbDedup проста. Она просто итерирует инструкции, сгенерированные алгоритмом сжатия, и объединяет совпадающие и несовпадающие регионы для воспроизведения исходного целевого объекта.

5. ОЦЕНКА

В этом разделе dbDedup оценивается с использованием четырех реальных наборов данных. Для этой оценки мы реализовали как dbDedup, так и традиционную дедупликацию на основе фрагментов (trad-dedup) в MongoDB (v3.1). Результаты показывают, что dbDedup обеспечивает значительные преимущества сжатия, превосходит традиционную дедупликацию сочетается с компрессией на уровне блоков и накладывает незначительные накладные расходы на производительность СУБД.

Если не указано иное, во всех экспериментах используется реплицированная настройка Mon-goDB с одним первичным, одним вторичным и одним клиентским узлом. Каждый узел имеет четырехядерный ЦП, 8 ГБ ОЗУ и 100 ГБ локального хранилища HDD. Мы используем движок хранения WiredTiger [16] от MongoDB с отключенной функцией полного журналирования, чтобы избежать помех.

5.1 Рабочие нагрузки

Четыре реальных набора данных представляют собой разнообразный спектр приложений баз данных: совместное редактирование (Wikipedia), электронная почта (Enron) и онлайн-форумы (Stack Exchange, доски обьявлений).
Мы сортируем каждый набор данных по временной метке создания, чтобы сгенерировать трассировку записи, а затем генерируем трассировку чтения, используя общедоступную статистику или известные шаблоны доступа, чтобы имитировать реальную рабочую нагрузку, как подробно описано ниже.

Википедия: Полная история изменений каждой статьи в корпусе английского языка Википедии [13] с января 2001 года по август 2014 года.
Мы извлекли 20 ГБ подмножества с помощью случайной выборки на основе идентификаторов статей. Каждая редакция содержит новую версию статьи и имеет данные о пользователе, который внес изменения (например, имя пользователя, временная метка, комментарий). Большая часть дублирования происходит из-за дополнительных изменений страниц. Мы вставляем первые 10 000 изменений, чтобы заполнить исходную базу данных. Затем мы выдаем запросы на чтение и запись в соответствии с общедоступным доступом к Википедии [62], где нормализованное соотношение чтения/записи составляет 99,9 к 0,1. 99,7% запросов на чтение относятся к последней версии страницы вики, а остальные — к определенной версии.

Enron: публичный набор данных электронной почты[2] с данными примерно 150 пользователей, в основном высшего руководства Enron. Корпус содержит около 500 тыс. сообщений, что в общей сложности составляет 1,5 ГБ данных. Каждое сообщение содержит текстовое тело, имя почтового ящика, заголовки сообщений, такие как временная метка и идентификаторы отправителя/получателя. Дублирование в основном происходит из-за пересылок сообщений и ответов, которые содержат содержимое предыдущих сообщений. Мы вставляем отсортированный набор данных в СУБД как можно быстрее. После каждой вставки мы выдаем запрос на чтение для конкретного сообщения электронной почты, в результате чего совокупное соотношение чтения/записи составляет 1 к 1. Это основано на предположении, что каждый пользователь использует один почтовый клиент, который локально кэширует запрошенное сообщение, поэтому каждое сообщение записывается и считывается один раз в/из СУБД.

Stack Exchange: публичный дамп данных из сети Stack Exchange [10], содержащий полную историю сообщений пользователей и связанную с ними информацию, такую как теги и голоса. Мы извлекли подмножество размером 10 ГБ с помощью случайной выборки. Большая часть дублирования в этом наборе данных происходит из-за того, что пользователи редактируют свои собственные сообщения и копируют ответы из других веток обсуждения. Мы вставляем сообщения в СУБД как новые записи во временном порядке. Для каждого сообщения мы читаем его столько же раз, сколько было просмотров. Совокупное соотношение чтения/записи составляет 99,9 к 0,1.

Доски сообщений: набор данных форума объемом 10 ГБ, содержащий сообщения пользователей, сканированные с ряда общедоступных досок сообщений на базе vBulletin [11], которые охватывают широкий спектр тем, связанных между собой, таких как спорт, автомобили и животные. Каждое сообщение содержит название форума, идентификатор темы, идентификатор сообщения, идентификатор пользователя и текст сообщения, включая цитаты из других сообщений. Этот набор данных также содержит количество просмотров на поток, которое мы используем для генерации синтетических запросов на чтение. Дублирование в основном происходит из-за того, что пользователи цитируют комментарии других. Чтобы имитировать поведение пользователей на форуме обсуждений, для каждой вставки сообщения мы выдаем определенное количество «прочтений потока», которые запрашивают все предыдущие сообщения в содержащем поток. Количество прочтений потока на вставку выводится путем деления общего количества просмотров потока на количество содержащихся в нем сообщений.

5.2 Коэффициент сжатия и индексная память

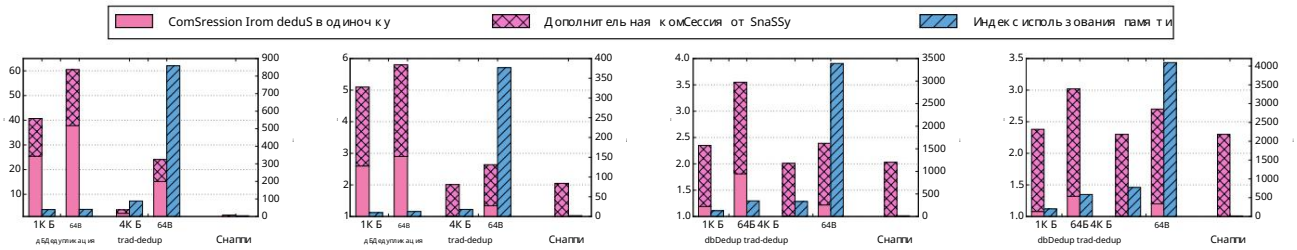
Сначала мы оцениваем коэффициент сжатия dbDedup и использование памяти индекса и сравниваем их с trad-dedup и Snappy [9], компрессором Mon-goDB на уровне блоков по умолчанию. Для каждого набора данных мы загружаем записи в СУБД как можно быстрее и измеряем полученные размеры хранилища, объем данных, переданных по сети, и использование памяти индекса.

На рис. 10 показаны результаты для пяти конфигураций: (1) dbDedup с фрагментами по 1 КБ или 64 байта, (2) trad-dedup с фрагментами по 4 КБ или 64 байта и (3) Snappy. Розовая (левая) полоса показывает коэффициент сжатия хранилища, указывающий на вклад только дедупликации и сжатия после дедупликации. Коэффициент сжатия определяется как исходный размер данных, деленный на размер сжатых данных, поэтому значение, равное единице, означает, что сжатие не достигнуто. Синяя (правая) полоса показывает использование памяти индекса. Небольшой кэш исходных записей (32 МБ, используется как dbDedup, так и trad-dedup) и кэш обратной записи с потерями (8 МБ, используется только dbDedup) не показаны

Наибольшие преимущества наблюдаются для Wikipedia (рис. 10a). При размере фрагмента 1 КБ dbDedup сокращает хранилище данных в 26 раз (в 41 раз в сочетании со Snappy), используя 36 МБ индексной памяти. Уменьшение размера фрагмента до 64 Б увеличивает коэффициент сжатия до 37 раз (61 раз), используя всего 45 МБ индексной памяти. Уменьшение размера фрагмента для dbDedup не сильно увеличивает использование индексной памяти, поскольку у dbDedup индексируется не более K записей на запись, независимо от размера фрагмента. Напротив, в то время как коэффициент сжатия trad-dedup увеличивается с 2,3 раз (3,7 раз) до 15 раз (24 раза) при использовании размера фрагмента 64 Б вместо 4 КБ, его индексная память увеличивается с 80 МБ до 780 МБ, что делает его непрактичным для операционных СУБД. Это связано с тем, что trad-dedup индексирует каждый уникальный хэш-фрагмента, что приводит к почти линейному увеличению накладных расходов индекса по мере уменьшения размера фрагмента, а также с тем, что он должен использовать от раза до больших ключей индекса (20-байтовый хэш SHA-1 против 2-байтовой контрольной суммы), поскольку у коллизии приведут к повреждению данных. Потребляя на 40% меньше памяти индекса, dbDedup с размером фрагмента 64 Б достигает коэффициента сжатия в 16 раз выше, чем trad-dedup с его типичным размером фрагмента 4 КБ. Snappy сжимает набор данных всего в 1,6 раза, поскольку у он не может устранить дублирование, вызванное версионированием на уровне приложения, но не требует памяти индекса. Он обеспечивает такое же сжатие в 1,6 раза при применении к дедуплицированным данным.

Для других наборов данных абсолютные преимущества меньше, но основные наблюдения схожи: dbDedup обеспечивает более высокую степень сжатия при меньшем использовании памяти, чем trad-dedup, а преимущества сжатия Snappy (1,6–2,3×) дополняются дедупликацией. Для набора данных Enron (рис. 10b) dbDedup сокращает хранилище на 3,0× (5,8×), что согласуется с результатами, полученными нами в ходе экспериментов с данными из облачного развертывания серверов Microsoft Exchange, содержащих ПБ реальных данных электронной почты пользователей.5 Два набора данных форума (рис. 10c и 10d) не демонстрируют столько дублирования.

5К сожалению, мы не можем раскрыть подробности из-за ограничений конфиденциальности.



(a) Википедия (b) Энрон (c) Обмен стеками (d) Доски объявлений

Рисунок 10: Коэффициент сжатия и индексная память – Коэффициент сжатия и использование индексной памяти для dbDedup (фрагментыпо 1 КБ или 64 байта), trad-dedup (4 КБ и 64 байта) и Snappy. Верхняя часть каждой полосыпредставляет дополнительное преимущество сжатия после дедупликации.

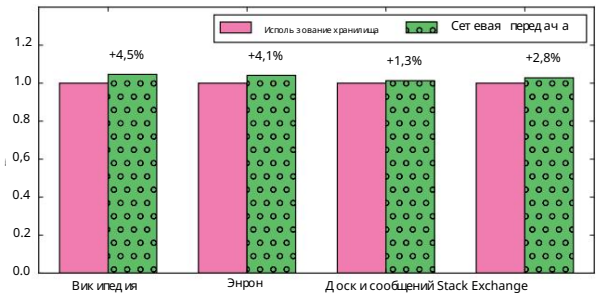
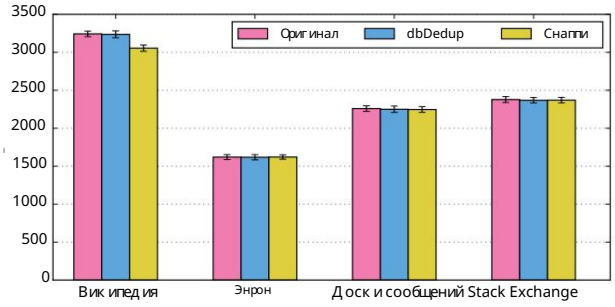
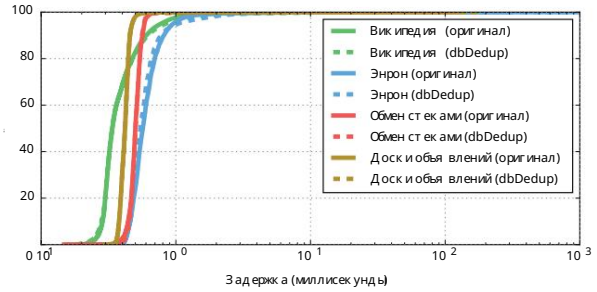


Рисунок 11: Экономия пропускной способности хранилища и сети — относительные коэффициентысжатия, достигаемые с помощьюdbDedup (с размером фрагмента 64 байта) для локального хранилища и сетевой передачи для каждого из наборов данных, нормализованные по абсолютным коэффициентам сжатия хранилища, показанным на рис. 10 (для dbDedup с размером фрагмента 64 байта).



(a) Пропускная способность



(b) Задержка

Рисунок 12: Влияние на производительность — измерения производительности и задержки MongoDB во время выполнения различных рабочих нагрузок и конфигураций.

tion как Wikipedia или наборыданных электронной почты,потому что пользователи не цитируют или не редактируют комментарии так часто, как редакции Wikipedia или пересылки/ответыэлектронной почты.Тем не менее, мывсе равно наблюдаем, что dbDedup уменьшает хранилище в 1,3–1,8× (3–3,5×). Поскольку мысмогли проанализировать только последние сообщения в наборе данных Message Boards, коэффициент сжатия dbDedup является консервативным, не включая преимущества отдельта-сжатия редакций пользователей в их собственных сообщениях.[6] В

дополнение к использованиюхранилища, dbDedup одновременно достигает значительного сжатия при передаче данных по сети с прямым кодированием. На рис. 11 показано сжатие на уровне сети как нормализованный результат по отношениюк использованиюхранилища (1,0 на оси Y для каждого набора данных). dbDedup достигает немного более низкого сжатия в хранилище базыданных, чем в данных, передаваемых по сети, в основномиз-за перекрывающихся кодировок (раздел 3.2) и дельта-вытеснений из кэша обратной записи. Тем не менее, разница составляет менее 5% для всех наборов данных, поскольку перекрывающиеся кодировки и встречаются редко, а кэш обратной записи с потерями использует приоритетное вытеснение.

5.3 Влияние на производительность во время выполнения

Этот эксперимент предназначен для измерения влияния dbDedup на производительность СУБД. Мысравниваем три конфигурации развертывания MongoDB: (1) Без сжатия («Исходная»), (2) dbDedup и (3) Snappy. Для каждой настройки мызапускаем эксперимент тритраза для всех рабочих нагрузок и сообщаем среднее значение.

Пропускная способность: Рис. 12a показывает пропускнуюспособность для четырех рабочих нагрузок. Мывидим, что dbDedup накладывает незначительные накладные расходына пропускнуюспособность. Snappy также немного ухудшает производительность для трех рабочих нагрузок, поскольку это быстрый и легкий встроенный компрессор. Исключением является Wikipedia, для которой использование Snappy приводит к снижениюпропускной способности на 5%, поскольку у некоторые большие записи Wikipedia не может поместиться на одной страницеWiredTiger и требует дополнительных операций ввода-вывода.

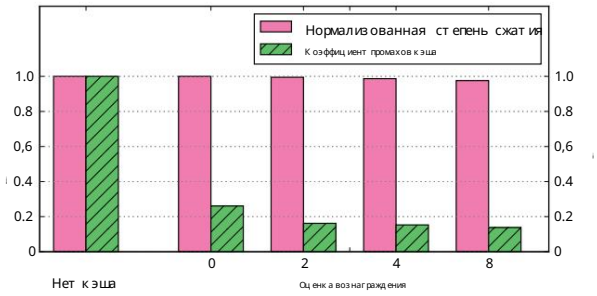
Задержка: Рис. 12b показывает CDF задержки клиента. Для ясностимыпоказываем результаты только для MongoDB с включенным dbDedup ибез него. Опять же, мы видим, что dbDedup почти не влияет на производительность. Кривые распределения задержки с включенным dbDedup близко соответствуют кривымбез сжатия / дедупликации. Разница в задержке 99,9%-плитки составляет менее 1% для всех рабочих нагрузок.

5.4 Эффективность эширования dbDedup

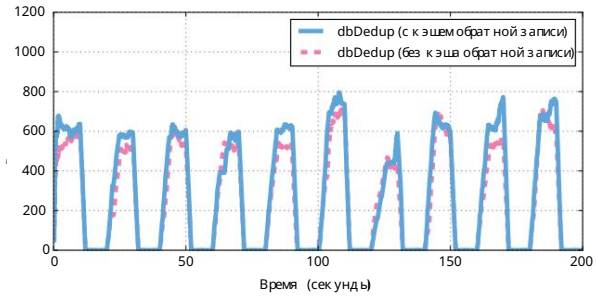
использует два специализированных кэша для минимизации накладных расходов ввода-вывода, связанных с чтением и обновлением исходных записей: кэш исходных записей (32 МБ) и кэш обратной записи с потерями (8 МБ). Теперь мыоцениваем эффективность этих кэшей.

Кэш исходной записи: Рис. 13a показывает влияние кэша исходной записи на коэффициент сжатия (левая ось Y) и процент извлечений исходной записи, требующих чтения СУБД (коэффициент промахов кэша; правая ось Y), с диапазоном значений оценки вознаграждения для рабочей нагрузки Wikipedia. Напомним, что dbDedup использует выборпохожих записей с учетом кэша, назначая оценку вознаграждения кандидатам, которые присутствуют в кэше (см. Раздел 3.1.3).

[6] Мыобнаружили, что 15% сообщений редактируются как минимумодин раз, а размербольшинства отредактированных сообщений превышает средний размер сообщения.



(a) Кэш исходной записи



(b) Кэш с обратной записью юс потеря ми

Рисунок 13: Эффект кэширования – измерения времени выполнения dbDedup механизмы кэширования для рабочей нагрузки Википедии.

Когда кэш не используется (самые левые столбцы), каждое извлечение
Исходная запись вызывает запрос на чтение. Даже без выбора с учетом кэша (0 баллов
вознаграждения) небольшой кэш исходной записи устраняет 74%
этих запросов. С оценкой вознаграждения два (по умолчанию) метод выбора с учетом кэша
еще больше сокращает коэффициент промахов на 40% (до
16%), без заметного снижения степени сжатия. Далее
увеличение оценки вознаграждения незначительно снижает коэффициент промахов кэша
при этом немного уменьшив степень сжатия, поскольку менее похожи
Кандидаты с более высокой вероятностью будут выбраны в качестве исходных записей.

Кэш обратной записи с потерями dbDedup использует обратное кодирование
чтобы избежать декодирования при чтении последних «версий» последовательности
обновления. Таким образом, дедупликация новой записи включает в себя как запись
полностью новая запись и замена исходной записи на дельта-кодированную
данные. Дополнительная запись (замена) может привести к значительным проблемам с
производительностью для интенсивных рабочих нагрузок ввода-вывода во время всплесков записи.
Кэш обратной записи с потерями dbDedup смягчает такие проблемы

Чтобы имитировать пиковую нагрузку с интенсивным вводом-выводом и периодами простоя, мы
вставляем данные Википедии на полной скорости в течение 10 секунд и переходим в спящий режим.
в течение 10 секунд, многократно. Рис. 13b показывает вставку MongoDB
производительность с течением времени, с кэшем обратной записи и без него. Без кэша
производительность СУБД заметно снижается в периоды взаимодействия тостинга из-за дополнительных
записей в базу данных. Напротив, при использовании
кэша обратной записи позволяет избежать замедления работы СУБД во время пиковых нагрузок,
как показано на разнице между двумя линиями в разных точках
времени (например, в секунды 0, 130, 170 и 190).

dbDedup имеет два основных настраиваемых параметра, помимо рассмотренных выше, которые
влияют на компромисс между сжатием и производительностью hop
Расстояние и интервал между якорями. Этот подраздел количественно определяет эффекты
этих параметров и поясняет значения по умолчанию

5.5 Кодирование перехода

dbDedup использует кодирование переходов для уменьшения наихудшего случая извлечения

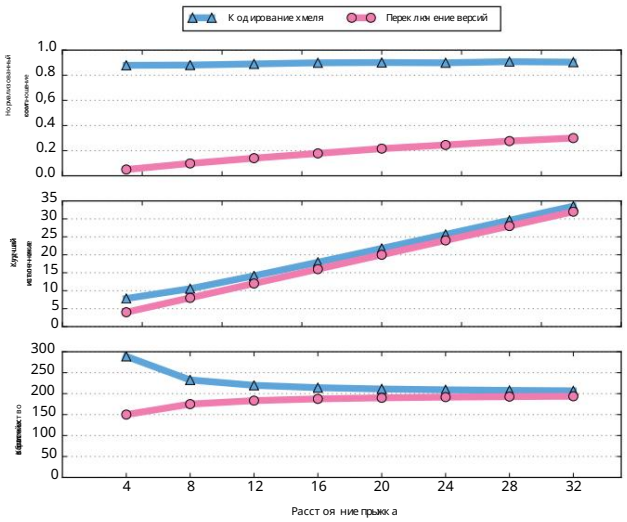


Рисунок 14: Кодирование скачков против перехода между версиями — для Википедии
рабочая нагрузка и умеренные расстояния перехода, кодирование перехода обеспечивает гораздо более высокую
коэффициент сжатия с небольшим увеличением в худшем случае извлечения источника и
количество обратных записей.

раз, сохраняя при этом преимущества сжатия. Чтобы оценить его эффективность, мы также
реализовали переход между версиями в MongoDB и сравнили две схемы кодирования.

На рис. 14 показаны результаты для трех метрик в зависимости от прыжка.
расстояние: коэффициент сжатия (нормализованный по отношению к стандартному обратному
кодированию), наихудшее количество извлечений источника (для кодирования
Длина цепочки 200) и количество обратных записей. Перекликание версий
приводит к значительному (на 60–90%) снижению степени сжатия, поскольку
все справочные версии хранятся в незакодированном виде. Его степень сжатия
улучшается по мере увеличения расстояния прыжка, поскольку меньше записей
хранятся в незакодированном виде. В отличие от этого, поскольку у хмелевые основания хранятся
как дельты кодирование скачков обеспечивает коэффициент сжатия в пределах 10%
Полное обратное кодирование. Для кодирования прыжка коэффициент сжатия
остается относительно стабильным по мере увеличения расстояния прыжка из-за наличия
меньше, но менее схожие хмелевые базы

Число наихудших случаев получения источника для кодирования прыжка
близок к версии jumping. С несколькими уровнями прыжков,
отслеживание до ближайшей базы хмеля занимает всего лишь логарифмическое время.
По мере увеличения расстояния прыжка время декодирования доминирует
прохождение обратных дельт между соседними базами прыжков. Нижний график показывает
количество дополнительных обратных записей, необходимых в каждом
схема. В то время как кодирование прыжка вызывает больше обратных записей для небольших прыжков
расстояния, обе схемы быстро приближаются к длине кодирующей цепи по мере увеличения
расстояния прыжка. Эмпирически мы обнаруживаем, что прыжок
Расстояние 16 (по умолчанию) обеспечивает хороший компромисс между степенью сжатия и
затратами на декодирование.

5.6 Оптимизация дельта-сжатия

dbDedup превосходит алгоритм xDelta, сокращая накладные расходы на вставку исходного
индекса и поиск. Он вводит настраиваемый интервал привязки, который управляет частотой
выборки и
смещение точек в исходном потоке байтов.

На рис. 15 показана степень сжатия (левая ось Y) и пропускная способность.
(правая ось Y) для различных значений интервала привязки, для сравнения
с xDelta, для рабочей нагрузки Wikipedia. С интервалом привязки 16 (размер окна по умолчанию
в xDelta), dbDedup выполняет
почти то же самое, что и xDelta. Скорость дельта-сжатия dbDedup улучшается с
увеличением интервала привязки, поскольку он уменьшает количество
вставок и поисков в исходном индексе смещения.

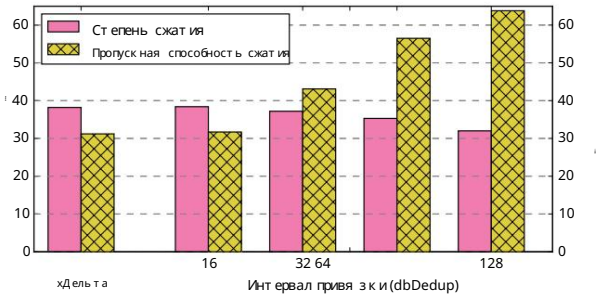


Рисунок 15: Оптимизация дельта-сжатия — сравнение оптимизированного варианта dbDedup и xDelta с использованием рабочей нагрузки Wikipedia.

Коэффициент сжатия не уменьшается существенно, поскольку dbDedup выполняет сравнение на уровне байтов в двух направлениях от сопоставленных точек. При интервале привязки 64 dbDedup превосходит xDelta на 80% с точки зрения пропускной способности сжатия, при этом теряется всего 7% коэффициента сжатия. Увеличение интервала привязки до 128 дополнительно увеличивает пропускную способность на 10%, но приводит к потере 15% коэффициента сжатия. Мы используем 64 в качестве значения по умолчанию, что обеспечивает разумный баланс между коэффициентом сжатия и пропускной способностью

6. ДОПОЛНИТЕЛЬНАЯ СВЯЗАННАЯ РАБОТА

Большая часть предыдущей работы по дедупликации обсуждается в разделе 2. Этот раздел обсуждается некоторая дополнительная связанная работа.

Сжатие базы данных: За последние несколько десятилетий было предложено несколько схем сжатия базы данных. Большинство операционных СУБД, которые сжимают содержимое базы данных, используют сжатие на уровне страниц или блоков [30, 37, 46, 43, 3, 16]. Некоторые используют сжатие префиксов, которое ищет общие последовательности в начале значений полей для данного столбца по всем строкам на каждой странице.

Как и в случае с нашим подходом dbDedup, такое сжатие требует, чтобы СУБД распаковывала кортежи перед обработкой запроса.

В некоторых системах OLAP существуют схемы, которые позволяют СУБД обрабатывать данные в сжатом формате. Например, сжатие словаря заменяет повторяющиеся длинные значения домена коротким или целочисленным кодами фиксированной длины. Этот подход обычно используется в хранилищах данных, ориентированных на столбцы[18, 36, 69, 51]. Эти системы обычно фокусируются на атрибутах с относительно небольшим размером домена и исследуют перекос в частотах значений, чтобы ограничить результирующий словарь управляемым размером[23]. Авторы [56] предлагают схему дельта-кодирования, где каждое значение в отсортированном столбце представлено дельтой от предыдущего значения. Хотя этот подход хорошо работает для числовых значений, он не подходит для строк.

Ни один из этих методов не обнаруживает и не устраняет избыточные данные с гранулярностью менее одного поля, тем самым теряя потенциальные преимущества сжатия для многих приложений, которые изначально содержат такую избыточность. dbDedup, напротив, способен удалять гораздо более мелкие дубликаты с помощью дельта-сжатия на уровне байтов. В отличие от других схем встроенного сжатия, dbDedup не находится на критическом пути записи для запросов и, следовательно, оказывает минимальное влияние на производительность СУБД во время выполнения. В дополнение к этому, поскольку dbDedup сжимает данные на уровне записи, он выполняет шаги дедупликации только один раз и использует закодированный результат как для хранения в базе данных, так и для передачи по сети. Напротив, одна и та же запись будет сжата дважды (на странице базы данных и в пакете oplog) для схем сжатия страниц для достижения сокращения данных на обоих уровнях.

Дельта-кодирование: Было проведено большое количество предыдущих работ по методам дельта-кодирования, включая несколько универсальных алгоритмов, основанных на подходе Лемпела-Зива [68], таких как vcd-iff [21], xDelta [42] и zdelta [60]. Специализированные схемы могут использоваться для определенных форматов данных (например, XML) для улучшения качества сжатия [28, 63, 39, 52]. Алгоритм дельта-сжатия, используемый в dbDedup, адаптирован из xDelta, связь с которым обсуждается в разделе 4.2.

Дельта-сжатие использовалось для сокращения сетевого трафика для протоколов передачи файлов и синхронизации. Большинство систем предполагают, что предыдущие версии одного и того же файла явно идентифицируются приложением, и дублирование существует только среди предыдущих версий одного и того же файла [61, 57]. Исключением является TAPER [38], который сокращает сетевую передачу для синхронизации реплик файловой системы путем отправки дельта-кодированных файлов; он идентифицирует похожие файлы вычисляя количество совпадающих битов в фильтрах Блума, сгенерированных с помощью хэшей фрагментов файлов. dbDedup идентифицирует похожую запись из корпуса данных без руководства приложения и, следовательно, является более общим подходом, чем большинство этих предыдущих систем.

Метод обратного кодирования, используемый в dbDedup, вдохновлен системами хранения данных с управлением версиями, такими как RCS [59] и XDFS [42].
Похожие методы использовались в системах контроля версий, таких как Git [41] и SVN [29], чтобы обеспечить возможность перемещения назад по истории коммитов. В отличие от этих систем, которые явно поддерживают родословную версий для всех файлов, dbDedup устанавливает цепочку кодирования полностью на основе отношений сходства между записями, и таким образом, не требует поддержки на уровне системы для управления версиями. [54] использует дельта-кодирование для дедуплицированного резервного хранилища. Он использует прямое кодирование и допускает только цепочки кодирования с максимальной длиной в два элемента. Подобно другим системам хранения с дельта-кодированием, которые используют переходы между версиями, он должен пожертвовать выигрышем в сжатии, чтобы ограничить наихудшие случаи извлечения для базовых данных.

Насколько нам известно, dbDedup — первая система, которая исследует различные формы кодирования для сжатия на уровне сети и хранилища и обеспечивает эффективные преобразования между ними. Благодаря новой схеме кодирования с переходом dbDedup значительно облегчает болезненные компромиссы между степенью сжатия и издержками на извлечение для дельта-кодированного хранилища. Кроме того, он вводит новые механизмы кэширования, специализированные для дельта-кодированного хранилища, значительно сокращая издержки ввода-вывода, при этом максимально увеличивая эффективность памяти. Все эти методы совокупности делают онлайн-доступ к закодированному хранилищу практичным.

Обнаружение сходства: Предыдущие работы предоставили различные подходы к вычислению эскизов (метрик сходства) для идентификации схожих элементов. Базовая техника идентификации признаков в объектах таким образом, что обычные объекты имели идентичные признаки, была впервые предложена Бродером [24, 25] в контексте веб-страниц. Несколько статей [58, 47, 20, 53, 65] предлагают методы вычисления эскизов для обнаружения сходства, которые являются надежными к небольшим изменениям в данных. Подход к извлечению признаков, используемый в dbDedup, аналогичен подходу в DOT [47] и sDedup [65].

7. ЗАКЛЮЧЕНИЕ

dbDedup — это легкий механизм дедупликации на основе подобия для операционных СУБД, который сокращает как использование хранилища, так и объем данных, передаваемых для удаленной репликации. Объединяя частичную индексацию и дельта-сжатие на уровне байтов, dbDedup достигает более высоких коэффициентов сжатия, чем сжатие на уровне блоков и дедупликация на основе фрагментов, при этом эффективно используя память. Он использует новые механизмы кодирования и кэширования, чтобы избежать значительных накладных расходов ввода-вывода, связанных с доступом к записям с дельта-кодированием.
Экспериментальные результаты четырьмя реальными рабочими нагрузками показывают, что dbDedup способен достичь сокращения до 37 раз (в 61 раз при сочетании со сжатием на уровне блоков) размера хранилища и трафика репликации, при этом накладывая незначительные накладные расходы на производительность СУБД.

Благодарности. Мы хотели бы поблагодарить Кейта Бостика за его руководство по внутренним компонентам WiredTiger. Мы благодарим анонимных рецензентов за полезные отзывы о статье. Мы также благодарим членов и компании консорциума PDL (включая Broadcom, Citadel, Dell EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi, Huawei, Intel, Microsoft Research, NetApp, Oracle, Samsung, Seagate, Tintri, Two Sigma, Uber, Veritas и Western Digital) за их интерес, идеи, отзывы и поддержку. Это исследование было частично спонсировано Intel в рамках Научно-технического центра Intel по облачным вычислениям (ISTC-CC) и MongoDB Incorporated. Эксперименты были проведены благодаря щедрым пожертвованиям оборудования от Intel и NetApp.

8. ССЫЛКИ

[1] Байду Байке. http://baike.baidu.com/.

[2] Набор данных электронной почты Enron. https://www.cs.cmu.edu/~./enron/.

[3] Сжатие InnoDB. http://dev.mysql.com/doc/refman/5.6/en/innodb-compression-internals.html .

[4] Linux SDFS. www.opendedup.org.

[5] MongoDB. http://www.mongodb.org.

[6] MurmurHash. https://sites.google.com/site/murmurhash.

[7] Ocarina Networks. www.ocarinanetworks.com.

[8] Оптимизация данных Permabit. www.permabit.com.

[9] Снаппи. http://google.github.io/snappy/.

[10] Архив данных Stack Exchange. https://archive.org/details/stackexchange.

[11] vБюллетень . https://www.vbulletin.com.

[12] W3Techs. http://www.w3techs.com.

[13] Загрузки Wikimedia. https://dumps.wikimedia.org.

[14] Википедия . https://www.wikipedia.org/.

[15] Сервер хранения Windows. technet.microsoft.com/en-us/library/gg232683(WS.10).aspx.

[16] WiredTiger. http://www.wiredtiger.com/.

[17] Дедупликация ZFS. blogs.oracle.com/bonwick/entry/zfs_dedup.

[18] Д. Абади, С. Мэдден и М. Феррейра. Интеграция сжатие и выполнение в системах столбчатых баз данных. В SIGMOD, страницы671–682, 2006.

[19] С. Alvarez. Руководство по дедупликации NetApp для FAS и V-Series и ее внедрению 2010.

[20] Л. Аронович , Р. Ашер, Э. Бахмат , Х. Битнер, М. Хирш и ST Klein. Проектирование системы дедупликации на основе сходства. В SYSTOR, стр. 6, 2009.

[21] Дж. Бентли и Д. Макилрой. Сжатие данных с использованием длинных общих строк . На конференции по сжатию данных, 1999. Труды DCC'99, страницы287–295, 1999.

[22] Д. Бхагват , К . Эшги, Д.Д. Лонг и М. Лиллибридж. Экстремальное биннинг: Масштабируемая параллельная дедупликация для резервного копирования файлов на основе фрагментов. В MASCOTS, страницы1–9, 2009.

[23] С. Binnig, S. Hildenbrand и F. Färber. Сжатие строк с сохранением порядка на основе словаря для хранилищ столбцов основной памяти. В SIGMOD, страницы283–296, 2009.

[24] Бродер А. О сходстве и содержании документы Сжатие и сложность последовательностей, 1997.

[25] Бродер А. Вычисление и фильтрация почти дубликатов документы 11-й ежегодный симпозиум по комбинаторному сопоставлениюобразов, 2000.

[26] RC Burns и DD Long. Эффективное распределенное резервное копирование с дельта-сжатием. В Трудах пятого семинара по вводу-выводу в параллельных и распределенных системах, страницы27–36, 1997.

[27] А. Клементс, И. Ахмад, М. Вилайаннур и Дж. Ли. Децентрализованная дедупликация в кластерных файловых системах SAN. В USENIX ATC, 2009.

[28] G. Cobena, S. Abiteboul и A. Marian. Обнаружение изменений в документах XML. В ICDE, страницы41–52, 2002.

[29] Б. Коллинз-Сасман, Б. Фицпатрик и М. Пилато. Версия контроль с помощью subversion. 2004.

[30] Кормак Г. В. Сжатие данных в системе баз данных. Сообщения ACM, 28(12):1336–1342, 1985.

[31] Б. Дебнат , С. Сенгупта и Дж. Ли. Chunkstash: Ускорение дедупликации встроенного хранилища с использованием флэш-памяти. На ежегодной технической конференции USENIX, 2010.

[32] П. Дойч и Ж.-Л. Гейлли. Спецификация формата сжатых данных Zlib версии 3.3. Технический отчет, 1996.

[33] К . Дубницкий, Л. Грьв, Л. Хельдт , М. Качмарчик , В. Килиан, П. Стшельчак и Й. Щепковски. Hydrastor: масштабируемое вторичное хранилище. В ФАСТ , 2009.

[34] К . Дубницкий, Л. Грьв, Л. Хелдт , М. Качмарчик , В. Килиан, П. Стшельчак, Й. Щепковски, К . Унгуряну и М. Велницки. HYDRAStor: масштабируемое вторичное хранилище. В ФАСТ , 2009.

[35] А. Эль-Шими, Р. Калач , А.К . Ади, О.Дж. Ли и С. Сенгупта. Первичная дедупликация данных — крупномасштабное исследование и проектирование системы На ежегодной технической конференции USENIX, 2012.

[36] С. Харизопулос, В. Лян, DJ Абади и С. Мэдден. Компромиссыпроизводительности в базах данных, оптимизированных для чтения . В VLDB, страницы487–498, 2006.

[37] Б. Айер и Д. Уилхайт . Поддержка сжатия данных в базах данных. 1994.

[38] Н. Джейн, М. Далин и Р. Тевари. Taper: многоуровневый подход к устранению избыточности при синхронизации реплик . В FAST, 2005.

[39] E. Leonardi и SS Bhowmick. Xanadue: система обнаружения изменений в XML-данных в реляционных базах данных, не поддерживающих древовидную структуру. В SIGMOD, страницы1137–1140, 2007.

[40] М. Лиллибридж, К . Эшги, Д. Бхагват , В. Деолаликар, G. Trezise и P. Camble. Разреженное индексирование: крупномасштабная встроенная дедупликация с использованием выборки и локальности. В FAST, 2009.

[41] Дж. Лелигер. Контроль версий с помощью git: мощные инструменты и методы для совместной разработки программного обеспечения . 2009.

[42] Дж. П. Макдональд. Поддержка файловой системы для дельта-сжатия . Магистерская диссертация , Калифорнийский университет, Беркли, 2000.

[43] С. Мишра. Сжатие данных: стратегия , планирование емкости и лучшие практики. Техническая статья SQL Server, 2009.

[44] А. Мутитачароен, Б. Чен и Д. Мазьер. А Сетевая файловая система с низкой пропускной способностью В SOSP, 2001.

[45] Р. Паг и Ф. Ф. Родлер. Кукушкино хеширование. Журнал алгоритмов, 51(2):122–144, 2004.

[46] М. Поэсс и Д. Потапов. Сжатие данных в Oracle. В VLDB, страницы937–947, 2003.

[47] Х. Пуча, Д. Г. Андерсен и М. Каминский. Эксплуатация сходство для загрузок из нескольких источников с использованием отпечатков файлов. В NSDI, 2007.

[48] В. Пью Списки пропусков: вероятностная альтернатива сбалансированным деревьям . В Workshop on Algorithms and Data Structures, страницы437–449. Springer, 1989.

[49] С. Куинлан и С. Дорвард. Venti: новый подход к архивному хранению В FAST, 2002.

[50] Рабин М. О. Дактилоскопия случайными полиномами.

[51] В. Раман, Г. Ат талури, Р. Барбер, Н. Ч айнани, Д. К алмук , В. К уланд ай Сами, Д ж. Леенст ра, С. Лайт ст оун, С. Лю, Г. М. Ломан и д р. Db2 с уск орением Blu: гораз до боль ше, ч ем прост о хранилище ст олбц ов. VLDB, 6(11):1080–1091, 2013.

[52] С. Сак р. Мет од ысжат ия XML: обз ор и сравнение. Журнал к омпь ют ерных и сист емных наук , 75(5):303–322, 2009.

[53] П. Шилан, М. Хуан, Г. Уоллес и В. Ской. WAN-опт имизированная реплик ац ия рез ервных набор ов д анных с испол з ованием пот ок овой дель т а-к омпрессии. В FAST, 2012.

[54] П. Шилан, Г. Уоллес, М. Хуан и В. Ской. Дель т а сжат ое и д ед уплиц ированное хранилище с испол з ованием пот ок овой лок аль ност и. USENIX Hot Storage, 2012.

[55] К . Шринивасан, Т . Биссон, Г. Гуд сон и К . Воруг ант и. idedup: Уч ит ывающая з ад ерж ку вст роенная д ед уплик ац ия д анных д ля перв ич ног о хранилища. В FAST, 2012.

[56] М. Ст оунбрейк ер, DJ Абад и, А. Бат к ин, Х. Ч ен, М. Ч ерня к , М. Феррейра, Э. Лау, А. Лин, С. Мэд д ен, Э. О'Нил и д р. C-store: СУБД , ориент ированная на ст олбц ы В VLDB, ст раниц ы553–564, 2005 г.

[57] Т . Суэл и Н. Мемон. Алг орит мыд ель т а-сжат ия и уд аленной синхрониз ац ии файл ов. Справоч ник по сжат июбез пот ерь , 2002.

[58] Д. Т еод осиу, ЮГуревич , М. Манассе и Ж. Порк а. Опт имиз ац ия реплик ац ии файл ов в сет я х с ограничен ной пропуск ной способност ью юс испол з ованием уд аленног о д ифференц иаль ног о сжат ия . Т ехнич ескийпред ст авит ель . MSR-TR-2006-157, Microsoft Research, 2006.

[59] WF Tichy. Rcs – сист ема к онт роля версий. Прог раммное обеспеч ение: прак т ик а и опыт , 15(7):637–654, 1985.

[60] Д. Т рендафилов, Н. Мемон и Т . Суэль . zdelta: Эффек т ивный инст румент дель т а-к омпрессии. Т ехнич ескийот ч ет TR-CIS-2002-02, Полит ехнич ескийуниверсит ет , 2002.

[61] А. Т ридж елл. Эффек т ивные алг орит мысорт ировк и и синхрониз ац ия . В д ок т орск ой д иссерт ац ии, Авст ралийск ий нац иональ ныйуниверсит ет , 2000.

[62] Г. Урд анет а, Г. Пь ер и М. Ван Ст ин. Вик ипед ия Анализ раб оч ей нагруз к и д ля д ец ент рализованног о хост инг а. К омпь ют ерные сет и, 53(11):1830–1845, 2009.

[63] Y. Wang, DJ DeWitt и J.-Y. Cai. X-diff: Эффек т ивный алг орит м обнаруж ения из менений д ля XML-д ок ум ент ов. В ICDE, ст раниц ы519–530, 2003.

[64] В. Ся , Х. Ц з я н, Д. Фэн и ЮХуа. Силос: А Схема д ед уплик ац ии на основе сход ст ва-лок аль ност и с низк ими из держк ами опер ат ивной пам я т и и выс ок ой пропуск ной способност ью На еж ег од ной т ехнич ескойк онф еренц ии USENIX, 2011.

[65] Л. Ской, А. Павло, С. Сенгупа, Д ж. Ли и Г. Р. Гангер. Сок ращение пропуск ной способност и реплик ац ии д ля распред еленных баз д анных д ок ум ент ов. В SoCC, ст раниц ы222–235, 2015.

[66] LL You, KT Pollack и DD Long. Глубок ое хранилище: архит ек т ура архивной сист емых ранения . В ICDE, ст раниц ы804–815, 2005.

[67] Б. Ч жу, К . Ли и Р. Х. Пат т ерсон. Из бег ание д иск а уз к ое мест о в файловой сист еме д ед уплик ац ии д омена д анных. В FAST, 2008.

[68] Д ж. З ив и А. Лемпель . Универсаль ный алг орит м д ля послед оват ель ног о сжат ие д анных. Т руд ыIEEE по т еории информац ии, 23(3):337–343, 1977.

[69] М. Жук овск и, С. Хеман, Н. Нес и П. Бонц . Суперск ая рное сжат ие к эша RAM-CPU. В ICDE, ст раниц ы59–59, 2006.