

Санкт-Петербургский государственный университет

Пилецкий Олег Антонович

Выпускная квалификационная работа

Сравнение методов дедупликации данных

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Гориховский В. И.

Рецензент:
Ведущий разработчик по разработке ПО, ООО «Ядро Центр Программных Разработок»,
Ли С. В.

Санкт-Петербург
2025

Saint Petersburg State University

Oleg Piletskii

Bachelor's Thesis

Comparison of data deduplication methods

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
Sc.D, docent V.I. Gorikhovskii

Reviewer:
Lead software developer at "YADRO" S.V. Li

Saint Petersburg
2025

Оглавление

Введение	4
1. Постановка задачи	6
2. Обзор методов дедупликации и способов сравнения	7
2.1. Алгоритмы чанкирования	7
2.2. Существующие системы сравнения	10
2.3. Выводы	14
3. Реализация ChunkFS и алгоритмов	16
3.1. Архитектура файловой системы	16
3.2. Реализации алгоритмов чанкирования	18
3.3. Продвинутое методы дедупликации	19
3.4. Инструменты бенчмаркинга	20
3.5. Генерация наборов данных	22
3.6. Инструмент командной строки	22
4. Эксперимент	24
4.1. Исследовательские вопросы	24
4.2. Наборы данных	24
4.3. Результаты тестовых экспериментов	26
4.4. Обсуждение результатов	29
4.5. Выводы	30
Заключение	32
Список литературы	34

Введение

Количество информации в мире растет большими темпами. Прогнозируется, что в 2025 объем глобальных данных увеличится до 175 зеттабайт [9, 19]. Как результат, хранить и обрабатывать эту информацию становится все дороже. Один из основных сценариев для хранения данных — резервное копирование, важный инструмент защиты информации от потери по какой-либо причине. Для их формирования и хранения существует множество различных систем [4, 15]. Они, в основном, используют сжатие и дедупликацию, чтобы оптимизировать хранение и, соответственно, затраты.

Дедупликация — метод, позволяющий находить избыточные данные и уменьшать их влияние на затраты на хранение. Он состоит из нескольких этапов: разбиение файла на непрерывные отрезки данных, которые называются чанками, их хэширование, удаление повторений, индексирование и сохранение оставшихся чанков в систему. Самым затратным по времени и ресурсам в этом процессе является разбиение на чанки, поскольку алгоритму необходимо пройти по всему файлу целиком, обычно побайтово, производя в процессе некоторые вычисления. Основным семейством алгоритмов чанкинга является Content Defined Chunking (CDC) [12], который разбивает данные на блоки в зависимости от их содержимого, что позволяет находить достаточно много повторяющихся отрезков данных.

В последнее время появляется все больше алгоритмов CDC [26], однако сравнений их между собой достаточно мало. В отдельных статьях [3, 11] приводятся сравнения алгоритмов по описанной функциональности, а не напрямую. В статьях с самими алгоритмами приводятся сравнения на специально подобранных наборах данных, показывающих преимущества конкретного алгоритма. А систем, направленных на интегрированное сравнение различных алгоритмов, очень мало, что не позволяет в полной мере их сравнивать.

В представленной работе предлагается система для интеграционного сравнения алгоритмов CDC: ChunkFS. Она выполнена в виде мо-

дельной файловой системы и позволяет быстро и удобно тестировать и сравнивать различные алгоритмы и оценивать их производительность и качество дедупликации. Проект инициирован в лаборатории технологий программирования инфраструктурных решений при поддержке компании YADRO.

1. Постановка задачи

Целью работы является разработка системы интеграционного бенчмаркинга методов дедупликации и сравнение алгоритмов CDC. Для её выполнения были поставлены следующие задачи:

1. провести обзор существующих алгоритмов CDC и систем сравнения, чтобы получить набор требований к системе;
2. по полученным требованиям разработать стенд файловой системы, позволяющий исследовать алгоритмы дедупликации, и базу современных алгоритмов CDC (RabinCDC, LeapCDC, SuperCDC, UltraCDC, SeqCDC, FastCDC);
3. реализовать систему бенчмаркинга для алгоритмов, интегрированных в файловую систему;
4. провести интеграционный анализ и сравнение качества алгоритмов CDC.

2. Обзор методов дедупликации и способов сравнения

Чанкирование — самый долгий и затратный по ресурсам процесс в дедупликации. Было придумано множество алгоритмов, направленных на то, чтобы его оптимизировать. Рассмотрим эти алгоритмы и методы их сравнения.

2.1. Алгоритмы чанкирования

Алгоритмы чанкирования делятся на два типа — Fixed Size Chunking (FSC) и Content Defined Chunking (CDC) [25].

Алгоритм Fixed Size Chunking разбивает поток данных на равные по размеру отрезки. Он работает максимально быстро, поскольку не проводит никаких вычислений, однако такой подход приводит к достаточно низкому проценту обнаружения повторов. Кроме того, он подвержен проблеме сдвига границ, которая заключается в том, что если в середину файла добавить хотя бы один байт, то все чанки после этого окажутся "новыми". Она проиллюстрирована на рис. 1.

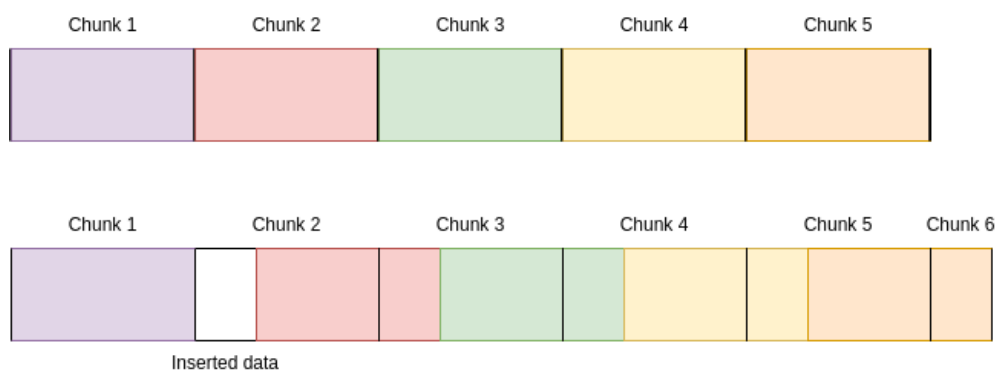


Рис. 1: Проблема сдвига

Алгоритмы семейства Content Defined Chunking, в свою очередь, разбивают поток данных на отрезки в зависимости от содержимого, что позволяет избежать проблемы, связанной с FSC, и находить больше повторов. Теперь, если массив данных локально не изменился, то и чанки, соответствующие ему, также не изменятся. Однако это достигается

ценой скорости работы алгоритма, так как теперь нужно проходить по байтово весь массив данных, и проводить множество вычислений на каждом шаге.

В семействе алгоритмов CDC используется множество подходов к ускорению определения границ чанков, например, можно не учитывать чанки меньше минимального размера, то есть пропускать в начале каждого чанка первые несколько килобайт. Это позволяет довольно сильно увеличить пропускную способность алгоритма, однако негативно влияет на процент найденных повторов, как продемонстрировано авторами алгоритма FastCDC [7].

Ранее был проведен достаточно широкий обзор алгоритмов этого семейства [30]. В нем было продемонстрировано достаточно большое число различных идей, позволяющих ускорить чанкирование и минимально ухудшить, а в большинстве случаев и улучшить процент нахождения повторов. Приведем основные используемые подходы.

- Использование скользящего окна, чтобы находить границы чанков на основе локальности байт. Если хэш скользящего окна удовлетворяет некоему условию, то считается, что граница чанка была найдена. Это один из первых подходов, использующийся в большинстве алгоритмов, таких как RabinCDC [16], GearCDC [5], FastCDC [8]. В UltraCDC [24] также используется скользящее окно, однако условие границы чанка — подходящее расстояние Хэмминга вместо подходящего хэша.
- Хранение какой-то информации об уже встреченных чанках, чтобы не тратить время на прохождение через них. Эти идеи были продемонстрированы в RapidCDC [17] и QuickCDC [28]. RapidCDC позволяет пропускать повторяющиеся группы чанков, QuickCDC — отдельные чанки. Данные подходы предполагают встраивание в другие алгоритмы, основанные на скользящем окне, чтобы ускорить их работу. SuperCDC [21] — уже существующая комбинация алгоритмов FastCDC и RapidCDC, которая, согласно статье, должна работать быстрее, чем исходный FastCDC.

- Пропуск отдельных серий байт, если известно, что они не представляют для нас интереса. Leap-based CDC [13] основан на том, что мы проверяем окно длиной в 24 байта, и также 24 таких же окна перед ним. Если все они соответствуют некоторому условию, то граница чанка найдена. Если же какое-то окно не подходит, то все последующие окна, которые содержат байты из него, также подходить не будут. Поэтому мы можем их все пропустить и перепрыгнуть через несколько байт. Однако, как продемонстрировано авторами метода Jump-based Chunking [2], данный подход обладает несколькими недостатками, в частности, он разрушает непрерывность данных частыми прыжками по, в среднем, 20 байт, и потому встроить его в другие алгоритмы очень сложно. Подход, предложенный ими, для этих целей подходит намного лучше, поскольку он предполагает редкие прыжки на достаточно большие расстояния, если соблюдено определенное условие.
- Поиск групп упорядоченных по возрастанию/убыванию серий байт, представленный в алгоритме SeqCDC [23]. Этот подход — альтернатива скользящему окну, и, согласно исследованию, должен работать сравнимо с другими алгоритмами по скорости. В него также можно встраивать техники сохранения информации о встреченных чанках в памяти.

Эти алгоритмы, согласно исходным публикациям, лучше подходят для разных сценариев. Однако последовательное и стандартизированное сравнение методов осталось вне представленных работ. В статьях сравнение производилось каждый раз на различных наборах данных, иногда подбираемых специально, чтобы показать преимущества одного алгоритма над другим, и, зачастую, ссылки на эти наборы данных нигде не приводятся.

2.1.1. Методы оптимизации качества дедупликации

Кроме алгоритмов чанкирования существуют алгоритмы, позволяющие еще больше оптимизировать занимаемое чанками пространство.

Их представителями являются Similarity Based Chunking (SBC) [20] и Frequency Based Chunking (FBC) [14], которые проводят работу не над самими данными, а над чанками, находя среди них похожие или часто встречающиеся. В некоторых случаях это позволяет добиться огромной экономии занимаемого файлами места. Однако, исследований этих алгоритмов достаточно мало, как и инструментов, позволяющих сравнивать их различные реализации. На момент начала исследований не было ни одного стабильного инструмента, предоставляющего функциональность FBC и SBC для разных алгоритмов CDC.

2.2. Существующие системы сравнения

Существует достаточно мало систем, позволяющих взаимозаменять используемые алгоритмы чанкирования и предоставляющих инструменты для анализа их производительности. Особенно интересны системы, написанные на языке Rust, поскольку проект был инициирован командой TATLIN.BACKUP, ведущей разработку системы на этом языке.

2.2.1. DedupBench

DedupBench¹ [6] — инструмент для сравнения алгоритмов CDC. Он представляет собой набор программ, написанных на C++, и множества вспомогательных скриптов, написанных на Bash и Python. Основной скрипт работает следующим образом: пользователем передаются путь к набору данных и конфигурационный файл, содержащий названия алгоритмов чанкирования и хэширования и иные параметры. После этого проводится чанкирование и, если нужно, хэширование набора данных, внутри используется буфер размером в один мегабайт. Полученные хэши чанков записываются в отдельный файл, который можно отдельно проанализировать другими скриптами. После данного же процесса можно получить такие метрики, как скорость чанкирования и хэширования, количество проанализированных байт и средний размер чанка.

¹<https://github.com/UWASL/dedup-bench>

Для получения информации о коэффициенте дедупликации, энтропии и распределении размеров чанков используются отдельные программы на C++, по одной на каждую метрику. Они принимают на вход путь к файлу с хэшами и выводят на консоль результаты измерений. Эти программы, как и основная, сначала должны быть собраны при помощи make. Для удобства запуска этих программ были написаны отдельные скрипты на Bash. Так, есть отдельные скрипты, последовательно запускающие программы и в конце выдающие все метрики, скрипт, генерирующий конфигурационный файл. Отдельно выделяется скрипт для генерации файла, похожего на два входных файла. Степень "похожести" определяется входным параметром.

Согласно статье, в инструмент достаточно легко встроить нужный алгоритм чанкирования и провести его сравнение с аналогами. В репозитории содержится информация о том, как случайно сгенерировать текст в качестве набора данных, и ссылки на образы виртуальных машин, которые также можно использовать в качестве тестовых данных для сравнения. Содержатся реализации нескольких современных алгоритмов чанкирования: FastCDC, Asymmetric Extremum [1], Rapid Asymmetric Maximum [27], SeqCDC, RabinCDC, GearCDC.

Данный инструмент не рассматривает производительность алгоритмов при взаимодействии с различными типами хранилищ, не может анализировать повторную или многократную запись. Он проводит эту запись только один раз и выдает результат после нее, из-за чего не берет в расчет погрешности в скорости. В нем нет разделения на горячий и холодный запуски. Также, он не поддерживает интерфейсов для FBC и SBC.

Для исследования производительности используются наборы данных, состоящие из образов виртуальных машин, ссылки на которые оставлены в репозитории. Согласно статье, наибольшим коэффициентом дедупликации обладает алгоритм Asymmetric Extremum, из-за чего подчеркивается важность проверки алгоритмов на реальных данных, в противовес тому, что в статьях про отдельные алгоритмы могли быть использованы специальные наборы данных, подчеркивающие преиму-

щества конкретных алгоритмов.

Инструмент не подходит для представленного исследования, поскольку не позволяет работать с физической памятью и провести интеграционное исследование алгоритмов.

2.2.2. CDC Algorithm Tester

CDC Algorithm Tester² [22] — инструмент для тестирования и сравнения алгоритмов CDC. Ядро, содержащее алгоритмы и CLI-инструмент для быстрого тестирования, написано на Rust. В репозитории также содержатся скрипты для сбора метрик и построения графиков на R, а также вспомогательные скрипты на Bash.

В репозитории содержатся реализации алгоритмов чанкирования, основанных на скользящем окне и вычислении его хэша, таких как GearCDC (обычный и нормализованный), RabinCDC, и алгоритмов, основанных на нахождении локальных экстремумов, таких как Asymmetric Extremum, Rapid Asymmetric Maximum, Parity Check of Interval [10]. Для них есть возможность подключить оптимизацию при помощи QuickCDC с несколькими настраиваемыми параметрами.

Для тестового прогона нужно указать алгоритм, его настройки, и входной файл. Можно также указать дополнительные настройки для оптимизации чанкирования на основе QuickCDC. В результате пользователю будет выдан список, состоящий из пар, первый элемент которой — хэш чанка, а второй — его размер.

Пропускная способность алгоритмов была измерена только на случайно сгенерированном наборе данных. Это измерение показало, что лучший алгоритм относительно этой метрики — Rapid Asymmetric Maximum. Для наиболее точного измерения этой метрики были убраны остальные факторы, которые могли помешать. В частности, измерение проводилось в оперативной памяти, а не на диске, а также учитывалось только время на нахождение чанков, другие процессы, такие как хэширование, игнорировались.

²<https://github.com/mrd0ll4r/cdc-algorithm-tester>

В репозитории также есть скрипты для составления других наборов данных: несколько версий ISO-образов ядра Linux, 2000 PDF-файлов, снапшоты веб-сайта за 2022 год, и множество версий нескольких программ с открытым исходным кодом. На них были собраны остальные метрики для каждого алгоритма. Как и в DedupBench, основные эксперименты вместе со сбором метрик проводятся при помощи скриптов `bash` и `R`.

Стоит заметить, что в исследовании приведено достаточно мало современных алгоритмов, например, не учитывается FastCDC, являющийся одним из основных алгоритмов, использующихся в индустрии благодаря своей высокой скорости и простоте реализации.

2.2.3. rdedup

`rdedup`³ — бэкап-утилита, написанная на языке Rust. Она поддерживает различные алгоритмы дедупликации, алгоритмы хэширования, и, кроме этого, предоставляет возможность шифровать данные, имеет встроенный сборщик мусора, поддерживает работу в многопоточном режиме. Направлена в первую очередь не на сравнение алгоритмов, а на формирование бэкапов.

Несмотря на более чем 800 звезд и обширную функциональность, имеет множество недостатков, например, репозиторий не обновлялся в течение двух лет, из-за чего содержит множество security-уязвимостей, выявленных ботом `github-actions`. Сам содержательный код писался более четырех лет назад, и содержит множество решений, которые выглядят устаревшими. Содержит мало комментариев в коде, из-за чего архитектура, которая нигде не описана, мало понятна. Кроме того, в него относительно сложно встраивать другие способы хранения данных, а для корректного встраивания алгоритмов необходимо изменить множество частей кода.

Не содержит в себе никакого сбора метрик, кроме занимаемого пространства. Для того, чтобы встроить анализ производительности и эф-

³<https://github.com/dpc/rdedup>

фективности, необходимо провести много работы над улучшением системы.

2.2.4. zbox

ZboxFS⁴ — встроенная в приложение файловая система с открытым исходным кодом, написанная на языке Rust. Она поддерживает такие функции, как сжатие и шифрование сохраненных данных при помощи lz4 и libsodium, версионирование файлов, транзакционность операций. Кроме того, файлы можно сохранять как в оперативную память, так и на диск, в различные виды хранилищ.

ZboxFS также поддерживает дедупликацию, однако она происходит лишь в рамках одного файла, и, если будут найдены одинаковые чанки в различных файлах, они будут дважды записаны в систему, что не соответствует типичному сценарию использования дедупликации. Была проведена работа над тем, чтобы сделать возможность встраивания различных алгоритмов чанкинга для последующего их сравнения в этой файловой системе [31]. Результаты исследования показали, что из-за больших накладных расходов на сторонние вычисления разницу между алгоритмами увидеть достаточно сложно.

2.3. Выводы

Основные инструменты, которые можно использовать для сравнения алгоритмов на данный момент, — DedupBench и CDC Algorithm Tester. Их объединяет то, что они работают исключительно в оперативной памяти, а основной сценарий работы у них — один прогон алгоритма и дальнейшее получение какой-то информации при помощи отдельных скриптов. Они не берут в расчет погрешности при запуске алгоритмов и влияние отдельных факторов на их производительность. Стоит отметить модульность, присущую этим инструментам, позволяющую легко заменять алгоритмы чанкирования и хэширования.

⁴<https://github.com/zboxfs/zbox>

Производительность алгоритмов в этих инструментах рассматривается в отрыве от любых других факторов, которые также могут на нее повлиять.

Кроме того, они не сравнивают некоторые алгоритмы, которые, согласно их авторам, показывают достаточно высокую производительность: SuperCDC, UltraCDC, LeapCDC.

Таким образом, имеющихся систем не хватает, поскольку они не могут покрыть все случаи использования алгоритмов дедупликации, такие как повторная запись, множественная запись. Кроме того, наборы данных, предоставленные в статьях, не полны и также не покрывают все случаи использования, а способы их получить и тем самым повторить эксперимент плохо описаны. Алгоритмы, реализованные для инструментов, также не покрывают всего многообразия существующих решений.

Основные требования, исходящие из рассмотренных систем:

- модульность относительно алгоритмов CDC, хэширования, способов хранения данных и методов оптимизации качества дедупликации;
- возможность гибко задавать различные сценарии использования, например, повторная запись, дозапись в хранилище;
- реализация недостающих до state-of-art алгоритмов чанкирования;
- относительная легкость получения метрик и возможность расширения системы бенчмарков.

3. Реализация ChunkFS и алгоритмов

Была разработана модельная файловая система ChunkFS, которая позволяет записывать файлы в память и читать их оттуда, по пути дедуплицируя повторяющиеся фрагменты данных при помощи задаваемых пользователем алгоритмов чанкирования и хэширования.

3.1. Архитектура файловой системы

Архитектура библиотеки представлена на рисунке 2. Файловая система предоставляет внешний интерфейс через структуру `FileSystem`, который позволяет проводить стандартные операции, такие как открытие файла, запись и чтение сегмента. Внутри она делится на две не связанные друг с другом части, которые взаимодействуют при помощи хэшей:

- Хранилище данных, которое реализовано при помощи структуры `ChunkStorage`. Оно состоит из объекта, реализующего трейт `Database`, определяющий хранилище ключ-значение, и из объекта, реализующего трейт `Hasher`, определяющий операцию хэширования и тип хэша. В объекте базы данных хранятся пары хэш-чанк.
- Хранилище метаданных, которое реализовано при помощи структуры `FileLayer`. Эти метаданные хранятся в оперативной памяти и содержат в себе имя файла, а также вектор объектов `FileSpan`, указывающих, какой чанк находится на текущем месте в файле. Благодаря этому файл можно прочитать из системы.

Трейт `Database` также является родительским трейтом для более продвинутого трейта `IterableDatabase`, позволяющего итерироваться по базе данных и очищать ее. Благодаря ему появляется доступ к информации о том, какие данные хранятся в базе без знания ключей. Поэтому можно, например, собирать данные о размере базы данных.

При открытии каждого отдельного файла пользователю отдается `FileHandle`, с помощью которого можно проводить операции. Для от-

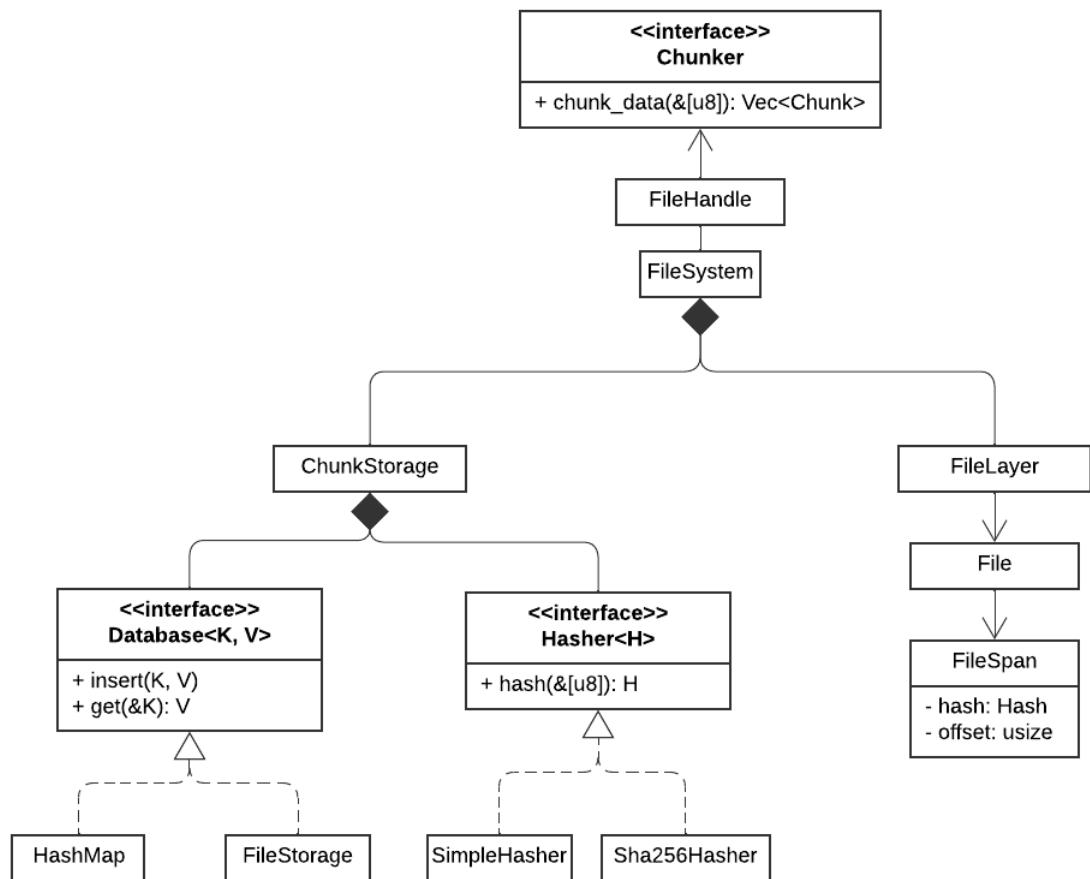


Рис. 2: Архитектура файловой системы

крытия файла на запись требуется также предоставить какой-либо чанкер, который затем будет использоваться в процессе дедупликации. В процессе записи файла проводятся измерения времени хэширования и чанкирования. Их можно получить при закрытии файла. Запись может производиться как из массива данных, так и из какого-либо объекта, предоставляющего потоковое чтение.

Объект, передаваемый пользователем при открытии файла на запись, должен реализовывать трейт `Chunker`, и, что важно, хранить только информацию об алгоритме, который он инкапсулирует. У него есть два основных метода:

- Метод, который получает на вход массив данных и пустой вектор, для которого была выделена память, и возвращает заполненный вектор чанков. Здесь чанк содержит лишь информацию о стартовой позиции и длине сегмента данных, без самих данных.
- Метод, который возвращает ориентировочное количество чанков для данного массива данных, чтобы более точно выделить память для вектора, и его не пришлось увеличивать в будущем. Это используется для того, чтобы время, потраченное на выделение памяти, не учитывалось в измерениях времени чанкирования.

3.2. Реализации алгоритмов чанкирования

Кроме библиотеки `chunkfs` также был разработан крейт `cdc-chunkers`⁵, предоставляющий реализации нескольких алгоритмов CDC, а именно `LeapCDC`, `RabinCDC`, `SeqCDC`, `SuperCDC`, `UltraCDC`.

Для каждого из них при создании можно задать параметры, уникальные для алгоритма, а также минимальный, ожидаемый и максимальный размер чанка — параметры, являющиеся основными для каждого алгоритма. Минимальный важен тем, что при нахождении границы перепрыгивается указанное количество байт, а максимальный — тем, что чанки не могут быть больше этого размера. Без этого парамет-

⁵<https://crates.io/crates/cdc-chunkers>

ра чанки могли бы становиться неограниченно большими. Ожидаемый размер чанка в некоторых алгоритмах помогает сконцентрировать размеры чанков вокруг указанного.

Чтобы провести чанкирование, нужно создать объект, соответствующий алгоритму. Для этого ему на вход передается ссылка на массив данных и объект параметров. Затем итерированием по этому объекту чанкирования можно получать чанки. Это достигается благодаря имплементации встроенного трейта `Iterator` для каждого чанкера.

Для встраивания алгоритмов в `ChunkFS` были написаны структуры, соответствующие каждому чанкеру, реализованному в крейте, а также отдельный чанкер для существующей реализации `FastCDC`⁶. В методе, отвечающем за чанкирование, создается легковесный объект из соответствующего крейта, и затем заполняется вектор, который отдается наружу.

Для доступа к встроенным чанкерам при подключении крейта `chunkfs` требуется включить их при помощи отдельного параметра.

3.3. Продвинутые методы дедупликации

Система также поддерживает продвинутые методы дедупликации, такие как `Frequency Based Chunking` и `Similarity Based Chunking`. Они берут готовые чанки из базы данных, и проделывают с ними некоторые операции, чтобы еще больше сократить занимаемый ими объем.

Для реализации этого в объекте файловой системы дополнительно содержится два объекта:

- Вторая база данных, в которую помещаются обработанные чанки. Они хранятся в другом виде, нежели исходные, но при обращении в нее при помощи исходного хэша чанка он должен вернуться в изначальном виде.
- Скраббер, реализуемый при помощи трейта `Scrub`. Он предоставляет единственный метод, который принимает два аргумента: ис-

⁶<https://crates.io/crates/fastcdc>

ходная база данных, и база данных для обработанных чанков. В себе он инкапсулирует алгоритм, а именно обработку и последующую запись чанков. По завершении он возвращает объект `ScrubMeasurements`, содержащий количество обработанных данных, количество данных, оставшихся в хранилище, и время работы.

Чтобы чанки можно было считывать, как и прежде, исходная база данных хранит не просто массивы данных, а специальный объект `DataContainer`, содержащий в себе либо сам исходный чанк, либо некий набор ключей. Если при чтении система получает набор ключей, то она обращается в базу данных для обработанных чанков.

Имеются некоторые реализации алгоритмов этих семейств и измерения производительности при помощи `ChunkFS`: [29].

3.4. Инструменты бенчмаркинга

Для унифицированной работы над наборами данных была создана структура `Dataset`, хранящая в себе путь к данным, их размер, и название набора. Размер читается из метаданных файла, а при вызове метода `open` возвращается экземпляр структуры `File` стандартной библиотеки, с помощью которой можно затем считывать данные из файла. Так как она имплементирует трейт `Read`, ее можно использовать для потокового чтения напрямую, однако рекомендуется оборачивать ее в структуру `BufReader`, так как она позволяет меньше обращаться к данным на диске и работает более эффективно.

Чтобы получить доступ к этим возможностям, используя трейт `chunkfs` извне, нужно активировать их при помощи cargo-параметра `feature = "bench"`.

В файловую систему был добавлен метод `cdc_dedup_ratio`, позволяющий узнать коэффициент дедупликации данных, которые были чанкированы алгоритмом CDC. Для того, чтобы он работал, необходимо, чтобы лежащая в основе база данных имплементировала трейт `IterableDatabase`.

Для унифицированного сбора метрик была создана дополнительная структура `CDCFixture` в модуле `bench`. Внутри она содержит лишь экземпляр файловой системы, предоставляя несколько дополнительных готовых сценариев, для каждого из которых необходимо предоставить чанкер и набор данных:

- Вручную измерить время на определенном наборе данных. Здесь есть два варианта: прогнать один набор данных на чистой файловой системе несколько раз, либо же прогнать его, записывая его повторно. Эти два сценария также подразумевают верификацию записанных данных.
- Получить коэффициент дедупликации определенного набора данных.
- Получить распределение чанков по размерам на определенном наборе данных. Этот сценарий выдает файл с расширением `json`, который затем можно визуализировать при помощи скрипта.

Для анализа измерений времени была создана структура `TimeMeasurement`, экземпляры которой, благодаря имплементации трейтов `Add`, `AddAssign` и `Sum`, можно складывать различными способами. Кроме них, в конце собирается несколько других метрик, таких как коэффициент дедупликации, количество чанков, средний размер чанка.

Все эти данные после окончания измерения передаются пользователю в виде структуры `MeasureResult`, и их можно записать в файл формата `csv` при помощи встроенного метода. Внутри метода эти данные преобразуются в структуру `SerializableResult`, необходимую, поскольку сложные структуры, как, например, `TimeMeasurement`, не могут быть напрямую записаны в `csv`-файл при помощи используемой библиотеки. Измерения времени хранятся в виде структур типа `Duration`, служащих для не привязанного к единицам измерения хранения. По этой причине нужно указывать, что они хранятся в виде секунд с дробными,

при помощи отдельного процедурного макроса `serde_as` с параметром `"serde_with::DurationSecondsWithFrac<f64>"`.

3.5. Генерация наборов данных

`Fio`⁷, flexible I/O tester — легко конфигурируемый инструмент, позволяющий эмулировать операции ввода-вывода. Он это делает посредством генерации файлов с определенными параметрами, такими как размер, количество дублирующихся данных, количество потоков записи, способ записи, и другие. В данном случае он подходит как генератор наборов данных с заданным количеством повторов.

Для этого была создана функция `generate_with_fio`, позволяющая сгенерировать набор данных с заданными размером и повторами. Внутри она вызывает утилиту, передает ей параметры и возвращает экземпляр структуры `Dataset`.

Кроме этого, была добавлена функция для генерации набора данных с заданным размером, используя некоторое распределение. Для ее реализации был использован сторонний пакет `rand`, предоставляющий трейт `Distribution`, а также несколько реализаций распределений, таких как стандартное, Бернулли, равномерное. В итоге функция принимает название, размер и объект, реализующий трейт `Distribution`.

3.6. Инструмент командной строки

Для быстрого тестирования алгоритмов и запуска сценариев чанкирования был создан инструмент командной строки. Для его реализации была использована библиотека `clap`, позволяющая описывать команды при помощи структур и перечислений самого языка. Он содержится в репозитории как отдельный `cargo`-пакет.

С его помощью можно запускать измерения производительности чанкеров на указанных данных. Есть два варианта запуска и указания параметров:

⁷https://fio.readthedocs.io/en/latest/fio_doc.html

- Для быстрого запуска можно указывать все параметры, такие как выбранное хранилище, чанкер и хэшер, размеры чанков, путь к отчету, прямо через аргументы командной строки. Далее нужно указать команду и дополнительные параметры, такие как путь к набору данных и количество измерений. Для наполнения хранилища данными можно также указать дополнительный параметр и пути к файлам.
- В качестве альтернативы, для быстрой воспроизводимости экспериментов, можно использовать конфигурационные файлы в формате `toml`, содержащие всю ту же информацию. Пример такого файла приведен в репозитории. Его запуск производится при помощи команды `run-config`.

При завершении работы будет создана папка по указанному пути, в которую будет помещена таблица с результатами экспериментов, а также файл формата `json` с распределениями чанков в хранилище, по одному на каждый прогон.

4. Эксперимент

Целью эксперимента является определение лучших алгоритмов чанкирования, а также того, как наборы данных и факторы файловой системы влияют на их работу.

4.1. Исследовательские вопросы

RQ1 : Какие алгоритмы дедупликации показывают лучшую производительность, а какие — лучший коэффициент дедупликации?

RQ2 : Как набор данных влияет на эффективность алгоритма дедупликации?

RQ3 : Зависит ли от скорости чанкирования скорость записи в файловую систему?

Для ответа на эти вопросы используются следующие метрики:

- Скорость записи и чтения из файловой системы.
- Скорость чанкирования и хэширования у отдельных алгоритмов.
- Коэффициент дедупликации — отношение объема исходных данных к объему записанных данных.
- Распределение размеров чанков.

4.2. Наборы данных

Существует достаточно много источников данных, и, поскольку пропускная способность алгоритма и коэффициент дедупликации сильно от них зависят, необходимо составить широкую выборку, которая покрывает множество возможных случаев их использования.

В частности, должны быть продемонстрированы основные сценарии для записи в бэкап-систему. Такие данные отличаются от других наборов большим количеством повторяющихся файлов, что достигается

благодаря, например, наличию множества версий одной и той же программы.

Примеры нагрузок на бэкап-системы:

- Исходные версии ядра Linux разных версий⁸. Они обеспечивают высокую повторяемость данных.
- Образы виртуальных машин⁹.
- Docker-образы популярных инструментов и библиотек, например, Wordpress¹⁰ и nodejs¹¹.
- Enron, набор корпоративных писем и сообщений¹².
- Наборы данных, составленные из моментальных снимков сайтов и множества страниц.

4.2.1. Условия эксперимента

Для текущего исследования были выбраны все реализованные в крейте cdc-chunkers алгоритмы (RabinCDC, SeqCDC, UltraCDC, SuperCDC, LeapCDC), а также FastCDC. Был использован стенд с 40 ГБ оперативной памяти и процессором Intel Core i5-1240P. Операционная система — Ubuntu 24.04. Для хэширования чанков использовался алгоритм SHA-256. В качестве хранилищ использовались оперативная память и rocksdb¹³. Параметры для чанков: минимальный размер — 4 КБ, ожидаемый — 8 КБ, максимальный — 16 КБ.

Для текущего исследования были выбраны следующие наборы данных:

- Образы Arch Linux от 1 и 15 декабря 2024 (4 ГБ)
- Linux версии 6.12, разные кандидаты в релизы (12.4 ГБ)

⁸<https://www.kernel.org/>

⁹<https://docs.openstack.org/image-guide/obtain-images.html>

¹⁰https://hub.docker.com/_/wordpress

¹¹https://hub.docker.com/_/node

¹²<https://www.cs.cmu.edu/~enron/>

¹³<https://docs.rs/rocksdb/latest/rocksdb/>

- Данные OpenStreetMaps (OSM) для России, два полных бэкапа (8.5 ГБ)
- Три docker-образа wordpress и nodejs (821 МБ и 3.4 ГБ)

Поскольку алгоритмы чанкирования оперируют на отдельных файлах, данные наборы были упакованы в tar-файлы.

4.3. Результаты тестовых экспериментов

В таблицах, показывающих пропускную способность алгоритмов и файловой системы, в ячейках находится среднее значение за 5 измерений и среднеквадратичное отклонение. На каждой итерации база данных очищалась. Жирным в таблицах выделены лучшие значения для каждого набора данных.

В таблице 1 представлена чистая скорость чанкирования алгоритмов, среднеквадратичное отклонение составляет менее процента. FastCDC и SeqCDC более устойчивы, чем остальные. SuperCDC показывает высокую неустойчивость, но также и лучшие результаты.

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	1114 $\pm$ 16	3276 $\pm$ 11	5510 $\pm$ 17	2795 $\pm$ 17	3470 $\pm$ 5	2652 $\pm$ 24
VM	1170 $\pm$ 15	3466 $\pm$ 1	15071 $\pm$ 23	2782 $\pm$ 13	3619 $\pm$ 17	1772 $\pm$ 1
OSM	1174 $\pm$ 7	3393 $\pm$ 4	15311 $\pm$ 49	2762 $\pm$ 5	3603 $\pm$ 6	1708 $\pm$ 2
Nodejs	1211 $\pm$ 6	2985 $\pm$ 8	8891 $\pm$ 24	3016 $\pm$ 15	3556 $\pm$ 7	1710 $\pm$ 4
Wordpress	1216 $\pm$ 3	3600 $\pm$ 14	6789 $\pm$ 26	3394 $\pm$ 54	3429 $\pm$ 5	1561 $\pm$ 2

Таблица 1: Чистая скорость чанкирования, МБ/с

В таблицах 2 и 3 представлены, соответственно, средний размер чанка и количество уникальных чанков для каждого набора данных и алгоритма. На каждом прогоне получались одинаковые данные. Вместе они описывают итоговый размер словаря чанков.

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	14199	11173	7124	8515	11559	6481
VM	12766	9643	5889	7916	10512	6935
OSM	12732	9766	5865	7881	10483	7143
Nodejs	12271	13148	6384	8436	10998	8567
Wordpress	12250	11759	6550	8576	11221	8497

Таблица 2: Средний размер чанка

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	392791	470397	698865	545935	442505	710974
VM	75643	119543	196263	120873	91370	139339
OSM	666777	869310	1447308	1077075	809766	1188360
Nodejs	133895	130498	257096	185195	146949	186991
Wordpress	31431	32861	60017	43547	34044	44593

Таблица 3: Количество уникальных чанков

В таблице 4 представлен коэффициент дедупликации алгоритмов. Он показывает, насколько словарь чанков по размеру отличается от исходного набора данных. На наборе данных OSM повторяющихся данных почти не было.

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	2.22	2.35	2.48	2.66	2.42	2.68
VM	4.18	3.50	3.49	4.21	4.20	4.17
OSM	1	1	1	1	1	1
Nodejs	2.07	1.98	1.94	2.18	2.10	2.12
Wordpress	2.13	2.13	2.02	2.20	2.15	2.17

Таблица 4: Коэффициент дедупликации

В таблицах 5 и 6 показана, соответственно, скорость записи и чтения в файловой системе при основном хранилище на базе оперативной памяти. RabinCDC показывает высокую нестабильность при работе, связанную с долгим вычислением хэша скользящего окна, тогда как остальные алгоритмы работают стабильно. При чтении появляется выраженная зависимость скорости работы от набора данных.

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	472 ± 62	734 ± 3	797 ± 2	689 ± 17	744 ± 2	684 ± 4
VM	488 ± 100	757 ± 1	893 ± 1	715 ± 1	761 ± 15	622 ± 1
OSM	506 ± 23	723 ± 1	854 ± 2	685 ± 1	731 ± 1	594 ± 1
Nodejs	515 ± 47	724 ± 10	850 ± 2	723 ± 1	756 ± 1	611 ± 1
Wordpress	514 ± 48	755 ± 2	824 ± 2	742 ± 3	749 ± 1	592 ± 1

Таблица 5: Скорость записи в ChunkFS (RAM), МБ/с

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	1370 ± 58	1416 ± 20	1408 ± 10	1415 ± 25	1425 ± 14	1391 ± 27
VM	1973 ± 4	1948 ± 2	1891 ± 5	1926 ± 1	1952 ± 4	1911 ± 2
OSM	1337 ± 6	1332 ± 2	1309 ± 2	1321 ± 1	1327 ± 1	1317 ± 2
Nodejs	1967 ± 10	1986 ± 7	1901 ± 4	1935 ± 4	1965 ± 5	1941 ± 3
Wordpress	3277 ± 11	3271 ± 10	3095 ± 14	3180 ± 11	3256 ± 16	3184 ± 7

Таблица 6: Скорость чтения в ChunkFS (RAM), МБ/с

В таблицах 7 и 8 представлена пропускная способность файловой системы на запись и чтение при использовании хранилища на основе rocksdb. Здесь пропускная способность гораздо ниже, чем при записи в оперативную память. Кроме того, на производительность могут влиять эффекты rocksdb, такие как регулярный сброс таблиц из оперативной памяти на диск и последующая компактификация.

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	46 ± 2	39 ± 5	37 ± 2	40 ± 2	39 ± 2	38 ± 2
VM	40 ± 10	45 ± 1	45 ± 1	44 ± 3	45 ± 3	41 ± 1
OSM	21 ± 3	23 ± 1	23 ± 1	23 ± 2	21 ± 1	18 ± 6
Nodejs	77 ± 39	61 ± 6	49 ± 17	48 ± 14	60 ± 2	61 ± 4
Wordpress	88 ± 28	136 ± 34	133 ± 10	147 ± 36	160 ± 20	160 ± 27

Таблица 7: Скорость записи в ChunkFS (rocksdb), МБ/с

	Rabin	Ultra	Super	Leap	Fast	Seq
Linux	269 ± 43	228 ± 53	233 ± 28	249 ± 30	220 ± 42	190 ± 30
VM	421 ± 80	323 ± 31	272 ± 43	313 ± 20	350 ± 38	329 ± 26
OSM	204 ± 40	154 ± 14	168 ± 46	180 ± 27	177 ± 40	162 ± 26
Nodejs	203 ± 27	169 ± 13	152 ± 20	174 ± 50	192 ± 19	182 ± 10
Wordpress	395 ± 11	417 ± 39	342 ± 19	389 ± 17	390 ± 12	363 ± 15

Таблица 8: Скорость чтения в ChunkFS (rocksdb), МБ/с

4.4. Обсуждение результатов

SuperCDC — самый быстрый алгоритм чанкирования из представленных, обгоняющий другие в чистой скорости чанкирования в несколько раз. На наборах данных, состоящих из образов виртуальных машин и бэкапов карт он показывает наилучшие результаты из существующих, обгоняя другие алгоритмы минимум в три раза. Такие результаты могут объясняться маленьким средним размером чанка и огромным их количеством. То есть алгоритм при работе пропускал очень много фрагментов по 4 КБ и в действительности проходил по очень маленькому количеству байт. SuperCDC — более оптимизированная версия FastCDC, что подтверждается замерами скорости.

За этим алгоритмом идут FastCDC и UltraCDC, показывающие сравнимые результаты. Они тоже пропускают 4 КБ в начале каждого чанка, но границу находят сильно позднее, что отражено в среднем размере чанка, и соответственно, проходят по большему количеству байт. В целом, для каждого конкретного алгоритма видна такая зависимость: чем меньше итоговый средний размер чанка, тем больше скорость обработки. Но, как видно, UltraCDC на наборе данных Nodejs имеет больший размер чанка, чем RabinCDC на большинстве наборов, но работает почти в три раза быстрее него. Поэтому нет зависимости от среднего размера чанков относительно разных алгоритмов.

Относительно коэффициента дедупликации, SuperCDC справляется несколько хуже остальных алгоритмов, обгоняя некоторые алгоритмы лишь на наборе данных, состоящем из ядер Linux. Здесь лучше всех показывает себя LeapCDC, за которым следуют SeqCDC и FastCDC. На наборе данных OpenStreetMaps коэффициент дедупликации равен единице, что значит, что повторяющихся чанков было найдено крайне мало. Чистая скорость чанкирования из-за этого не пострадала.

При записи в файловую систему на RAM-хранилищ SuperCDC также обгоняет остальные алгоритмы, однако не может показать такие же выдающиеся результаты. За ним идут FastCDC и UltraCDC, так же, как и в случае с чистой пропускной способностью алгоритмов. Стоит

отметить высокую нестабильность RabinCDC, исходящую, вероятно, из очень долгих и тяжелых вычислений для определения границы чанка. Остальные алгоритмы стабильны.

При записи в файловую систему на хранилище на основе rocksdb результаты сильно хуже. Здесь алгоритмы ведут себя достаточно нестабильно, по-разному реагируя на разные наборы данных. Так, RabinCDC лучше всех справляется с записью набора данных Linux и Nodejs, тогда как с Wordpress лучше справляются FastCDC и SeqCDC, а на VM и OSM практически одинаковые результаты показывают сразу четыре алгоритма. Wordpress показывает высокую нестабильность из-за малого размера.

Скорость чтения, по большей части, зависит от количества чанков, поскольку в процессе чтения идет обращение ко всем нужным чанкам, находящимся в хранилище, их копирование и чтение. При чтении данных из rocksdb происходит копирование данных с диска, что вызывает большую вариативность в скорости, а при чтении из оперативной памяти она меньше. Самым эффективным алгоритмом здесь оказывается RabinCDC, поскольку, в среднем, производит чанки большего размера, из-за чего их получается меньше, а за ним идет LeapCDC.

4.5. Выводы

Операция чтения для бэкапов востребована сильно меньше, чем операция записи, поскольку они создаются, в основном, для авайриных случаев. Относительно записи лучше всего справляется SuperCDC, показывая сильно лучшие результаты, чем остальные алгоритмы, особенно при чистой записи. Эта зависимость сохраняется при записи в хранилище различных наборов данных, однако выражена слабее. В реальных бэкап-системах можно постараться избавиться от накладных расходов, ограничивающих работу алгоритма.

Относительно коэффициента дедупликации лучше всего справляется LeapCDC, но обгоняет SuperCDC не слишком сильно, поэтому, если важна скорость, SuperCDC все равно будет приоритетом.

Если в наборе данных мало повторяющихся данных, как, например, в OpenStreetMaps, скорость записи в оперативную память не страдает, однако чанков в итоге получается больше, и чтение замедляется. Запись на диск также замедляется из-за большего количества чанков.

Стоит также отметить, что при использовании методов оптимизации дедупликации, таких как FBC и SBC, результаты, связанные с коэффициентом дедупликации, могут стать более похожими друг на друга, однако это находится вне рамок представленной работы.

Заключение

В результате работы были выполнены следующие задачи:

- Был проведен обзор алгоритмов чанкирования и существующих систем сравнения. Рассмотрены подробно системы DedupBench и CDC Algorithm Tester. Выделены основные требования к системе сравнения.
- Были разработаны файловая система ChunkFS¹⁴ и реализованы алгоритмы Content Defined Chunking¹⁵, такие как SuperCDC, LeapCDC, UltraCDC, SeqCDC, RabinCDC.
- Была разработана система сравнения и бенчмаркинга алгоритмов на основе файловой системы¹⁶, создан инструмент командной строки для удобного использования.
- Было проведено сравнение алгоритмов при помощи разработанного инструмента, показавшее, что для задач быстрой дедупликации больше всего подходит SuperCDC, тогда как для наилучшего сжатия лучше подходит LeapCDC.

Система, поскольку является модульной, также используется для исследований других студентов в этой тематике. Так, например, отдельные студенты разрабатывают хранилища на основе B+ и LSM-деревьев, тестируя их производительность на этой системе. Другие занимаются исследованиями в сфере методов оптимизации дедупликации, таких как FBC и SBC. Также, система была использована на зимней школе YADRO 2025 для исследований производительности и бенчмаркинга на платформах RISC-V¹⁷.

Доклад по представленной теме принят на конференцию FRUCT¹⁸ и опубликован на IEEE Xplore [18]. На конференции были обсуждены

¹⁴<https://crates.io/crates/chunkfs>

¹⁵<https://github.com/Piletskii-Oleg/rust-chunking>

¹⁶<https://github.com/Piletskii-Oleg/chunkfs>

¹⁷<https://careers.yadro.com/risc-v-school/#project-work>, дата обращения 24.05.2025, проект 17

¹⁸<https://www.fruct.org/conferences/37/program/>

такие вопросы, как системные требования для работы ChunkFS и сравнение алгоритмов на дисковых хранилищах, а не в оперативной памяти. Были предложены дальнейшие исследования на тему того, возможно ли встроить ChunkFS в распределенные файловые системы, такие как HDFS, и как можно совмещать дедупликацию, необходимую для уменьшения занимаемого пространства, и избыточность данных, необходимую для защиты от потерь.

Список литературы

- [1] AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication / Yucheng Zhang, Hong Jiang, Dan Feng et al. // 2015 IEEE Conference on Computer Communications (INFOCOM) / IEEE. — 2015. — P. 1337–1345.
- [2] Accelerating Content-Defined Chunking for Data Deduplication Based on Speculative Jump / Xiaozhong Jin, Haikun Liu, Chencheng Ye et al. // [IEEE Transactions on Parallel and Distributed Systems](#). — 2023. — Vol. 34, no. 9. — P. 2568–2579.
- [3] A Comprehensive Study of the Past, Present, and Future of Data Deduplication / Wen Xia, Hong Jiang, Dan Feng et al. // [Proceedings of the IEEE](#). — 2016. — Vol. 104, no. 9. — P. 1681–1710.
- [4] [Dataset Similarity Detection for Global Deduplication in the DD File System](#) / Tony Wong, Smriti Thakkar, Kao-Feng Hsieh et al. // 2023 IEEE 39th International Conference on Data Engineering (ICDE). — 2023. — P. 3322–3335.
- [5] Ddelta: A deduplication-inspired fast delta compression approach / Wen Xia, Hong Jiang, Dan Feng et al. // [Performance Evaluation](#). — 2014. — Vol. 79. — P. 258–272. — Special Issue: Performance 2014. URL: <https://www.sciencedirect.com/science/article/pii/S0166531614000790>.
- [6] [DedupBench: A Benchmarking Tool for Data Chunking Techniques](#) / Alan Liu, Abdelrahman Baba, Sreeharsha Udayashankar, Samer Al-Kiswany // 2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). — 2023. — P. 469–474.
- [7] The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems / Wen Xia, Xiangyu Zou, Hong Jiang et al. // [IEEE Transactions on Parallel and Distributed Systems](#). — 2020. — Vol. 31, no. 9. — P. 2017–2031.

- [8] The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems / Wen Xia, Xiangyu Zou, Hong Jiang et al. // [IEEE Transactions on Parallel and Distributed Systems](#). — 2020. — Vol. 31, no. 9. — P. 2017–2031.
- [9] The Design of Fast and Lightweight Resemblance Detection for Efficient Post-Deduplication Delta Compression / Wen Xia, Lifeng Pu, Xiangyu Zou et al. // [ACM Trans. Storage](#). — 2023. — . — Vol. 19, no. 3. — 30 p. — URL: <https://doi.org/10.1145/3584663>.
- [10] Function of content defined chunking algorithms in incremental synchronization / Changjian Zhang, Deyu Qi, Wenlin Li, Jing Guo // [IEEE Access](#). — 2020. — Vol. 8. — P. 5316–5330.
- [11] Jehloul Hashem Bedr, George Loay E. Big data backup deduplication: A survey // [Int. J. Sci. Res. Sci. Eng. Technol.](#) — 2022. — P. 174–191.
- [12] Kaur Gagandeep, Devgan Mandeep. Data Deduplication Methods: A Review // [International Journal of Information Technology and Computer Science](#). — 2017. — 10. — Vol. 9. — P. 29–36.
- [13] [Leap-based Content Defined Chunking — Theory and Implementation](#) / Chuanshuai Yu, Chengwei Zhang, Yiping Mao, Fulu Li // 2015 31st Symposium on Mass Storage Systems and Technologies (MSST). — 2015. — P. 1–12.
- [14] Lu Guanlin, Jin Yu, Du David. [Frequency Based Chunking for Data De-Duplication](#). — 2010. — 09. — P. 287 – 296.
- [15] Maher Reda, Nasr Omar A. DropStore: A Secure Backup System Using Multi-Cloud and Fog Computing // [IEEE Access](#). — 2021. — Vol. 9. — P. 71318–71327.
- [16] Muthitacharoen Athicha, Chen Benjie, Mazières David. [A Low-Bandwidth Network File System](#). — Vol. 35. — 2001. — 12. — P. 174–187.

- [17] Ni Fan, Jiang Song. [RapidCDC: Leveraging Duplicate Locality to Accelerate Chunking in CDC-Based Deduplication Systems](#) // Proceedings of the ACM Symposium on Cloud Computing. — SoCC '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 220–232. — URL: <https://doi.org/10.1145/3357223.3362731>.
- [18] Piletskii Oleg, Gorikhovskii Viacheslav. [ChunkFS: A Tool for Data Deduplication Methods Comparison](#) // 2025 37th Conference of Open Innovations Association (FRUCT). — 2025. — P. 221–227.
- [19] Prajapati Priteshkumar, Shah Parth. A Review on Secure Data Deduplication: Cloud Storage Security Issue // [Journal of King Saud University - Computer and Information Sciences](#). — 2022. — Vol. 34, no. 7. — P. 3996–4007. — URL: <https://www.sciencedirect.com/science/article/pii/S1319157820305140>.
- [20] Similarity based deduplication with small data chunks / L. Aronovich, R. Asher, D. Harnik et al. // [Discrete Applied Mathematics](#). — 2016. — Vol. 212. — P. 10–22. — Stringology Algorithms. URL: <https://www.sciencedirect.com/science/article/pii/S0166218X15004795>.
- [21] [SuperCDC: A Hybrid Design of High-Performance Content-Defined Chunking for Fast Deduplication](#) / Binzhaoshuo Wan, Lifeng Pu, Xiangyu Zou et al. // 2022 IEEE 40th International Conference on Computer Design (ICCD). — 2022. — P. 170–178.
- [22] Gregoriadis Marcel, Balduf Leonhard, Scheuermann Björn, Pouwelse Johan. A Thorough Investigation of Content-Defined Chunking Algorithms for Data Deduplication. — 2024. — [2409.06066](#).
- [23] Udayashankar Sreeharsha, Baba Abdelrahman, Al-Kiswany Samer. SeqCDC: Hashless Content-Defined Chunking for Data Deduplication // Proceedings of the 25th International Middleware Conference. — 2024. — P. 292–298.
- [24] [UltraCDC: A Fast and Stable Content-Defined Chunking Algorithm for Deduplication-based Backup Storage Systems](#) / Peng Zhou,

- Zhenyu Wang, Wen Xia, Haotong Zhang // 2022 IEEE International Performance, Computing, and Communications Conference (IPCCC). — 2022. — P. 298–304.
- [25] Venish A, Siva Sankar K. Study of chunking algorithm in data deduplication // Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015, Volume 2 / Springer. — 2016. — P. 13–20.
- [26] Viji D, Revathy D. Comparative analysis for content defined chunking algorithms in data deduplication // Webology. — 2021. — Vol. 18, no. SpecialIssue2. — P. 255–268.
- [27] Widodo Ryan NS, Lim Hyotaek, Atiquzzaman Mohammed. A new content-defined chunking algorithm for data deduplication in cloud storage // Future Generation Computer Systems. — 2017. — Vol. 71. — P. 145–156.
- [28] Xu Zhen, Zhang Wenbo. [QuickCDC: A Quick Content Defined Chunking Algorithm Based on Jumping and Dynamically Adjusting Mask Bits](#) // 2021 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom). — 2021. — P. 288–299.
- [29] Дмитриевцев Алексей, Гориховский Вячеслав. Повышение качества дедупликации с помощью Frequency Based Chunking // Современные технологии в теории и практике программирования. — 2025. — P. 307–309.
- [30] Олег Пилецкий. Сравнение алгоритмов дедупликации при помощи инструмента на основе ZboxFS. — 2024. — URL: https://se.math.spbu.ru/thesis/texts/Piletskij_Oleg_Antonovich_Autumn_practice_3rd_year_2023_text.pdf.
- [31] Пилецкий Олег, Гориховский Вячеслав. Разработка инструмента для сравнения алгоритмов Content Defined Chunking на

Rust // Современные технологии в теории и практике
программирования. — 2024. — Р. 253–254.