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
2024

Оглавление

Введение

Дедупликация данных – вычислительный процесс, который фокусируется на поиске и сохранении уникальных единиц информации. Данный метод направлен на устранение повторяющихся копий данных на одном или группе носителей с целью снижения накладных расходов при хранении и передаче информации. Используется в системах хранения резервных копий.

Различают дедупликацию двух видов: файловую и блоковую. Файловая дедупликация работает по принципу хранения уникальных файлов и указателей на повторяющиеся. Блоковая дедупликация, она же чанковая, дробит фрагмент данных на небольшие блоки и сохраняет уникальные, заменяя остальные указателями на хранящийся.

Основные преимущества дедупликации:

- эффективное использование пространства для хранения данных
- эффективная передача данных по сетям с низкой пропускной способностью
- возможность создания большего количества резервных копий и их длительного хранения

Content-Defined Chunking (CDC) — метод разделения на блоки для первичной дедупликации, в котором данные используются для определения границ будущих чанков. Существует множество различных CDC: FastCDC, Leap-based CDC, SuperCDC и другие.

Для улучшения дедупликации существуют два подхода: Similarity Based Chunking [1] и Frequency Based Chunking [3]. Первый основывается на сходстве хранимых чанков, второй на частотном анализе. Работы по исследованию этих методов уже проводились, но они не сравнивались ранее для различных видов CDC, также не производился анализ взаимодействия этих алгоритмов с файловой системой. Для исследования был выбран метод FBC, так как работы с SBC ведутся параллельно [5].

Основная идея в основе Frequency Based Chunking – анализ частот подчанков в данном CDC-разбиении. На основе результатов частотного анализа происходит дальнейшее дробление блоков.

ChunkFS [4] — файловая система с встроенной поддержкой методов CDC, которая подходит для интеграционного анализа алгоритмов дедупликации, поэтому она была выбрана для интеграции и анализа алгоритма. Исходя из того, что ChunkFS написана на Rust, был сделан выбор в пользу именно этого языка программирования.

1. Постановка задачи

Целью работы является разработка и интеграция медодики дедупликации Frequency Based Chunking в файловую систему и проведение исследований эффективности. Для её выполнения были поставлены следующие задачи:

1. Выполнение обзора существующих реализаций алгоритмов FBC с открытым кодом, предпочтительно написанных на Rust (раздел 2.1);
2. Интеграция алгоритма ChunkFS:;
   (a) прототипирование алгоритма;
   (b) повышение эффективности прототипа алгоритма (раздел 3.1);
   (c) проектирование архитектуры решения и интеграция алгоритма в файловую систему (раздел 3.2);
3. Проведение первичных экспериментов (раздел 4).

2. Обзор

2.1. Существующие решения

В открытом доступе нет реализаций подобных алгоритмов на Rust, поэтому рассматривались реализации на других языках программирования, в частности, Python. В открытом репозитории на GitHub существует реализация Frequency Based Chunking на Python [2]. Данная реализация не является содержательной в контексте поставленной задачи, так как работает только с текстовыми данными и используется для сбора особенностей в нейросетевом NLP-пайплайне.

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 и CDC. Интеграционный анализ было принято решение проводить с помощью файловой системы ChunkFS [4], поскольку из представленных только она располагает достаточной функциональностью, которая позволяет сравнивать различные реализации алгоритмов. Руководствуясь тем, что ChunkFS написана на языке Rust, для реализации алгоритмов был выбран именно этот язык программирования.

3. Описание решения

Ключевые этапы алгоритма:

1. Разбиение входных данные на чанки.
2. Вычисление частот, их запись в словарь.
3. Поиск частовстречаемых чанков.
4. Выделение новых чанков из уже существующих.

На этапе прототипирования, был реализован алгоритм, решающий задачи составления словаря частот, сравнения и дробления блоков с помощью полного перебора всех возможных подчанков.

3.1. Улучшения алгоритма

Примитивная реализация алгоритма при помощи полного перебора всех возможных подчанков оказалась недостаточно эффективной для проведения экспериментов на серьезных объемах данных, поэтому было принято решение внедрить несколько улучшений.

3.1.1. Обновление словаря

В базовой реализации словарь растет крайне быстро, причем конечный его размер будет во много раз превосходить исходный размер набора CDC-чанков. Решением данной проблемы стало периодическое обновление словаря, при котором из него будут удаляться записи, которые наиболее редко встречались в проанализированном наборе данных. Это позволило как сократить размер словаря, так и улучшить среднее время операций, связанных с поиском частых чанков. Периодичность обновления словаря зависит как от объема обработанных данных, так и от количества полученных на вход алгоритма CDC-чанков.

3.1.2. Поиск чанков фиксированного размера

При реализации алгоритма полным перебором, мы ищем все возможные подчанки в исходном наборе данных. Вместо этого, было принято решение рассматривать блоки фиксированного размера. Анализ подчанков фиксированного размера позволил уменьшить как размер словаря, так и скорость его составления. Притом снижение коэффициента дедупликации оказалось незначительным, из-за того что потери при выделении чанков меньшего размера являются минимальными.

3.1.3. Разрешение коллизий

Во время составления словаря, мы производим поиск всех возможных подчанков фиксированной длины. При этом, если на некотором шаге поиска мы обнаружили чанк, уже находящийся в словаре, тогда, если он не находится в начале или в конце CDC-чанка, мы можем быть уверены, что следующие n-1 (n – длина найденного подчанка) попыток найти чанк окажутся бессмысленными, ведь при дроблении текущий чанк гарантированно будет замещён. Следовательно, когда мы во время создания словаря обнаружили чанк с записью в словаре, мы переместим наш поиск на n-1 шаг вперед.

3.1.4. Фильтр Блума

Внедрение Фильтра Блума позволило эффективно проверять отсутствие элемента в словаре, благодаря этому, получилось значительно эффективнее выполнять как составление словаря, так и поиск чанков-кандидатов для дробления.

3.2. Проектирование архитектуры и интеграция в ChunkFS

Архитектура представлена на Рис. 1

- FBCMap — хранилище типа ключ-значение, которое реализует

интерфейс DataBase для получения, добавления, удаления и проверки наличия чанка соответственно.

- FBCScrubber — объект, содержащий анализатор частот FrequencyAnalyser и чанкер ChunkerFBC, который будет дробить CDC-блоки на основе данных частотного анализа. Для FBCScrubber реализован интерфейс Scrub, который имеет один метод scrub, отвечающий за применение алгоритма Frequency Based Chunking.

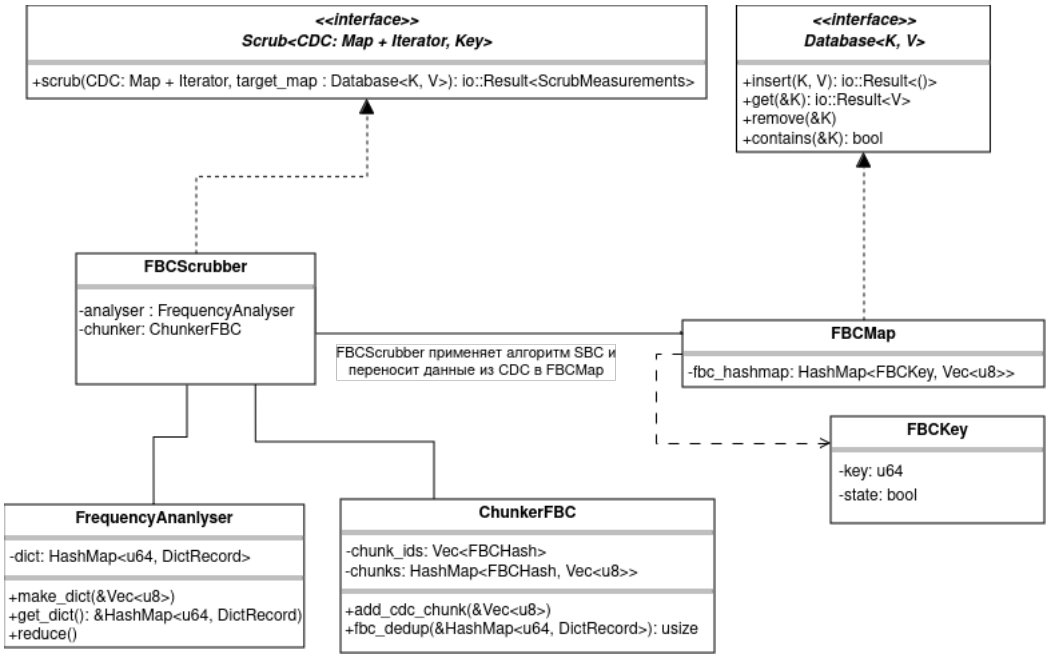

Рис. 1: Архитектура проекта

Архитектура была согласована с разработчиком файловой системы ChunkFS. В коде файловой системы были объявлены интерфейсы Scrub и DataBase с соответствующими методами и добавлена поддержка оптимизаций дедупликации SBC и FBC [6].

Для применения алгоритма FBC создается новый объект FBCMap. Далее создается FBCScrubber. Скраббер имеет метод scrub, который принимает в качестве первого параметра набор имеющихся чанков, полученных на первом этапе после CDC, второго — созданную FBCMap. Чанки из первой структуры будут перемещены и заменены на ключи FBCMap.

3.2.1. Scrubber

Итерируясь по базе данных, анализатор производит поиск подчанков, после чего строит словарь, в котором указаны самые частые чанки и их число вхождений в набор, после этого имеющиеся CDC-блоки дробятся на мелкие и сохраняются в FBCMap. Скраббер собирает информацию о времени работы, количестве сжатых и неизменных данных.

4. Эксперимент

4.1. Условия эксперимента

ОС Linux, дистрибутив Manjaro 24.1.1, процессор Intel® Core™ i7-12650H, объем оперативной памяти 16 Гиб. Алгоритмы первичной дедупликации RabinCDC и SuperCDC. Для анализатора частот был выбран размер чанка 128 байт. 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)

4.2. Исследовательские вопросы

RQ1 : насколько метод FBC эффективен при работе с чанками CDC?

RQ2 : как выбор алгоритма CDC влияет на коэффициент дедупликации при использовании FBC?

4.3. Метрики

Метрика: коэффициент дедупликации (считается как начальный размер данных, деленный на размер данных после применения алгоритма)

4.4. Результаты

Результаты экспериментов приведены в таблице **??**.

4.4.1. RQ1

Применение FBC после RabinCDC и SuperCDC для датасета Enron дало ощутимый прирост коэффициента дедупликации. Это обусловлено тем, что электронные письма содержат большое количество повторя-

Датасет	Коэффициент дедупликации	Алгоритм
Enron	1.06±0.015	SuperCDC
Enron	1.05±0.018	RabinCDC
Enron	1.44±0.029	FBC + SuperCDC
Enron	1.41±0.031	FBC + RabinCDC
Linux	1.42±0.013	SuperCDC
Linux	1.32±0.017	RabinCDC
Linux	1.6±0.03	FBC + SuperCDC
Linux	1.57±0.034	FBC + RabinCDC

ющихся элементов, к примеру, заголовки, поэтому FBC находит большое количество повторяющихся чанков и дедуплицирует их. Для датасета, содержащего ядра Linux, применение FBC не дало такого серьезного прироста коэффициента дедупликации, как это было в случае с почтами. Это обусловлено тем, что CDC-алгоритм значительно лучше справился с поставленной задачей, и в первичных чанках оказалось не так много повторов.

4.4.2. RQ2

Для датасета Enron, выбор алгоритма CDC не оказывает существенного влияния на коэффициент дедупликации. Применение FBC после RabinCDC на датасете с ядрами Linux дало несколько лучший прирост коэффициента дедупликации в сравнении с аналогичным применением после SuperCDC. Это обусловлено тем, что эффективность RabinCDC на данном наборе хуже, поэтому и повторов в чанках CDC для данного алгоритма находится больше.

Заключение

- В ходе обзора выявлено отсутствие существующих решений для FBC, применимых для сравнения.
- Реализован алгоритм FBC на Rust, валидный вне файловой системы.
- В коде алгоритма FBC реализованы интерфейсы Scrub и Database для интеграции с файловой системой. Исходный код алгоритма есть в открытом доступе на github https://github.com/admitrievtsev/fbc_scrubber.
- Проведены первичные эксперименты, результаты которых показали интересные возможности для применения метода в реальных системах.

План дальнейшей работы:

- внедрение асинхронного Key-Value хранилища;
- повышение эффективности дробления чанков;
- применение машинного обучения для вычисления частот;

Список литературы

[1] Aronovich L. Similarity based deduplication with small data chunks. — URL: https://www.sciencedirect.com/science/article/pii/S0166218X15004795 (дата обращения: 24 декабря 2024 г.).

[2] Castella Sergi. Compressing semantic representations and evaluating them in downstream tasks. — URL: https://github.com/sergicastellasape/semantic-compression (дата обращения: 26 декабря 2024 г.).

[3] Lu Guanlin Jin Yu Du David. Frequency Based Chunking for Data De-Duplication. — 2010. — URL: https://ieeexplore.ieee.org/document/5581583 (дата обращения: 25 декабря 2024 г.).

[4] Oleg Piletskii. ChunkFS. — URL: https://github.com/Piletskii-Oleg/chunkfs.git (дата обращения: 24 декабря 2024 г.).

[5] Scherbakov Maxim. Similarity Based Chunking Scrubber. — URL: https://github.com/maxscherbakov/sbc_algorithm (дата обращения: 24 декабря 2024 г.).

[6] Олег Пилетский. Разработка инструмента для сравнения алгоритмов дедупликации. — URL: https://se.math.spbu.ru/thesis/texts/Piletskij_Oleg_Antonovich_Spring_practice_3rd_year_2024_text.pdf?ysclid=m4sub8rcdf702483786 (дата обращения: 2024).