Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Разработка инструмента для сравнения алгоритмов дедупликации

Пилецкий Олег Антонович

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

В мире существует большое число файловых систем и систем хранения данных, и объемы данных, которые через них проходят, огромны. Чтобы сократить объем хранимых данных, а вместе с ним и стоимость обслуживания этих систем, используются различные методы. Одним из самых распространенных методов является дедупликация, позволяющая находить повторяющиеся данные, которых в хранилищах достаточно много.

Дедупликация состоит из нескольких этапов, а именно — разбиение файла на непрерывные отрезки данных, также называемые чанками, их хэширование, удаление повторений, индексирование и сохранение оставшихся чанков в систему. Самым затратным по времени и ресурсам в этом процессе является первый этап, также называемый чанкингом или чанкированием, поскольку алгоритму необходимо пройтись по всему файлу целиком, обычно побайтово, производя в процессе некоторые вычисления. Основным семейством алгоритмов чанкинга является Content Defined Chunking[1]. Их существует достаточно много, однако сравнений их между собой достаточно мало.

Некоторые из алгоритмов также позволяют дедуплицировать не сам файл, а сохраненные в системе чанки, что позволяет добиться еще большего выигрыша в занимаемом пространстве. Примерами таких методов являются Frequency Based Chunking [2] и Similarity Based Chunking [4], исследований которых проводилось очень мало. По этой причине они представляют большой интерес.

Основная проблема заключается в том, что отсутствует стандартизованная система для сравнения всего многообразия алгоритмов дедупликации. В существующих исследованиях сравнения алгоритмов проводились без привязки к файловой системе, с совершенно разными реализациями одних и тех же алгоритмов.

Для компании YADRO особый интерес представляет язык программирования Rust, дающий гарантии безопасной работы с памятью без

[1] https://joshleeb.com/posts/content-defined-chunking.html

использования сборщика мусора. Для более точного понимания, каике алгоритмы будут лучше в тех или иных сценариях использования, необходимо создать легко модифицируемый стенд в виде in-process файловой системы на Rust, в который будет возможно встраивание и дальнейшее сравнение разных алгоритмов дедупликации.

1. Постановка задачи

Целью работы является создание стенда файловой системы на языке Rust для сравнения алгоритмов дедупликации и их оптимизации. Для её выполнения были поставлены следующие задачи:

1. провести обзор существующих файловых систем с открытым исходным кодом, поддерживающих дедупликацию, предпочтительно, написанных на языке Rust;
2. спроектировать архитектуру стенда, поддерживающую конфигурируемость дедупликации;
3. реализовать стенд, поддерживающий разные алгоритмы Content Defined Chunking;
4. встроить в него поддержку систем оптимизации дедупликации, таких как FBC и SBC;
5. провести первичное сравнение алгоритмов на стенде.

2. Обзор

В работе прошлого семестра [6] в качестве основы файловой системы для стенда использовалась ZboxFS, однако из-за огромных накладных расходов на шифрование и сжатие данных смена алгоритма дедупликации не давала почти никакого изменения в производительности операций чтения и записи. Поскольку на Rust больше нет подобных файловых систем, поддерживающих дедупликацию, было принято решение создавать свою.

2.1. Требования

Стенд должен представлять собой in-process файловую систему на языке Rust, в которой пользователю будет предоставлен выбор алгоритма чанкинга, способа хранения данных и алгоритма хэширования. Кроме того, у пользователя должна быть возможность реализовывать свои алгоритмы и использовать эти реализации с файловой системой.

Основными операциями являются запись и чтение больших файлов в систему, поскольку на них лучше всего виден эффект дедупликации. Чтобы получать эту информацию, необходим встроенный бенчмаркинг, позволяющий получать информацию сразу же после записи или чтения файла.

2.2. Методы дедупликации

Основная вариативность методов дедупликации заключена в первом этапе — разбиении на чанки. Content Defined Chunking — семейство алгоритмов, которые, проходя по массиву данных целиком, находят границы чанков. В работе прошлого семестра проводились обзор и сравнение некоторых самых многообещающих алгоритмов этого семейства в модифицированной файловой системе Zbox.

Кроме них, существуют также различные подходы к дедупликации уже существующих чанков, такие как Frequency Based Chunking и Similarity Based Chunking.

Frequency Based Chunking использует информацию о частоте чанков в потоке данных, чтобы увеличить коэффициент дедупликации. Он состоит из двух компонент: алгоритма статистической оценки частоты появления чанков для определения часто встречающихся в глобальном масштабе, и двухэтапного алгоритма разбиения на чанки, который использует эти частоты для получения лучшего результата.

Similarity Based Chunking основан на том, что можно хранить вместо похожих чанков лишь какой-то один вместе с чанками-разницами.

2.3. Существующие файловые системы

Данный стенд необходим для того, чтобы иметь возможность сравнивать алгоритмы дедупликации, определять самые эффективные, и встраивать их в существующие файловые системы. Чтобы выявить основные их особенности, которые не содержатся в требованиях, но важны, были рассмотрены некоторые существующие файловые системы с поддержкой дедупликации.

2.3.1. ZboxFS

ZboxFS [5] — встроенная в приложение файловая система с открытым исходным кодом, написанная на языке Rust. Она поддерживает такие функции, как версионирование файлов, папки, сжатие и шифрование сохраненных данных при помощи lz4 и libsodium, транзакционность операций. Кроме того, файлы можно сохранять как в оперативную память, так и на диск.

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. В прошлом семестре была проведена работа над тем, чтобы сделать возможность встраивания различных алгоритмов чанкинга для последующего их сравнения в этой файловой системе. Результаты исследования показали, что из-за больших накладных расходов на сторонние вычисления разницу между алгоритмами увидеть достаточно сложно.

2.3.2. borg

Borg [1] — бэкап-система, написанная на Python, с поддержкой дедупликации, сжатия и шифрования данных. Дедупликация основана на Content Defined Chunking алгоритме, проходящемся по всему файлу скользящим окном. Для определения границы чанков используется хэш циклическими полиномами (buzhash).

В отличие от ZboxFS, чанки в ней общие на весь репозиторий, а не только в пределах одного файла.

2.3.3. restic

Restic [3] — быстрая и безопасная бэкап-система, написанная на языке Go.

Данные в этой системе хранятся в репозиториях. Репозиторий может хранить различные типы данных. Доступ к ним запрашивается при помощи идентификатора, которым является SHA-256 хэш содержимого файла. Все файлы записываются единожды и затем не модифицируются. Запись — атомарная операция, благодаря чему возможны чтение и запись в один репозиторий несколькими клиентами одновременно.

Данные хранятся в блобах — набор байт с его длиной и идентифицирующей информацией, т.е. SHA-256 хэшем.

При создании бэкапа сканируются все файлы в папке и ее подпапках. Данные из них всех делятся на блобы разной длины, определяемой при помощи скользящего окна в 64 байта. В текущей реализации используется Rabin Fingerprinting, для которого при создании репозитория определяется неприводимый многочлен.

Файлы размером 512 КБ не делятся на блобы. Блобы имеют размер от 512 КБ до 8 МБ, средний размер — 1 МБ.

Если файл был модифицирован, при следующем бэкапе сохраняются лишь модифицированные блобы. Это работает даже в том случае, когда байты были вставлены в случайные позиции посреди файла.

2.4. Результаты

Рассмотренные бэкап-системы используют внутри лишь алгоритм RabinCDC семейства Content Defined Chunking, не поддерживают смену алгоритмов чанкинга и не рассматривают идеи FBC и SBC.

Помимо собственно дедупликации, другие файловые системы поддерживают множество функций, таких как шифрование, сжатие и версионирование данных. Поскольку данная работа фокусируется именно на дедупликации, сторонняя функциональность, требующая отдельного исследования, не была реализована.

В рассмотренных бэкап-системах повторяющиеся данные сохраняются на диск лишь единожды, и если один и тот же чанк находится в различных файлах, в систему он также будет сохранен только один раз. В этих файловых системах для этой функцональности вводится понятие репозитория. Отрезки данных хранятся вместе с соответствующим им хэшем.

Также, в рассмотренных файловых системах используются чанки с переменной длиной, и, в зависимости от файловой системы и реализации конкретного алгоритма, различаются минимальный, средний и максимальный размеры чанков. Это может оказывать влияние на эффективность алгоритма и, как следствие, всей файловой системы.

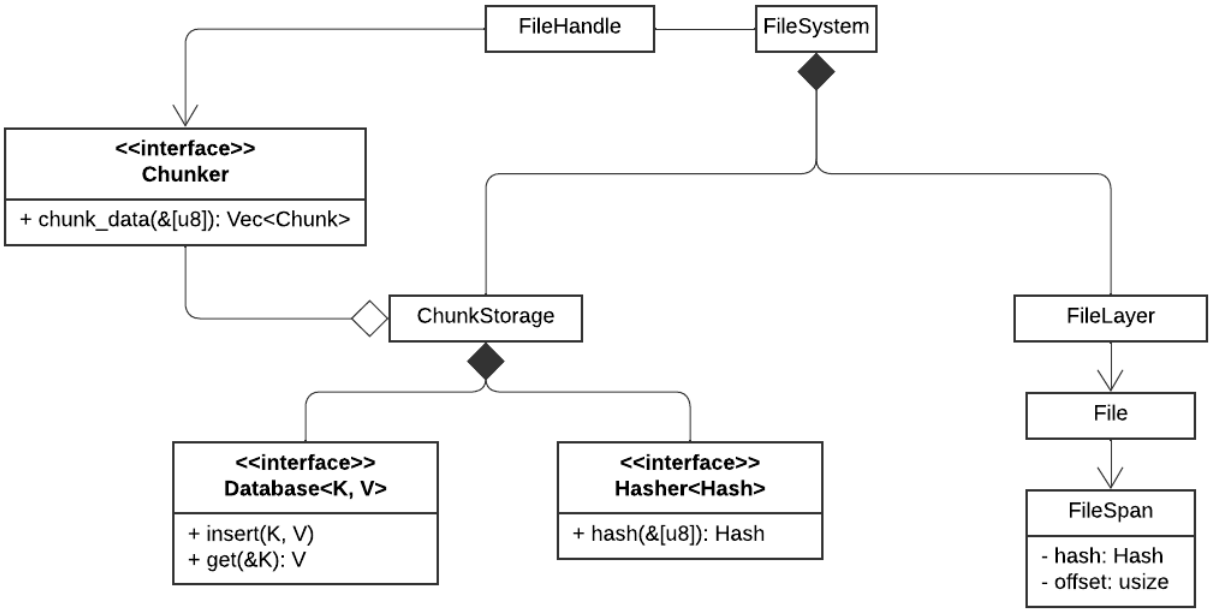

Рис. 1: Архитектура файловой системы

3. Описание решения

Архитектура файловой системы представлена на рисунке 1.

Файловая система — структура FileSystem, состоящая из двух не связанных друг с другом частей — хранилища данных и хранилища файлов:

- Данные сохраняются при помощи структуры ChunkStorage в виде пар хэш-чанк.
- Файлы хранятся при помощи структуры FileLayer в словаре, где ключом является название файла. Сам файл не хранит в себе свои данные, он хранит лишь набор сегментов, которые состоят из хэша и расстояния от начала файла до этого сегмента.

Данные две структуры общаются друг с другом при помощи хэшей сегментов. Данный подход позволяет хранить одни и те же фрагменты данных для разных файлов, не дублируя их.

Для модульности были выделены 3 основных интерфейса: Database, Chunker и Hasher:

- Database — хранилище вида ключ-значение, оно имеет методы для, соответственно, добавления, получения значения по ключу и проверки, содержится ли переданный ключ внутри. Содержится в хранилище данных.

- Chunker — объект, который производит разбиение массива данных на чанки. С его помощью реализации алгоритмов Content Defined Chunking встраиваются в файловую систему. Последний чанк хранится отдельно от всех остальных, доступ к нему происходит при помощи метода remainder. Это сделано для того, чтобы исключить возможность получения неполного чанка, извлеченного не благодаря работе алгоритма, а лишь из-за достижения конца массива данных. Содержится не в хранилище данных, а в объекте, связанном с открытым файлом.

- Hasher — объект, производящий хэширование переданного ему фрагмента данных. Содержится в хранилище данных.

Чтобы создать структуру типа FileSystem, пользователем должны быть переданы в его конструктор объекты, реализующие трейты Database и Hasher, которые будут в ней использоваться.

Работа с файлами пользователем происходит при помощи методов FileSystem. Для открытия файла используется метод open_file, в который передается его имя и объект, реализующий трейт Chunker. Метод возвращает объект типа FileHandle<C: Chunker>, с помощью которого затем можно записывать в файл данные или читать из него. При закрытии файла происходит сброс содержащихся данных в хранилище, и затем возвращаются результаты измерений, содержащие время чанкинга и хэширования данных при записи. Чтобы иметь возможность складывать эти измерения, для содержащей их структуры были реализованы трейты Add и AddAssign.

Перезапись файла доступна только целиком. Для удобства в файловой системе нет поддержки папок, все файлы находятся на одном уровне.

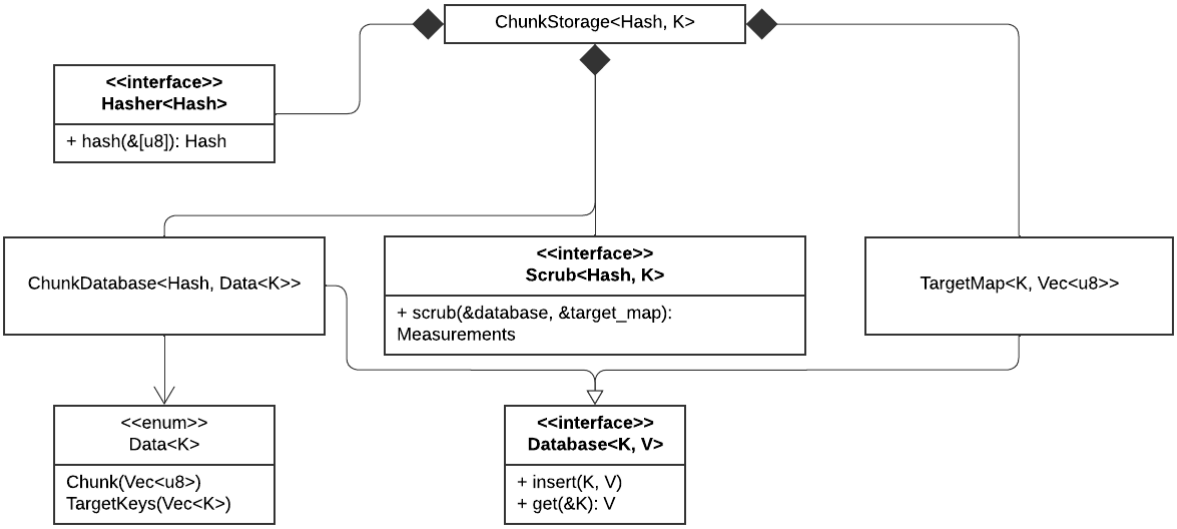

Рис. 2: Поддержка методов оптимизации

Поскольку файловая система должна предоставлять только интерфейсы и способ их взаимодействия между собой, для доступа к встроенным реализациям некоторых алгоритмов хэширования и чанкирования пользователь при добавлении зависимости от данного крейта должен передать для тега features параметры ”chunkers” и ”hashers”.

3.1. Поддержка продвинутых алгоритмов чанкинга

Архитектура, созданная для поддержки продвинутых алгоритмов чанкинга, представлена на рисунке 2.

FBC и SBC — подходы, позволяющие сократить объем уже дедуплицированных данных, проводя некоторую работу над записанными в систему чанками. Чтобы предоставить возможность изучать различные реализации этих алгоритмов, было принято решение встроить их поддержку в разрабатываемую файловую систему.

Для этого хранилище данных было переделано так, что сама база данных, реализующая трейт Database, имеет в качестве значений не чанки, а специальное перечисление Data, обернутое в структуру DataContainer. Перечисление содержит в себе либо исходный чанк, либо набор ключей, полученных в ходе работы алгоритма, с помощью кото-

рых можно обратно собрать этот чанк. Если значение — набор ключей, то хранилище данных обращается к новой базе данных. Обращение к лежащему внутри перечислению из контейнера производится при помощи методов extract и extract_mut, возвращающих, соответственно, иммутабельную и мутабельную ссылку на содержимое контейнера.

Для реализации самих алгоритмов был создан трейт Scrub, имеющий один метод — scrub. Он принимает два параметра: исходную базу данных, имеющую такую особенность, что по ней можно итерироваться, и новую базу данных. Чанки из первой анализируются и, если нужно, переносятся во вторую. Чтобы указать компилятору, что по первой базе данных можно итерироваться, на нее были наложены ограничения: этот объект должен реализовывать трейт IntoIterator. Этот итератор затем должен будет возвращать мутабельные ссылки на пары хэш-контейнер.

Для того, чтобы иметь возможность обращаться к перенесенным чанкам, для контейнера данных в исходной базе был создан метод make_target, принимающий набор ключей и перезаписывающий хранящиеся в контейнере данные. Для получения итогового чанка используется метод get у новой базы данных, который собирает его по переданному набору ключей.

В итоге, в файловую систему при создании, кроме основной базы данных и хэшера, нужно также передавать скраббер и базу данных для результатов работы алгоритма. Скраббер отвечает за перезапись чанка, база данных — за его сбор и передачу пользователю. Остальные детали были инкапсулированы в реализации ChunkStorage, благодаря чему пользователь должен лишь вызвать метод scrub у объекта типа FileSystem и получить измерения скорости и количества просмотренных и оставшихся в хранилище данных.

3.2. Поддержка различных хэшей

Необходимо также поддерживать различные типы хэшей для использования с трейтом Hasher. Для этого внутри него был создан ассоциированный тип Hash, который затем должен быть явно обозначен

пользователем при реализации этого трейта. Метод hash возвращает объект этого типа.

Чтобы хэш вел себя как хэш, на этот ассоциированный тип были наложены ограничения в виде trait bounds. Так, этот тип также должен реализовывать трейты Hash, Clone, PartialEq, Eq и Default из стадартной библиотеки. Если это не будет выполняться, произойдет ошибка компиляции.

Чтобы каждый раз не писать эти ограничения на трейты, был создан отдельный трейт ChunkHash, автоматически реализованный для всех структур, которые удовлетворяют всем пяти трейтам. После этого во все необходимые структуры было вписано ограничение, что тип-параметр Hash должен удовлетворять трейту ChunkHash.

4. Эксперимент

Необходимо провалидировать файловую систему и провести первичное исследование эффективности чанкеров в ней.

4.1. Условия эксперимента

Бенчмаркинг проводится при помощи пакета criterion. Запуск бенчмарков производится при помощи команды cargo bench.

Чтобы сравнить то, как хорошо различные алгоритмы чанкинга справляются с дедупликацией, были выбраны следующие стандартные наборы данных:

- Несколько копий исходного кода ядра Linux версий с 6.0 по 6.3 (5.1 ГБ).
- Образ Ubuntu 22.04 (1.4 ГБ).

4.2. Метрики

- Метрики для CDC
  - Коэффициент дедупликации, показывающий, сколько повторяющихся данных смог найти и устранить алгоритм. Считается как размер данных до / размер данных после.
  - Скорость работы.
- Метрики для методов оптимизации
  - Время работы.
  - Количество просмотренных данных.
  - Сколько данных осталось.
- Интеграционные метрики
  - Скорость записи файла в систему.
  - Скорость прочтения файла в системе.

4.3. Исследовательские вопросы

RQ1 : насколько результаты бенчмаркинга воспроизводимы на стенде?

RQ2 : какой алгоритм из реализованных обеспечивает лучшие характеристики?

4.4. Результаты

	RabinCDC	UltraCDC	SuperCDC	LeapCDC
Linux	0.708	0.648	0.599	0.566
Ubuntu	0.945	0.984	0.964	0.927

Таблица 1: Коэффициент дедупликации

	RabinCDC	UltraCDC	SuperCDC	LeapCDC
Linux	1030	1709	3756	4106
Ubuntu	1265	1672	14176	4136

Таблица 2: Чистая скорость чанкирования, МБ/с

В следующих двух таблицах указано среднее время и среднеквадратичное отклонение.

	RabinCDC	UltraCDC	SuperCDC	LeapCDC
Linux	9.74 ± 0.09	7.46 ± 0.11	6.88 ± 0.14	6.63 ± 0.12
Ubuntu	2.37 ± 0.04	1.96 ± 0.07	1.52 ± 0.08	1.61 ± 0.03

Таблица 3: Время записи файла в систему, секунды

	RabinCDC	UltraCDC	SuperCDC	LeapCDC
Linux	15.05 ± 0.66	15.58 ± 1.03	16.45 ± 0.75	15.50 ± 1.07
Ubuntu	0.75 ± 0.04	0.71 ± 0.06	0.89 ± 0.02	0.70 + 0.04

Таблица 4: Время чтения файла в системе, секунды

4.5. Выводы

В данной файловой системе есть возможность сравнивать алгоритмы как в чистом виде, так и при взаимодействии с остальными механизмами файловой системы. На данный момент наблюдается прямая зависимость скорости записи от алгоритма чанкирования, и, пусть не достигается десятикратного прироста производительности, как это происходит с чистыми алгоритмами, разница существенна.

По скорости записи и коэффициенту дедупликации лучше всего себя проявили SuperCDC и LeapCDC, и эти результаты совпадают с полученными на ZboxFS.

Скорость чтения достаточно слабо зависит от алгоритма — при чтении происходит получение данных из внутреннего хранилища при помощи набора ключей и их последующая конкатенация.

Заключение

В течение семестра был спроектирован, реализован и протестирован стенд файловой системы для сравнения алгоритмов дедупликации.

- был проведен обзор существующих файловых систем, в результате которого выяснилось, что архитектур, подходящих для целей сравнения алгоритмов, нет;
- была спроектирована архитектура стенда, поддерживающая легкую замену алгоритмов Content Defined Chunking, FBC и SBC;
- по ней был реализован универсальный и легко расширяемый стенд, представляющий собой библиотеку с набором интерфейсов, которые можно реализовывать отдельно и использовать вместе со всей системой;
- стенд был провалидирован и было проведено сравнение алгоритмов CDC.

По результатам работы прошлого семестра ZboxFS оказалась неподходящей для сравнения алгоритмов дедупликации. Поскольку на Rust очень мало файловых систем с поддержкой дедупликации, но была необходимость сравнивать различные алгоритмы, было принято решение написать стенд в виде in-process файловой системы, в которую можно встраивать реализации алгоритмов и интеграционно сравнивать их производительность. Были собраны требования и обозначены особенности существующих файловых и бэкап-систем, после чего была разработана архитектура, поддерживающая легкую замену алгоритмов чанкирования, хэширования и способов хранения данных.

В дальнейшем планируется использование этой файловой системы для сравнения методов оптимизации алгоритмов чанкирования, таких как Frequency Based Chunking и Similarity Based Chunking.

Код реализации можно найти в репозитории GitHub[2].

[2] https://github.com/Piletskii-Oleg/chunkfs

Список литературы

[1] Borg. Deduplicating archiver with compression and encryption. — URL: https://www.borgbackup.org/.

[2] Lu Guanlin, Jin Yu, Du David. Frequency Based Chunking for Data De-Duplication. — 2010. — 09. — P. 287 – 296.

[3] Restic. Fast, secure, efficient backup program. — URL: https://restic.net/.

[4] 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.

[5] Zbox. Zero-details, privacy-focused in-app file system. — URL: https://github.com/zboxfs/zbox.

[6] 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.