Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б15-мм

Исследование реализаций лог-структурированного блочного устройства в ядре Linux

Гавриленко Михаил

Отчёт по учебной практике
в форме «Эксперимент»

Научный руководитель:
инженер-исследователь лаборатории технологий программирования инфраструктурных решений СПбГУ, Васенина А. И.

Санкт-Петербург
2025

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Блочные устройства 6

2.2. Логический адрес блока 6

2.3. Лог-структурированное хранение данных 7

2.4. Структуры данных 8

3. Реализация **11**

3.1. Отображение LBA 11

3.2. Реализации структур данных 12

4. Эксперимент **15**

4.1. Оценка 15

4.2. Метрики производительности 15

4.3. Методология тестирования и настройка рабочего процесса 16

4.4. Инструменты для бенчмаркинга и конфигурация 17

4.5. Конфигурация тестовой платформы 18

4.6. Тестирование целостности 19

4.7. Результаты тестирования производительности и анализ . 19

Заключение **35**

Список литературы **36**

Введение

Растущая потребность в высокопроизводительных решениях для хранения большого количества информации привела к высоким требованиям к аппаратной и программной составляющей систем хранения данных (СХД). В отличие от аппаратных решений, программные предоставляют возможность более гибкой модификации при низких затратах. В качестве накопителей в СХД зачастую используются жесткие диски (HDD) из-за низкой стоимости гигабайта данных и высокого уровня надежности хранения данных. Одна из особенностей HDD – низкая производительность операций случайного доступа из-за механических задержек, связанных с нахождением сектора путем перемещения головки диска. В отличие от жестких дисков, у твердотельных накопителей (SSD) нет этой проблемы, но они подвержены собственным проблемам, таким как усиление записи [19] и износ ячеек памяти.

Одним из подходов, который может быть использован для устранения проблемы ухудшения производительности при случайном обращении, является лог-структурированное хранение. Данный подход преобразует случайные записи в последовательные, используя формат журнала, тем самым минимизируя механические задержки на HDD и эффективно используя пропускную способность диска. Для SSD этот метод помогает уменьшить износ ячеек и повысить эффективность внутренней организации записи [12]. Помимо оптимизации операций на накопителях, принцип лог-структурированного хранения лежит в основе многих современных технологий, включая файловые системы [18, 12, 15], базы данных [7, 13] и различных функций СХД, таких как мгновенные снимки, кэши, журналы защиты от пробелов записи (write hole) и других.

Несмотря на то, что лог-структурированный подход к хранению данных известен давно и использовался в разных продуктах [18, 1, 20, 12, 15, 11, 7, 13], детали реализации не были достаточно изучены. Эффективность системы с подобной организацией хранения данных напрямую зависит от реализации механизма журнала – таблицы соотношения Logical Block Address (LBA). Кроме того, реализация подобного

механизма для многопоточной среды с использованием блокировок может стать бутылочным горлышком в системе. В результате чего, зачастую отдается предпочтение более продвинутым неблокирующим реализациям. В данной работе представлен фреймворк LS-BDD для тестирования реализаций лог-структурированности на блочном уровне хранения данных. Основой фреймворка является модуль ядра Linux, реализующий лог-структурированное виртуальное блочное устройство с возможностью подключения различных структур данных, реализующих операции поиска, добавления и удаления элемента. В рамках первой версии фреймворка, в качестве структур данных для реализации были выбраны хеш-таблица и список с пропусками, использующие неблокирующую синхронизацию. Для тестирования и анализа производительности реализован набор утилит на языках Shell и Python. На основе реализованного фреймворка было проведено сравнение производительности реализованных структур данных по пропускной способности (throughput), количеству операций ввода-вывода в секунду (IOPS) и задержке (latency).

1. Постановка задачи

Целью работы является исследование особенностей реализации логструктурированной адресации на основе неблокирующих структур данных в блочных устройствах. Для достижения цели были поставлены следующие задачи:

1. реализовать неблокирующие структуры данных для модуля LS-BDD;
2. протестировать драйвер на основе реализованных структур данных;
3. проанализировать полученные результаты.

2. Обзор

2.1. Блочные устройства

Блочные устройства — вид устройств, которые предоставляют программную абстракцию накопителя с доступом к данным в виде блоков определенного размера. Преимущественно блоки бывают размера 4 килобайт (КБ) и 512 байт (Б). Наименьшим адресуемым элементом блочных устройств является сектор, обычно объемом 512 Б. Размер блока данных кратен размеру сектора. Примерами блочных устройств являются жесткие диски и тома LVM [9]. Помимо физических блочных устройств, также существуют виртуальные, позволяющие программно задать логику работы устройства. Один из вариантов использования виртуальных блочных устройств это реализация логики блочного устройства в условиях отсутствия физического накопителя, например, с использованием оперативной памяти. Помимо этого, виртуальные блочные устройства могут использоваться для реализации более сложной логики хранения данных поверх физических накопителей, например, для организации RAID-массивов [17]. В ядре Linux реализован заголовочный файл `blkdev.h`, позволяющий работать с блочными устройствами. Данный заголовочный файл содержит в себе основную структуру - `block_device`, которая предоставляет доступ к инициализированному блочному устройству. Базовая единица запроса записи/чтения (IO - Input/Output) - структура `bio` (Block IO), определена в заголовочном файле `linux/blk_types.h`. Она содержит информацию об операции на низком уровне: тип операции, целевое устройство, размер запроса, начальный сектор и так далее [3]. Каждая структура `bio` содержит массив векторов ввода-вывода (`bio_vec`), элементы которого указывают на страницы оперативной памяти, участвующие в IO.

2.2. Логический адрес блока

Логический адрес блока — общепринятый стандарт, используемый для указания местоположения блоков данных, хранящихся на блочных

устройствах. Это схема адресации, преимущественно используемая операционной системой и файловыми системами при обработке операций чтения или записи. В отличие от физических адресов блоков (PBA — Physical Block Address), которые указывают на фактическое физическое местоположение на носителе, LBA позволяет обращаться к носителю информации как к линейной последовательности логических блоков. Эти блоки нумеруются последовательно, от нуля до максимального LBA, поддерживаемого устройством. Такая абстракция скрывает от пользователя сложную физическую геометрию (например, цилиндры, головки и сектора) или детали внутреннего отображения, предоставляя универсальный интерфейс взаимодействия с блочными устройствами.

2.3. Лог-структурированное хранение данных

Лог-структурированное хранение данных — технология, основанная на принципе журнала. Случайная запись сводится к последовательной, что потенциально увеличивает скорость записи на HDD [12]. Также стоит отметить, что лог-структурированное хранение лежит в основе многих других технологий, часто используемых в СХД. Реализация подобной технологии подразумевает под собой наличие таблицы хранения соотношения LBA базового устройства и и LBA лог-структурированного устройства. Лог-структурированный LBA — логический адрес, который используется как ключ для внутренней системы управления данными журнала, где новые записи записываются последовательно, в конец лога. Для хранения может быть выбрана любая структура данных с оптимальными асимптотическими оценками скорости выполнения основных операций — вставки, поиска и обновления. Такой подход обладает некоторыми недостатками. Одним из них является повышение расхода вычислительных ресурсов для обращения к структуре данных, которая лежит в основе журнала. Повышенный расход памяти для хранения таблицы соотношения LBA также является существенным недостатком.

2.4. Структуры данных

Выбор структуры данных в формате «ключ-значение» для реализации таблицы соотношения LBA напрямую влияет на производительность операций поиска, вставки и обновления, а также на потребление памяти. Учитывая большой объем современных накопителей и высокие требования к скорости ввода-вывода, структура данных должна обеспечивать:

- быстрый поиск по ключу;
- быструю вставку или обновление;
- эффективное использование памяти;
- возможность параллельного доступа из нескольких потоков с минимальными затратами на синхронизацию.

Часто в решениях для высоконагруженных систем, ориентированных на производительность, используются реализации алгоритмов и структур данных без блокировок (lock-free) или без ожидания (wait-free)[14]. Алгоритмы без блокировок обладают лучшей масштабируемостью и гарантируют, что по крайней мере один поток всегда достигает прогресса, устраняя риск полного зависания потока. В отличие от блокирующих алгоритмов, которые могут вызывать взаимоблокировки или инверсию приоритетов, lock-free алгоритмы сохраняют работоспособность при сбоях отдельных потоков. Хотя wait-free алгоритмы предоставляют более сильные гарантии, для высокопроизводительных систем хранения данных часто предпочитают алгоритмы без блокировок. Они обеспечивают существенные преимущества перед алгоритмами с блокировками при более управляемой сложности, чем у алгоритмов без ожидания, достигая практического баланса для большинства популярных требований к структурам данных с многопоточным доступом.

В ходе исследования для данной реализации были выбраны две структуры данных, с потенциально оптимальными асимптотическими оценками сложности выполнения операций и возможностью реализации версий с неблокирующей синхронизацией: хеш-таблица и список с

пропусками. Несмотря на то, что выбранные структуры данных имеют различную асимптотическую оценку временной сложности операций, что предполагает преимущество хеш-таблицы в среднем времени поиска по сравнению со списком с пропусками, фактическая производительность и пригодность могут сильно зависеть от характеристик потребления памяти[16]. Хеш-таблицы могут требовать предварительного выделения памяти или демонстрировать менее предсказуемое использование памяти во время изменения размера по сравнению с более постепенным ростом потребления памяти списка с пропусками. Эффективное использование памяти является неоспоримо важным свойством. На первый взгляд, данное свойство может показаться тривиальным, однако оно осложнено из-за требований по высокой производительности, которые результируют в использовании lock-free вариаций структур данных. Реализации полноценных lock-free структур данных требуют наличия систем освобождения памяти (memory reclamation system, далее MRS).

2.4.1. Список с пропусками

Список с пропусками — структура данных, состоящая из нескольких слоев — списков. Количество списков — параметрическое. Нижний слой представляет собой обычный список, состоящий из всех элементов. Каждый вышележащий слой $(i + 1)$ (где i — количество слоев в списке с пропусками) представляет собой список, в котором элементы слоя i встречаются с некоторой вероятностью p. В конце каждого списка находится специальный элемент, который отмечает конец списка. В итоге, список с пропусками обеспечивает выполнение всех операций в среднем с временной сложностью $O(logn)$ и $O(n)$ в худшем случае.

2.4.2. Хеш-таблица

Хеш-таблица — это структура данных, представляющая собой ассоциативный массив, который устанавливает отношение «ключ/значение» для пар. Хеш-функция используется для нахождения индекса и последующей вставки значения в указанную ячейку массива. Часто реализации

хеш-таблиц используют несовершенную хеш-функцию. Эта функция имеет недостаток в виде коллизий — отображения различных ключей в один и тот же индекс массива с некоторой вероятностью. Хеш-таблица обеспечивает достаточно высокую скорость операций удаления, поиска и вставки, со средней временной сложностью $O(1)$ и $O(n)$ в худшем случае.

3. Реализация

В рамках данной работы был доработан драйвер виртуального устройства — LS-BDD. Этот драйвер получает на вход некоторые запросы ввода-вывода, после чего перенаправляет их на целевое устройство, изменяя вид нагрузки ввода-вывода с произвольной на последовательную.

3.1. Отображение LBA

В основе реализации принципа журнала в этом драйвере лежит таблица отображения адресов. В нашем случае таблица реализована с помощью различных структур данных с хранением в формате ключ-значение. Ключом является исходный LBA, определённый системой и полученный из структуры `bio`. Значением является указатель на структуру `value_redir`, состоящую из лог-структурированного LBA и метаданных. В данной работе используется подход Redirect-On-Write с мгновенным удалением неактуальных данных. Таким образом, при добавлении функциональности для хранения многочисленных одинаковых ключей, может быть использован подходы Copy-On-Write или Redirect-On-Write со сборщиком мусора, лежащие в основе технологий мгновенных снимков [21].

Обработка операции записи включает поиск следующего свободного блока базового блочного устройства, который становится новым лог-структурированным LBA для текущего BIO запроса. Блоки данных записываются последовательно, и следующий свободный блок обычно определяется на основе конца журнала — как следующий доступный последовательный блок. Если отображение для данного LBA базового устройства уже существует в структуре данных, существующее соотношение обновляется с новым лог-структурированным LBA. Запросы на чтение обрабатываются иначе. Сначала используется таблица соотношений для поиска LBA лог-структурированного устройства, соответствующего запрошенному LBA базового устройства. Если соотношение существует, то чтение направляется по соответствующему LBA на ниже-

лежащее устройство. В противном случае, если отображение не найдено, обрабатывается случай обращения внутрь записанного блока данных. Для этого происходит поиск предыдущего блока данных и обработка его части, начиная с LBA, по которому происходит чтение.

3.2. Реализации структур данных

Для решения проблем, описанных в обзоре предметной области, мы использовали lock-free структуры данных. Несмотря на то, что реализации структур данных без блокировок начали появляться как минимум 30 лет назад, не все структуры данных имеют полезные реализации с открытым исходным кодом или качественные научные источники.

3.2.1. Список с пропусками

В LS-BDD lock-free список с пропусками была реализована на основе работы Мориса Херлихи [10] и работы Кейзера Фрейзера [6]. Основным, использующимся подходом является использование помеченных указателей. Младший значащий бит указателя `next` в каждом узле используется как «бит маркировки». Если бит установлен, узел считается логически удаленным. Это позволяет разделить удаление на логический и физический этап. Логическая часть представляет собой удаление связей с соседними элементами. Физическая часть удаления выполняется благодаря дополнительной реализации MRS [4, 5]. В текущей версии MRS срабатывает исключительно при завершении работы драйвера. Для реализации подобного механизма был добавлен дополнительный стек, в который добавляются элементы списка с пропусками при логическом удалении. При завершении работы драйвера производится проход по данному стеку и основной структуре для освобождения элементов. Такая реализация позволяет минимизировать влияние конкретной реализации MRS на общую производительность драйвера.

3.2.2. Хеш-таблица

После обзора и исследования доступных реализаций хеш-таблиц было принято решение модифицировать существующую в ядре RCU (Read-Copy-Update) хеш-таблицу. Функции добавления и другие операции над RCU списком реализованы с использованием атомарных инструкций и не требуют внешних блокировок для добавления элементов в начало списка. Удаление произвольного элемента из `llist` в RCU версии более ограничено, обычно позволяя только удаление из начала списка (`llist_del_first()`) или очистку всего списка (`llist_del_all()`). Кроме этого, такие RCU функции часто требуют тщательной синхронизации или специфических паттернов использования (например, один производитель/один потребитель) во избежание состояний гонки, если необходимо рассматривать для удаления элементы, отличные от головного. Эти ограничения не подходят для нашего случая из-за необходимости полноценного логического и физического удалений для любого ключа. Модификация состояла в замене двусвязного RCU списка, лежащего в основе бакетов, на односвязный lock-free список. В качестве односвязного списка была выбрана реализация, предложенная Т. Харрисом [8]. В своей статье Харрис представляет практический алгоритм на основе атомарной инструкции CAS (Compare-And-Swap) с использованием техники помеченных указателей. Наиболее существенным ограничением в использовании этой реализации является отсутствие механизма безопасного освобождения памяти удаленных узлов. По этой причине, так же, как и в списке с пропусками, была реализована MRS, использующая подход на основе стека.

Изменение размера хеш-таблицы без блокировок — сложная задача. В текущей реализации размер таблицы фиксирован и определяется при инициализации. В случае с использованием структуры в качестве основы лог-структурированного подхода — можно рассчитать примерный объем структуры данных. Одной из возможных оптимизаций предложенного драйвера может быть реализация двусвязного lock-free списка. Частота обращений к предыдущему элементу может достигать значительного

уровня при обработке блоков разного размера.

4. Эксперимент

4.1. Оценка

Основной целью эксперимента является количественная оценка характеристик производительности и накладных расходов, связанных с реализованным модулем ядра Linux — LS-BDD, в частности, при использовании хеш-таблицы без блокировок и списка с пропусками без блокировок для управления отображениями LBA. Также будет проведен сравнительный анализ с базовым сценарием, включающим прямой доступ к нижележащему устройству хранения. Ключевые исследовательские вопросы, направляющие это исследование, следующие:

1. Производительность ввода-вывода: В какой степени модуль LS-BDD повышает пропускную способность и количество операций ввода-вывода в секунду (IOPS) для рабочих нагрузок со случайной записью/чтением по сравнению с прямым доступом к нижележащему устройству хранения?

2. Влияние размера блока: Как изменяется производительность модуля LS-BDD при увеличении размера блока запроса?

3. Масштабируемость и параллелизм: Как производительность модуля LS-BDD масштабируется с увеличением числа параллельных потоков ввода-вывода и глубины очереди? Демонстрируют ли структуры данных без блокировок заметные преимущества в сценариях с высокой степенью состязания (contention)?

4. Сравнение потребления памяти lock-free структур данных: Каковы пиковые и средние объемы потребляемой ядром памяти, необходимые для поддержания отображения LBA?

4.2. Метрики производительности

Для обеспечения всесторонней оценки производительности будут использоваться следующие основные метрики:

1. Throughput: Определяется как объем данных, переданных за единицу времени, обычно измеряемый в гигабайтах в секунду (ГБ/с). Пропускная способность будет измеряться для чтения и записи с использованием больших размеров блоков.

2. IOPS: Представляет количество запросов ввода-вывода, обработанных в секунду. IOPS является ключевой метрикой для оценки паттернов случайного доступа, операций с малыми блоками и транзакционных рабочих нагрузок. Она будет измеряться для различных размеров блоков и различных сочетаний рабочих нагрузок чтения/записи.

3. Latency: Указывает время, необходимое для выполнения, отправления и обработки одного запроса ввода-вывода. Важно измерять не только среднюю задержку, но и ключевые перцентили, такие как 99-й перцентиль. Это позволяет оценить предсказуемость производительности и выявить выбросы задержек или хвостовые задержки (tail latencies).

4. Потребление памяти: Оценка объема памяти, занимаемого структурами данных, ответственными за отображение LBA.

4.3. Методология тестирования и настройка рабочего процесса

Методология тестирования блочных устройств основана на спецификации тестирования производительности твердотельных накопителей (Solid State Drive Performance Testing Specification) ассоциации индустрии сетей хранения данных (SNIA). Из-за ограниченной функциональности блочного устройства и ограничений эксперимента использовались некоторые правила тестирования, описанные в спецификации SNIA. Детали рабочего процесса представлены в таблице 1.

Таблица 1: Описание конфигураций тестов производительности

Метрика	IOPS	Throughput	Latency
Block size	4-128	128, 1024	4, 8
R/W mix	0/100, 65/35, 100/0		
Time x Run	30x25		
Jobs num	1, 2, 4, 8	1, 2, 4, 8	1, 8
IO depth	1, 4, 16, 32		
Warm up	да, нет		

4.4. Инструменты для бенчмаркинга и конфигурация

Основным инструментом, выбранным для сравнительного анализа реализованного блочного устройства, является fio (Flexible I/O Tester) [2]. fio — это специализированная утилита, предназначенная для генерации рабочих нагрузок ввода-вывода с целью тестирования производительности систем хранения и оценки характеристик запоминающих устройств [2]. Она предлагает широкий спектр опций для оценки производительности блочных устройств, включая точный контроль над процессом отправки запросов ввода-вывода. Основные параметры fio, использованные в данном исследовании, перечислены ниже:

- `direct=1`: Обход страничного кэша системы для измерения чистой производительности устройства/драйвера.
- `iodepth=<значение>`: Устанавливает глубину очереди (например, 32) для имитации параллельных запросов.
- `numjobs=<значение>`: Задает количество параллельных потоков ввода-вывода.
- `time_based=1, runtime=30`: Выполняет каждый тест в течение фиксированного времени (30 секунд).
- `registerfiles=1, fixedbufs=1`: Минимизация накладных расходов на настройку и управление буферами путем предварительной

регистрации файлов и использования фиксированных буферов.

- `prio=0`: Назначает наивысший приоритет ввода-вывода тестовым запросам.
- `cpus_allowed_policy=shared`: Распределяет использование ядер ЦП.
- `cpus_allowed=<номера-ядер-цп>`: Указывает номера ядер ЦП, разрешенные для использования, например 0-7).

Параметры, специфичные для рабочей нагрузки, такие как соотношение записи к чтению (`--rwmix`) и размер блока (`--bs`), варьировались в соответствии с конкретным тестовым случаем, как подробно описано в Таблице 1.

4.5. Конфигурация тестовой платформы

Оценка производительности проводилась на следующей конфигурации платформы:

- Процессор: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (4 ядра, 8 потоков)
- ОЗУ: 16 ГБ DDR4
- Нижележащее устройство хранения: null-диск, предоставленный модулем `null_blk` ядра Linux, сконфигурированный с размером 400 ГБ.

Важно отметить, что показатели производительности null-диска — это не скорость устройства, а скорее скорость, с которой операционная система может обрабатывать запросы ввода-вывода, которые никуда не ведут. Использование null-диска позволяет устранить узкие места производительности, связанные с более медленными физическими носителями информации, тем самым обеспечивая сфокусированный анализ накладных расходов и эффективности самого драйвера LS-BDD и его

механизмов отображения LBA. Однако важно отметить, что такая конфигурация не отражает напрямую характеристики производительности на физических HDD или SSD. Поэтому результаты тестирования производительности не могут быть сравнены ни с чем, кроме тестов на чистом null-диске без использования лог-структурированного подхода.

4.6. Тестирование целостности

Для проверки целостности данных на реализованном блочном устройстве использовался fio. Путем генерации и добавления контрольной суммы, которая вычисляется в соответствии с уникальным содержимым блока данных, становится возможным проверить корректность содержимого блока данных. Тестирование производилось на RAM-диске, из-за особенности работы с null-диска.

4.7. Результаты тестирования производительности и анализ

В рамках тестирования производительности были получены многочисленные графики результатов тестирования. Производительность модуля нас интересует в двух сценариях — без и с предварительной записью на накопитель, для заполнения таблицы LBA (далее — прогрев). Первый случай явно показывает базовое поведение лог-структурированного подхода, а именно таблицы отображения LBA при первых записях. Такое поведение является базовым, так как в основном — лог-структурированное хранение является промежуточным этапом в кэшировании, Redirect-On-Write или других технологиях. Количество операций в таких случаях различно, так как большинство структур данных не предоставляют функцию односоставной операции update, поэтому запись по уже существующему в таблице LBA требует больше действий.

В качестве основной конфигурации была выбрана структура данных — список с пропусками. Количество потоков IO было выбрано равным 8, с глубиной IO равной 32.

4.7.1. Производительность LS-BDD в сравнении с прямым доступом

Для полноценного сравнения производительности LS-BDD и прямого доступа к null-диску — рассмотрим тестовые случаи без предварительного прогрева диска и с, на основе списка с пропусками. Для начала рассмотрим первый вариант.

На графиках, описывающих среднее значение пропускной способности операций записи (рис. 1, 2) при размерах блока 128 КБ можно заметить довольно хаотичное поведение графика прямого доступа, в отличие от доступа к lsvbd1. Различное относительное положение графиков при разных размерах блока подтверждает тезис об особенностях измерения производительности null-диска. Предположительно, подобное поведение может быть обусловлено эффективностью кэшей для малых блоков. Блоки размера 128 КБ могут хорошо попадать в L1/L2 кэши, показывая высокую производительность. Улучшение производительности в конце при записи на LS-BDD может сигнализировать о заполнении таблицы LBA и кэшировании ее значений. Для подтверждения данной гипотезы — взглянем на графики при прогреве накопителя (рис. 3, 4).

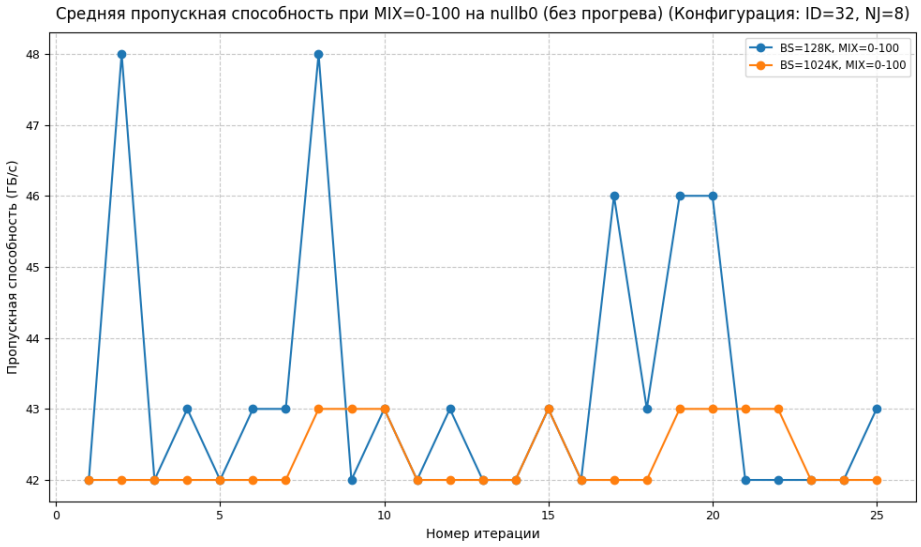

Рис. 1: Пропускная способность прямой записи

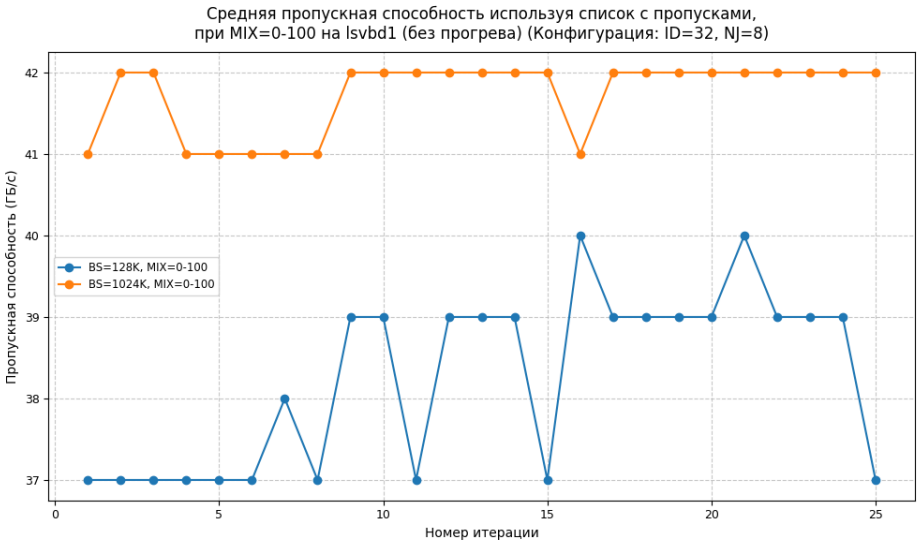

Рис. 2: Пропускная способность лог-структурированной записи

Результаты тестирования с прогревом показывают похожий уровень производительности. Причиной снижения производительности записи блоками размера 1024 КБ на null-диск является особенность тестовой системы. Несмотря на повышение нагрузки на LS-BDD из-за повышения кол-ва операций со таблицей отображений LBA — сохраняется примерный уровень производительности, отмеченный в конце графика 2. Поведение записи на null-диск сохранило особенности, выявленные при записи без прогрева. Тем не менее, можно заметить выравнивание линии графика при записи блоками размера 1024 КБ.

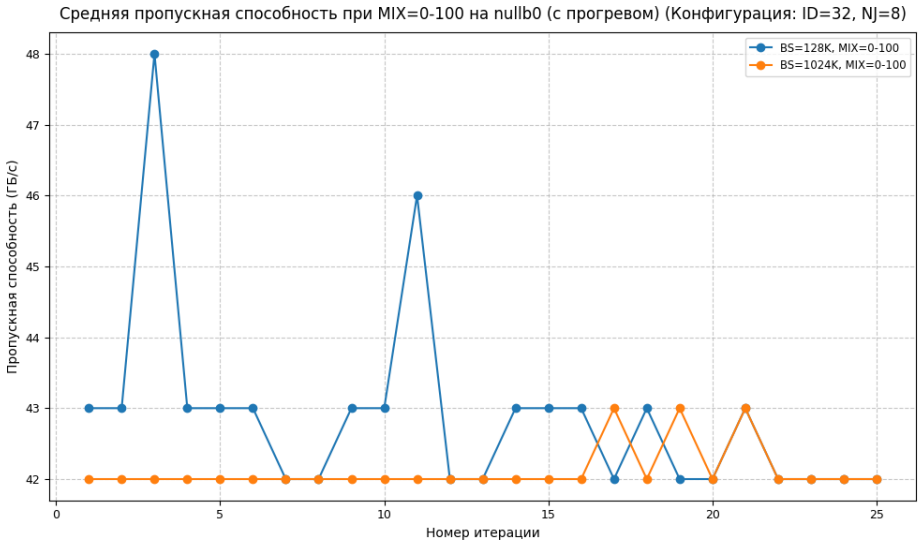

Рис. 3: Пропускная способность прямой записи с прогревом (список с пропусками)

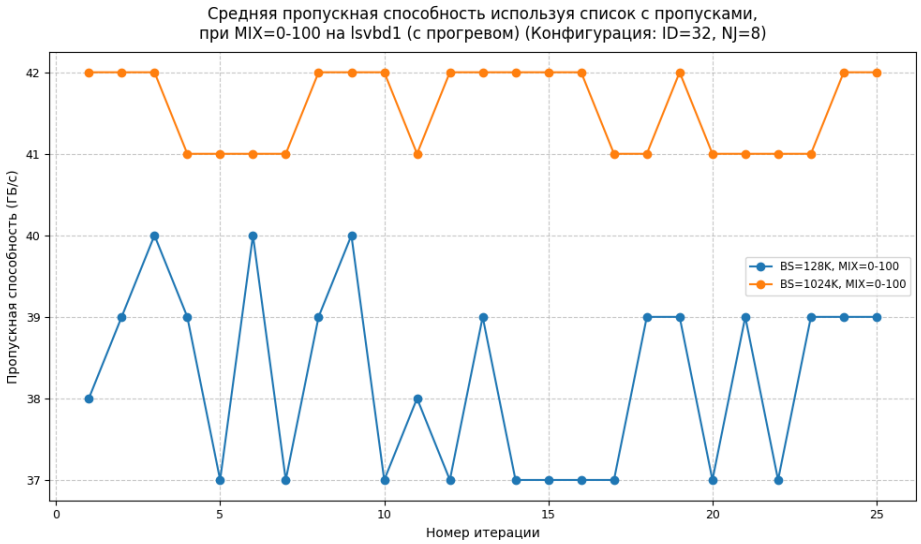

Рис. 4: Пропускная способность лог-структурированной записи с прогревом (список с пропусками)

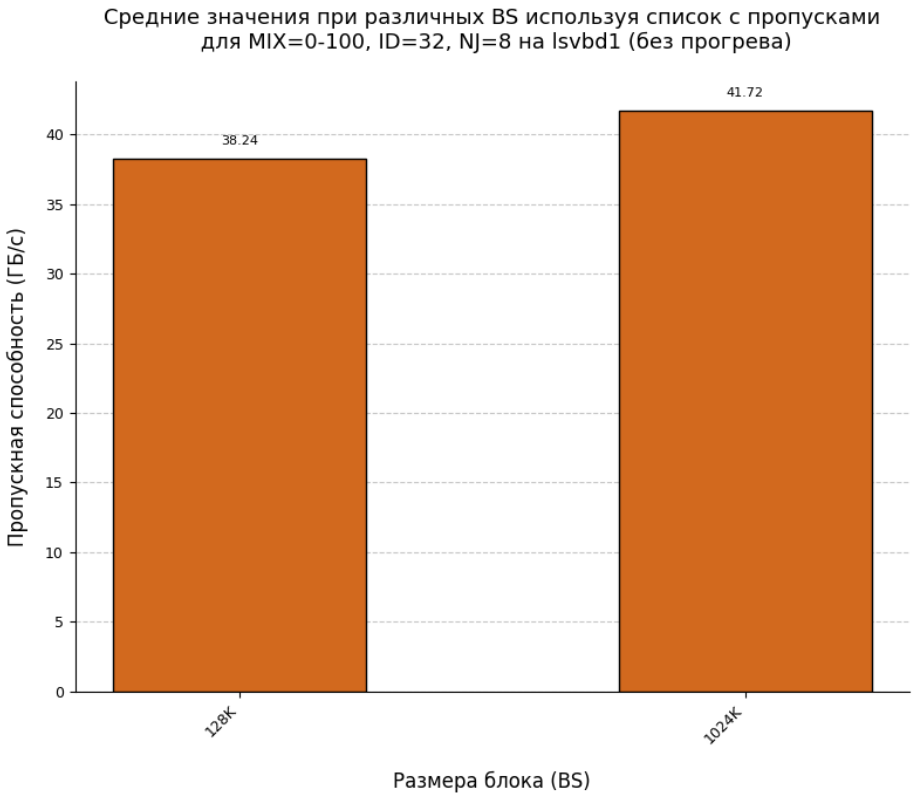

Рис. 5: Средние IOPS лог-структурированной записи без прогрева (список с пропусками)

Измерения при операциях чтения показали высокий уровень производительности. Производительность чтения на lsvbd1 не упала, что указывает на низкую степень дополнительной нагрузки, которая появляется за счёт взаимодействия с таблицей LBA. Графики равномерные.

Тестирование производительности реализации лог-структурированного блочного устройства на основе хеш-таблицы (рис. 6) показало более эффективную работу — в среднем на 5-7% при размере блоков в 128КБ (рис. 5) вне зависимости от того, происходил ли прогрев устройства или нет.

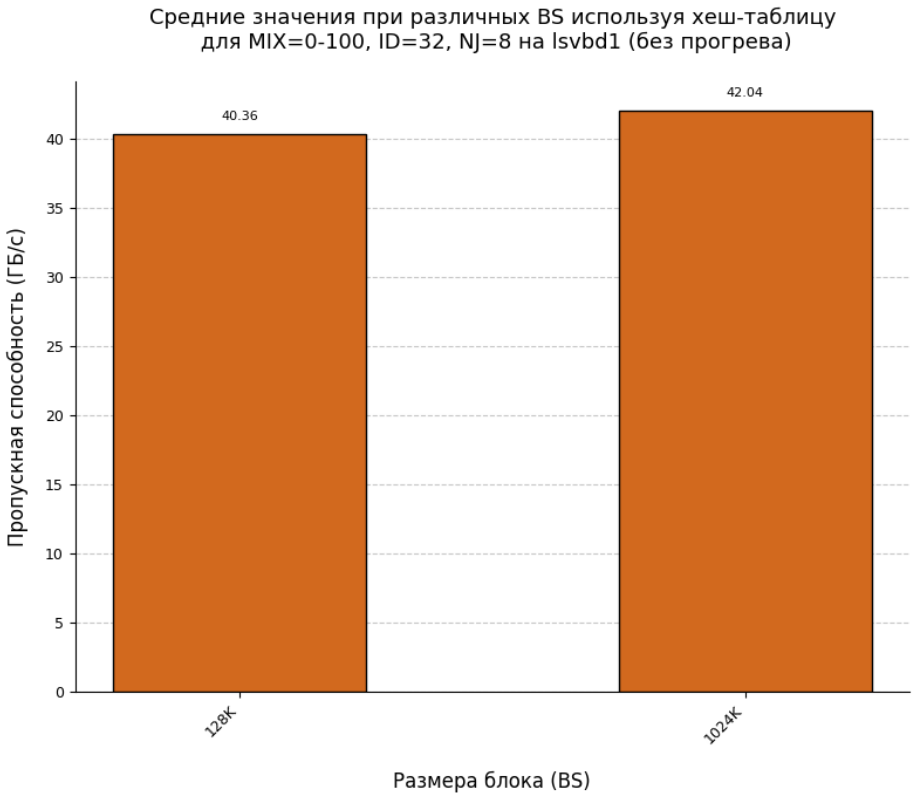

Рис. 6: Средние IOPS лог-структурированной записи без прогрева (хеш-таблица)

Несмотря на явное превосходство хеш-таблицы над списком с пропусками при операциях записи — список с пропусками может быть более эффективен в других случаях. Списки с пропусками превосходят в быстром извлечении диапазонов данных, что критично для блочных устройств при чтении последовательных блоков и работе сборщиков мусора. Графики чтения (рис. 7, 8) подтвердили данную гипотезу, показав превосходство списка с пропусками при чтении блоков размера 128 КБ.

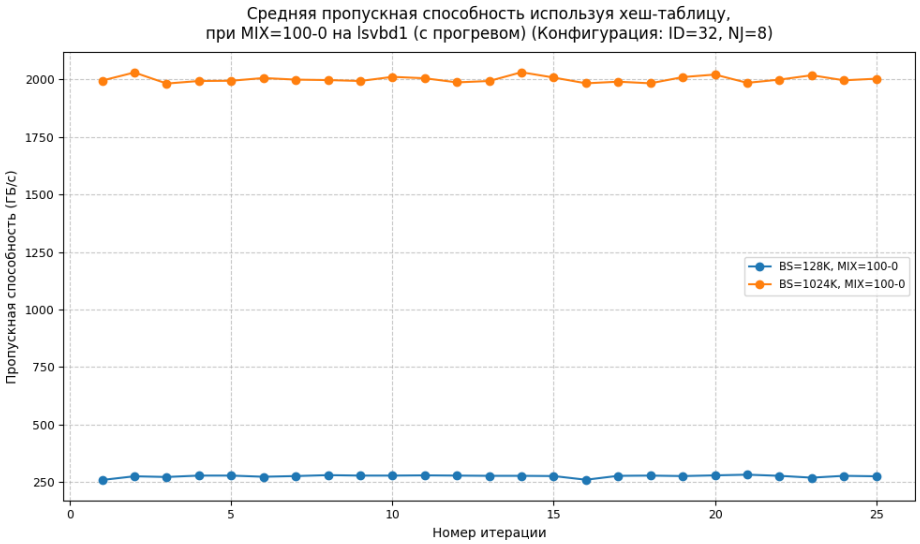

Рис. 7: Пропускная способность чтения (хеш-таблица)

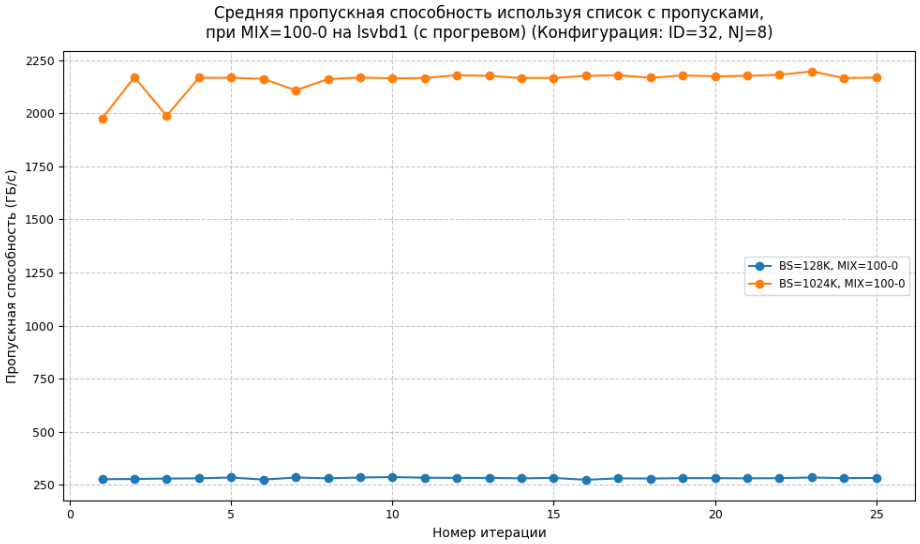

Рис. 8: Пропускная способность чтения (список с пропусками)

4.7.2. Влияние размера блока

Производительность (IOPS) обоих устройств показывает схожую закономерность — снижение с увеличением размера блока. Большие блоки требуют больше времени на копирование данных в/из буферов ядра, обработку в драйвере и передачу по шине данных.

На малых размерах блоков lsvbd1 показывает производительность примерно в 2 раза меньше, чем nullb0, что указывает на значительные

расходы направленные на лог-структурированное хранение. При размерах блоков 512K и 1024K производительность устройств практически уравнивается, что говорит о том, что на больших блоках доминируют другие факторы.

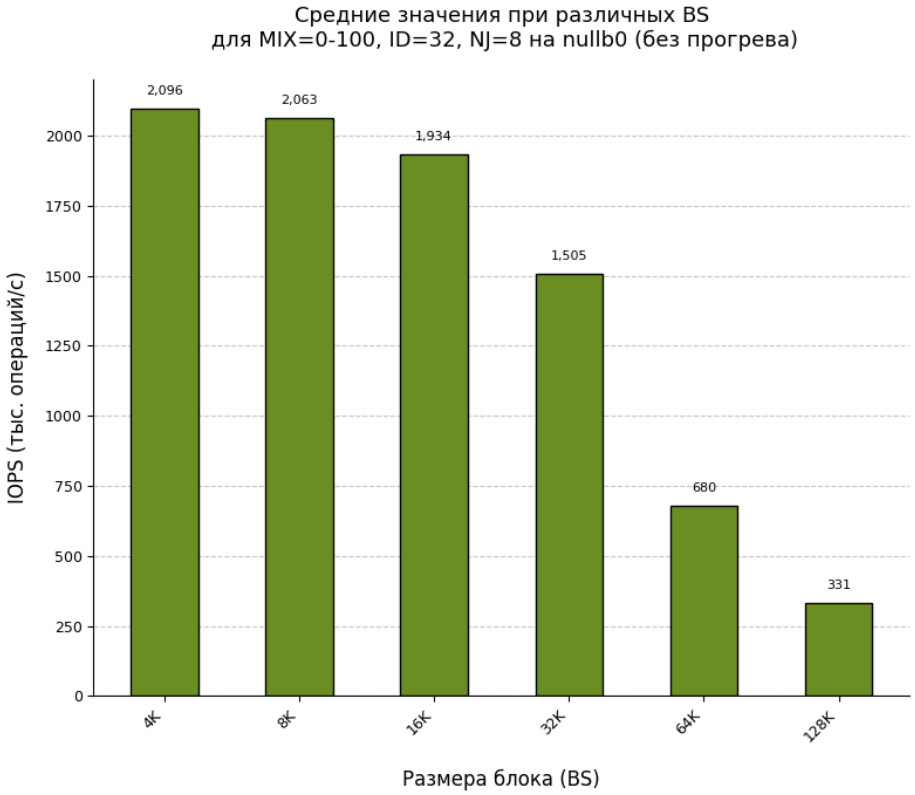

Рис. 9: Производительность прямой записи при различных BS

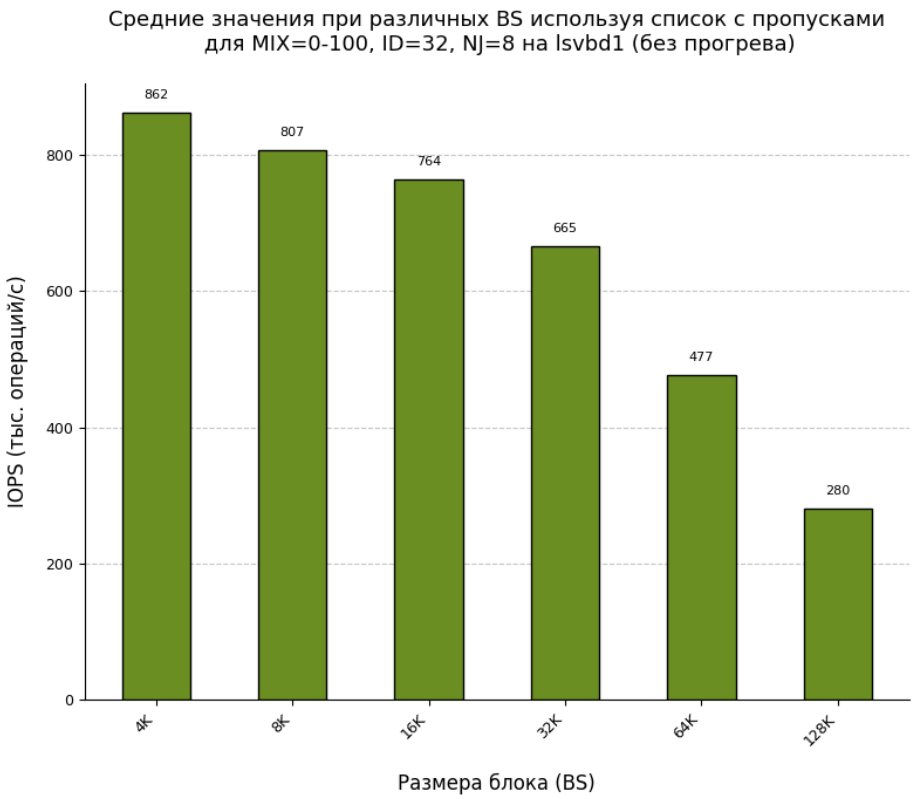

Рис. 10: Производительность лог-структурированной записи при различных BS

Графики описывающие 99-й перцентили задержек (рис. 11, 12) показывают, что задержки увеличиваются с ростом размера блока, что соответствует тенденции производительности. Список с пропусками демонстрирует в 1.5-2 раза более высокие задержки по сравнению с хеш-таблицей на малых блоках (4-16 КБ) из-за накладных расходов на поиск свободного пространства. На больших блоках (64 - 128 КБ) разница сокращается, поскольку доминируют операции копирования данных, а не управление метаданными. В случае с хеш-таблицей — оптимальный баланс производительности и задержек для lsvbd1 достигается при размерах блоков 8-16 КБ.

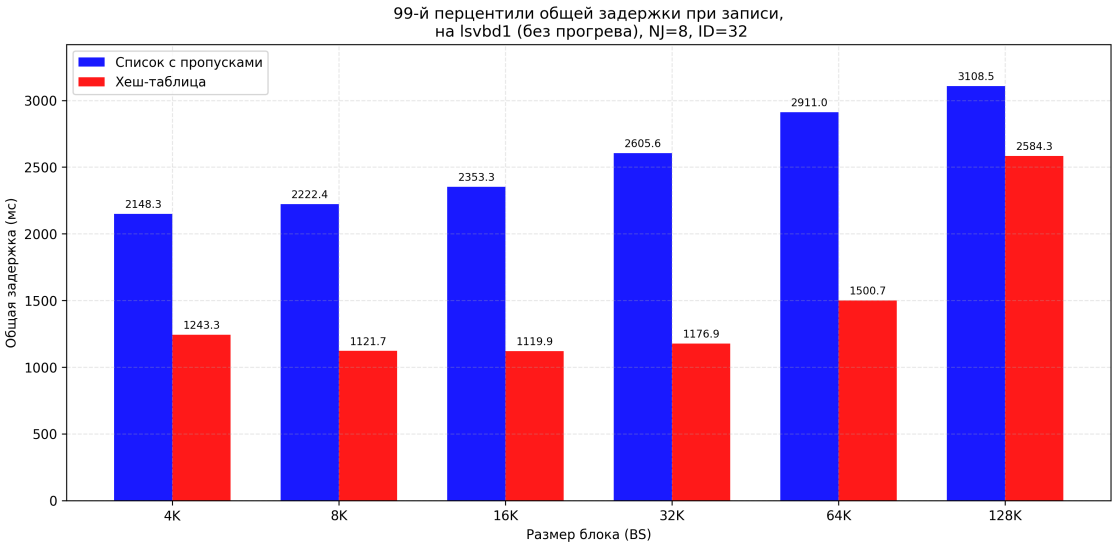

Рис. 11: Общие задержки при лог-структурированной записи

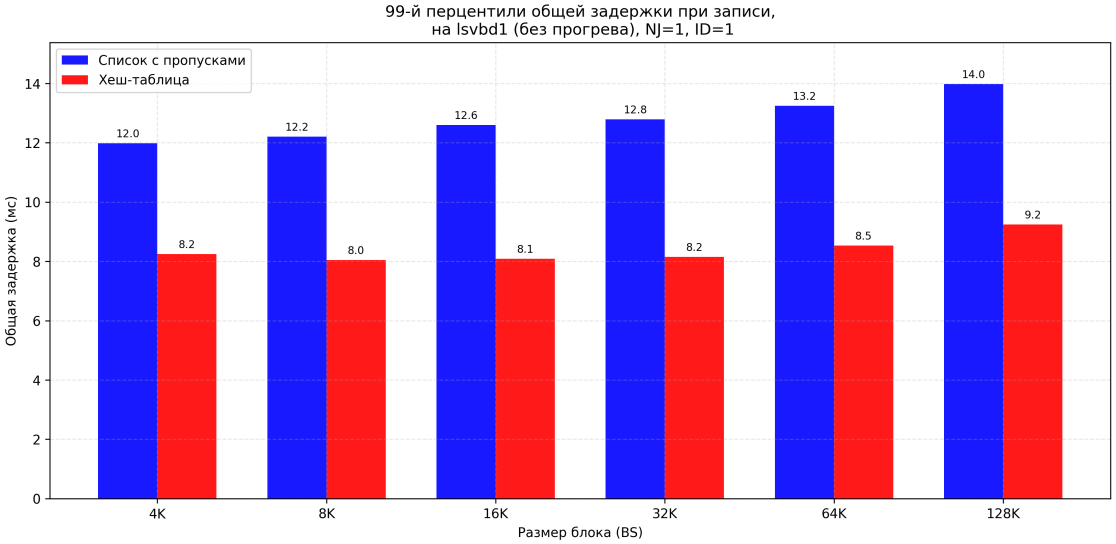

Рис. 12: Общие задержки при лог-структурированной записи (ID=1,NJ=1)

4.7.3. Масштабируемость и параллелизм

Для выявления особенностей работы реализованного модуля была проанализирована производительность при различном количестве единиц выполнения (Jobs number) и величине очереди IO запросов (IO depth).

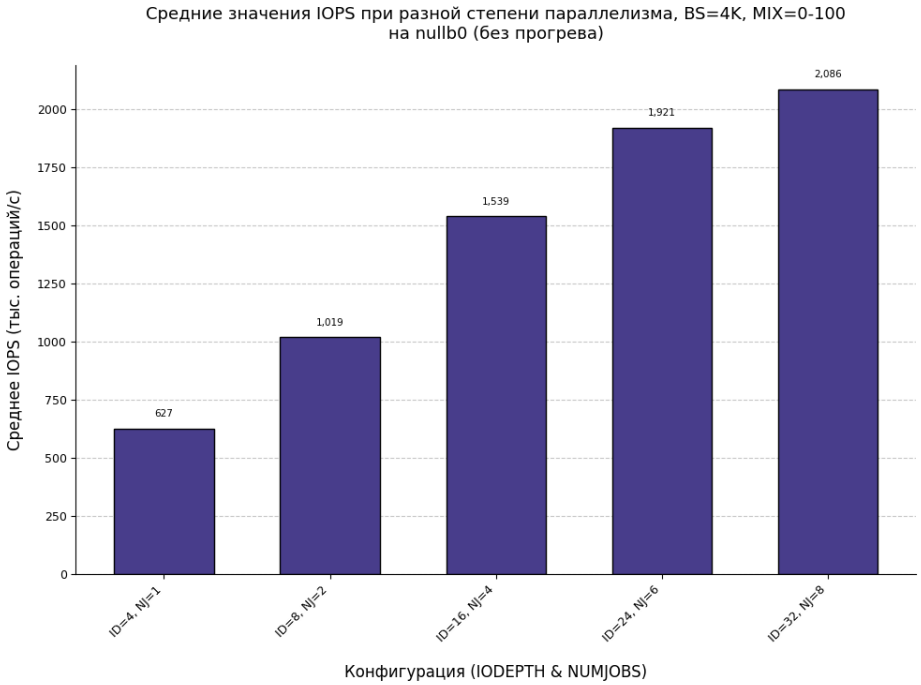

Рис. 13: Производительность прямой записи при различных NJ/ID (список с пропусками)

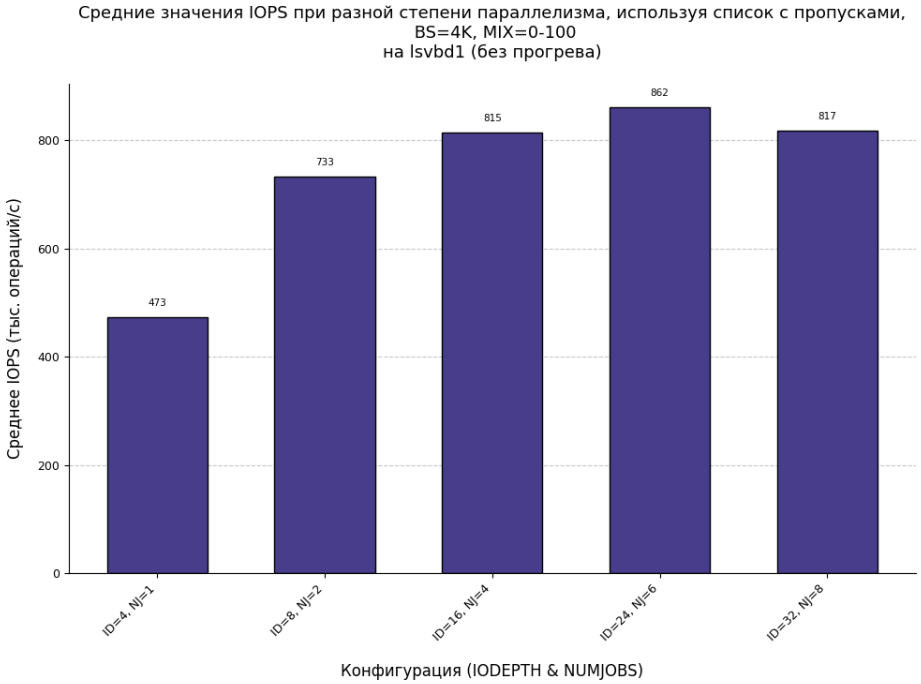

Рис. 14: Производительность лог-структурированной записи при различных NJ/ID (список с пропусками)

При тестировании производительности виртуального блочного устройства, nullb0 демонстрирует практически линейный рост произво-

дительности с некоторым замедлением при высоких значениях параллелизма, что характерно для виртуальных устройств без физических ограничений. Результаты лог-структурированного доступа показывают принципиально другую динамику. Абсолютные значения производительности nullb0 превышают показатели lsvbd1 в 2.4-2.6 раза, что объясняется накладными расходами лог-структурированного драйвера. lsvbd1 показывает более консервативное масштабирование с коэффициентом роста примерно 1.8x и наличием точки оптимума при `IODEPTH=24`. Снижение производительности может произойти в результате деградации при превышении оптимального числа конкурирующих операций, увеличения числа повторных попыток в lock-free операциях и влиянии кэш-когерентности процессора.

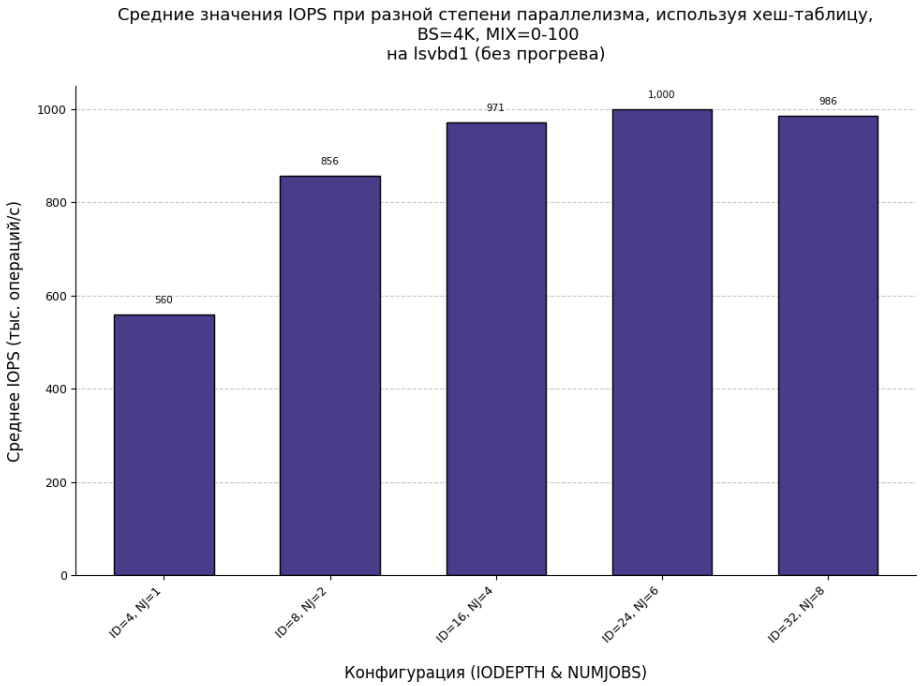

Рис. 15: Производительность лог-структурированной записи при различных NJ/ID (хеш-таблица)

График производительности lsvbd1 с использованием хеш-таблицы (рис. 15) показал более высокий уровень значений IOPS по сравнению со списком с пропусками(рис. 14) — в среднем на 17%. Такое различие в производительности обусловлено положительным отличием в ассимптотической сложности операций. На графике также можно заметить

наличие точки оптимума при `IODEPTH=24`, что говорит об оптимальном балансе между параллелизмом и конкуренцией за ресурсы. При классической реализации хеш-таблицы на основе одномерного массива — причиной снижения производительности хеш-таблицы `IODEPTH>24` может быть увеличение уровня кэш-промахов при большем числе операций.

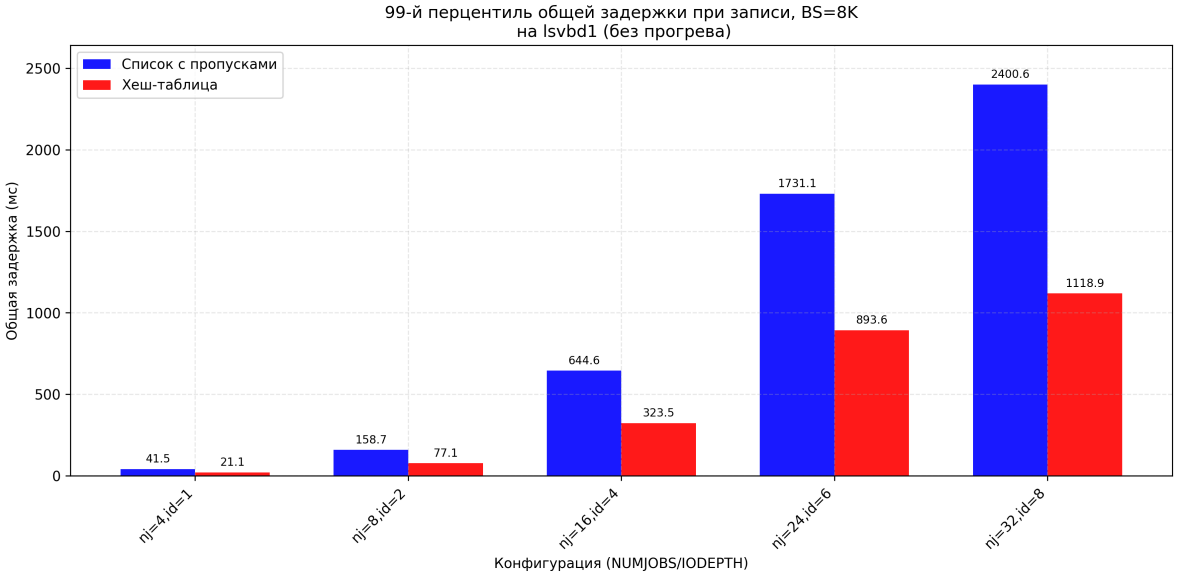

Рис. 16: Задержка лог-структурированной записи при различных NJ/ID

При увеличении общего уровня параллелизма с 6 до 8 одновременных операций (переход от конфигурации `ID=24, NJ=6` к `ID=32, NJ=8`) наблюдается классическая деградация производительности системы ввода-вывода. Снижение IOPS и экспоненциальный рост latency (рис. 16) обусловлены превышением оптимальной нагрузки системы и развитием конкуренции за ресурсы, включая конкуренцию за блокировки, накладные расходы на переключение контекста и деградацию эффективности кэш-памяти. Результаты подтверждают существование оптимальной рабочей точки системы в диапазоне 4-6 одновременных операций и размера очереди равной 16-24, за пределами которой дальнейшее увеличение параллелизма приводит к снижению общей производительности согласно закону убывающей отдачи.

4.7.4. Сравнительный анализ потребления памяти lock-free структур: хеш-таблицы и списка с пропусками

Потребление памяти реализованных структур данных существенно различается вследствие фундаментальных различий в их архитектуре (см. таблицу 2).

Узел списка с пропусками имеет переменный размер, поскольку количество указателей на следующие узлы определяется высотой текущего узла в вероятностной структуре. В отличие от этого, хеш-таблица организована как массив фиксированного размера, где каждый элемент (бакет) представляет односвязный список для разрешения коллизий. Основные затраты памяти в хеш-таблице связаны с необходимостью предварительной аллокации всех бакетов. В данной работе — размер хеш-таблицы установлен в 262144 бакетов, что обеспечивает стабильную, корректную производительность при выполнении операций записи без предварительного прогрева. Количество бакетов является важным в терминах производительности и оптимизации структуры данных. Большое количество бакетов результирует в большом количестве разбросанных по памяти структур, что может влиять на локальность кэшей. Также возможно избыточное выделение памяти, если размер нагрузки не совпадает с исходно предусмотренным значением, и некоторые бакеты не получают значений.

Анализ проводился для сценария записи 1 ТБ данных блоками по 8 КБ, что требует хранения 134217728 записей в таблице LBA отображений. Экспериментальные измерения показывают трёхкратное превосходство хеш-таблицы по эффективности использования памяти при равномерном распределении ключей. Средний размер узла списка с пропусками составляет 132 байта, что при полном заполнении структуры требует 16,5 ГБ оперативной памяти (1,65% от объёма хранимых данных). Хеш-таблица демонстрирует значительно более экономное потребление ресурсов — разница составляет приблизительно 12,5 ГБ в пользу хеш-таблицы.

Результаты анализа демонстрируют существенное преимущество хеш-

Таблица 2: Объемы памяти необходимые для аллокации в различных структурах данных. В процентах обозначено соотношение к общему объему записи.

Структура	Список с пропусками	Хеш-таблица
Узел	224 Б (худший случай)	32 Б
Голова	32 Б	16 Б + 262144 * 40 Б
Всего	28 ГБ = 2,8%	4 ГБ = 0,4%

таблицы в эффективности использования памяти. Это преимущество обусловлено детерминированным размером узлов и отсутствием избыточных указателей, характерных для вероятностных структур данных типа списка с пропусками.

4.7.5. Заключение

Эксперимент показал, что лог-структурированное блочное устройство LS-BDD на основе неблокирующих структур данных демонстрирует сопоставимую с прямым доступом производительность при работе с крупными блоками (512–1024 КБ), особенно при использовании хеш-таблицы. Однако при малых блоках (4–128 КБ) IOPS снижается примерно вдвое, что связано с накладными расходами на управление таблицей отображений LBA.

Размер блока критически влияет на эффективность. При увеличении блока производительность снижается. Оптимальный баланс производительности и задержек для lsvbd1 достигается при размерах блоков 8 КБ - 16 КБ. Масштабируемость LS-BDD ограничена: прирост производительности менее линейный, с оптимумом при глубине очереди 16-24 элемента. Хеш-таблица оказалась в среднем на 17% быстрее списка с пропусками в многопоточных сценариях.

По потреблению памяти хеш-таблица значительно эффективнее — в 3 раза экономичнее при записи 1 ТБ данных, благодаря фиксированному размеру узлов. Рекомендуется использовать хеш-таблицу в

системах с высокой нагрузкой и ограниченными ресурсами, а список с пропусками — для последовательного чтения и задач, связанных с извлечением диапазонов. Для составления выводов о практическом применении необходимы тесты на физических устройствах.

Заключение

В рамках данной работы были получены следующие результаты:

- реализованы неблокирующие список с пропусками и хеш-таблица;
- протестирован драйвер на основе реализованных структур;
- проведен анализ полученный результатов замеров производительности.

С реализованной функциональностью можно ознакомиться на Github[1].

Развитие данной работы может включать сравнительный анализ производительности с другими lock-free структурами данных, такими как B+-деревья и красно-черные деревья. Кроме того, можно рассмотреть реализацию функций персистентности для сохранения состояния отображения в энергонезависимой памяти. Было бы полезно реализовать полноценный Redirect-On-Write и сравнить производительность. Необходимо провести повторное тестирование производительности с использованием жестких накопителей в качестве накопителей.

[1]LS-BDD: https://github.com/qrutyy/ls-bdd

Список литературы

[1] Apesteguía. F. Chapter 22. The Z File System (ZFS). — URL: https://docs.freebsd.org/en/books/handbook/zfs/ (дата обращения: 25 июня 2025 г.).

[2] Axboe J. fio - Flexible I/O tester. — URL: https://fio.readthedocs.io/ (дата обращения: 25 июня 2025 г.).

[3] Brown N. A block layer introduction part 1: the bio layer. — URL: https://lwn.net/Articles/736534/ (дата обращения: 25 июня 2025 г.).

[4] Brown Trevor Alexander. Reclaiming Memory for Lock-Free Data Structures: There has to be a Better Way // Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing. — PODC '15. — New York, NY, USA : Association for Computing Machinery, 2015. — P. 261–270. — URL: https://doi.org/10.1145/2767386.2767436.

[5] Cohen Nachshon, Petrank Erez. Automatic memory reclamation for lock-free data structures // SIGPLAN Not. — 2015. — . — Vol. 50, no. 10. — P. 260–279. — URL: https://doi.org/10.1145/2858965.2814298.

[6] Freyer K. Practical lock-freedom, Technical Report UCAM-CL-TR-579, University of Cambridge Computer Laboratory. — URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-579.pdf (дата обращения: 25 июня 2025 г.).

[7] Google. LevelDB. — URL: https://github.com/google/leveldb (дата обращения: 25 июня 2025 г.).

[8] Harris Timothy L. A Pragmatic Implementation of Non-blocking Linked-lists // Distributed Computing / Ed. by Jennifer Welch. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2001. — P. 300–314.

[9] Hat. Red. Chapter 3. LVM Components. — URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/logical_volume_manager_administration/lvm_components (дата обращения: 25 июня 2025 г.).

[10] Herlihy M. The Art of Multiprocessor Programming. — URL: https://cs.ipm.ac.ir/asoc2016/Resources/Theartofmulticore.pdf (дата обращения: 25 июня 2025 г.).

[11] Hu Yiming, Yang Qing. DCD—disk caching disk: a new approach for boosting I/O performance // SIGARCH Comput. Archit. News. — 1996. — . — Vol. 24, no. 2. — P. 169–178. — URL: https://doi.org/10.1145/232974.232991.

[12] Jia Wenqing, Jiang Dejun, Xiong Jin. A High-Performance and Scalable Userspace Log-Structured File System for Modern SSDs // ACM Trans. Storage. — 2025. — . — Just Accepted. URL: https://doi.org/10.1145/3728645.

[13] Vo Hoang Tam, Wang Sheng, Agrawal Divyakant et al. LogBase: A Scalable Log-structured Database System in the Cloud. — 2012. — 1207.0140.

[14] Meyer H. Lock-free Data Structures for Data Stream Processing. — URL: https://btw.informatik.uni-rostock.de/download/workshopband/D1-6.pdf (дата обращения: 25 июня 2025 г.).

[15] Ousterhout John, Douglis Fred. Beating the I/O bottleneck: a case for log-structured file systems // SIGOPS Oper. Syst. Rev. — 1989. — . — Vol. 23, no. 1. — P. 11–28. — URL: https://doi.org/10.1145/65762.65765.

[16] Parallelizing Skip Lists for In-Memory Multi-Core Database Systems / Zhongle Xie, Qingchao Cai, H.V. Jagadish et al. // 2017 IEEE 33rd International Conference on Data Engineering (ICDE). — 2017. — P. 119–122.

[17] RAID arrays. — URL: https://www.kernel.org/doc/html/v4.14/admin-guide/md.html (дата обращения: 25 июня 2025 г.).

[18] Rosenblum Mendel, Ousterhout John K. The design and implementation of a log-structured file system. — 1992. — . — Vol. 10, no. 1. — P. 26–52. — URL: https://doi.org/10.1145/146941.146943.

[19] Write amplification analysis in flash-based solid state drives / Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas et al. // Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference. — SYSTOR '09. — New York, NY, USA : Association for Computing Machinery, 2009. — 9 p. — URL: https://doi.org/10.1145/1534530.1534544.

[20] de Jonge Wiebren, Kaashoek M. Frans, Hsieh Wilson C. The logical disk: a new approach to improving file systems // SIGOPS Oper. Syst. Rev. — 1993. — . — Vol. 27, no. 5. — P. 15–28. — URL: https://doi.org/10.1145/173668.168621.

[21] wcurtispreston. Snapshot 101: Copy-on-write vs Redirect-on-write. — URL: https://storageswiss.com/2016/04/01/snapshot-101-copy-on-write-vs-redirect-on-write/ (дата обращения: 25 июня 2025 г.).