Онлайн Дедупликация для Базы данных

# АБСТРАКТНЫЙ

Лянхун Сюй Эндрю Павел

Карнеги Меллон Университет Карнеги Меллон Университет

[lianghon@andrew.cmu.edu](mailto:lianghon@andrew.cmu.edu)  [pavlo@cs.cmu.edu](mailto:pavlo@cs.cmu.edu)

Судипта Сенгупта Грегори Р. Гангер

Майкрософт Исследования Карнеги Меллон Университет

[sudipta@microsoft.com](mailto:sudipta@microsoft.com)  [ganger@ece.cmu.edu](mailto:ganger@ece.cmu.edu)

90



Compression from dedup alone Additional compression from Snappy Index memory usage

Index memory usage (MB)

dbDedup — это схема дедупликации, основанная на сходстве данных, для систем управления базами данных (СУБД) в режиме онлайн. Помимо сжатия на уровне блоков из индивидуальный база данных страницы или операция бревно (oplog) сообщения, используемые в современных СУБД, dbDedup использует дельта-данные на уровне байтов кодирование из индивидуальный записи в пределах база данных к добиться большей экономии. Метод однопроходного кодирования dbDedup может быть интегрирован в компоненты хранения и ведения журнала СУБД. обеспечить два преимущества: (1) уменьшение размера данных, хранящихся на диске, за пределами что традиционный сжатие схемы предоставлять, и (2) уменьшенный количество из данные передано над сеть для репликация

80

70

Compression ratio

60

50

40

30

20

10

1КБ 64Б 4КБ

64Б

1200

1000

800

600

400

200

0

услуги. Для оценивать наш работа, мы реализовано dbDedup в а распределенный NoSQL СУБД и проанализировал его характеристики с использованием четыре реальных

dbDedup

trad -dedup

Снаппи

наборы данных. Наши результаты показывать что dbDedup достигает вверх к 37 *×* снижение в хранилище размер и репликация трафик из база данных на свой собственный и вверх к 61 *×* снижение когда в паре с СУБД уровень блока сжатие. dbDedup обеспечивает оба преимущества с незначительное влияние на пропускную способность СУБД или задержку клиента (среднее и конечное).

# ВВЕДЕНИЕ

Темпы роста объемов данных превышают темпы снижения стоимости оборудования. Одним из решений этой проблемы является сжатие базы данных. Для хранения базы данных, помимо экономии места, сжатие помогает уменьшить число из диск Вводы/выводы и улучшать производительность, потому что спросили данные подходит в меньше страниц. Для распределенный базы данных реплицируются в разных географических регионах, также существует острая необходимость в сокращении объема передачи данных, используемого для синхронизации реплик.

The большинство широко использовал подход для данные снижение в Операционные СУБД — это сжатие на уровне блоков [[30,](#_bookmark73) [37,](#_bookmark81) [46,](#_bookmark90) [43,](#_bookmark87) [3,](#_bookmark47) [16].](#_bookmark60) Такие СУБД используются для поддержки пользовательских приложений, которые выполняют простые запросы для извлечения небольшого количества записей за раз (в отличие от выполнения сложных запросов, сканирующих большие сегменты базы данных). Хотя сжатие на уровне блоков является простым и эффективным, оно не решает проблему избыточности между блоками и, следовательно , листья значительный комната для улучшение для много приложений (например, из-за версионирования на уровне приложений в вики или частичного копирования записей на досках объявлений). Дедупликация (дедупликация) стала популярной в резервное копирование системы для устранение дубликат содержание через

Разрешение на создание цифровых или печатных копий всей или части этой работы для личного или класс использовать является предоставленный без платеж предоставил что копии являются нет сделал или распределенный для получения прибыли или коммерческой выгоды и что копии содержат это уведомление и полную цитату на первой странице. Авторские права на компоненты этой работы принадлежат третьим лицам. АКМ должен быть почтенный. Абстрагирование с кредит является разрешено. копия в противном случае, или повторно публиковать, к почта на серверы или к перераспределять к списки, требует прежний специфический разрешение и/или плата. Запросите разрешения на [permissions@acm.org.](mailto:permissions@acm.org)

*СИГМОД'17, Может 14-19, 2017, Роли, NC, США*

© 2017 АКМ. ISBN 978-1-4503-4197-4/17/05. . . 15,00 $

ДОИ: <http://dx.doi.org/10.1145/3035918.3035938>

**Рисунок 1:** Коэффициент сжатия и использование индексной памяти для данных Википедии хранится в пяти конфигурациях MongoDB: с dbDedup (размер фрагмента 1 КБ) и 64 Б), с традиционный дедупликация (4 КБ и 64 Б), и с Снаппи (блокировать- уровень сжатие). dbDedup обеспечивает выше сжатие соотношение и ниже индекс память накладные расходы чем традиционный дедупликация. Снаппи обеспечивает такой же

1,6 *×* сжатие для пост-дедупликация данные или исходные данные.

целый корпус данных, часто достигая гораздо более высоких степеней сжатия. Резервный поток делится на фрагменты, и в качестве идентификатора каждого фрагмента используется устойчивый к коллизиям хэш (например, SHA-1). Система дедупликации ведет глобальный индекс всех хэшей и использует его для обнаружения дубликатов. Дедупликация хорошо работает как для основных, так и для резервных наборов данных, состоящих из больших файлов, которые редко изменяются (а если и изменяются, то изменения незначительны).

К сожалению, традиционные схемы дедупликации на основе фрагментов данных не подходят для операционных СУБД, где приложения выполняют запросы на обновление, которые изменяют отдельные записи. Количество дублирующихся данных в а индивидуальный записывать является вероятный незначительно. Но большой кусок размеры (например, 4–8 КБ) являются норма к избегать огромный в памяти индексы и большое количество чтений с диска.

Этот бумага представляет **dbDedup** , а легкий схема для онлайновые системы баз данных, которые используют *дедупликацию на основе сходства* [[65]](#_bookmark109) для сжатия индивидуальный записи. Вместо из индексация каждый кусок хэш, dbDedup выбирает небольшое подмножество хэшей фрагментов для каждой новой записи базы данных, а затем использует этот образец для идентификации похожей записи в база данных. Это затем использует байтовый уровень дельта сжатие на​ два записи к уменьшать оба онлайн хранилище использовал и удаленный пропускная способность репликации . dbDedup обеспечивает более высокую степень сжатия при меньших затратах памяти, чем дедупликация на основе фрагментов, и хорошо сочетается со сжатием на уровне блоков, как показано на рис. [1.](#_bookmark0)

Мы представлять и объединить несколько методы к достигать этот эффективность. Прежде всего, мы представляем новое *двустороннее кодирование* эффективно передавать закодированные новые записи (прямое кодирование) в удаленные реплики, сохраняя при этом новые записи с закодированными формами выбранных источник записи (назад) кодировка). Как а результат, нет де-

код является необходимый для общий случай из доступ большинство последняя запись в цепочке кодирования (например, последняя версия Википедии). К избегать производительность накладные расходы от обновление источник записи, мы также представляем *дельта-кэш с обратной записью с потерями,* настроенный на максимальную степень сжатия и избегающий конфликтов ввода-вывода. Наш подход также использует а новый техника, называется *прыгать кодирование* , что минимизирует наихудшее число шагов декодирования, необходимое для доступа к определенной записи в а длинный кодирование цепь. Наконец, мы описывать как к адаптивно отключайте дедупликацию для баз данных и записей, где ожидается небольшая экономия.

К оценивать наш подход, мы реализовано dbDedup в СУБД MongoDB [ [5]](#_bookmark49) и измерили ее эффективность с использованием четырех реальных наборов данных. Наши результаты показывают, что он обеспечивает снижение до 37 *×* (61 *×* когда комбинированный с блочный уровень сжатие) в хранилище размер и трафик репликации. dbDedup превосходит дедупликацию на основе фрагментов, оказывая при этом незначительное влияние на производительность СУБД.

Этот бумага делает следующий взносы.

1. К лучший из наш знания, мы подарок первый дедупликация система​ для оперативный СУБД что уменьшает оба база данных использование полосы пропускания при хранении и репликации. Это также первая система дедупликации хранилища баз данных, использующая дедупликацию на основе сходства.
2. Мы внедряем новые методы, которые имеют решающее значение для достижения приемлемого дедупликация эффективность, включение практичный использовать для онлайн-хранилище баз данных.
3. Мы оцениваем полную реализацию системы в распределенной СУБД NoSQL, используя четыре реальных набора данных.

Остальная часть статьи организована следующим образом. Раздел [2](#_bookmark1) мотивирует использовать из основанный на сходстве дедупликация для база данных приложения и классифицирует наш подход родственник к другой дедупликация системы. Раздел [3](#_bookmark4) описывает dbDedup's дедупликация рабочий процесс и механизмы. Раздел [4](#_bookmark20) Подробности реализации dbDedup, включая его интеграцию в хранилище и репликация фреймворки из а СУБД. Мы затем оцените наши подход с использованием несколько реальный мир данные наборы в Раздел [5.](#_bookmark26) Наконец, в разделе [6 мы завершаем](#_bookmark44) обсуждение соответствующей работы.

# ФОН И МОТИВАЦИЯ

Дедупликация заключается в выявлении и удалении дублирующегося контента в массиве данных. В этом разделе обосновывается его потенциальная ценность. в СУБД, объясняет два начальный категории (точный подходов к дедупликации на основе соответствия и сходства и почему дедупликация на основе сходства лучше подходит для дедупликации в СУБД, а также помещает dbDedup в контекст путем категоризации предыдущих систем дедупликации.

## Почему Дедупликация для База данных Приложения?

The большинство общий способ что оперативный СУБД уменьшать Размер хранилища данных достигается путем сжатия на уровне блоков на отдельных страницах базы данных. Например, InnoDB от MySQL может сжимать страницы когда они являются выселенный от память и написано к диск [[3].](#_bookmark47) Когда эти страницы возвращаются в память, система может сохранить страницы сжатый как длинный как нет запрос пытается к читать его содержание. Поскольку объем из сжатие алгоритм является только а на одной странице, степень сокращения, которую может обеспечить система, невелика.

Аналитические СУБД используют более агрессивные схемы (например, сжатие словаря, кодирование длин серий), которые значительно сокращают размер из а база данных [[18]](#_bookmark62) .Это является потому что эти системы сжимать индивидуально колонны, и таким образом там является выше вероятность из дублирующиеся данные. И в отличие от в выше MySQL например, они также поддержка обработки запросов непосредственно на сжатых данных.

Этот тип сжатия непрактичен в операционной СУБД. Эти системы предназначены для высококонкурентных рабочих нагрузок, которые выполняют запросы, извлекающие небольшое количество записей в а время. Если СУБД имел к компресс каждый атрибут каждый время

а новый записывать был вставлено, затем они бы быть слишком медленный к поддержка онлайн-приложений на базе веб-технологий.

Однако мы видим, что многие приложения баз данных могли бы выиграть от дедупликации из-за сходства между несовмещенными записями. чей отношение является нет известный к лежащий в основе 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 изучена . в оперативный базы данных. The начальный причина является что Записи базы данных обычно невелики 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 контролем валют (MVCC) поддерживают исторические версии для поддержки параллельных транзакций, они обычно очищают старые версии, как только они перестают быть видимыми для какой-либо активной транзакции. Как результат, немного приложения брать преимущество из версионирование поддерживать предоставил к СУБД к выполнять «путешествие во времени» запросы». Вместо, большинство приложений осуществлять версионирование на их собственный когда необходимый. Общей особенностью этих приложений является то, что различные версии одного элемента данных записываются в СУБД как совершенно не связанные между собой записи, что приводит к значительной избыточности, которая не фиксируется. путем простого сжатия страницы. Примеры таких приложений включают в себя веб-сайты питание к Вордпресс, который включают в себя 25% из весь Интернет [[12],](#_bookmark56) а также совместные вики-платформы, такие как Wikipedia [[14]](#_bookmark58) и Baidu Baike [[1].](#_bookmark45)

Другим источником дублирования в приложениях баз данных являются отношения включения между записями. Например, ответ по электронной почте или пересылка обычно включает в себя содержание из предыдущий сообщение в теле сообщения. Другим примером являются онлайн-доски объявлений, где пользователи часто цитировать каждый чужой комментарии в их посты. Как и версионирование, этот копирование является а артефакт из приложение что не может быть легко раскрыта для базовой СУБД. В результате, эффективная избыточность удаление также требует а дедупликация техника что выявляет и устраняет избыточность во всем массиве данных.

Это является важный к примечание что там являются также много база данных приложения что бы нет выгода от дедупликация. Для пример, некоторый делать не имеют достаточной избыточности, и, таким образом, накладные расходы на поиск возможности к удалять избыточный данные является нет ценность ИТ. Типичные примеры включают большинство рабочих нагрузок OLTP, где много записей вписываются в один база данных страница и большинство увольнения среди поля может быть устранены с помощью схем сжатия на уровне блоков. Для приложений, которым это не выгодно, dbDedup автоматически отключает функции дедупликации, чтобы снизить их влияние на производительность системы.

## Основанный на сходстве Дедупликация против. Точный Дедупликация

Подходы к дедупликации можно разделить на две категории. первый и большинство общий ("точный дедупликация") выглядит для точные совпадения по единице дедупликации (например, фрагменту) [[67,](#_bookmark110) [40,](#_bookmark84) [27,](#_bookmark71) [34,](#_bookmark78) [35].](#_bookmark79) The второй («основанный на сходстве дедупликация") выглядит для похожий единиц (кусков или файлов) и применяет к ним дельта-сжатие [[61,](#_bookmark105) [53,](#_bookmark97) [22].](#_bookmark66) Для те база данных приложения что делать выгода от дедупликация, мы находим что основанный на сходстве дедупликация превосходит на основе кусков дедупликация с точки зрения степени сжатия и использования памяти, хотя это может включать дополнительный Ввод/вывод и вычисление накладные расходы. Это раздел кратко описывает на основе кусков дедупликация, почему это делает нет работа хорошо для СУБД и почему основанный на сходстве дедупликация Раздел [3](#_bookmark4) подробности Рабочий процесс dbDedup и его методы для смягчающий потенциал накладные расходы. Традиционная схема дедупликации файлов, основанная на точном совпадении фрагментов данных («дедупликация на основе фрагментов») [[44,](#_bookmark88) [49,](#_bookmark93) [67]](#_bookmark110) работает следующим образом. Входящий файл (соответствующий к а новый записывать в контекст из

Кусок Граница

Входящий записывать

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  | |  | |  | | |  |  | | |
| 1 | | 2 | | 3 | | | 4 | 5 | | |

Оригинал записывать

 Измененный Область

 Дубликат кусок ИДЕНТИФИКАТОР  Кодирование метаданные



Chunk-based deduplication

Delta compression

Output

**Фигура 2:** Сравнение между на основе кусков дедупликация и сходство-

дедупликация на основе использования дельта-сжатия для типичных рабочих нагрузок базы данных с небольшими и разбросанными изменениями.

СУБД) является первый разделенный в куски с использованием Рабин-дактилоскопия [[50];](#_bookmark94) Хэши Рабина вычисляются для каждого скользящего окна в потоке данных, и граница фрагмента объявляется, если младшие биты хэша ценить соответствовать а предопределенный шаблон. The средний кусок размер может быть контролируемый к число из биты использовал в шаблон. Как правило, шаблон соответствия из *n* бит приводит к среднему размеру фрагмента 2 *n* B. Для каждого фрагмента система вычисляет уникальный идентификатор, используя устойчивый к коллизиям алгоритм. хэш (например, SHA-1). Это затем чеки а глобальный индекс для видеть ли это имеет видимый этот хэш до. Если а соответствовать является найденный, затем фрагмент объявляется дубликатом. В противном случае кусок считается уникальный и является добавлен к индекс и лежащий в основе данные store. Хотя дедупликация на основе фрагментов обычно хорошо подходит для рабочих нагрузок резервного копирования, она редко подходит для рабочих нагрузок базы данных. Из нашего наблюдения, дубликат регионы для база данных рабочие нагрузки являются обычно небольшие (порядка десятков-сотней байт) и разбросанные по всему а записывать. В этот маленький размер, на основе куска дедупликация с а типичный кусок размер на заказ из КБ является не могущий к идентифицировать много дубликат кусками. Сокращение кусок размер к соответствовать вверх с длина дубликации может улучшать системы сжатие соотношение, но отслеживание фрагментов индекс становится чрезмерно большой и отрицает любой перфо-

манс преимущества приобрел к Ввод/вывод снижение.

Напротив, дедупликация dbDedup на основе сходства идентифицирует одну похожую запись из корпуса базы данных и выполняет дельта-сжатие. между новый записывать и похожий один. Как показано На рис. [2](#_bookmark2) дельта-сжатие dbDedup на уровне байтов позволяет идентифицировать гораздо более мелкие дубликаты и, таким образом, обеспечивает более высокую степень сжатия, чем дедупликация на основе фрагментов.

## Категоризация Дедупликация Системы

Таблица [1](#_bookmark3) иллюстрирует один из взглядов на то, как dbDedup соотносится с другими системами, использующими дедупликацию, на основе двух осей: подход дедупликации (точное совпадение или основанное на сходстве) и цель дедупликации (первичное хранилище или вторичные/резервные данные). Насколько нам известно, dbDedup — это первый инструмент, основанный на сходстве. дедупликация система для начальный данные хранилище, как хорошо как первая система дедупликации для онлайновых СУБД, охватывающая как первичное хранилище, так и вторичные данные (oplog).

Много предыдущей работы по дедупликации данных [[67,](#_bookmark110) [40,](#_bookmark84) [20,](#_bookmark64) [53,](#_bookmark97) [54]](#_bookmark98) было сделано в контекст из резервное копирование данные (как выступил против к начальный хранение) где дедупликация делает нет нуждаться к держать вверх с начальный данные прием и его не нужно запускать на основном (обслуживающем данные) узле. Более того, такие резервные рабочие нагрузки часто выполняются на устройствах с премиальным оборудованием. dbDedup, работающий в контексте операционных СУБД, должен работать на основных узлах обслуживания данных на общедоступном оборудовании и экономно использовать ресурсы ЦП, памяти и ввода-вывода.

Там был недавний интерес к начальный дедупликация данных на первичный (обслуживание данных) сервер но решения являются по большей части в уровне хранения (а не на уровне управления данными, как в нашей работе). такой системы, в зависимости на выполнение, дедупликация может происходят либо в строке с новыми данными (Sun ZFS [[17],](#_bookmark61) Linux SDFS [[4],](#_bookmark48) iDedup [[55])](#_bookmark99) или в фон как постобработка на хранится

**Стол 1:** Категоризация из связанный работа

|  |  |
| --- | --- |
|  | **Exact Dedup Similarity-based Dedup** |
| Primary | iDedup [[55]](#_bookmark99)  ZFS [[17]](#_bookmark61)  SDFS [[4]](#_bookmark48)  Windows server 2012 [[15]](#_bookmark59) ***dbDedup***  NetApp ASIS [[19]](#_bookmark63)  Ocarina [[7]](#_bookmark51)  Permabit [[8]](#_bookmark52) |
| Secondary | DDFS [[67]](#_bookmark110) Extreme binning [[22]](#_bookmark66)  Venti [[49]](#_bookmark93) Sparse Indexing [[40]](#_bookmark84)  ChunkStash [[31]](#_bookmark75) Silo [[64]](#_bookmark108) SDS [[20]](#_bookmark64)  DEDE [[27]](#_bookmark71) SIDC [[53]](#_bookmark97) sDedup [[65]](#_bookmark109)  HydraStor [[33]](#_bookmark77) DeepStore [[66]](#_bookmark111) |

данных (Windows Server 2012 [[35])](#_bookmark79) или предоставить оба варианта (NetApp [[19],](#_bookmark63) Ocarina [[7],](#_bookmark51) Permabit [[8]).](#_bookmark52)

Системы в нижнем среднем столбце используют комбинацию точных и основанных на подобии методов дедупликации с различной степенью детализации, но по сути являются системами дедупликации на основе фрагментов, поскольку они хранят хэши для каждый кусок. К лучший из наш знания, dbD-edup — это первая система дедупликации на основе сходства для первичного хранения рабочие нагрузки что достигает данные снижение на хранилище и пропускная способность сети требование в такой же время. Этот является потому что дельта на уровне байтов сжатие является традиционно обдуманный дорогой для онлайн-базы данных из-за дополнительных затрат на ввод-вывод и вычисления по сравнению со сравнениями хэшей. В результате предыдущие системы либо полностью избегать это или использовать это когда диск Ввод/вывод является нет а главный беспокойство. Например, SIDC [[53]](#_bookmark97) и sDedup [[65]](#_bookmark109) используют дельта-сжатие для сетевой дедупликации потоков репликации; SDS [[20]](#_bookmark64) применяет дельта-сжатие к большим 16 МБ фрагментам в потоках резервного копирования, извлеченных к последовательный диск читает. Пока dbDedup занимает преимущество дельта сжатие к достигать начальство сжатие соотношение, это использует число из методы к уменьшать накладные расходы вовлеченный, изготовление это практичный механизм дедупликации для онлайновых СУБД.

## dbDedup ДИЗАЙН

В этом разделе описывается рабочий процесс дедупликации dbDedup, методы кодирования, механизмы снижения накладных расходов ввода-вывода и подходы к предотвращению напрасной траты усилий на действия по дедупликации, не приносящие никакой пользы.

## Дедупликация Рабочий процесс

dbDedup использует основанный на сходстве дедупликация к достигать хороший компрессия соотношение и низкий память использование одновременно. Рис. [3](#_bookmark6) показывает дедупликация кодировать рабочий процесс использовал когда подготовка обновлено записывать данные для локального хранения и удаленной репликации. Во время запросов на вставку или обновление новые записи записываются в локальный oplog, а dbDedup кодирует их в фоновом режиме, вне критического пути. Четыре ключевых шага: (1) извлечение признаков сходства из новой записи, (2) поиск в дедупликация индекс к находить а список из кандидат похожие записи в корпусе базы данных, (3) выбор одной лучшей записи из кандидаты и (4) выполнение дельта сжатие между новый​ и похожий записывать к вычислить закодированный формы для местный синхронизация хранения и реплик.

### Особенность Извлечение

Как а первый шаг в находка похожий записи в база данных, dbDedup извлекает признаки сходства из новой записи, используя зависящий от содержимого подход. dbDedup делит новый записывать в несколько фрагментов данных переменного размера с использованием алгоритма снятия отпечатков пальцев Рабина [[50]](#_bookmark94) что является широко использовал в много на основе кусков дедупликация системы. В отличие от этих систем, которые индексируют устойчивый к коллизиям хэш (например, SHA-1) для каждого уникального фрагмента, dbDedup вычисляет (слабее, но вычислительно дешевле) MurmurHash [[6]](#_bookmark50) для каждый кусок и



① *Feature extraction*

Sketch (top-K features)

⑤ *Index lookup*

Candidate similar records

➂ *Source selection*

Source record

➃ *Delta*

Δ *compression*

(From local oplog)

Source record cache

Lossy delta cache

*Database Remote*

*update replication*

Feature

index Database

Encoded record

New record

**Фигура 3: dbDedup Рабочий процесс** – (1) Особенность Извлечение, (2) Индекс Искать, (3) Источник Выбор, и (4) Дельта Сжатие.

только индексы а представитель подмножество из кусок хэши. dbDedup адаптирует технику, называемую *последовательной выборкой* [[47]](#_bookmark91) , для выбора репрезентативных кусок хэши, который обеспечивает лучше сходство характеристика чем случайный отбор проб. Это сортирует хэш ценности в а последовательно (например, по величине от большего к меньшему) и выбирает верхние [K1](#_bookmark7) *хэши* в качестве *эскиза подобия* для записи. Каждый фрагмент хеша в эскизе называется *признаком* — если две записи имеют один или несколько общих признаков, они считаются похожими.

Индексируя только хэши выбранных фрагментов, dbDedup ограничивает память накладные расходы из его дедупликация индекс к быть в большинство Индекс *К* записей на запись. Это важное свойство позволяет dbDedup использовать небольшие кусок размеры для лучше сходство обнаружение пока нет потребляющий излишний БАРАН нравиться в на основе кусков дедупликация. Более того, потому что dbDedup делает нет полагаться на точный соответствовать из кусок хэши для дедупликация, это является более терпимый из хэш столкновения. Это является почему это может используйте алгоритм MurmurHash вместо SHA-1, чтобы сократить накладные расходы при расчете хэша фрагмента. Хотя это может привести к а небольшой болезнь в сжатие ставка должный к более ЛОЖЬ положительные моменты: использование более слабого хеша не влияет на корректность, поскольку dbDedup выполняет дельта-сжатие на последнем этапе.

### Индекс Искать

Для каждого извлеченного признака dbDedup находит существующие записи, которые разделяют этот признак с новой записью. Поскольку dbDedup — это онлайн- дедупликация система, это является императив что этот индекс искать процесс быстрый и эффективный. dbDedup достигает этот к здание а функция в памяти индекс что использует а вариант из Кукушка хеширование [[45,](#_bookmark89) [31]](#_bookmark75) к особенности карты к записи. Это подход использует несколько хеширование функции, которые сопоставляют ключ с несколькими слотами-кандидатами, что увеличивает коэффициент загрузки таблицы, ограничивая время поиска константой. В особенности индекс, каждый вход является состоял из из а 2-байтовый ключ что является а компактный контрольная сумма из особенность и а 4-байтовый ценить что является а указатель к месту расположения соответствующей записи в базе данных.

На особенность искать, dbDedup первый вычисляет а хэш из значение признака с использованием одной из функций хеширования Cuckoo, которая сопоставляется с потенциальным слотом, содержащим несколько записей индекса (корзин). Затем он повторяет над ведра, сравнивает их контрольные суммы с данная функция, и добавляет любой совпало записи к список из похожий записи. Это процесс повторяется с другой хеширование функции до это находит пустой ведро указывающий конец из поиск.dbDedup затем вставляет​ особенность и а ссылка к новый записывать к пустой ведро для будущего искать. Окончательно, dbDedup комбайны искать результаты для все топ- *K* особенности и генерирует а список из существующий похожий записи как ввод для следующего шага. Для дальнейшего снижения использования ЦП и памяти dbDedup ограничивает максимальное количество похожих записей, которые проверяются для каждый особенность. Однажды порог является достиг, процесс поиска заканчивается и вход содержащий наименее недавно использовавшийся

1. Находим *К* = 8 ударов разумный компромисс между степенью сжатия и использование памяти, и мы используем его как значение по умолчанию для всех экспериментов если не указано иное.

(ЛРУ) записывать является выселенный от особенность индекс.

### Источник Выбор

The индекс искать результаты может содержать несколько кандидат похожие записи, но dbDedup выбирает только одну из них для дельта-сжатия новый записывать в заказ к минимизировать накладные расходы вовлеченный. В то время как большинство предыдущий сходство выбор алгоритмы делать такой решения чисто основанный на на сходство метрики из входы, dbDedup добавляет рассмотрение производительности системы, отдавая предпочтение записям-кандидатам, которые присутствуют в кэше исходных записей (см. раздел [3.3).](#_bookmark14) Мы называем этот метод выбора выбором *с учетом кэша* . Конкретно, dbDedup первый присваивает а исходный счет для каждый кандидат похож на запись на основе количества общих признаков с новый запись. Затем, dbDedup увеличивается что оценка по вознаграждению, если запись кандидата уже находится в кэше. Кандидат с наивысшим баллом затем выбирается в качестве входных данных для дельта-сжатия. В то время как выбор с учетом кэша может в конечном итоге привести к выбору а записывать что является неоптимальный в условия из сходство, мы найди это в значительной степени уменьшает Ввод/вывод накладные расходы к принести источник записи от база данных. Мы оцениваем эффективность выбора с учетом кэша и его чувствительность к оценке вознаграждения в разделе [5.4.](#_bookmark39)

### Дельта Сжатие

The последний шаг в dbDedup рабочий процесс является к выполнять дельта сжатие между новой записью и выбранной похожей записью. Подробности методов кодирования описаны в разделе [3.2](#_bookmark9) , а алгоритмы сжатия — в разделе [4.2.](#_bookmark24)

## Кодирование для Онлайн Хранилище

Эффективный доступ из дельта-кодированный хранилище является а давний испытание должный к Ввод/вывод и вычисление накладные расходы вовлеченный в этапы кодирования и декодирования. В частности, восстановление закодированных данных может требовать чтение все дельты вдоль а длинный кодирование цепочке до тех пор, пока не будет достигнут незакодированный (сырой) элемент данных. Чтобы обеспечить разумные гарантии производительности, большинство онлайн-систем используют дельта-кодирование. только к уменьшать сеть передача инфекции (уход хранилище некодированный) или использовать это к а очень ограниченный степень в хранилище компонентов (например, путем ограничения максимальной длины цепочки кодирования небольшим значением). Однако такой подход в значительной степени не позволяет использовать потенциальную экономию пространства, которую можно было бы достичь.

dbDedup значительно облегчает болезненный компромисс между сжатием прибыль и доступ скорость в дельта закодированный хранилище с два новых кодирование схемы. Это использует а *двусторонний кодирование* техника что сокращает как пропускную способность удаленной репликации, так и объем хранилища базы данных, одновременно оптимизируя запросы общего характера. Кроме того, он использует *хмель кодирование* к уменьшать худший случай источник извлечения для чтение закодированные записи, при этом в значительной степени сохраняя преимущества сжатия.

### Двусторонний Кодирование

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

Р 0 Р 1

**Write order**

Р 2

Р 0

Р 1 Р 2



R0

Δ1,0

R1

Δ2,0

R2



R0

Δ0,1

Δ0,2

**Write order**

Вперед кодирование в обратном направлении кодирование

* + - 1. Вперед кодирование



R0

Δ0,1

Δ1,2



R0

Δ1,0

R1

Δ2,1

R2

* + - 1. Назад кодирование

**Фигура 5: Перекрывающиеся кодирование** – Назад кодирование может вести к ком-

**Фигура 4: Иллюстрация из двусторонний кодирование** – dbDedup использует вперед ru- кодирование для уменьшения пропускной способности сети для синхронизации реплик, в то время как использование обратного кодирования для сжатия хранилища базы данных.

новую запись в двух направлениях, используя технику, которую мы называем *двусторонним кодированием* . Для сетевой передачи dbDedup выполняет *пересылку кодирование* (Инжир. [4а),](#_bookmark10) который использует старше (т.е., выбрано

давление потеря когда а старше записывать является выбрано как источник. В этот пример, когда *R* 0 — это выбрано как источник для *Р* 2 , назад кодирование листья *Р* 1 и *R* 2 оба некодированы.

Хранение #Худший случай

извлечения использования #Writebacks

**Назад кодирование** *S б* + ( *Н −* 1 ) *· С д  Н Н*

**Версия прыжки** *N · С б* + ( *Н − Н* ) *· С д  Ч Н − Н*

*Ч Ч Ч*

(*H−*1)

кандидат) записывать как источник и новый записывать как цель. После кодирование, источник останки в его оригинальный форма, в то время как цель является закодированный как а ссылка к источник плюс дельта от источник к цель. dbDedup посылает закодированный данные, вместо из оригинала новой записи, к удаленным репликам. Использование прямого кодирования для сетевой уровень дедупликация является а естественный дизайн выбор, поскольку это позволяет репликам легко декодировать целевую запись, используя локально сохраненную исходную запись.

dbDedup может просто использовать ту же закодированную форму для локальной базы данных хранилище. Делает так, однако, бы вести к значительный снижение производительности для запросов на чтение к последней записи в кодировке цепь, который мы наблюдать к быть общий случай с управление версиями и включениями на уровне приложений . Поскольку все промежуточные записи в прямой цепочке хранятся в закодированной форме с использованием предыдущей в качестве источника, декодирование последней записи требует извлечения всех дельт по цепочке, вплоть до первой записи, которая хранится в незакодированном виде.

Вместо этого dbDedup использует *обратное кодирование* (рис. [4б)](#_bookmark12) для локального хранилища, чтобы оптимизировать запросы на чтение последних записей. То есть, для локального хранилища dbDedup выполняет дельта-сжатие в обратном временном заказ, используя новый записывать как источник и аналогичный послужной список кандидата, как у цели. В результате самая последняя запись в цепочке кодирования всегда сохраняется незакодированной. Читайте запросы к последний версия таким образом нести нет декодирование накладные расходы в все. Хотя обратное кодирование оптимизировано для чтения, оно создает две потенциальные проблемы. Во-первых, это увеличивает количество операций записи, поскольку старую запись, выбранную в качестве источника, необходимо обновить до закодированной формы. Чтобы смягчить усиление записи, dbDedup кэширует обратно закодированные записи для записи обратно в базу данных и задерживает обновления до система Ввод/вывод является относительно праздный, который мы обсудим более подробно в разделе [3.3.](#_bookmark14) Вторая проблема возникает, когда в качестве источника выбирается более старая запись. Существующие данные (дельта от его текущий база записывать) является заменен к дельта от новый рекорд. С тех пор назад кодирование понимает космос сбережения к обновление дельта-источников, например, *перекрывающееся кодирование* (рис. [5)](#_bookmark11) на одном и том же источнике записи может вести к некоторый сжатие потеря. Вперед кодирование, в контраст, естественно избегает этот проблема с нет обратная запись требуется. К счастью, мы находить перекрывающиеся кодирование является нет распространено в реальный мир приложения — большинство ( *>* 95%) обновления являются инкрементный на основе последней версии (см. раздел [5.2).](#_bookmark28)

dbDedup выполняет дельта-кодирование между новыми записями и записями-кандидатами в двух направлениях, однако при этом возникают накладные расходы на вычисления только одного прохода кодирования. Это достигается путем первоначальной генерации данных с прямым кодированием, а затем их эффективного преобразования в данные с обратным кодированием. дельта в память скорость. Мы вызов этот процесс *перекодировать* и подробно описать алгоритм в разделе [4.2.](#_bookmark24)

### Прыгать Кодирование

**Прыгать кодирование** *С б* + ( *Н −* 1 ) *· С д ЧАС* + лог *Н Н Н* + *Н · ЧАС* 2

**Таблица 2: Краткое изложение различных схем кодирования** – Кодирование прыжка в значительной степени устраняет болезненный компромисс между космос сбережения и декодирование скорость. *Н* является длина из кодирование цепь, и *ЧАС* обозначает прыгать расстояние (размер кластера для перехода между версиями). *С б* и *С д* ссылаться на размер базы запись и дельта соответственно, где *S b С д* в большинстве случаев. Эти размеры очевидно, различаются для разных записей. Здесь мы используем общее обозначение для простота рассуждения.

Как обсудили выше, с использованием назад кодирование минимизирует декодирование​ накладные расходы для чтение недавний записи, но это может все еще нести излишний​ источник извлечение время для случайный запросы к старше записи (например, а специфический версия из а Википедия статья). Предыдущая работа на дельта-кодированное хранилище [[26,](#_bookmark70) [42]](#_bookmark86) использовал технику, называемую *переходом между версиями* , чтобы справиться с этой проблемой, ограничивая наихудшее количество извлечений исходных данных за счет снижения эффективности сжатия. Идея является к разделять кодирование цепь в фиксированный размер кластеры, где последняя запись в каждом кластере, называемая *эталонной версией* , хранится в своей исходной форме, а остальные записи хранятся как обратно закодированные дельты. Это ограничивает время поиска в худшем случае. к размеру кластера, но приводит к более низкой степени сжатия, поскольку эталонные версии не сжимаются. Как кодирующий кластер размер уменьшается, сжатие потеря может увеличивать существенно, поскольку дельты обычно намного меньше базовых записей.

dbDedup использует новую технику, которую мы называем *кодированием переходов* , которая консервы сжатие соотношение закрывать к стандартный обратное кодирование, при этом достигая сопоставимого времени извлечения в худшем случае с версия прыжки подход. Как иллюстрированный в Инжир. [6,](#_bookmark15) дополнительный дельты есть вычисленный между особый записи и другие некоторый расстояние назад в цепочке, подобно спискам пропусков [[48].](#_bookmark92) Мы называем эти записи *базами прыжков* , а минимальный интервал между ними — *расстоянием прыжка* , обозначаемым как *H.* Кодирование перехода использует несколько уровней косвенности для ускорения процесса декодирования, с интервалом уровень *Л* существование *H L.* Декодирование а записывать включает в себя первый трассировка обратно к ближайшей базе перехода за логарифмическое время, а затем следовать цепочке кодирования, начиная с нее.

В таблице [2](#_bookmark13) приведены компромиссы между тремя методами кодирования с точки зрения использования хранилища, наихудшего числа извлечений и дополнительный число из обратные письма. Для прыгать кодирование, количество худший случай источник извлечения является закрывать к что из версия прыжки

( *Н* ). Но поскольку базы переходов хранятся в закодированной форме, достигаемая степень сжатия намного выше, чем при переходе между версиями, и сопоставима со стандартным обратным кодированием. Все три кодировки схемы нести некоторый количество из писать усиление, но разница становится незначительный как прыгать расстояние увеличивается. Мы представьте более подробное сравнение в разделе [5.](#_bookmark26)

## Кэширование для Дельта-кодированный Хранилище

Дельта закодированный хранилище, должный к его «прикованный» свойство, заслуги спец-

Версия прыжки …

Δ2,1

Δ3,2

Δ4,3

R4

Δ6,5

Δ7,6

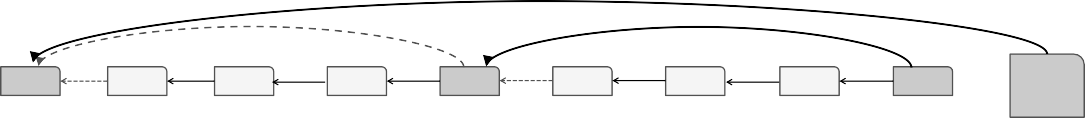
Δ8,7

R8

R0

R16

Прыгать кодирование



Δ16,0

Δ2,1

Δ3,2

Δ4,3

Δ8,4

Δ6,5

Δ7,6

Δ8,7

Δ12,8

…

R

16

**Рисунок 6: Кодирование скачков** – сравнение кодирования скачков и перехода между версиями с цепочкой кодирования из 17 записей. Затененные записи ( *R* 0 , *R* 4 и т. д.) – это скачки базы (эталонные версии) с расстоянием прыжка (размером кластера) 4. Кодирование скачков обеспечивает сопоставимую скорость декодирования с переходом между версиями, достигая при этом Степень сжатия близка к стандартному обратному кодированию.

специализированные механизмы кэширования. Используя это свойство, dbDedup кэширует только несколько ключевых узлов в заданной цепочке кодирования, максимизируя память эффективность пока устранение большинство Ввод/вывод накладные расходы для доступ к закодированным записям. Он использует два специализированных кэша: исходную запись кэш что уменьшает число из база данных читает в течение en-кодировать и а с потерями обратная запись дельта кэш что смягчает писать усиление, вызванное обратным кодированием.

### Источник Записывать Кэш

Основная проблема в dbDedup, как и в других системах с дельта-кодированием, является Ввод/вывод накладные расходы к забрать база данные от диск в качестве входных данных для дельта-сжатия. В частности, чтение выбранного похожего записывать может вовлекать а дополнительный диск доступ, спорящий с обработка клиентских запросов и другие действия с базой данных.

dbDedup использует небольшой, но эффективный кэш записей, чтобы избежать большинства операций чтения с диска для исходных записей. Конструкция кэша записей использует высокую степень временной локальности при обновлении записей рабочих нагрузок, которые хорошо поддаются дедупликации. Например, обновления статьи в Википедии, сообщения на форуме по определенной теме или обмен электронными письмами в том же духе. нить обычно происходить в пределах а короткий время кадр. Итак, Вероятность нахождения недавней похожей записи в кэше высока, даже при а относительно маленький кэш размер. Другой ключ наблюдение является что обновления обычно являются инкрементными (основанными на непосредственном предыдущем обновлять), значение что два записи иметь тенденцию к быть более похожий если они ближе по времени создания.

Основанный на на наблюдения выше, источник записывать кэш сохраняет последнюю запись цепочки кодирования в кэше. Для ускорения движения назад кодирование из прыгать базы, dbDedup кроме того кэши последний прыгать базы в каждый прыгать уровень [2](#_bookmark16) Когда а новый записывать приходит, если dbDedup идентифицирует похожую запись в кэше (что является нормальным случаем из-за метода выбора с учетом кэша, описанного в Раздел [3.1),](#_bookmark5) это заменяет существующий записывать с новый 1. Если новая запись является базой перехода, dbDedup заменяет ее смежные базы соответствующим образом. Если аналогичный источник не найден, dbDedup просто добавляет новую запись в кэш и удаляет самую старую запись методом LRU, если кэш заполняется.

### Потерянный Обратная запись Дельта Кэш

Как обсуждалось в разделе [3.2,](#_bookmark9) обратное кодирование оптимизирует запросы на чтение, но вводит некоторое усиление записи — вставка записи запускает дельта-сжатие и обновление исходной записи. на диск. проблема является обострился несколько с прыгать кодирование, где вставка а прыгать база причины обратная запись нет только к источник записывать, но также к соседний базы на каждый прыгать уровень. Для тяжелого вставка всплески, этот мог существенно увеличивать количество операций записи на диск, приводящее к заметному снижению производительности.

dbDedup использует а *с потерями обратная запись кэш* к адрес этот Проблема. ключ наблюдение является что обратные письма являются нет строго необходимый для обратно-кодированного хранилище. Отказ или задерживать в применение такой написать ответ операции делает нет ухудшать данные последовательность или честность-

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

обновлено записи оставаться нетронутый и только последствие является потенциальная потеря сжатия. Это уникальное свойство «потерь» обеспечивает естественную отказоустойчивость и позволяет dbDedup гибко планировать, когда и в каком порядке будут применяться обратные записи.

На записывать вставка, dbDedup пишет новый записывать к база данных как нормальный, и магазины дельта из источник записывать в кэш. Это задержки действительный обратная запись операция до система Ввод-вывод становится относительно бездействующим. Метрика простоя может быть разной, но мы используем Ввод/вывод очередь длина как а указание в наш текущий реализация. Чтобы сохранить максимальное сжатие при ограниченной памяти, dbDedup сортирует дельты в кэш к абсолютный количество из экономя место, они вносят свой вклад и соответственно расставляют приоритеты в порядке обратной записи . Когда Ввод/вывод становится праздный, более ценный дельты являются выписано​ первый. Когда кэш становится полный до система получает достаточно праздный , вход с наименее сжатие прирост является выброшен без​ воздействие правильность. К расстановка приоритетов обновлять и постановления о выселении, dbDedup более эффективно пожинает сжатие преимущества

от кэшированный дельты.

## Избегание Непродуктивный Дедупликация Работа

dbDedup использует два подхода, чтобы избежать применения дедупликации с низкой вероятностью получения существенной выгоды. Во-первых, регулятор дедупликации отслеживает коэффициент сжатия во время выполнения и автоматически отключает дедупликацию для баз данных, которые не приносят достаточно пользы. Во-вторых, фильтр на основе размера адаптивно пропускает дедупликацию для более мелких записей, которые вносят небольшой вклад в общий коэффициент сжатия.

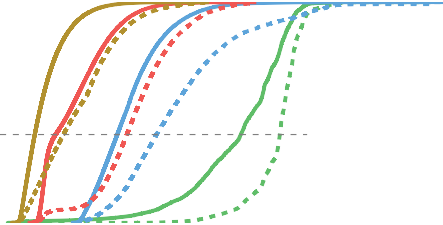
### Автоматический Дедупликация губернатор

База данных приложения экспонат разнообразный дедупликация характеристики. Для тех, кому это не приносит особой пользы, dbDedup автоматически отключает дедупликацию, чтобы избежать траты ресурсов. По нашему опыту, большая часть дублирования существует в пределах одной базы данных, то есть дедупликация несколько другой базы данных обычно урожайность маленький предельные выгоды по сравнению с их индивидуальной дедупликацией. Таким образом, dbDedup разбивает свой индекс дедупликации в памяти по базам данных и внутренне отслеживает степень сжатия для каждой из них. Если уровень сжатия базы данных остается ниже определенного порогового значения (например, 1,1 *×* ) в течение достаточно длительного периода (например, вставки 100 тыс. записей), регулятор дедупликации отключает дедупликацию для нее и удаляет соответствующий раздел индекса. Будущие записи, принадлежащие этой базе данных, обрабатываются. как нормальный, обходя дедупликация двигатель, пока уже закодированные данные остаются нетронутыми. dbDedup не активирует повторно базу данных, для которой дедупликация уже отключена, потому что мы этого не замечаем. драматический изменять в сжатие соотношение над время для любой определенная рабочая нагрузка, которую мы считаем нормой.

### Адаптивный На основе размера Фильтр

В ходе наших наблюдений за несколькими реальными наборами данных баз данных (см. раздел [5.1)](#_bookmark27) мы обнаружили, что наибольшая экономия за счет дедупликации достигается за счет небольшой доли из записи что являются больше в размер. Инжир. [7](#_bookmark17) показывает кумулятивная функция распределения (CDF) размера записи и взвешенная CDF по вкладу в экономию пространства для четырех рабочих нагрузок, используемых в наш эксперименты. Для эти наборы данных, 60% самый большой записи ac-

100



Wikipedia (#records) Wikipedia (space saving) Enron (#records)

Enron (space saving)

Stack Exchange (#records) Stack Exchange (space saving) Message Boards (#records) Message Boards (space saving)

80

CDF (percentage)

Источник

Якорь Дубликат сегмент  Побайтовое сравнение

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |

60

40

Цель

ВСТАВИТЬ КОПИЮ



ВСТАВЛЯТЬ

КОПИЯ

ВСТАВЛЯТЬ

20

0 2 10 3  10 4  10 5

10

Записывать размер (байты)

10 6 10 7

**Фигура 9:** Иллюстрация из дельта сжатие в dbDedup.

ondary получает обновления, добавляет их в свой локальный oplog и воспроизводит новые записи oplog для обновления своей локальной базы данных.

**Фигура 7:** На основе размера дедупликация фильтр.

С помощью dbDedup первичный узел сначала сохраняет новую запись в своем местный oplog. Позже, когда подготовка к магазин записывать или отправлять в реплику, он обрабатывается кодировщиком dbDedup, следуя шагам дедупликации, описанным в разделе [3.1.](#_bookmark5) Если dbDedup успешно выбирает похожую запись из существующего корпуса данных, он извлекает содержание из похожий записывать к первый проверка Кэш исходной записи. При промахах кэша он считывает запись из базового хранение. Это затем применяется двунаправленный дельта сжатие к источник и цель записи к генерировать вперед-кодированный форма новой записи и обратно закодированная форма аналогичной записи. dbDedup вставки новый записывать к начальный база данных в исходном виде и кэширует обратно закодированную аналогичную запись в с потерями обратная запись кэш до система Ввод/вывод становится праздный. Затем,

Начальный Узел​



*Reads*

Client

*Inserts & Updates*

*Reads*

Oplog

Oplog

Record cache

Delta cache

dbDedup Re-encoder

Delta cache

Record cache

Database

Backward- encoded records

Database

dbDedup Encoder

Oplog

syncer

dbDedup Decoder

dbDedup Decoder

Вторичный Узел​

dbDedup добавляет вперед-кодированный записывать к начальный оплог

**Фигура 8:** Интеграция из dbDedup в а СУБД.

считать для примерно 90–95% из данные сокращение. В другой слова, если мы только дедупликация записи больше чем 40%-плитка записывать размер, мы может уменьшать дедупликация накладные расходы к 40% пока только проигрыш 5–10% степени сжатия.

dbDedup подвиги этот наблюдение, с использованием а на основе размера дедупликация Фильтр, который обходит (рассматривает как уникальные) записи, размер которых меньше определенного порога. В отличие от специализированный дедупликация системы чей рабочая нагрузка характеристики известны заранее, dbDedup определяет размер отсечения для каждой базы данных, используя простую эвристику. Для каждой базы данных порог дедупликации сначала инициализируется на нуле, что означает, что все входящий записи являются дедуплицировано. Это ценить является затем периодически обновлено с 40%-плитка записывать размер из база данных каждые 1000 вставок записей.

# ВЫПОЛНЕНИЕ

Этот раздел описывает dbDedup выполнение подробности, включая как это подходит в СУБД хранилище и репликация фреймворки и внутреннее устройство его алгоритма дельта-сжатия.

## СУБД Интеграция

Пока выполнение подробности отличаться через СУБД, мы проиллюстрировать интеграцию dbDedup с использованием простой распределенной установки, состоящей из одного клиента, одного основного узла и одного вторичного узла, как показано на рис. [8.](#_bookmark19) Для простоты мы предполагаем, что только первичный узел обслуживает запросы на запись [3](#_bookmark21) и что он отправляет обновления асинхронно. к вторичный узел в форма из оплог пакеты. Теперь опишем поведение dbDedup для основных операций СУБД.

**Вставлять:** Основной узел записывает новую запись в свою локальную базу данных. и добавляет записывать к его оплог. Каждый оплог вход включает в себя а временная метка и а полезная нагрузка что содержит вставлен запись. Когда размер из несинхронизированный оплог записи достигает а порог, начальный посылает их в а партия к вторичный узел. The сек-

1. Когда вторичные устройства также выполняют функцию записи, каждое из них будет поддерживать отдельный ставка дедупликация индекс. Эти индексы бы быть обновлено в течение реплика синхронно- низации и в конечном итоге сближаются.

что является передано к вторичный в течение реплика синхронизация. На вторичном узле синхронизатор oplog СУБД получает и распространяет закодированные записи oplog на перекодировщик dbDedup. Перекодировщик сначала декодирует новую запись, считывая базовый аналог записывать от его местный база данных [4](#_bookmark22) (или источник записывать кэш, на хиты) и применение вперед-кодированный дельта. Это затем дельта компрессы​ похожий записывать с использованием недавно реконструированный новый запись как источник, нравиться в начальный, и генерирует такой же дельта-кодирование в обратном порядке для аналогичной записи. Наконец, dbDedup записывает новый записывать к вторичный база данных и обновления похожий запись в его дельта-кодированный форма. Эти шаги обеспечивает что вторичный

магазины такой же данные как начальный узел.

dbDedup поддерживает а *ссылка считать* для каждый хранится записывать который отслеживает количество записей, ссылающихся на него как на базу декодирования. Потому что dbDedup использует назад кодирование для база данных хранилище, после вставки, ссылка считать из новый записывать является набор к один, в то время как это из похожий записывать является без изменений. ссылка считать из исходная база аналогичной записи, если таковая имеется, уменьшается на единицу.

**Обновлять:** На обновлять, dbDedup первый чеки ссылка количество спрошено записывать. Если считать является ноль, значение нет другой записи относятся к это для декодирование, dbDedup напрямую применяется обновлять как нормальный. В противном случае dbDedup сохраняет текущую запись нетронутой и добавляет к ней обновление. Это гарантирует, что другие записи, использующие это как а ссылка может все еще быть раскодировано успешно. Когда количество ссылок достигает нуля, dbDedup сжимает все обновления записи и заменяет ее новыми данными.

dbDedup использует а обратная запись кэш к задерживать обновлять из а дельта-кодированная исходная запись. Чтобы предотвратить перезапись обычных клиентских обновлений, dbDedup всегда проверяет кэш для каждого обновления. Если он находит запись с тем же идентификатором (чтобы быть записанным позже), он аннулирует вход и доходы обычно с клиент обновлять.

**Удалить:** Если количество ссылок на удаляемую запись равно нулю, то удаление доходы как нормальный. В противном случае, dbDedup знаки

1. Поскольку вторичные и первичные узлы в основном синхронизированы, базовая запись используется в первичный кодировать запись почти всегда также подарок в вторичный. В редкий случаи где это является нет, вторичный запросы основной узел для новой записи, чтобы избежать дополнительных затрат на декодирование.

**Алгоритм 1** Дельта Компресс

1: **функция** D ELTA C OMPRESS ( *источник , тгт* )

2: *я ←* 0 *дн.* Инициализация

3: *дж ←* 0

4: *поз. ←* 0

5: *вс ←* 16

6: *sИндекс ← пустой*

7: *tInsts ← пустой*

8: **пока** *я* + *вс <* = *длина источника​* **делать** *д* Строить индекс для источник якоря

9: *хэш ←* РАБИН ХЭШ ( *источник ,*​​ *я , я* + *вс* )

10: **если** ЯВЛЯЕТСЯ ЯКОРЕМ ( *хэш* )​​ **затем**

11: *sIndex* [ *хэш* ] *← я*

12: **конец если**

13: *я ← я* + 1

14: **конец пока**

15: **пока** *дж* + *вс <* = *длина тгт .* **делать** *д* Сканировать тгт для самый длинный соответствовать

16: *хэш ←* РАБИН ХЭШ ( *tgt , j ,*​​ *дж* + *вс* )

17: **если** ЯВЛЯЕТСЯ ЯКОРЕМ ( *хэш* )​​ **и** *хэш* **в** *sИндекс* **затем**

18: ( *так что ф ф , к ф ф , л* ) *←* БАЙТ КОМП ( *ист ,*​​ *тгт , sИндекс* [ *ф п* ] *, к* )

19: **если** *поз < к ф ф* **затем**

20: *инстИнст ← ИНСТ* ( *ВСТАВЛЯТЬ , поз , к ф ф − поз* .)

21: *memcpy* ( *insInst . data , тгт , к ф ф − поз* .)

22: *tInsts . добавить* ( *insInst* )

23: **конец если**

24: *cpInst ← ИНСТ* ( *КОПИЯ , так ф ф , л* )

25 *: tInsts.append* ( *cpInst* )*​*

26: *поз. ← к ф ф* + *л*

27: *дж ← к ф ф* + *л*

28: **иначе**

29: *дж ← дж* + 1

30: **конец если**

31: **конец пока**

32: **возвращение** *tInsts*

33: **конечная функция**

он удален, но сохраняет свое содержимое. Любое чтение клиентом удаленной записи возвращает пустой результат, но она по-прежнему может служить базой декодирования для других записей, ссылающихся на нее. Когда количество ссылок на записывать капли к ноль, dbDedup удаляет это от база данных и уменьшить счетчик ссылок его базовой записи на единицу.

**Читать:** Если запрашиваемая запись хранится в необработанном виде, то она напрямую отправляется клиенту, как и в обычном случае. Если запись является закодирован, затем dbDedup's декодер возвращается это назад к его оригинальную форму перед возвратом клиенту. Во время декодирования декодер извлекает базовую запись из кэша исходных записей (или хранилища в случае промаха кэша) и восстанавливает запрошенную запись, используя сохраненную дельту. Если сама базовая запись закодирована, декодер повторяет шаг выше итеративно до это находки а база записывать сохранены в полном объеме.

**Мусор Коллекция:** Каждый запись ссылка считать обеспечивает что кодирование цепь воля нет быть испорченный на обновления или удаления. Для облегчения сбора мусора dbDedup проверяет наличие удаленных объектов при чтении. В частности, по пути декодирования, если запись видна после удаления dbDedup создает дельту между двумя соседними записями, и уменьшает его ссылка считать к один. Когда нет другие записи зависеть на это для декодирование, записывать может быть безопасно удалено из базы данных.

## Дельта Сжатие

К гарантировать легкий дедупликация, это является важный к делать Дельта-сжатие dbDedup быстрое и эффективное. Алгоритм сжатия дельта, используемый в dbDedup, адаптирован из xDelta [[42],](#_bookmark86) классического алгоритма кодирования копирования/вставки, использующего технику сопоставления строк для нахождения совпадающих смещений в исходном и целевом потоках байтов. Оригинальный алгоритм xDelta в основном работает в два этапа. В первом шаг, xДельта делит источник транслировать в фиксированный размер (к де-

**Алгоритм 2** Дельта Перекодировать​

1: **функция** ДЕЛЬТА - КОДИРОВАНИЕ ( *src ,*​​​ *тгт , tInsts* )

2: *sPos ←* 0

3: *тПос ←* 0

4: *копироватьСегменты ← пустой*

5: *сИнсты ← пустой*

6: **для каждый** *инст* в *tInsts* **делать**

7: **если** *инст . тип* = *КОПИЯ* **затем**

8 *: copySegs.append* ( *inst.sO​​​ ф ф , тПос , инст . лен* )

9: **конец если**

10: *тПос ← тПос* + *инст . лен*

11: **конец для**

12: *копироватьСегменты . сортироватьПо* ( *sO фф* ) 13: **для каждый** *сегмент* в *copySegs* **делать** 14: **если** *сПос < сег . sO ф ф* **затем**

15: *инстИнст ← ИНСТ* ( *ВСТАВЛЯТЬ , сПос , так ф ф − сПос* )

16: *memcpy* ( *insInst . data , источник , так ф f − сПос* )

17 : *sInsts.append* ( *insInst )​*

18: **конец если**

19: *cpInst ← ИНСТ* ( *КОПИЯ , сег . тО ф ф , сег . len* )

20 : *sInsts.append* ( *cpInst )​*

21: *сПос ← сег . sO ф ф* + *сег . длина*

22: **конец для**

23: **возвращение** *sInsts*

24: **конечная функция**

вина, 16-байтовый) блоки. Это затем вычисляет а Ольха32 [[32]](#_bookmark76) контрольную сумму (та же функция отпечатка пальца, которая используется в gzip ) для каждого блока байтов и создает временный индекс в памяти, сопоставляющий контрольные суммы с соответствующими им смещениями в источнике. На втором этапе xDelta сканирует цель объект байт к байт от начало, с использованием а скольжение окно из такой же размер как байт блоки. Для для каждого целевого смещения вычисляется контрольная сумма Alder32 байтов в скользящем окно и консультирует источник индекс населенный в первый шаг. Если это находки а соответствовать, xДельта расширяется поиск процесс от сопоставленные смещения, используя двунаправленное побайтовое сравнение для определения самой длинной общей последовательности (LCS) между источником и цель потоки. Это затем пропускает совпало область к продолжать итеративный поиск. Если совпадение не найдено, он перемещает скользящее окно к один байт и перезапускает соответствие. Вдоль этот процесс, xDelta кодирует совпало регионы в цель в КОПИЯ инструкции и несовпадающие регионы в INSERT инструкции.

Как показано в Алгоритм [1](#_bookmark23) и Инжир. [9,](#_bookmark18) dbDedup's дельта сжимает-

сион алгоритм является а измененный версия из xДельта основанный на на наблюдение​ что а большой дробь из его время является потраченный в источник индекс строительство и поиски. В первый кодирование шаг, dbDedup образцы а подмножество позиций смещения, называемых *якорями* , младшие биты контрольных сумм которых соответствовать а предопределенный узор. интервал между якоря указывают выборка соотношение и является контролируемый к длина из битовый шаблон. В второй шаг, dbDedup выполняет индекс поиски только для якоря в цель, избегая нуждаться к проконсультироваться индекс источника в каждый цель смещение. якорь интервал обеспечивает а настраиваемый компромисс между степенью сжатия и скоростью кодирования, и мы оцениваем его эффекты в Раздел [5.](#_bookmark26) Мы пропускать некоторый оптимизации в псевдокод, приведенный выше, из-за ограничений по пространству. Например, смежные и перекрывающиеся КОПИЯ инструкции являются объединенный; короткая КОПИЯ инструкции являются преобразованный в эквивалент ВСТАВЛЯТЬ инструкции , когда затраты на кодирование превышают экономию места.

Как обсудили в Раздел [3.2,](#_bookmark9) после вычисление вперед-

закодированный данные с использованием алгоритм выше, dbDedup использует дельта-перекодирование (алгоритм [2)](#_bookmark25) для эффективной генерации исходной записи с обратным кодированием. Вместо переключения исходных и целевых объектов и повторного выполнения дельта-сжатия dbDedup повторно использует КОПИЯ инструкции сгенерировано до и сортирует их к их соответствующие смещения источника. Затем он заполняет несовпадающие области в источник с ВСТАВЛЯТЬ инструкции. Пока это может результат в немного

неоптимальная скорость сжатия (например, из-за перекрытия COPY (инструкции, которые объединяются), процесс перекодирования происходит чрезвычайно быстро (со скоростью памяти), поскольку не выполняются вычисления контрольной суммы или индексные операции.

Дельта декомпрессия в dbDedup является простой. Это просто итерирует инструкции, сгенерированные алгоритмом сжатия и объединяет совпало и бесподобный регионы к воспроизвести исходный целевой объект.

# ОЦЕНКА

В этом разделе dbDedup оценивается с использованием четырех реальных наборов данных. этот оценка, мы реализовано оба dbDedup и традиционная дедупликация на основе фрагментов (trad-dedup) в MongoDB (v3.1). Результаты показывают что dbDedup обеспечивает значительный сжатие преимущества, превосходит традиционную дедупликацию, сочетается со сжатием на уровне блоков и оказывает незначительное влияние на производительность СУБД.

Если не указано иное, во всех экспериментах используется реплицированная база данных MongoDB. настраивать с один начальный, один вторичный, и один клиент узел. Каждый узел имеет четыре ядра ЦП, 8 ГБ ОЗУ и 100 ГБ локального жесткого диска хранилище. Мы использовать MongoDB's WiredTiger [[16]](#_bookmark60) хранилище двигатель с отключенной функцией полного ведения журнала во избежание помех.

## Рабочие нагрузки

Четыре реальных набора данных представляют собой разнообразный спектр приложений баз данных: совместное редактирование (Wikipedia), электронная почта (Enron) и онлайн-форумы (Stack Exchange, Message Boards). Мы сортируем каждый набор данных по временной метке создания, чтобы сгенерировать трассировку записи , и затем генерировать а читать след с использованием публичный статистика или известный доступ узоры к подражать а реальный мир рабочая нагрузка, как подробный ниже.

**Википедия:** Полная история изменений каждой статьи в Википедии Английский корпус [[13]](#_bookmark57) от январь 2001 к Август 2014. Мы извлекли подмножество объемом 20 ГБ с помощью случайной выборки на основе идентификаторов статей. Каждая редакция содержит новую версию статьи. и метаданные о пользователе, внесшем изменения (например, имя пользователя, временная метка, комментарий). Большая часть дублирования происходит из-за дополнительных изменений. к страниц. Мы вставлять первый 10,000 пересмотры к заполнить исходный база данных. Мы затем проблема читать и писать запросы Согласно общедоступному доступу к Википедии [[62],](#_bookmark106) где нормализованное чтение/запись соотношение является 99,9 к 0,1.99,7% из читать запросы являются к последний версия из а вики страница, и остаток к а специфический пересмотр.

**Энрон:** Публичный набор данных электронной почты [[2]](#_bookmark46) с данными примерно 150 пользователей, в основном высшего руководства Enron. Корпус содержит около 500 тыс. сообщений общим объемом 1,5 ГБ. Каждое сообщение содержит текстовое сообщение, имя почтового ящика, заголовки сообщения, такие как временная метка и идентификаторы отправителя/получателя. Дублирование в основном происходит из-за сообщение вперед и ответы что содержать содержание из предыдущие сообщения. Мы максимально быстро вставляем отсортированный набор данных в СУБД. После каждой вставки мы отправляем запрос на чтение определенному электронная почта сообщение, в результате в а совокупный читать/писать соотношение из 1 к 1. Это основано на предположении, что каждый пользователь использует единый гле электронная почта клиент что кэши запрошено сообщение локально, так каждое сообщение записывается и считывается в/из СУБД один раз.

**Куча Обмен:** А публичный данные свалка от Куча Сеть обмена [[10]](#_bookmark54) , которая содержит полную историю сообщений пользователей и связанную с ними информацию, такую как теги и голоса. Мы извлекли подмножество объемом 10 ГБ с помощью случайной выборки. Большая часть дублирования в этом наборе данных возникает из-за того, что пользователи редактируют свои собственные сообщения и копируют ответы. от другой обсуждение нити. Мы вставлять сообщения в СУБД как новый записи в временный заказ. Для каждый почта, мы читать

**Доски объявлений:** Набор данных форума объемом 10 ГБ, содержащий сообщения пользователей, извлеченные из ряда публичных сообщений, созданных с помощью vBulletin [[11]](#_bookmark55) доски что крышка а разнообразный диапазон из резьбовой темы, такие 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 цитирования пользователями комментариев других людей. Чтобы имитировать поведение пользователей на форуме для обсуждения, для каждой вставки сообщения мы выдаем определенное количество «чтений ветки», которые запрашивают все предыдущие сообщения в содержащей ее ветке. Количество чтений потока за вставка является полученный к разделение общий вид считать из ветку по количеству содержащихся в ней сообщений.

## Сжатие Соотношение и Индекс Память

Мы первый оценивать dbDedup's сжатие соотношение и индекс использование памяти и сравните их с trad-dedup и Snappy [[9],](#_bookmark53) MongoDB по умолчанию блочный уровень компрессор. Для каждый набор данных, мы загрузите записи в СУБД как можно быстрее и измерьте полученные размеры хранилища, объем данных, переданных по сети, и использование памяти индекса.

На рис. [10](#_bookmark30) показаны результаты для пяти конфигураций: (1) dbDedup с куски из 1 КБ или 64 байты, (2) trad-dedup с куски 4 КБ или 64 байта и (3) Snappy. Розовая (левая) полоса показывает коэффициент сжатия хранилища, указывающий на вклад только дедупликации и сжатия после дедупликации. Степень сжатия определяется как исходная данные размер разделенный к сжатый данные размер, так а ценить из одно означает нет сжатие достигнуто. синий (верно) бар показывает индекс использования памяти. Небольшой кэш исходных записей (32 МБ, используется обеими dbDedup и trad-dedup) и с потерями обратная запись кэш (8 МБ, используемые только dbDedup) не отображаются.

The преимущества являются самый большой для Википедия (Инжир. [10а).](#_bookmark31) С а Размер фрагмента 1 КБ, dbDedup сокращает хранилище данных в 26 *раз* (в 41 *раз* в сочетании с Snappy), используя 36 МБ индексной памяти. Уменьшение куска размер к 64 Б увеличивается сжатие соотношение к 37 *×* (61 *×* ) используя только 45 МБ индексной памяти. Уменьшение размера фрагмента для dbDedup не приводит к значительному увеличению использования памяти индекса, поскольку dbDedup индексирует не более *K* записей на запись, независимо от размера фрагмента. В отличие, пока trad-dedup's сжатие соотношение увеличивается от 2,3 *×* (3 *.* 7 *×* ) к 15 *×* (24 *×* ) когда с использованием а кусок размер из 64 Б вместо 4 КБ, его индексная память увеличивается с 80 МБ до 780 МБ, что делает это непрактично для операционных СУБД. Это происходит из-за того, что индексы с дедупликацией торговли каждый уникальный кусок хэш, ведущий к почти линейный увеличение индекс накладные расходы как кусок размер уменьшается, и также потому что это необходимо использовать много больше индекс ключи (20-байтовый ША-1 хэш против. 2-байтовый контрольная сумма), поскольку коллизии приведут к повреждению данных. Потребление 40% меньше индекс память, dbDedup с 64 Б кусок размер достигает сжатие соотношение 16 *×* выше чем trad-dedup с его типичный 4 Размер фрагмента КБ. Snappy сжимает набор данных всего в 1,6 *раза* , потому что он может нет устранять дублирование вызванный к уровень приложения версионирование, но не требует индексной памяти. Он обеспечивает такое же сжатие 1,6 *×* при применении к дедуплицированным данным.

Для другой наборы данных, абсолютный преимущества являются меньше, но

основные наблюдения схожи: dbDedup обеспечивает более высокую степень сжатия при меньшем использовании памяти, чем trad-dedup, а Snappy сжатие преимущества (1,6–2,3 *×* ) дополнять дедупликация. Для набора данных Enron (рис. [10b)](#_bookmark32) dbDedup сокращает хранилище в 3,0 *раза* (5,8 *×* ), что согласуется с результатами, полученными нами в ходе экспериментов с данными из облачного развертывания серверов Microsoft Exchange, содержащих петабайты реальных данных электронной почты пользователей. [5](#_bookmark29) Два форума наборы данных (рис. [10с](#_bookmark33) и [10г)](#_bookmark34) делать нет экспонат как много дубликат-

это для такой же число из раз как его вид считать. The совокупный

читать/писать соотношение является 99,9 до 0,1.

1. Грустно, мы не может раскрывать подробности должный к конфиденциальность ограничения.



Index memory usage

Additional comSression Irom SnaSSy

ComSression Irom deduS alone

60

50

Compression ratio

40

30

20

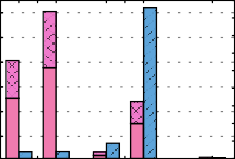
10

1КБ

64Б 4КБ

64Б

900

800

Index memory usage (MB)

700

600

500

400

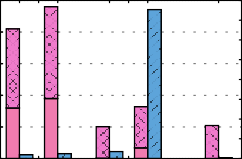
300

200

100

0

6

5

Compression ratio

4

3

2

1

1КБ

64Б 4КБ

64Б

400

350

Index memory usage (MB)

300

250

200

150

100

50

0

4.0

3.5

Compression ratio

3.0

2.5

2.0

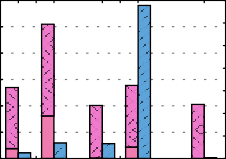
1.5

1.0

1КБ 64Б 4КБ

64Б

3500

3000

Index memory usage (MB)

2500

2000

1500

1000

500

0

3.5

3.0

Compression ratio

2.5

2.0

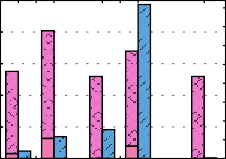
1.5

1.0

1КБ 64Б 4КБ

64Б

4000

3500

Index memory usage (MB)

3000

2500

2000

1500

1000

500

0

dbDedup

trad -dedup

Снаппи

dbDedup

trad -dedup

Снаппи

dbDedup

trad -dedup

Снаппи

dbDedup

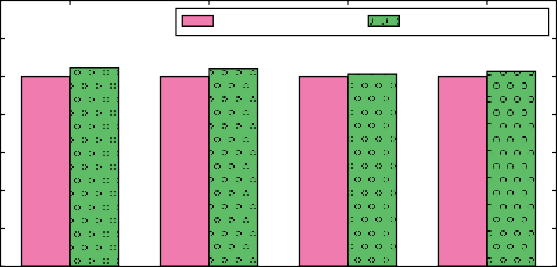
trad -dedup

Снаппи

* 1. Википедия
  2. Энрон
  3. Куча Обмен
  4. Сообщение Доски

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

3500



Storage usage

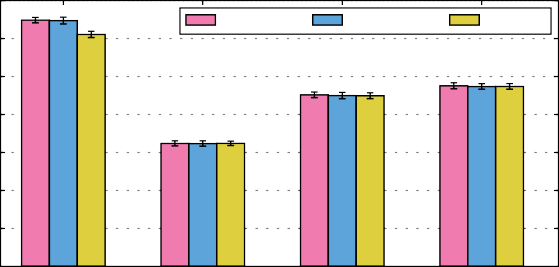
Network transfer

+4.5%

+4.1%

+1.3%

+2.8%



Original

dbDedup

Snappy

1.2

Normalized comp. ratio

1.0

0.8

0,6

0.4

0.2

0.0

Википедия Enron Stack Обмен Сообщение Доски

3000

2500

Throughput (ops)

2000

1500

1000

500

0

Википедия Enron Stack Обмен Сообщение Доски

**Рисунок 11: Экономия ресурсов хранения и пропускной способности сети** – Относительная экономия Коэффициенты сжатия, достигнутые с помощью dbDedup (с размером фрагмента 64 байта) для локального хранение и сетевая передача для каждого из наборов данных, нормализованных по Абсолютные коэффициенты сжатия хранилища показаны на рис. [10](#_bookmark30) (для dbDedup с 64-байтовый кусками).

ция как Википедия или электронная почта наборы данных, потому что пользователи делать нет цитата или редактировать комментарии как часто как Википедия пересмотры или электронная почта пересылает/отвечает. Даже так, мы все еще наблюдать что dbDedup уменьшает хранение в 1,3–1,8 *×* (3–3,5 *×* ). Поскольку мы смогли просканировать только последние сообщения в наборе данных досок объявлений, сжатие dbDedup соотношение является консервативный, нет включая преимущества от дельта-сжатие правок пользователей в их собственных сообщениях. [6](#_bookmark38)

Помимо использования хранилища, dbDedup одновременно достигает значительных результатов сжатие на данные передача инфекции над сеть с

100

80

CDF (percentage)

60

40

20

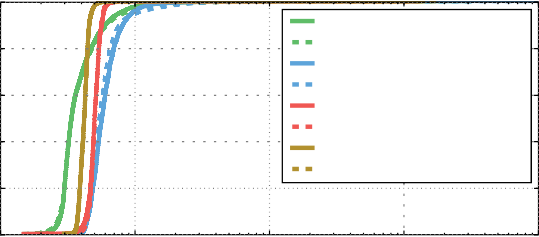
0 - 1

10

10 0

1. Пропускная способность

10 1



Wikipedia (original) Wikipedia (dbDedup) Enron (original) Enron (dbDedup)

Stack Exchange (original) Stack Exchange (dbDedup) Message Boards (original) Message Boards (dbDedup)

Задержка (миллисекунды)

1. Задержка

10 2

10 3

вперед кодирование. Инжир. [11](#_bookmark35) показывает сетевой уровень сжатие как нормализованный результат по отношению к использованию хранилища (1,0 по оси Y для каждого набора данных). dbDedup обеспечивает немного меньшее сжатие базы данных хранилище чем на данные передано над сеть, в основном из-за к перекрывающиеся кодировки (Раздел [3.2)](#_bookmark9) и дельта выселения из кэша обратной записи. Тем не менее, разница составляет менее 5%. для все наборы данных, потому что перекрывающиеся кодировки являются необычный и потому что кэш обратной записи с потерями использует приоритетное вытеснение.

## Время выполнения Производительность Влияние

Этот эксперимент является разработанный к мера dbDedup's влияние на производительность СУБД . Мы сравниваем три конфигурации развертывания MongoDB: (1) Нет сжатие («Оригинал»), (2) dbDedup, и

(3) Энергичный. Для каждой настройки мы проводим эксперименты три раза для всех рабочих нагрузок и выводим среднее значение.

**Пропускная способность:** Рис. [12а](#_bookmark36) показывает пропускная способность для четыре рабочие нагрузки. Мы видим, что dbDedup создает незначительные накладные расходы на пропускную способность. Snappy также немного снижает производительность для трех из рабочие нагрузки, так как это является а быстрый и легкий в соответствии компрессор. Исключением является Википедия, для которой использование Snappy вызывает 5% пропускная способность снижение, потому что некоторый большой Википедия записи

1. Мы обнаружили, что 15% сообщений редактируются по крайней мере один раз, а большинство отредактированных сообщений больше среднего размера поста.

**Фигура 12: Производительность Влияние** – Время выполнения измерения из MongoDB's пропускная способность и задержка для различных рабочих нагрузок и конфигураций.

не может соответствовать в а одинокий WiredTiger страница и требовать дополнительный Вводы/выводы.

**Задержка:** Рис. [12б](#_bookmark37) показывает СДФ из клиент задержка. Для ясность, мы только показывать результаты для MongoDB с и без dbDedup включен. Снова, мы видим, что dbDedup практически не оказывает никакого эффекта на производительность. задержка распределение кривые с dbDedup включено тесно отслеживать те для нет сжатие/дедупликация. The Разница в задержке 99,9%-го тайла составляет менее 1% для всех рабочих нагрузок.

## Эффекты из Кэширование

dbDedup использует два специализированный кэши к минимизировать Ввод/вывод накладные расходы, связанные с чтением и обновлением исходных записей: кэш исходной записи (32 МБ) и с потерями обратная запись кэш (8 МБ). Мы сейчас оценить эффективность этих кэшей.

**Кэш исходной записи:** на рис. [13а](#_bookmark40) показано влияние кэша исходной записи на коэффициент сжатия (левая ось Y) и процент извлечений исходной записи, требующих чтения СУБД (коэффициент промахов кэша; правая ось (ось Y), с а диапазон из награда счет ценности для Нагрузка Википедии. Отзыв что dbDedup использует кэш-с поддержкой выбор из похожие записи кандидатов, присваивая баллы вознаграждения кандидатам, присутствующим в кэше (см. раздел [3.1.3).](#_bookmark8)

Hop encoding

Version jumping

1.0



Normalized compression ratio Cache miss ratio

Normalized comp. ratio

0.8

0,6

0.4

0.2

0

0 2 4 8

1.0

0.8

Cache miss ratio

0,6

0.4

0.2

0

1.0

0.8

Normalized comp. ratio

0,6

0.4

0.2

0.0

35

Worst case source retrievals

30

25

20

15

Нет кэш

1200

dbDedup (w/ write-back cache) dbDedup (w/o write-back cache)

1000

Throughput (ops)

800

600

Оценка вознаграждения

1. Источник Записывать Кэш

10

5

0

Number of writebacks

300

250

200

150

100

50

0

4 8 12 16 20 24 28 32

Расстояние прыжка

400

200

0

0 50 100 150 200

Время (секунды)

1. Потерянный Обратная запись Кэш

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

Если кэш не используется (самые левые столбцы), каждое извлечение источника записывать несет а читать запрос. Даже без кэш-с поддержкой выбор (0 награда счет), маленький источник записывать кэш устраняет 74% таких запросов. При оценке вознаграждения два (по умолчанию) метод выбора с учетом кэша дополнительно снижает коэффициент промахов на 40% (до 16%), не снижая заметно коэффициент сжатия. Дальнейшее увеличение к награда счет незначительно уменьшать кэш скучать соотношение в то время как сокращение сжатие соотношение немного, потому что меньше в качестве исходных записей с большей вероятностью будут выбраны похожие кандидаты.

**Кэш обратной записи с потерями:** dbDedup использует обратное кодирование для избегать декодировать когда чтение последний «версии» из а обновлять последовательность. Таким образом, дедупликация а новый записывать включает в себя оба письмо полный новый записывать и замена источник записывать с Данные в дельта-коде. дополнительный писать (тот замена) может вести к значительный проблемы с производительностью для рабочих нагрузок с интенсивным вводом-выводом во время пиковых объемов записи. Кэш обратной записи с потерями dbDedup смягчает такие проблемы.

Чтобы эмулировать пиковую рабочую нагрузку с интенсивным вводом-выводом и периодами простоя, мы вставлять Википедия данные в полный скорость для 10 секунды и спать по 10 секунд, повторяя процедуру. На рис. [13б](#_bookmark42) показана пропускная способность вставки MongoDB. над время, с и без обратная запись кэш. Без кэш, СУБД пропускная способность видимо уменьшается в течение занятый периоды из-за дополнительных записей в базу данных. Напротив, использование обратной записи кэш избегает СУБД замедлять в течение рабочая нагрузка всплески, как показано к разница между два линии в различный моменты времени (например, в секунды 0, 130, 170 и 190).

dbDedup имеет два начальный настраиваемый параметры, вне те исследовал выше, что оказывать воздействие сжатие/производительность компромиссы: расстояние прыжка и якорь интервал. Это подраздел определяет количественно влияние этих параметров и поясняются значения по умолчанию.

## Прыгать Кодирование

dbDedup использует прыгать кодирование к уменьшать худший случай извлечение

**Рисунок 14: Кодирование прыжка против Переключение версий** – для Википедии рабочая нагрузка и умеренный прыгать расстояния, прыгать кодирование обеспечивает много выше сжатие соотношения с маленький увеличивается в худший случай источник извлечения и количество обратных записей.

раз пока поддержание сжатие выгоды. оценивать его эффективность , мы также реализовано версия прыжки в MongoDB и сравнил две схемы кодирования.

На рис. [14](#_bookmark41) показаны результаты для трех метрик в зависимости от расстояния прыжка: компрессия соотношение (нормализованный к стандартный назад кодирование), наихудшее количество извлечений источника (для цепочки кодирования длина из 200), и число из обратные записи. Версия прыжки результаты в существенно (60–90%) ниже сжатие соотношения, поскольку все справочные версии хранятся в незакодированном виде. Коэффициент сжатия улучшается по мере увеличения расстояния перехода, поскольку сохраняется меньше записей. в незакодированный форма. В контраст, потому что прыгать базы являются хранится как дельты, прыгать кодирование обеспечивает сжатие соотношения в пределах 10% полного обратного кодирования. При кодировании переходов коэффициент сжатия остается относительно стабильным по мере увеличения расстояния перехода из-за меньшего количества, но менее схожих баз переходов.

The число из худший случай источник извлечения для прыгать кодировка близка к той, что используется для перехода между версиями. При наличии нескольких уровней прыжков отслеживание до ближайшей базы прыжка занимает только логарифмическое время. По мере увеличения расстояния прыжка время декодирования доминирует за счет обхода обратных дельт между соседними базами прыжков. Нижний график показывает количество дополнительных обратных записей, необходимых в каждой схеме. прыгать кодирование несет более обратные записи для маленький расстояния прыжков, оба схемы быстро подход длина из кодирование цепь как прыгать расстояние увеличивается. Эмпирически, мы находить что а Расстояние перехода 16 (по умолчанию) обеспечивает хороший компромисс между степенью сжатия и накладными расходами на декодирование.

## Оптимизация из Дельта Сжатие

dbDedup превосходит xДельта алгоритм к сокращение накладные расходы на вычисления при вставке исходного индекса и поиске. Он вводит настраиваемый интервал привязки, который управляет частотой дискретизации точек смещения в исходном потоке байтов.

Инжир. [15](#_bookmark43) показывает сжатие соотношение (левый (ось Y) и пропускная способность (правая ось Y) для различных значений интервала привязки, в сравнении с xDelta, для рабочей нагрузки Википедии. С интервалом привязки 16 (размер окна по умолчанию в xDelta) dbDedup выполняет почти такой же как xДельта. dbDedup's дельта сжатие скорость улучшается как якорь интервал увеличивается, потому что это уменьшает количество вставки и поиски в источник компенсировать индекс . The ком-

60



Compression ratio Compression throughput

50

Compression ratio

40

30

20

10

0 16 32

60

50

40

30

20

10

64 128 0

если [[21],](#_bookmark65) xDelta [[42]](#_bookmark86) и zdelta [[60].](#_bookmark104) Могут быть использованы специализированные схемы для специфический данные форматы (например, XML-) к улучшать качество сжатия [[28,](#_bookmark72) [63,](#_bookmark107) [39,](#_bookmark83) [52].](#_bookmark96) Алгоритм дельта-сжатия, используемый в dbDedup, адаптирован из xDelta, связь с которым обсуждается в разделе [4.2.](#_bookmark24)

Дельта-сжатие используется для сокращения сетевого трафика при передаче файлов и протоколах синхронизации. Большинство систем предполагают, что предыдущие версии одного и того же файла явно идентифицированы приложением, и дублирование существует только среди предыдущих версий такой же файл [[61,](#_bookmark105) [57].](#_bookmark101) На исключение является КОНУС [[38],](#_bookmark82) который повторно

Compression throughput (MB/s)

xДельта

Якорь интервал (dbDedup)

дучес сеть передача для синхронизация файл система реплики к

**Фигура 15: Оптимизация из Дельта Сжатие** – Сравнение из оп- оптимизированный вариант dbDedup по сравнению с xDelta с использованием рабочей нагрузки Википедии.

Коэффициент сжатия существенно не уменьшается, поскольку dbDedup выполняет сравнение на уровне байтов в двух направлениях из сопоставленных точек. С а якорь интервал из 64, dbDedup превосходит xDelta на 80% по производительности сжатия, при этом потери составляют всего 7% потеря в сжатие Коэффициент. Увеличивается якорь интервал к 128 дополнительно увеличивает пропускную способность на 10%, но приводит к потерям в 15% по степени сжатия. Мы используем значение 64 в качестве значения по умолчанию, что обеспечивает разумный баланс между степенью сжатия и пропускной способностью.

# ДОПОЛНИТЕЛЬНЫЙ СВЯЗАННЫЙ РАБОТА

Большинство предыдущий дедупликация работа является обсудили в Раздел [2.](#_bookmark1) Этот в разделе обсуждаются некоторые дополнительные смежные работы.

**Сжатие базы данных:** За последние несколько десятилетий было предложено несколько схем сжатия баз данных. Большинство операционных СУБД, которые сжимают содержимое базы данных, используют страницу или блочный уровень сжатие [[30,](#_bookmark73) [37,](#_bookmark81) [46,](#_bookmark90) [43,](#_bookmark87) [3,](#_bookmark47) [16].](#_bookmark60) Некоторые использовать сжатие префикса, который выглядит для общий последовательности в начало значений полей для данного столбца по всем строкам на каждой странице. Просто как с наш dbDedup подход, такой сжатие требует СУБД распаковывает кортежи перед обработкой запроса.

Там являются схемы в некоторый OLAP системы что позволять СУБД для процесс данные в его сжатый формат. Для пример, словарь сжатие заменяет повторяющийся длинный домен ценности с короткий фиксированной длины целое число коды. Этот подход является обычно использовал в хранилищах данных, ориентированных на столбцы [[18,](#_bookmark62) [36,](#_bookmark80) [69,](#_bookmark113) [51].](#_bookmark95) Эти системы обычно фокус на атрибуты с относительно маленький домен размер и исследовать перекос в частотах значений, чтобы ограничить результирующий словарь к а управляемый размер [[23].](#_bookmark67) The авторы в [[56]](#_bookmark100) предложить схему дельта-кодирования, в которой каждое значение в отсортированном столбце представлено дельтой от предыдущего значения. Хотя этот подход работает хорошо для числовой ценности, это является непригодный для строки. Ни один из этих методов не обнаруживает и не устраняет избыточные данные с гранулярностью меньше одного поля, таким образом теряя потенциальное сжатие преимущества для много приложения что по своей сути содержать такой избыточность. dbDedup, в контраст, является способный к удалить много более мелкозернистый дубликаты с на уровне байтов дельта сжатие. В отличие от другой в соответствии сжатие схемы, dbDedup является нет 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 только дедупликация шаги один раз, и использует закодированный результат для оба хранение баз данных и сетевая передача. Напротив, та же самая запись будет сжатый дважды (в база данных страница и оплог партия), для страница

сжатие схемы к достигать данные снижение в оба слои.

**Дельта Кодировка:** Есть имеет был а большой тело из предыдущий работа над методами дельта-кодирования, включая несколько алгоритмов общего назначения основанный на на Лемпель-Зив подход [[68],](#_bookmark112) такой как vcd-

отправка дельта-кодированный файлы; это идентифицирует похожий файлы к вычисление число из соответствие биты на Цвести фильтры сгенерировано с хэши фрагментов файлов. dbDedup идентифицирует похожую запись из корпуса данных без руководства по применению и поэтому представляет собой более общий подход, чем большинство этих предыдущих систем.

Техника обратного кодирования, используемая в dbDedup, вдохновлена системами хранения данных с управлением версиями, такими как RCS [ [59]](#_bookmark103) и XDFS [[42].](#_bookmark86) методы иметь был использовал в версия контроль системы такой как Гит [[41]](#_bookmark85) и СВН [[29]](#_bookmark74) к давать возможность путешествие назад над история коммитов. В отличие от эти системы который явно поддерживать версионирование родословной для всех файлов, dbDedup устанавливает цепочку кодирования полностью основанный на на сходство отношения между записи, и таким образом делает нет требовать системный уровень поддерживать для версионирование. [[54]](#_bookmark98) использует дельта-кодирование для дедуплицированного резервного хранения. Он использует прямое кодирование. и только позволяет кодирование цепи с а максимум длина двух предметов. Подобно другим системам хранения данных с дельта-кодированием, которые используют версия прыжки, это имеет к жертва сжатие прибыль в чтобы ограничить наихудшие случаи извлечения базовых данных.

Насколько нам известно, dbDedup — первая система, исследующая различные формы из кодирование для сеть и уровень хранения сжатие и к предоставлять эффективный преобразования между два. С новая схема кодирования переходов, dbDedup, значительно облегчает болезненные компромиссы между сжатие соотношение и извлечение накладные расходы на дельта-кодированное хранилище. Кроме того, он вводит новые механизмы кэширования. специализированный для дельта закодированный хранилище, существенно сокращение накладных расходов на ввод-вывод при одновременном повышении эффективности использования памяти. Сочетание всех этих методов делает онлайн-доступ к закодированному хранилищу практичным.

**Обнаружение сходства:** Предыдущая работа предоставила различные подходы к вычисление эскизы (сходство метрики) для идентификация похожих предметов. Базовая техника идентификации особенностей объектов таким образом, чтобы похожие объекты имели идентичные особенности, была впервые предложена Бродер [[24,](#_bookmark68) [25]](#_bookmark69) в контекст из веб страниц. Несколько документы [[58,](#_bookmark102) [47,](#_bookmark91) [20,](#_bookmark64) [53,](#_bookmark97) [65]](#_bookmark109) предложить методы вычисления эскизов для подобия обнаружение что являются крепкий к маленький правки в данные. The извлечение признаков подход использовал в dbDedup является похожий к что в ДОТ [[47]](#_bookmark91) и sDedup [[65].](#_bookmark109)

# ЗАКЛЮЧЕНИЕ

dbDedup является а легкий основанный на сходстве дедупликация движок для операционных СУБД, который сокращает как использование хранилища, так и объем из данные передано для удаленный Репликация. Объединение Благодаря частичной индексации и дельта-сжатию на уровне байтов dbDedup достигает более высоких коэффициентов сжатия, чем сжатие на уровне блоков и дедупликация на основе фрагментов, при этом обеспечивая эффективное использование памяти. Он использует новые механизмы кодирования и кэширования, чтобы избежать значительных накладных расходов на ввод-вывод, связанных с доступом к записям с дельта-кодированием. Экспериментальные результаты с четырьмя реальными рабочими нагрузками показывают, что dbDedup способен достичь сокращения до 37 *× (61 ×* при сочетании с блочным уровнем сжатие) в хранилище размер и репликация трафик пока накладывая незначительные накладные расходы на производительность СУБД.

**Благодарности:** Мы хотели бы поблагодарить Кейта Бостика за его руководство на внутренности из WiredTiger. Мы благодарить анонимным рецензентам за полезные отзывы о статье. Мы также благодарим члены и компании из ПДЛ Консорциум (включая Broadcom, Citadel, Dell EMC, Facebook, Google, Hewlett-Packard Labs, Hitachi, Huawei, Intel, Microsoft Research, NetApp, Оракул, Samsung, Сигейт, Тинтри, Два Сигма, Убер, Веритас и Западный Цифровой) для их интерес, идеи, обратная связь, и поддерживать. Это исследование было частично спонсировано корпорацией Intel в рамках программы Intel Наука и Технологии Центр для Облако Вычислительная техника (ISTC-CC) и к MongoDB Зарегистрировано. Эксперименты были включено благодаря щедрым пожертвованиям оборудования от Intel и NetApp.

# ССЫЛКИ

1. Байду Байк. [http://baike.baidu.com/.](http://baike.baidu.com/)
2. Энрон Электронная почта Набор данных. [https://www.cs.cmu.edu/~./enron/.](https://www.cs.cmu.edu/~./enron/)
3. Сжатие InnoDB. [http://dev.mysql.com/doc/refman/5.6/](http://dev.mysql.com/doc/refman/5.6/en/innodb-compression-internals.html) [ru/innodb-compression-internals.html.](http://dev.mysql.com/doc/refman/5.6/en/innodb-compression-internals.html)
4. линукс СДФС. [www.opendedup.org.](http://www.opendedup.org/)
5. MongoDB. [http://www.mongodb.org.](http://www.mongodb.org/)
6. MurmurHash. [https://sites.google.com/site/murmurhash.](https://sites.google.com/site/murmurhash)
7. Окарина Сети. [www.ocarinanetworks.com.](http://www.ocarinanetworks.com/)
8. Пермабит Данные Оптимизация. [www.permabit.com.](http://www.permabit.com/)
9. Быстро. [http://google.github.io/snappy/.](http://google.github.io/snappy/)
10. Архив данных Stack Exchange. [https://archive.org/details/stackexchange.](https://archive.org/details/stackexchange)
11. vBulletin. [https://www.vbulletin.com.](https://www.vbulletin.com/)
12. W3Techs. [http://www.w3techs.com.](http://www.w3techs.com/)
13. Викимедиа Загрузки. [https://dumps.wikimedia.org.](https://dumps.wikimedia.org/)
14. Википедия. [https://www.wikipedia.org/.](https://www.wikipedia.org/)
15. Окна Хранилище Сервер.

[technet.microsoft.com/en-us/library/gg232683(WS.10).aspx.](http://technet.microsoft.com/en-us/library/gg232683(WS.10).aspx)

1. WiredTiger. [http://www.wiredtiger.com/.](http://www.wiredtiger.com/)
2. Дедупликация ZFS. [blogs.oracle.com/bonwick/entry/zfs\_dedup.](http://blogs.oracle.com/bonwick/entry/zfs_dedup)
3. Д. Абади, С. Мэдден и М. Феррейра. Интеграция сжатия и исполнение в ориентированный на столбцы системы баз данных. В *SIGMOD* , страницы 671–682, 2006.
4. С. Альварес. NetApp дедупликация для ФАС и Руководство по развертыванию и внедрению V-Series. 2010.
5. Л. Аронович, Р. Эшер, Э. Бахмат, ЧАС. Битнер, М. Хирш и СТ Кляйн. Проектирование системы дедупликации на основе сходства. В *SYSTOR* , стр. 6, 2009.
6. Дж. Бентли и Д. Макилрой. Сжатие данных с использованием длинного общего струны. В *Данные Сжатие Конференция, 1999. Труды. DCC'99* , страницы 287–295, 1999.
7. D. Bhagwat, K. Eshghi, DD Long и M. Lillibridge. Экстремальное биннинг: масштабируемая параллельная дедупликация для фрагментированных файл резервное копирование. В *ТАЛИСМАНЫ* , страницы 1–9, 2009.
8. С. Бинниг, С. Хильденбранд, и Ф. Фэрбер. Сжатие строк с сохранением порядка на основе словаря для хранилищ столбцов основной памяти. В *SIGMOD* , страницы 283–296, 2009.
9. А. Бродер. О сходстве и содержании документов. Сжатие и Сложность из Последовательности, 1997.
10. А. Бродер. Выявление и фильтрация почти дублирующихся документов. 11-й Ежегодный Симпозиум на Комбинаторное сопоставление образов, 2000.
11. RC Burns и DD Long. Эффективное распределенное резервное копирование с дельта сжатие. В *Труды из пятый семинар по вводу-выводу в параллельных и распределенных системах* , страницы 27–36,

1997.

1. А. Клементс, И. Ахмад, М. Вилайаннур и Дж. Ли. Децентрализованный Дедупликация в САН Кластер Файл Системы. В *USENIX ATC* , 2009.
2. Г. Кобена, С. Абитебул, и А. Мэриан. Обнаружение изменения в документах xml. В *ICDE* , страницы 41–52, 2002.
3. Б. Коллинз-Сассман, Б. Фицпатрик, и М. Пилато. Контроль версий с помощью Subversion. 2004.
4. Г. В. Кормак. Данные сжатие на а база данных система.

*Коммуникации из АКМ* , 28(12):1336–1342, 1985.

1. Б. Дебнат, С. Сенгупта, и Дж. Ли. Chunkstash: Ускорение вверх по строке хранилище дедупликация с использованием вспышка память. В *Ежегодная техническая конференция USENIX* , 2010.
2. П. немецкий и Ж.-Л. Гейлли. Злиб сжатый данные Спецификация формата версии 3.3. Технический отчет, 1996.
3. С. Дубницкий, Л. Грыз, Л. Хельдт, М. Качмарчик, В. Килиан,

П. Стржельчак, и Дж. Щепковский. Гидрастор: А масштабируемое вторичное хранилище. В *FAST* , 2009.

1. С. Дубницкий, Л. Грыз, Л. Хельдт, М. Качмарчик, В. Килиан,

П. Стржельчак, Дж. Щепковский, С. Унгуряну, , и

М. Вельницкий. HYDRAstor: а Масштабируемый Вторичный Хранилище. В

*ФАСТ* , 2009.

1. 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.
2. С. Харизопулос, В. Лян, Д. Дж. Абади, и С. Мэдден. Компромиссы производительности в базах данных, оптимизированных для чтения. В *VLDB* , страницы 487–498, 2006.
3. Б. Айер и Д. Уилхайт. Данные сжатие поддерживать в базах данных. 1994.
4. Н. Джайн, М. Далин, и Р. Тевари. Конусность: Многоуровневый подход к устранению избыточности при синхронизации реплик. В *FAST* , 2005.
5. Э. Леонарди и С. С. Бхоумик. Ксанаду: а система для обнаружения изменения к xml данные в не знающий о деревьях реляционные базы данных. В *SIGMOD* , страницы 1137–1140, 2007.
6. М. Лиллибридж, К. Эшги, Д. Бхагват, В. Деолаликар,

G. Trezise и P. Camble. Разреженная индексация: Крупномасштабная, встроенная дедупликация с использованием выборка и местность. В *ФАСТ* , 2009.

1. Дж. Лелигер. Контроль версий с помощью git: мощные инструменты и методы для совместный программное обеспечение разработка. 2009.
2. Дж. П. Макдональд. Файл система поддерживать для дельта сжатие. Магистерская диссертация, Калифорнийский университет, Беркли, 2000.
3. С. Мишра. Данные сжатие: Стратегия, емкость Планирование и передовой опыт. *Техническая статья SQL Server* , 2009.
4. А. Мутитачароен, Б. Чен и Д. Мазьер. Низкая пропускная способность сеть файл система. В *СОСП* , 2001.
5. Р. Пагх и Ф. Ф. Родлер. Кукушка хеширование. *Журнал Алгоритмов* , 51(2):122–144, 2004.
6. М. Поэсс и Д. Потапов. Данные сжатие в оракул. В

*ВЛДБ* , страницы 937–947, 2003.

1. Х. Пуча, Д. Г. Андерсен и М. Камински. Использование сходства для многоисточниковый загрузки с использованием файл отпечатки рук. В *NSDI* , 2007.
2. У. Пью. Списки пропусков: вероятностная альтернатива сбалансированным деревьям. В *Мастерская на Алгоритмы и Данные Структуры* , страницы 437–449. Springer, 1989.
3. С. Куинлан и С. Дорвард. Венти: А новый подход в архивное хранение. В *FAST* , 2002.
4. М. О. Рабин. *Дактилоскопирование к случайный полиномы* .
5. В. Раман, Г. Атталури, Р. Парикмахер, Н. Чайнани, Д. Калмыцкий,

В. КуландайСами, Дж. Леенстра, С. Лайтстоун, С. Лю, Г. М. Ломан и др. Db2 с ускорением Blu: гораздо больше, чем просто хранилище столбцов. *VLDB* , 6(11):1080–1091, 2013.

1. С. Сакр. Методы сжатия XML: обзор и сравнение. *Журнал из Компьютер и Система Науки* , 75(5):303–322, 2009.
2. P. Shilane, M. Huang, G. Wallace и W. Hsu. Wan-оптимизированный репликация из резервное копирование наборы данных с использованием потокового информирования дельта сжатие. В *БЫСТРЫЙ* , 2012.
3. П. Шилан, Г. Уоллес, М. Хуан и В. Сюй. Дельта-сжатие и дедуплицированный хранилище с использованием Потоковая информация о местоположении. *USENIX Hot Storage* , 2012.
4. К. Шринивасан, Т. Биссон, Г. Гудсон и К. Воруганти. idedup: С учетом задержки, в соответствии данные дедупликация для первичное хранилище. В *FAST* , 2012.
5. М. Стоунбрейкер, Д. Дж. Абади, А. Баткин, X. Чэнь,

М. Черняк, М. Феррейра, Э. Лау, А. Лин, С. Мэдден,

Э. О'Нил, эт ал. Магазин C-Store: а ориентированный на столбцы СУБД. В *VLDB* , страницы 553–564, 2005.

1. Т. Сьюл и Н. Мемон. Алгоритмы для дельта Сжатие и удаленная синхронизация файлов. *Справочник по сжатию без потерь* , 2002.
2. Д. Теодосиу, Ю. Гуревич, М. Манассе и Ж. Поркка. Оптимизация файл репликация над ограниченный пропускная способность сети с использованием удаленного дифференциального сжатия. *Технический представитель.*

*MSR-TR-2006-157, Майкрософт Исследовать* , 2006.

1. В. Ф. Тихий. Ркс–а система для версия контроль. *Программное обеспечение: практика и опыт* , 15(7):637–654, 1985.
2. Д. Трендафилов, Н. Мемон, и Т. Сьюэл. zdelta: Ан эффективный

дельта сжатие инструмент. *Технический Отчет TR-CIS-2002-02, Политехнический университет* , 2002.

1. А. Триджелл. Эффективные алгоритмы сортировки и синхронизации. В *доктор философии диссертация, австралийский Национальный университет* , 2000.
2. Г. Урданета, Г. Пьер и М. Ван Стин. Рабочая нагрузка Википедии анализ для децентрализованный хостинг. *Компьютерные сети* , 53(11):1830–1845, 2009.
3. Ю. Ван, Д. Дж. ДеВитт, и Ж.-Ю. Кай. X-разница: Ан эффективное изменение обнаружение алгоритм для xml документы. В *ICDE* , страницы 519–530, 2003.
4. В. Ся, ЧАС. Цзян, Д. Фэн, и Ю. Хуа. Силос: А

сходство-локальность основанный на почти точный дедупликация Схема с низкими накладными расходами оперативной памяти и высокой пропускной способностью. На *ежегодной технической конференции USENIX* , 2011.

1. Л. Сюй, А. Павло, С. Сенгупа, Дж. Ли и Г. Р. Гангер. Сокращение репликация пропускная способность для распределенный базы данных документов. В *SoCC* , страницы 222–235, 2015.
2. Л. Л. Ты, К. Т. Поллак, и Д. Д. Длинный. Глубокий store: Архитектура архивной системы хранения. В *ICDE* , страницы 804–815, 2005.
3. Б. Чжу, К. Ли и Р. Х. Паттерсон. Избегание узкого места на диске в данные домен дедупликация файл система. В *FAST* , 2008.
4. Дж. Зив и А. Лемпель. Универсальный алгоритм для последовательных данных сжатие. *ИИЭЭ Транзакции на информация теория* , 23(3):337–343, 1977.
5. М. Жуковский, С. Хеман, Н. Нес, и П. Бонц. Суперскалярное сжатие кэша ram-cpu. В *ICDE* , страницы 59–59, 2006.