Карпова Екатерина Олеговна
магистрант
МГТУ им. Н. Э. Баумана

Кострицкий Александр Сергеевич
старший преподаватель
МГТУ им. Н. Э. Баумана

МЕТОД ОПТИМИЗАЦИИ ПОТРЕБЛЕНИЯ ДИСКОВОГО ПРОСТРАНСТВА СИСТЕМОЙ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ POSTGRESQL

Аннотация: Разработан метод для оптимизации потребления дискового пространства СУБД PostgreSQL с помощью физического разделения хранения актуальных и исторических версий кортежей. Проведено сравнение потребления дискового пространства и скорости выполнения запросов от нагрузки на СУБД для разработанного метода и существующей реализации PostgreSQL.

Ключевые слова: PostgreSQL, дисковое пространство, «мертвые» кортежи, MVCC.

Введение. PostgreSQL — одна из наиболее популярных современных СУБД. Она заняла первую строчку рейтинга StackOverflow от 2025 года по используемости: 58% из 21 126 опрошенных профессиональных разработчиков применяют в работе PostgreSQL [1].

Таблицы и индексы в PostgreSQL значительно увеличиваются в размерах с течением времени [2]. Чрезмерное потребление диска негативно сказывается на работе приложений и самой СУБД и может привести к остановке работы системы, потере данных, а следовательно повлечь финансовые потери компании-разработчика [3]. Поэтому оптимизации его потребления СУБД PostgreSQL являются одним из приоритетных направлений разработки многих производителей ПО, например: Yandex Cloud [4], PostgresPro [5], OrioleDB [6] и других.

На основе существующих решений [7] был разработан модифицированный метод, позволяющий сократить потребление дискового пространства СУБД PostgreSQL.

Концепция метода. Далее будет рассмотрена математическая модель предлагаемого метода. Метод применим ко всем таблицам СУБД, а общий объем данных СУБД определяется суммарным размером всех таблиц. Поэтому достаточно рассмотреть оптимизацию целевой функции — потребление дискового пространства — на примере одной таблицы.

Пусть $\tau = 0,1,\dots,T$ — последовательность дискретных моментов времени. $r_n, n \in \mathbb{N}_0$ — множество версий кортежей, существующих в момент времени $t \in \tau$. Каждая версия из r_n в момент времени t может находиться в одном из трех состояний, определяемых функцией $S(r_n)$ в зависимости от следующих условий:

$$S(r_n) = \begin{cases} r_A, \text{если версия актуальна на момент времени } t \\ \quad \text{или удалена транзакцией на } t-1, \\ r_H, \text{если версия историческая, но используется} \\ \quad \text{активными транзакциями на момент времени } t, \\ r_V, \text{если версия историческая и не используется} \\ \quad \text{активными транзакциями на момент времени } t. \end{cases} \quad (1)$$

M — основное хранилище, $r_n \in M$, если $S(r_n) = r_A$. H — историческое хранилище, $r_n \in H$, если $S(r_n) \in r_H, r_V$. При этом $M \cap H = \emptyset$, а M_t, H_t — объемы основного и исторического хранилищ в момент времени $t \in \tau$ соответственно.

Пусть B — размер одного блока данных СУБД, а $\beta \in \mathbb{N}$ — минимальное количество блоков исторического хранилища, при достижении которого таблица рассматривается процедурой очистки. Тогда

$$b_t^H = \frac{H_t}{B} \quad (2)$$

— это количество блоков, занимаемых историческим хранилищем в момент времени $t \in \tau$. Пусть γ — период наложения эксклюзивной блокировки, δ — интервал ожидания завершения активных транзакций. Тогда условие завершения активных транзакций

$$g_t(\delta) = \begin{cases} 1, \text{если по истечении } \delta \text{ завершились} \\ \text{активные транзакции,} \\ 0, \text{иначе,} \end{cases} \quad (3)$$

условие принудительной активации процедуры очистки

$$b_t(\gamma, \delta) = \begin{cases} 1, \text{если } \gamma \mid t \text{ и } g_t(\delta) = 1, \\ 0, \text{иначе.} \end{cases} \quad (4)$$

Пусть ν — период запуска процедуры очистки «мертвых» кортежей. Тогда условие активации процедуры очистки

$$a_t(\nu) = \begin{cases} 1, \text{если } \nu \mid t, \\ 0, \text{иначе,} \end{cases} \quad (5)$$

условие прохождения порогового числа исторических блоков

$$p_t(\beta) = \begin{cases} 1, \text{если } b_t^H \geqslant \beta, \\ 0, \text{иначе,} \end{cases} \quad (6)$$

условие отсутствия в историческом хранилище используемых версий

$$c_t(\gamma, \delta) = \begin{cases} 1, \text{если } \forall r_n \in H: S(r_n) = r_V \text{ или } b_t(\gamma, \delta), \\ 0, \text{иначе.} \end{cases} \quad (7)$$

Тогда условие очистки исторического хранилища

$$u_t(\nu, \beta, \gamma, \delta) = a_t(\nu) p_t(\beta) c_t(\gamma, \delta). \quad (8)$$

При этом

$$H_{t+1} = \begin{cases} 0, \text{если } u_t(\nu, \beta, \gamma, \delta) = 1, \\ H_t, \text{иначе.} \end{cases} \quad (9)$$

Таким образом, параметры $\nu, \beta, \gamma, \delta$ оказывают влияние на объем исторического хранилища в момент времени $t + 1$. Тогда задачу минимизации потребления дискового пространства можно записать следующим образом.

$$F_{space}(\nu, \beta, \gamma, \delta) = \sum_{t \in \tau} \left(M_t + H_t(\nu, \beta, \gamma, \delta) \right) \to \min. \quad (10)$$

Необходимо выбрать такие значения параметров $\nu, \beta, \gamma, \delta$, чтобы значение F_{space} было минимальным.

$$(\nu^*, \beta^*, \gamma^*, \delta^*) = \arg \min_{\nu, \beta, \gamma, \delta} F_{space}(\nu, \beta, \gamma, \delta). \quad (11)$$

Пусть среднее время выполнения стандартных операций с данными для одного кортежа в PostgreSQL — $\overline{\tau}_{sel}, \overline{\tau}_{ins}, \overline{\tau}_{upd}, \overline{\tau}_{del}$, для предложенного метода — $\overline{\hat{\tau}}_{sel}, \overline{\hat{\tau}}_{ins}, \overline{\hat{\tau}}_{upd}, \overline{\hat{\tau}}_{del}$. Тогда ограничения на сохранение сравнимого времени выполнения стандартных операций с данными можно записать так.

$$\begin{cases} \frac{1}{T} \sum_{t \in \tau} \overline{\hat{\tau}}_{sel} \leqslant \frac{1{,}1}{T} \sum_{t \in \tau} \overline{\tau}_{sel}, \\ \frac{1}{T} \sum_{t \in \tau} \overline{\hat{\tau}}_{ins} \leqslant \frac{1{,}1}{T} \sum_{t \in \tau} \overline{\tau}_{ins}, \\ \frac{1}{T} \sum_{t \in \tau} \overline{\hat{\tau}}_{upd} \leqslant \frac{1{,}1}{T} \sum_{t \in \tau} \overline{\tau}_{upd}, \\ \frac{1}{T} \sum_{t \in \tau} \overline{\hat{\tau}}_{del} \leqslant \frac{1{,}1}{T} \sum_{t \in \tau} \overline{\tau}_{del}. \end{cases} \quad (12)$$

Обобщив все рассуждения, поставленную задачу можно записать следующим образом. Целевая функция (10):

$$F_{space}(\nu, \beta, \gamma, \delta) = \sum_{t \in \tau} \left(M_t + H_t(\nu, \beta, \gamma, \delta) \right) \to \min.$$

Ограничения (12):

$$\begin{cases} M_t \cap H_t = \emptyset, \\ \frac{1}{T}\sum_{t\in\tau}\bar{\hat{\tau}}_{sel} \leqslant \frac{1{,}1}{T}\sum_{t\in\tau}\bar{\tau}_{sel}, \\ \frac{1}{T}\sum_{t\in\tau}\bar{\hat{\tau}}_{ins} \leqslant \frac{1{,}1}{T}\sum_{t\in\tau}\bar{\tau}_{ins}, \\ \frac{1}{T}\sum_{t\in\tau}\bar{\hat{\tau}}_{upd} \leqslant \frac{1{,}1}{T}\sum_{t\in\tau}\bar{\tau}_{upd}, \\ \frac{1}{T}\sum_{t\in\tau}\bar{\hat{\tau}}_{del} \leqslant \frac{1{,}1}{T}\sum_{t\in\tau}\bar{\tau}_{del}. \end{cases}$$

Были сформулированы следующие требования и ограничения метода.

1. В файлах таблиц основного хранилища содержится только последняя или недавно удаленная версия кортежа и метаинформация о расположении предыдущих версий.
2. Исторические версии кортежей сохраняются только в одном отдельном хранилище, для которого предусмотрена периодическая очистка.
3. Хранилище исторических версий кортежей представлено отдельной от актуальных данных директорией, которая может физически располагаться на одном диске с основными данными или на отдельном.
4. Скорость выполнения базовых операций может быть снижена относительно существующей реализации не более, чем на 10% в среднем для операций с одним кортежем.

Схема расположения версий кортежей представлена на рисунке 1.

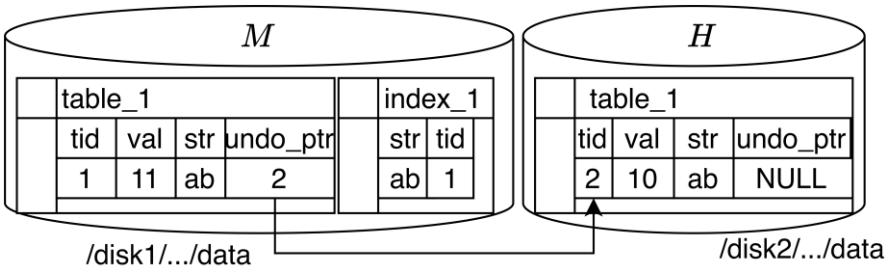

Рис. 1. Схема расположения версий кортежей

Для реализации раздельного хранения версий строк необходимо модифицировать схему хранения данных в PostgreSQL. Существует несколько способов внесения таких изменений: расширение с помощью интерфейса Table Access Methods, внешнее расширение для удаленного источника, модификация исходного кода ядра СУБД. Наиболее предпочтительным способом является расширение с помощью интерфейса TableAM, так как он обеспечивает наиболее полную совместимость при минимальных потерях в производительности.

Описание схемы физического хранения данных. Файлы каждой БД хранятся в отдельной директории внутри родительской директории, соответствующей кластеру СУБД. Директории БД называются по идентификатору, хранимому в системной таблице pg_database. Таблицы БД описываются системной таблицей pg_catalog. В СУБД существуют табличные пространства, описание которых хранится в системной таблице pg_tablespace. Они ссылаются на физическую директорию, в том числе на отдельном диске. В данной работе используется механизм табличных пространств для раздельного хранения актуальных и исторических версий, как показано на рисунке 2. Таблица split_history_storage хранится рядом с таблицами, созданными с помощью предложенного метода. Она содержит записи о соответствии файлов с актуальными данными таблицы файлам с историческими кортежами, физически расположенным в отдельной директории.

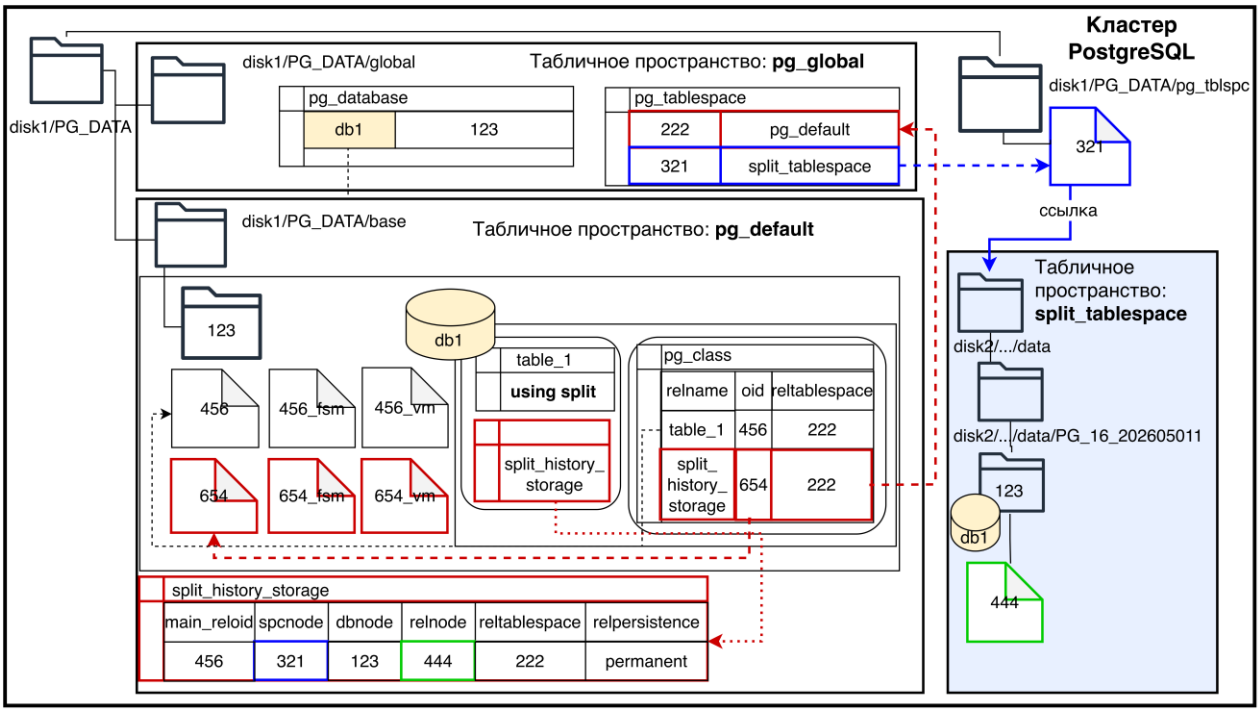

Рис. 2. Схема физического размещения данных

Структурная оптимизация операций с данными. Для оптимизации времени выполнения операций с данными был реализован следующий функционал.

1. При обновлении кортежа, приоритетной является запись новой версии строки на место прежней. Актуальная версия строки может быть помещена в другое место, только если ее новый размер превосходит доступный ранее. Это позволяет не обновлять индексы при каждой записи, что является достаточно трудоемкой операцией, а также снижает фрагментацию данных. Однако, если обновляемое поле является индексируемым, то индексы придется обновить, иначе историческая версия станет недоступной. Механика похожа на HOT-обновления СУБД PostgreSQL, но учитывает раздельное хранение исторических и актуальных кортежей [8]. При переносе версии на новое место, предыдущая версии сохраняется в историческом хранилище, а на прежнем месте создается ссылка на новую версию, размещаемую на той же странице.
2. Для поиска подходящей для записи страницы логично переиспользовать механизмы СУБД. Страницы таблиц при стандартных нагрузках заполняются чаще всего последовательно, поэтому при поиске нового места для записи данных в первую очередь на наличие свободного места проверяется та же страница, которая подходила для записи последний раз. Если страница не подходит, то перед созданием новой дополнительно проверяются карты свободного пространства FSM [9] и последняя страница заполняемого файла. Это позволяет не выполнять обход всех страниц и не создавать новые страницы, если остается место не недавно использованных существующих страницах.
3. Процесс поиска подходящей страницы для основного и исторического хранилища схож, но не одинаков. Так как историческое хранилище заполняется исключительно последовательно, для него не имеет смысла использовать карту свободных областей. При поиске проверяется только последняя страница файла.
4. Процесс очистки «мертвых» кортежей является периодическим, достаточно частым и реализуемым на стороне системы, а не пользователя. Это возможно, так как исторические версии строк лежат на отдельном диске, который можно очистить единовременно, удалив весь файл целиком, при этом анализировать его содержание не нужно. Достаточно проверить кортежи основного хранилища, а разрыв связи основной версии кортежа с историческими — нетрудоемкая операция. Период запуска данной процедуры, пороговый размер хранилища, период наложения эксклюзивной блокировки и время

ожидания завершения активных транзакций необходимо подобрать эмпирически: они должны позволять выявить наибольшее число моментов возможной очистки при наименьшем снижении эффективности операций над данными.

Определение оптимальных значений параметров. Для достижения поставленной цели с соблюдением перечисленных ограничений необходимо определить оптимальные значние параметров $\nu, \beta, \gamma, \delta$.

Параметры варьировались следующим образом.

1. Периоды очистки ν: 5 мс, 10 мс, 25 мс.
2. Размеры хранилища в блоках β: 8, 16, 32.
3. Периоды наложения эксклюзивной блокировки γ: 500 мс, 2000 мс, 5000 мс.
4. Интервалы ожидания завершения активных транзакций δ: 5 мс, 20 мс.

Замеры проводились следующим образом.

1. Для каждого значения периода создавался отдельный контейнер с СУБД PostgreSQL и расширением.
2. Создавалась таблица из 2 целочисленных полей и 1 текстового поля. В СУБД PostgreSQL 15.1 размер целочисленного типа integer составляет 4 байта. Тип text не имеет фиксированной длины, но накладывает ограничение на размер строки в 1 ГБ. Изначальный размер таблицы — 100 000 кортежей.
3. Последовательно выполнялись разные операции над данными: вставка нового кортежа с монотонно растущим первичным ключом целочисленного типа, чтение кортежа по добавленному ключу, обновление этого кортежа, удаление кортежа с минимальным значением первичного ключа. Максимальное количество операций каждого типа было установлено в 50000.
4. Операции выполнялись транзакциями по 100 операций в каждой.
5. Проводилось 3 независимых прогона.
6. Измерения производились внутри сервера PostgreSQL с использованием clock_timestamp().

На рисунке 3 представлена визуализация результатов замеров среднего потребления дискового пространства для разработанного метода при различных значениях параметров.

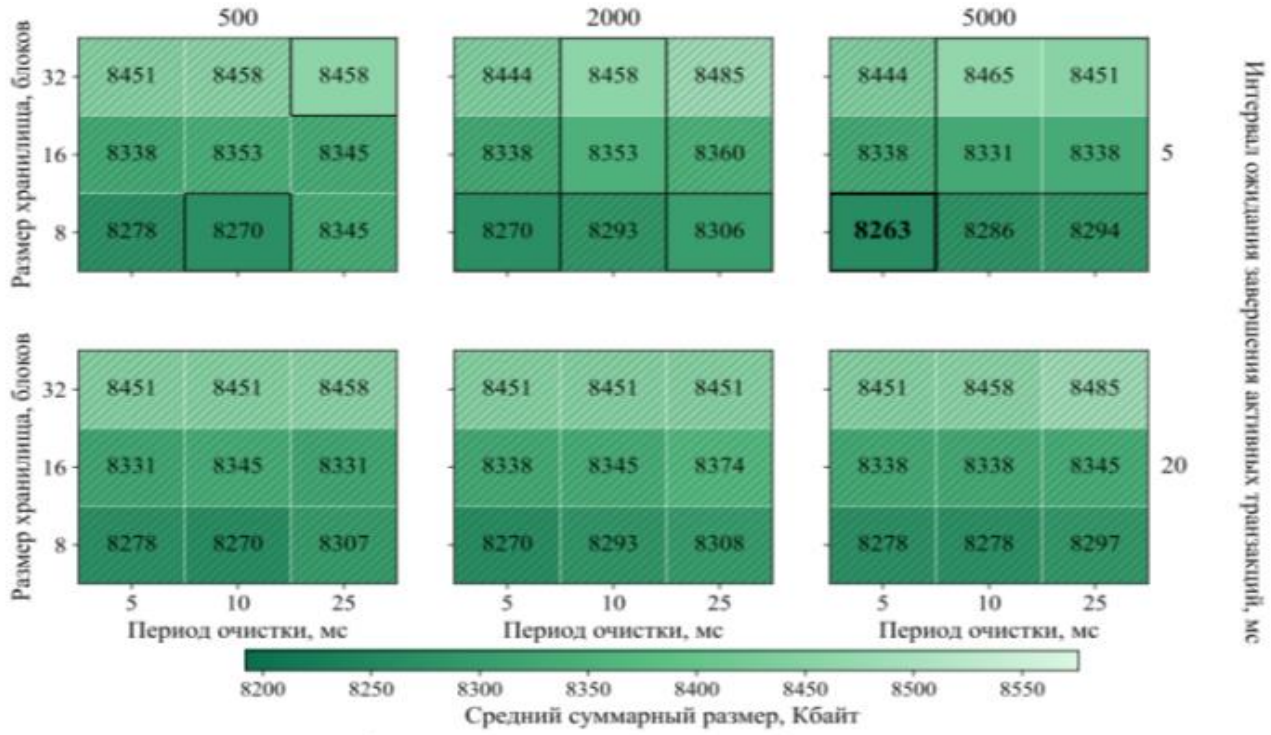

Рис.3. Среднее потребление дискового пространства хранилищем при различных значениях параметров

Минимальный средний размер хранилища — 8263 КБ — был достигнут при следующих значениях параметров.

1. Период очистки ν^*: 5 мс.
2. Размер хранилища β^*: 8 блоков.
3. Период наложения эксклюзивной блокировки γ^*: 5000 мс.
4. Интервал ожидания завершения активных транзакций δ^*: 5 мс..

Исследование производительности разработанного ПО. Целью исследования является оценка объемов потребления дискового пространства СУБД PostgreSQL и производительности основных операций по работе с данными с использованием разработанного метода. Также, нужно сравнить указанные показатели при одинаковой нагрузке с показателями в существующей реализации PostgreSQL.

Замеры проводились с использованием выбранных ранее значений параметров. Для PostgreSQL была выбрана периодичность очистки, выбранная компанией Amazon, как оптимальная — 1 секунда [10].

В качестве методики тестирования была выбрана YCSB [11], позволяющая сравнивать пропускную способность и задержки различных БД в рабочих нагрузках. Одной из типов нагрузки, стандартизируемых YCSB, является профиль интенсивного обновления, когда в равном соотношении производятся запросы на чтение и изменение данных, что соответствует требованиям исследования. YCSB оперирует простыми записями из некоторого ключа и набора полей фиксированной длины. Это близко к тем OLTP-сценариям, где приложение часто читает и обновляет отдельные кортежи по ключу. При этом обновления разделяются на два типа: 5% изменяет индексируемый атрибут, а 95% изменяет неиндексируемое поле. Такое соотношение является одним из стандартных в YCSB и соотносится с реальной практикой использования СУБД.

Для генерации нагрузки на СУБД использовался фреймворк BenchBase [12]. Он поддерживает взаимодействие с СУБД PostgreSQL, обеспечивая многопоточную генерацию нагрузки и предоставляя метрики производительности обработки запросов.

На рисунке 4 представлена визуализация зависимости среднего времени выполнения операций от нагрузки на СУБД. Различие составило не более 10% для всех операций при нагрузке от 500 до 8500 операций в секунду, как того требуют ограничения.

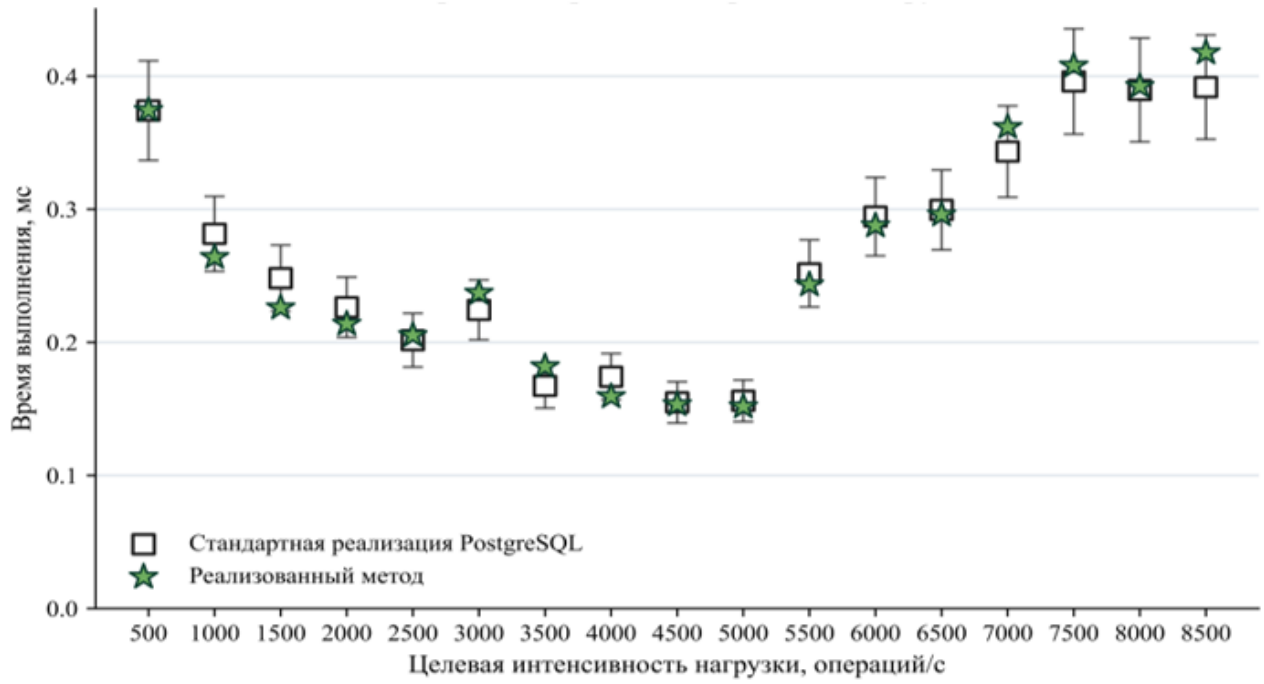

Рисунок 3. Зависимость среднего времени выполнения операций от нагрузки на СУБД

На рисунке 5 представлена визуализация зависимости потребления дискового пространства от нагрузки на СУБД.

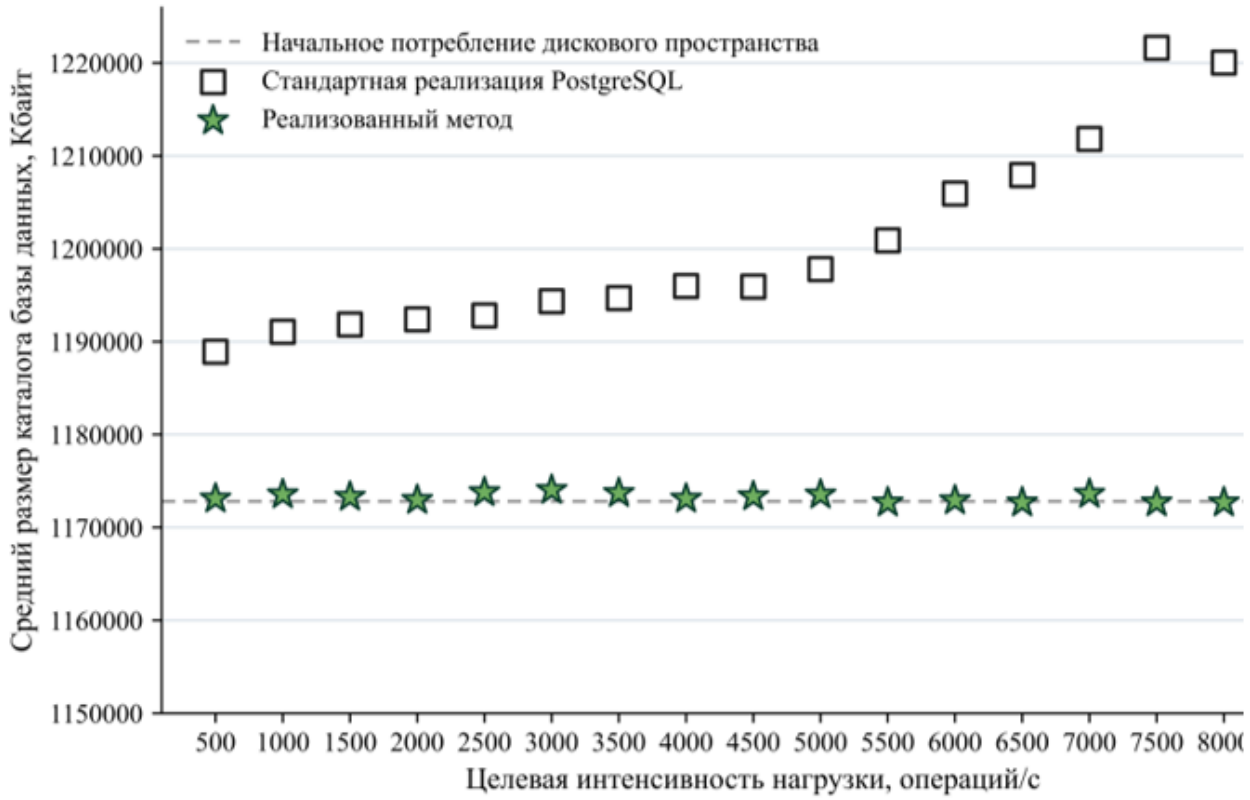

Рисунок 4. Зависимость потребления дискового пространства от нагрузки на СУБД

В разработанном методе среднее потребление дискового пространства при нагрузке от 500 до 8500 операций в секунду на 2,36% ниже, чем в стандартной реализации. При этом, можно дополнительно сравнить прирост потребления дискового пространства. Изначальный объем БД был одинаковым — 1 172 805,9 КБ. Для разработанного метода прирост составил 630,1 КБ, что на 98% меньше, чем в существующей реализации — 34 346,1 КБ.

Заключение. При сохранении сопоставимой скорости выполнения операций по работе с данными удалось достичь снижения потребления дискового пространства СУБД. Среднее потребление сокращено на 2,36%, при этом особо показательным является сокращение прироста с течением времени под постоянной нагрузкой на 90%. Это связано с тем, что раздельное хранение позволило целиком удалять исторические версии, возвращая занимаемое ранее пространство операционной системе. В PostgreSQL для этого требуется процесс VACUUM FULL, но он требует для своей работы длительной эксклюзивной блокировки, дополнительного пространства на диске и не может быть запущен при постоянной интенсивной нагрузке, в отличие от разработанного метода.

Практическую ценность решение представляет для OLTP-систем, учётных сервисов, каталогов и реестров с регулярным изменением записей. Применение метода менее оправдано в аналитических нагрузках, где преобладают полнотабличные чтения и диапазонные выборки.

Список литературы:

1. Результаты опроса разработчиков, проведённого в 2025 году StackOverflow [Электронный ресурс]. Режим доступа: https://survey.stackoverflow.co/2025/technology#1-databases (дата обращения: 08.06.2026).
2. Официальный сайт PGConf.Russia 2023 [Электронный ресурс]. Режим доступа: https://pgconf.ru/talk/1589479 (дата обращения: 08.06.2026).

3. Eun, Park Ju and Jang, Minyoung. A Study on Optimizing Disk Utilization of Software-Defined Storage. KIPS Transactions on Computer and Communication Systems, 2023, т. 12, № 4, с. 135–142.
4. Официальный сайт Yandex Cloud: миграция базы данных из Managed Service for PostgreSQL в Object Storage [Электронный ресурс]. Режим доступа: https://yandex.cloud/ru/docs/tutorials/dataplatform/mpg-to-objstorage (дата обращения: 08.06.2026).
5. Доклады PGConf.Russia 2022 [Электронный ресурс]. Режим доступа: https://pgconf.ru/talk/1589011 (дата обращения: 08.06.2026).
6. Официальный сайт OrioleDB [Электронный ресурс]. Режим доступа: https://www.orioledb.com (дата обращения: 08.06.2026).
7. Е. О. Карпова, А. С. Кострицкий. Существующие решения по уменьшению потребления дискового пространства системой управления базами данных в применении к PostgreSQL. Вектор научной мысли, 2025, № 7(24), с. 127 – 137.
8. Документация по механизму HOT-обновлений в СУБД PostgreSQL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/storage-hot.html (дата обращения: 08.06.2026).
9. Документация по механизму FSM в СУБД PostgreSQL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/storage-fsm.html (дата обращения: 08.06.2026).
10. Документация Amazon по выбранным настройкам СУБД PostgreSQL [Электронный ресурс]. Режим доступа: https://docs.aws.amazon.com/prescriptive-guidance/latest/postgresql-maintenance-rds-aurora/autovacuum.html (дата обращения: 08.06.2026).
11. Официальный репозиторий бенчмарков методики YCSB [Электронный ресурс]. Режим доступа: https://github.com/brianfrankcooper/YCSB/ (дата обращения: 08.06.2026).
12. Документация фреймворка BenchBase [Электронный ресурс]. Режим доступа: https://db.cs.cmu.edu/projects/benchbase/ (дата обращения: 08.06.2026).