Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б10-мм

Реализация хранилища чанков для файловой системы ChunkFS на основе B+ дерева

Закарлюка Иван Владимирович

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 существующих решений **5**

2.1. Sled 5

2.2. Nebari 6

2.3. RocksDB 6

2.4. Выводы 7

3. Описание решения **8**

3.1. B+-дерево 8

3.2. Прототип B+ дерева 8

3.3. Тестирование 9

4. Требования и архитектура **10**

4.1. Сценарии использования 10

4.2. Диаграммы активностей 10

Заключение **14**

Список литературы **15**

Введение

За последние годы темпы роста объемов хранимой информации резко увеличились. Для того чтобы справиться с этим, системы хранения данных используют различные способы. Одним из самых распространенных методов является дедупликация. Новые алгоритмы дедупликации и их эвристические оптимизации появляются каждый год. Распространенной проблемой таких исследований является недостаток интеграционного сравнения методов: алгоритмы сравниваются в чистой форме, без учета взаимодействия с файловой системой. Для систематического стандартизированного сравнения реализаций алгоритмов дедупликации на языке Rust была разработана модельная универсальная файловая система ChunkFS[7]. Key-value хранилище — это абстрактный тип данных, который позволяет хранить, а также быстро получать, добавлять и удалять значения по их уникальному ключу. Потребность в такой структуре естественно возникает для алгоритмов дедупликации, так как их работа после разбиения изначальных данных на чанки заключается в сравнении по хешу и сохранении только уникальных чанков, соответственно возникает потребность быстро добавлять и искать чанки по их хэшу. На текущий момент ChunkFS использует в качестве key-value хранилища хэш таблицу, которая хранит чанки в оперативной памяти,тогда как большинство файловых систем полагаются на постоянную память. Таким образом, чтобы приблизить это к реальной файловой системе, необходимо хранить чанки в постоянной памяти. Самыми популярными вариантами реализации key-value хранилищ являются LSM-дерево и B+-дерево. LSM-дерево в среднем показывает лучшую производительность при большом количестве запросов на запись, а B+-дерево при большом количестве запросов на чтение.

1. Постановка задачи

Целью работы является создание хранилища чанков на основе B+ дерева на языке Rust. Для её выполнения были поставлены следующие задачи:

1. Провести обзор существующих реализаций key-value хранилищ
2. Реализовать хранилище чанков на основе B+ дерева
3. Внедрить хранилище чанков в ChunkFS
4. Произвести сравнения хранилищ чанков, реализованный на основе различных структур данных

Однако эти задачи являются слишком большими, поэтому они будут выполнены в следующем семестре. На этот же семестр были поставлены следующие задачи

1. Провести обзор существующих реализаций key-value хранилищ
2. Выбрать структуру, на основе которой будет реализовано key-value хранилище для ChunkFS
3. Реализовать прототип key-value хранилища на основе B+ дерева
4. Интегрировать прототип в ChunkFS и оценить его работоспособность
5. Составить требования и спроектировать архитектуру итоговой реализации хранилища

2. Обзор существующих решений

2.1. Sled

Sled[4] - это высокопроизводительная база данных, написанная на Rust. Несмотря на то, что sled в первую очередь позиционируется как база данных, он предоставляет key-value хранилище на основе B+-дерева. Sled также использует современные оптимизации для B+-дерева.

2.1.1. Преимущества

1. Sled оптимизирован для быстрого чтения и записи данных, по утверждениям разработчиков - скорость чтения как в традиционных B+-деревьях, а скорость записи как в LSM-деревьях

2. Sled является полностью потокобезопасным, а все его операции являются атомарными

3. key-value хранилище реализует методы обычного BTreeSet из std, из-за чего с ним просто работать

2.1.2. Недостатки

1. Sled оптимизирован в первую очередь для долго работающих приложений по типу более высокоуровневых баз данных, отчего при других сценариях его использования могут быть проблемы с производительностью

2. Sled все ещё находится в разработке и на данный момент обновляется достаточно редко. Из-за этого в sled существует много критических багов, например утечки памяти, повреждение уже записанных данных при добавлении новых

3. Sled, по сравнению с аналогами, задействует очень много памяти

4. Очень сложно реализован, из-за чего для его модификации нужно будет потратить много времени

2.2. Nebari

Крейт Nebari[3] представляет собой хранилище ключ-значение, реализованное на основе B+ дерева, которое использует формат append-only для хранения данных.

2.2.1. Преимущества

1. Nebari позволяет использовать ключи длиной до 65,535 байт и значения до 4 гигабайт. Это делает его подходящим для хранения больших объемов данных и различных типов информации

2.2.2. Недостатки

1. Nebari блокирует текущий поток при доступе к файловой системе
2. Хоть Nebari считается стабильным, он все ещё находится на стадии альфа-разработки, что означает наличие потенциальных ошибок и рисков потери данных
3. Nebari является относительно новым проектом, из-за чего у проекта очень скудная документация

2.3. RocksDB

Крейт RocksDB[5] для языка Rust представляет собой обертку над популярной библиотекой RocksDB, которая является высокопроизводительным хранилищем ключ-значение.

2.3.1. Преимущества

1. Все операции потокобезопасны. Поддерживается ассинхронное чтение и запись

2. RocksDB является очень стабильной, она используется во многих больших проектах

2.3.2. Недостатки

1. RocksDB из-за своего объема сложна в использовании и особенно сложна в модификации

2. RocksDB занимает большое количество места на диске и потребляет большое количество оперативной памяти

Далее приведена таблица с кратким сравнением разных решений.

	Состояние проекта	Гарантии ACID	Асинхронность операций	Внутренняя реализация
Sled	Нет активной поддержки, есть критические баги, влекущие потерю данных	да	есть	B+ дерево
RocksDB	Стабильная версия	да	есть	LSM-дерево
LMDB	Стабильная версия	да	есть	B+ дерево
PickleDB	В разработке	нет	нет	Хэш-таблица
BonsaiDB	В активной разработке	да	нет	B+ дерево
LevelDB	Стабильная версия	да	есть	LSM-дерево
BerkleyDB	Стабильная версия	да	есть	B+ дерево/Хэш-таблица

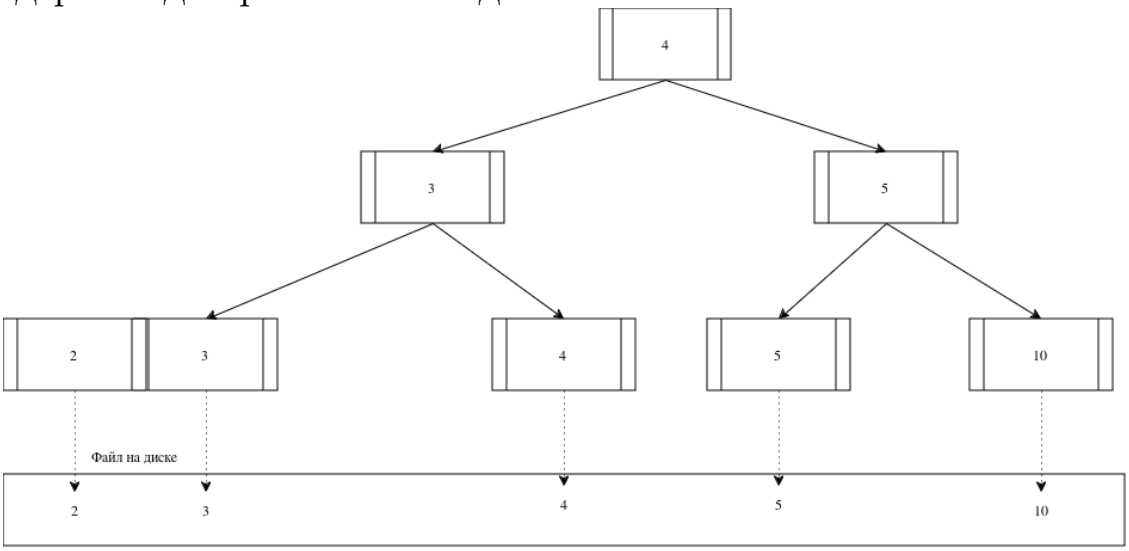
2.4. Выводы

Существующие реализации имеют как преимущества, так и недостатки, однако общим для них выступает то, что они будут работать с ChunkFS через различные адаптеры, из-за чего будут показывать худшую производительность. Для честного сравнения алгоритмов дедупликации нужна встраиваемая напрямую в ChunkFS реализация. Именно из-за этого было принято решение писать свою реализацию. Можно заметить, что практически во всех существующих решениях key-value хранилище выполнено на основе LSM или B+ деревьев, и эти деревья в целом на данный момент являются самыми распространенными решениями. Поэтому для интеграции в ChunkFS были выбраны именно эти две структуры данных, однако в этой работе будет рассмотрено именно B+-дерево, поскольку LSM-деревом занимается мой одногруппник, Иван Глазунов.

3. Описание решения

3.1. B+-дерево

B+-дерево[1] — структура данных на основе B-дерева, сбалансированного n-арного дерева поиска с переменным, зачастую с большим количеством потомков в узле. Обычно B+-дерево хранит указатели на данные, а сами данные записываются в постоянную память. Указатели на данные хранятся только в листьях B+-дерева, что позволяет последовательно читать информацию из листьев без необходимости каждый раз обходить дерево заново. B+-дерево применяется в таких СУБД, как DB2, Oracle Database, Microsoft SQL Server, SQLite, FoundationDB [2], WiredTiger[6]. B+-дерево также широко применяется в файловых системах — NTFS, ReiserFS, NSS, XFS, JFS, ReFS и BFS используют B+-деревья для различных задач.



3.2. Прототип B+ дерева

Прототип key-value хранилища на основе B+-дерева был написан на языке Rust, поскольку сам ChunkFS также написан на этом языке. Прототип хранит в качестве указателей на данные структуры ChunkPointer, которые хранят путь до файла с чанком, смещение и размер чанка. Структура имеет функцию чтения, которая считывает и возвращает чанк из файла. При вставке чанка в дерево оно создает новый экзем-

пляр ChunkPointer и передает ему чанк. Чанк записывается в файл, после чего дерево выполняет вставку пары <хэш чанка, экземпляр ChunkPointer>. При превышении определенного размера файла, который задается параметром, чанки начинают записываться в новый файл.

ChunkFS предоставляет интерфейс Database, реализации которого достаточно для интеграции прототипа, который и был реализован.

3.3. Тестирование

ChunkFS предоставляет несколько интеграционных тестов, на которых и был успешно протестирован прототип key-value хранилища.

Проверка производительности была проведена при помощи пакета criterion. В качестве метрик для проверки были выбраны скорость записи и скорость чтения, проверка проводилась на датасете из статей CNN объемом 107 МБ. Результаты следующие:

- Скорость записи в миллисекундах — 93 ± 6
- Скорость чтения в миллисекундах — 600 ± 404

4. Требования и архитектура

4.1. Сценарии использования

Для составления требований и архитектуры нужно рассмотреть сценарии использования key-value хранилища на основе B+ дерева. Такие хранилища используют для:

1. Кеширования
2. Хранения сессий в веб-приложениях
3. Хранения логов и метрик
4. Обработки очередей сообщений

Сценарий использования, специфичный для ChunkFS, заключается в следующем: ChunkFS может запросить наличие чанка с определённым хэшом, извлечь этот чанк из хранилища или добавить новый чанк с уникальным ключом в хранилище.

Исходя из сценариев использования, были сформулированы следующие требования:

1. Выполнение гарантий ACID
2. Возможность добавления, удаления чанков, очистки всего хранилища
3. Наличие неизменяемого итератора
4. Подгрузка близких чанков

4.2. Диаграммы активностей

Более подробное описание взаимодействий с B+ деревом представлено ниже на диаграммах активностей.

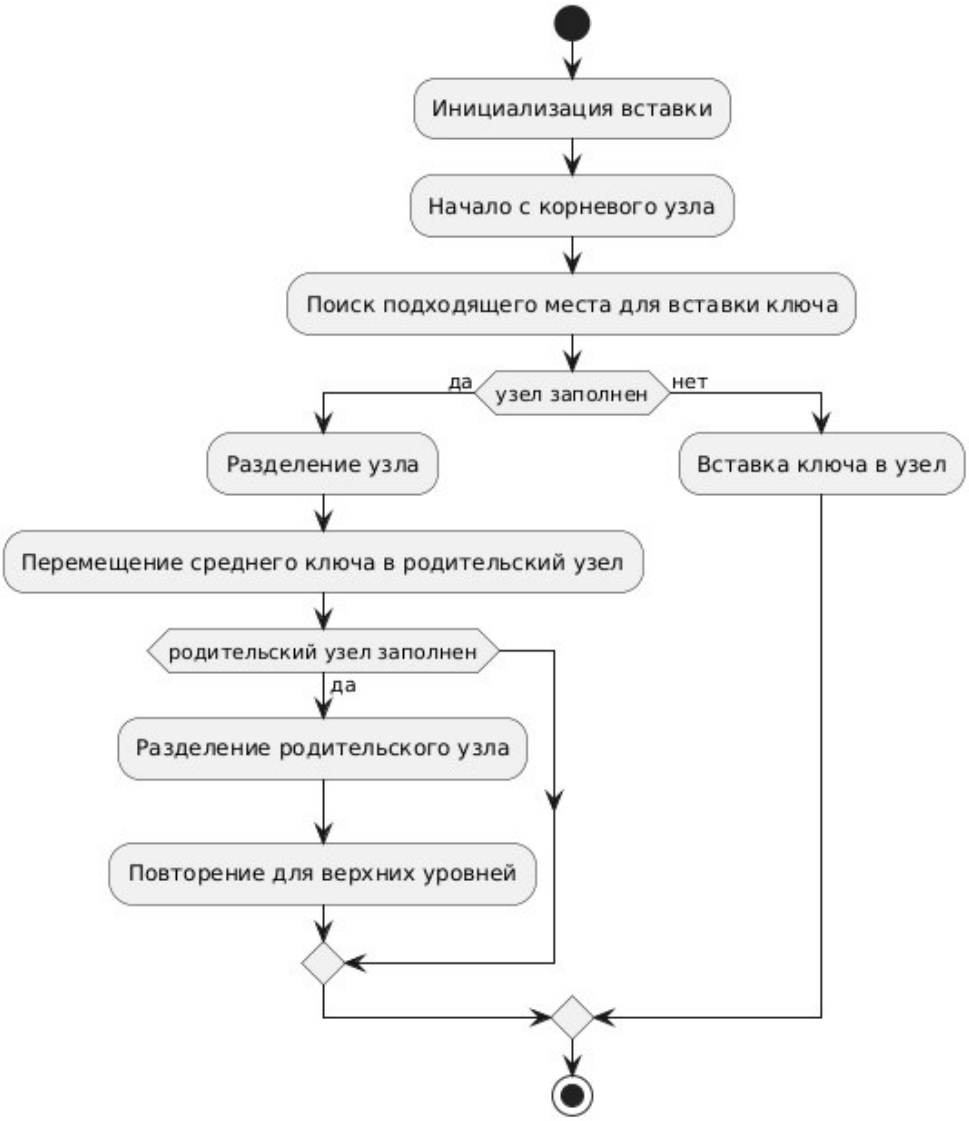


Рис. 1: Вставка


Инициализация поиска
Начало с корневого узла
листовой узел?
нет
Сравнение ключа с ключами в узле
Переход в дочерний узел
Поиск в листовом узле
Возврат результата

Рис. 2: Поиск

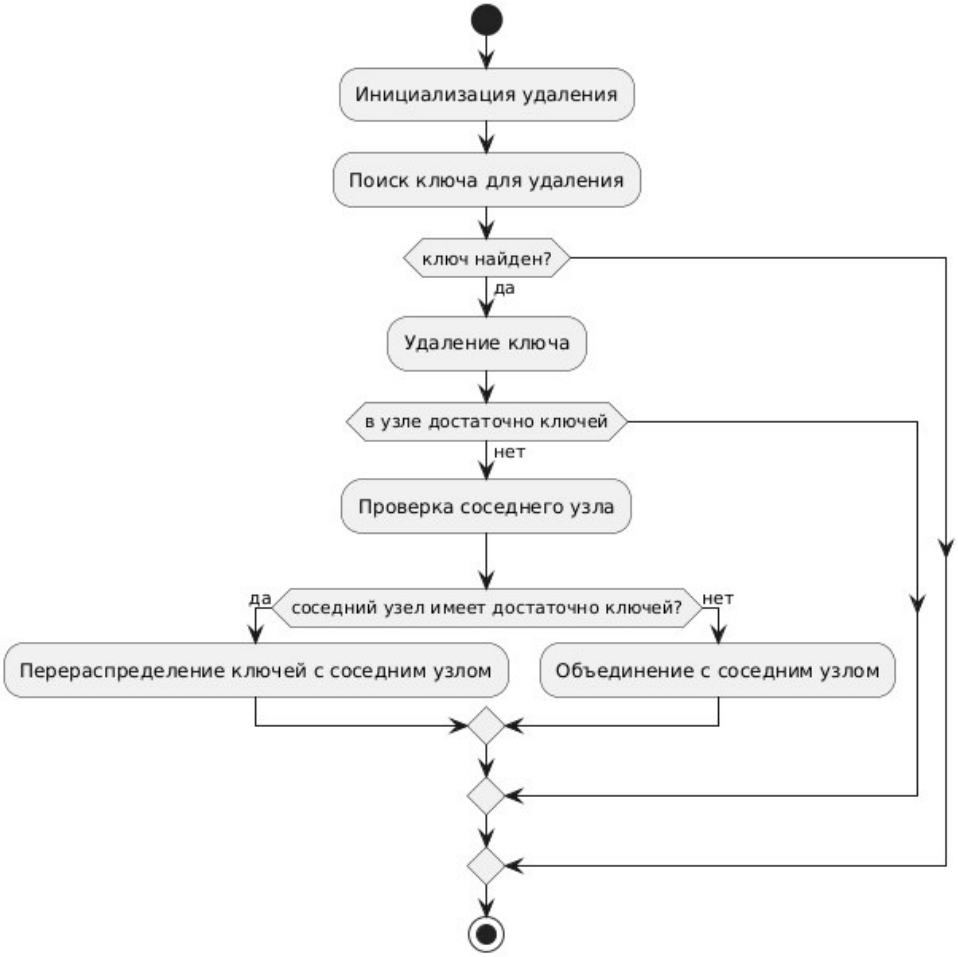


Рис. 3: Удаление

Заключение

В ходе данной работы были выполнены следующие задачи:

- Был проведен обзор существующих реализаций key-value хранилищ, однако ни одна из имеющихся не оказалась подходящей для встраивания в ChunkFS
- Для основы key-value хранилища была выбрана структура данных B+-дерево
- Был реализован прототип key-value хранилища на основе B+-дерева на языке Rust. Исходный код прототипа доступен на Github [1]
- Прототип key-value хранилища был интегрирован в ChunkFS, были проведены интеграционные и нагрузочные тесты
- Были сформулированы требования к итоговой реализации key-value хранилища

[1] https://github.com/kamenkremen/BPlusTree/pull/1

Список литературы

[1] Comer Douglas. Ubiquitous B-Tree. — URL: https://dl.acm.org/doi/pdf/10.1145/356770.356776.

[2] FoundationDB. B+ tree usage in database. — URL: https://github.com/apple/foundationdb/tree/main/contrib/sqlite (дата обращения: 24 ноября 2024 г.).

[3] Johnson Jonathan. Nebari. — URL: https://github.com/khonsulabs/nebari (дата обращения: 28 ноября 2024 г.).

[4] Neely Tyler. Sled. — URL: https://github.com/spacejam/sled (дата обращения: 28 ноября 2024 г.).

[5] Tyler Neely Oleksandr Anischenko. RocksDB. — URL: https://crates.io/crates/rocksdb (дата обращения: 28 ноября 2024 г.).

[6] WiredTiger. B+ tree usage in storage engine. — URL: https://github.com/wiredtiger/wiredtiger/tree/develop/src/btree (дата обращения: 24 ноября 2024 г.).

[7] Пилецкий Олег. Разработка инструмента для сравнения алгоритмов дедупликации. — URL: https://se.math.spbu.ru/thesis/texts/Piletskij_Oleg_Antonovich_Spring_practice_3rd_year_2024_text.pdf (дата обращения: 3 ноября 2024 г.).