

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б11-мм

Отчёт по учебной практике

Дмитриевцев Алексей Сергеевич

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2025

Оглавление

Введение	3
1. Постановка задачи	4
2. Обзор	5
2.1. Алгоритм FBC	5
2.2. Обзор используемых технологий	6
3. Описание решения	7
3.1. Параллельный подход к FBC	7
3.2. Выбор подхода синхронизации	8
4. Эксперимент	9
4.1. Условия эксперимента	9
4.2. Исследовательские вопросы	9
4.3. Метрики	9
4.4. Результаты	10
Заключение	12
Список литературы	13

Введение

Дедупликация данных – подход к сжатию данных, направленный на устранение повторяющихся копий на одном или группе носителей с целью снижения накладных расходов на хранение и передачу информации, широко используется в системах хранения резервных копий.

Content-Defined Chunking (CDC) – метод разделения на блоки для первичной дедупликации, в котором данные используются для определения границ будущих чанков. Для улучшения дедупликации существует подход Frequency Based Chunking [1], который основывается на анализе частот подчанков в данном CDC-разбиении и производит его измельчение. Такой подход позволяет значительно увеличить коэффициент дедупликации, однако на порядки замедляет скорость записи.

В представленной работе предлагается метод оптимизации FBC, основанный на параллельном вычислении частот. Этот метод интегрирован в ChunkFS [2] – модельную файловую систему, предназначенную для комплексного анализа методов дедупликации. Основной целью работы является исследование возможности применения FBC в системах резервного копирования, в особенности TATLIN.BACKUP от компании YADRO.

1. Постановка задачи

Целью работы является разработка и интеграция в ChhunkFS улучшения для алгоритма дедупликации Frequency Based Chunking. Использование вычисления частот в нескольких потоках должно улучшить скорость записи в файловой системе, поддерживающей такой подход. Для её выполнения были поставлены следующие задачи:

1. Интеграция улучшения алгоритма в файловую систему:
 - (a) реализация параллельного вычисления частот с помощью различных подходов к синхронизации;
 - (b) сравнительный анализ эффективности подходов к реализации улучшения;
2. Проведение экспериментального исследования для оценки влияния коллизий на эффективность алгоритма (раздел 4).

2. Обзор

2.1. Алгоритм FBC

Этапы работы Frequency Based Chunking представлены на 1. Скользящее окно проходит по чанкам, полученным после работы CDC-алгоритма. При нахождении частого подчанка, он добавляется в словарь. На основе частотного анализа проводится дальнейшее дробление: некоторые исходные CDC-чанки разделяются на несколько новых чанков меньшего размера, которые были найдены на предыдущем этапе. Таким образом, вследствие измельчения чанков, мы находим большее число дубликатов, что приводит к повышению качества дедупликации.

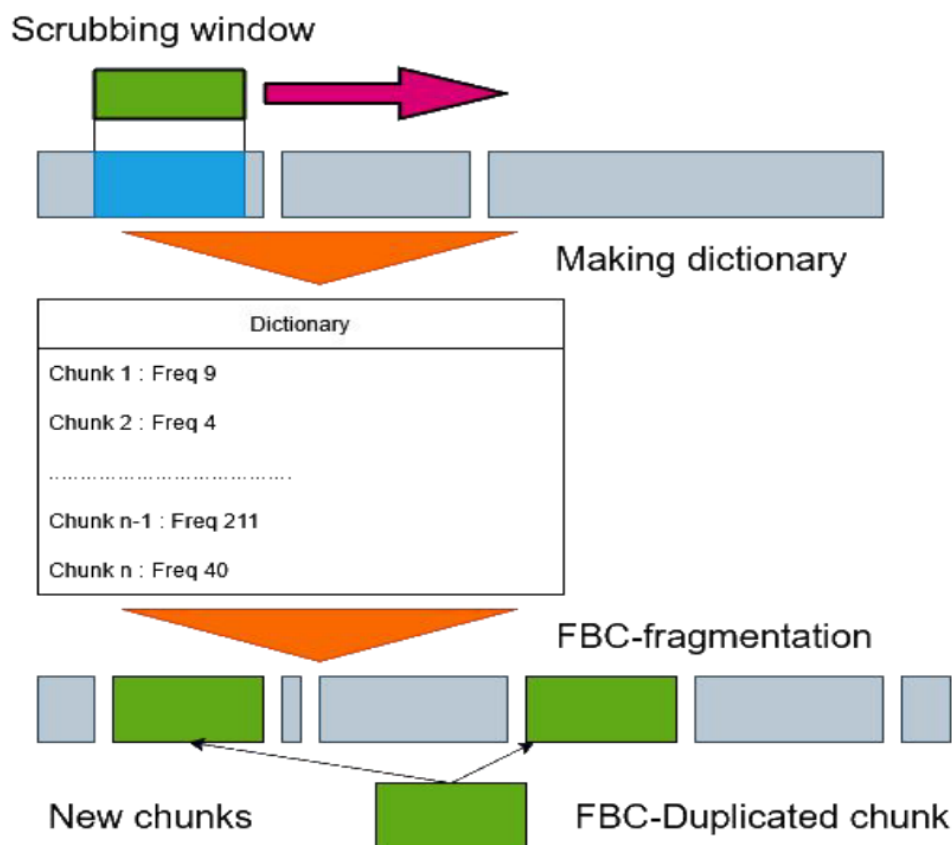


Рис. 1: Этапы алгоритма Frequency Based Chunking

2.2. Обзор используемых технологий

Для конструктивного сравнения необходимо иметь возможность получать информацию об изменении коэффициента дедупликации и времени работы в зависимости от параметров FBC. Интеграционный анализ было принято решение проводить с помощью файловой системы ChunkFS [2], поскольку из представленных только она располагает достаточной функциональностью, которая позволяет сравнивать различные реализации алгоритмов. Руководствуясь тем, что ChunkFS и более ранние реализации алгоритма написаны на языке Rust, для реализации улучшения алгоритма был выбран именно этот язык программирования.

3. Описание решения

Ранее была реализована и интегрирована в ChunkFS базовая[?] версия алгоритма. Проведенный комплексный анализ эффективности FBC выявил ряд особенностей алгоритма:

- измельчение чанков приводит к вплоть до стократному снижению скорости записи
- системы хранения резервных копий предполагают редкие, но крупные периоды записи данных
- запись может производиться независимо от чтения уже размещенных данных
- FBC – скраббирующий алгоритм, но при этом требуется существенное улучшение производительности записи
- чтение может легко выполняться параллельно, так как не является деструктивной операцией

Одной из наиболее трудоемких операций в алгоритме FBC является вычисление частот, поэтому наиболее естественным улучшением производительности является их параллельное вычисление, не предполагавшееся в оригинальной работе.

3.1. Параллельный подход к FBC

Алгоритм проходит чанки скользящим окном, периодически добавляя в словарь новые записи о наиболее частых подчанках из данного разбиения. Данные действия можно выполнять параллельно, обеспечив конкурентный доступ к словарю частых чанков, однако подобный подход к вычислению частот предвещает недостаток – при параллельной записи могут возникать коллизии. Источником коллизий является одновременное возникновение нового подчанка в разных потоках, затрудняющее корректное определение частоты этого подчанка. Из-за

этого может не произойти разбиение самих чанков. Однако выдвигается предположение, что коллизии несущественно влияют на качество дедупликации. Если подчанк действительно частый, он встретится в разных потоках неоднократно и будет добавлен в словарь. Чанки, уже находящиеся в FBC словаре, повторно в него не добавляются. Поэтому не будут участвовать в подразбиении только относительно редкие подчанки.

3.2. Выбор подхода синхронизации

Для решения задачи необходимо обеспечить эффективный конкурентный доступ к словарю FBC-чанков. Был проведен анализ эффективности различных реализаций: `Mutex<HashMap>`, `RwLock<HashMap>`, `DashMap`

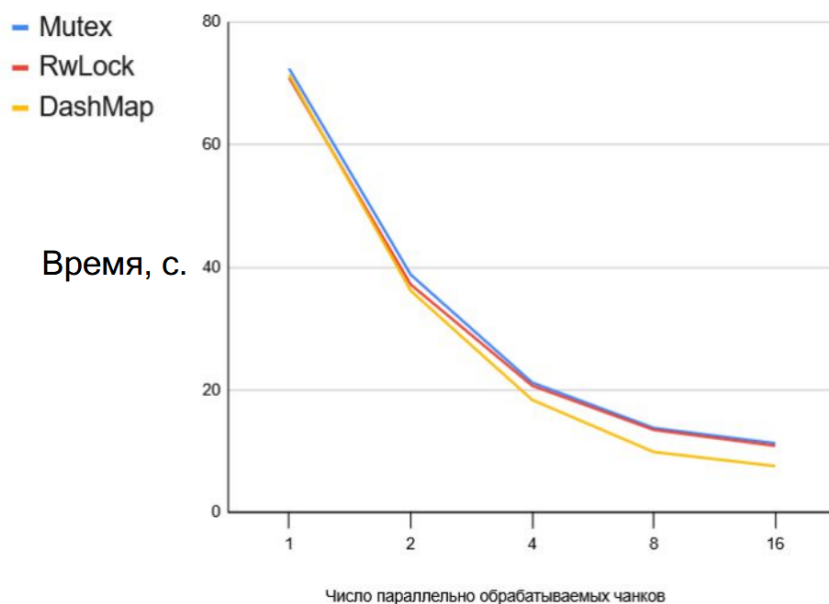


Рис. 2: Скорость вычисления частот для разных вариантов синхронизации, на фрагменте 20% от Enron, среднее за 10 запусков на устройстве из параграфа 4

Наибольшую эффективность показал вариант с `DashMap`, поэтому выбор был сделан именно в пользу данной реализации. как составление словаря, так и поиск чанков-кандидатов для дробления.

4. Эксперимент

4.1. Условия эксперимента

ОС Linux, дистрибутив Manjaro 24.1.1, процессор Intel® Core™ i7-12650H, объем оперативной памяти 16 Гиб. Чтобы сравнить, как алгоритмы справляются с задачей, были выбраны следующие наборы данных:

- Enron email (1,4 Гб)
- Две версии ядра Linux (3.4.6, 3.4.7) (945,3 Мб)

Алгоритмы первичной дедупликации RabinCDC и SuperCDC. Исследовались запуски на 1, 2, 4, 8 и 16 потоках, в которых весь датасет загружался в словарь и вычислялось время, затраченное на составление словаря, и объем такого словаря. Такие запуски проводились по 10 раз, из которых выбирались худшие, средние и лучшие результаты. Для анализатора частот был выбран размер чанка 128 байт.

4.2. Исследовательские вопросы

RQ1 : как влияет изменение числа потоков на качество дедупликации?

RQ2 : насколько эффективно параллельное вычисление частот чанков?

4.3. Метрики

Метрики: время записи целого датасета в секундах и коэффициент дедупликации (считается как начальный размер данных, деленный на размер данных после обработки).

4.4. Результаты

4.4.1. RQ1

На 3 приведены результаты влияния коллизий на качество дедупликации, для датасета Enron emails слева и ядер Linux справа. С ростом числа потоков наблюдается снижение коэффициента дедупликации за счет коллизий. При этом большее снижение заметно на датасете Enron emails. Это можно объяснить малым количеством дубликатов в этом датасете и, соответственно, меньшей частотой встречи подчанков. В датасете с ядрами Linux более вероятная последующая повторная встреча подчанка может нивелировать коллизию. Отсюда можно сделать вывод, что коллизии не оказывают существенного влияния на качество дедупликации: около 1-5% при записи новых чанков. Что значительно меньше выигрыша от использования FBC: около 30-40% [3].

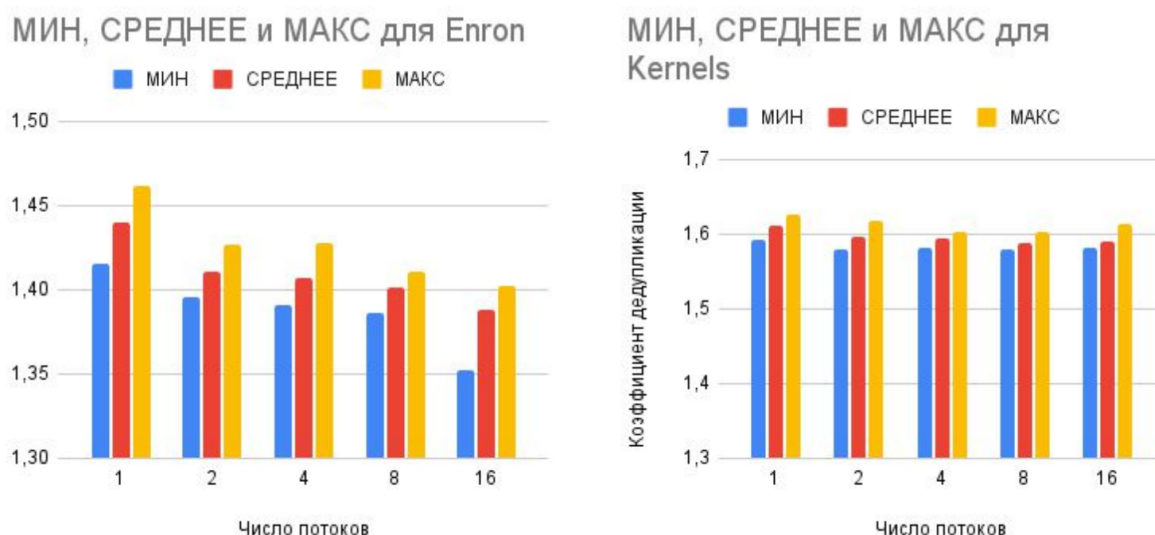


Рис. 3: Коэффициент дедупликации при параллельном исполнении

4.4.2. RQ2

Результаты оценки времени составления словаря представлены на 4. Разница между лучшими и худшими показателями незначительна (не превышает 2%), поэтому на графике приведены только средние значения. Параллельная реализация показала существенный прирост

в скорости составления словаря частых чанков, вплоть до 10 раз на 16 потоках. Однако, возможно, прирост эффективности между использованием 8 и 16 потоков является уже не настолько существенным, чтобы тратить столько ресурсов, которые можно было бы использовать на оптимизацию других частей ChunkFS. Например, на оптимизацию скорости записи чанков в хранилище. Более того, использование 16 потоков приводит к снижению коэффициента дедупликации ввиду возникновения большего количества коллизий.

Время работы алгоритма

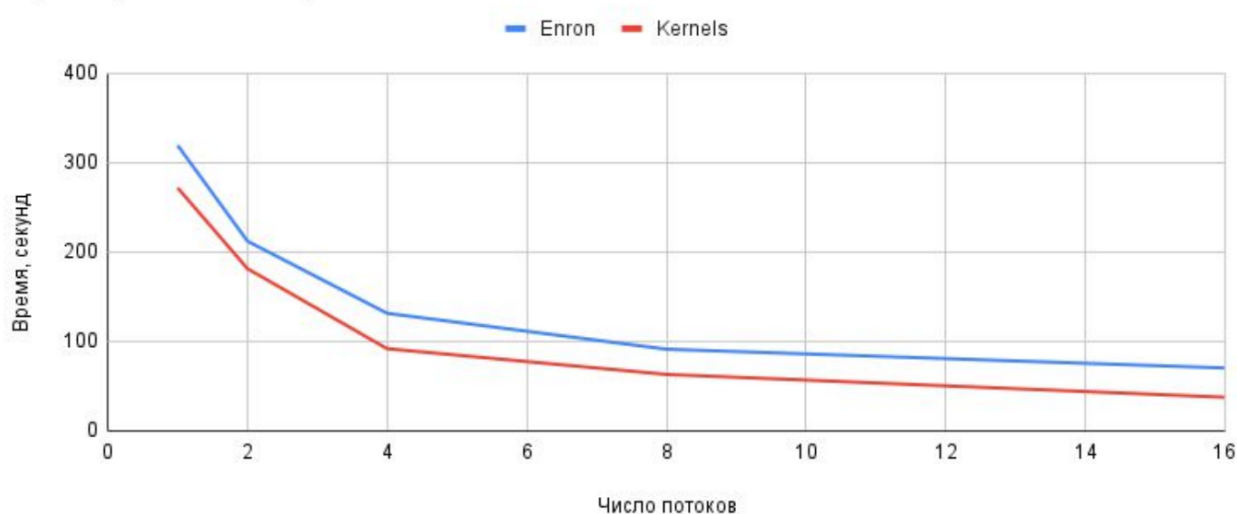


Рис. 4: Время на составление словаря для используемых датасетов

Для систем хранения резервных копий мы можем рекомендовать использовать параллельную реализацию на 8 потоках. Более того, для снижения количества коллизий можно предложить проводить запись первой версии последовательно. Это может уменьшить количество коллизий и снизить ухудшение качества дедупликации от использования параллелизма. Такой подход не ухудшит производительность регулярного использования системы, а только скорость ее инициализации.

Заключение

В работе предложен подход к распараллеливанию алгоритма FBC. Параллельное вычисление частот позволило достичь скорости алгоритма, сравнимой с базовыми CDC алгоритмами, при практически сохранении улучшения качества дедупликации. Также на основе экспериментального исследования предлагается эвристическое решение для использования в системах хранения резервных копий. Оно позволяет снизить количество коллизий при вычислении частот за счет последовательной записи первой версии и параллельной записи последующих.

Список литературы

- [1] Lu Guanlin Jin Yu Du David. Frequency Based Chunking for Data De-Duplication. — 2010. — URL: <https://ieeexplore.ieee.org/document/5581583> (дата обращения: 22 июня 2025 г.).
- [2] Oleg Piletskii. ChunkFS. — URL: <https://github.com/Piletskii-Oleg/chunkfs.git> (дата обращения: 22 июня 2025 г.).
- [3] Дмитриевцев А. С. Гориховский В. И. Повышение качества дедупликации с помощью Frequency Based Chunking. CFP: сборник материалов конференции. — 2025.