УДК 004.65

Сравнительный анализ реализации модели «ключ — значение» в PostgreSQL и Redis: оценка задержек и пропускной способности

Волошин Ратмир Анатольевич — студент бакалавриата 3-го курса направления 38.03.05 «Бизнес-информатика». Научные интересы: информационные системы, разработка и исследование характеристик приложений для работы с базами данных. E-mail: vratmir@yandex.ru

Хомоненко Анатолий Дмитриевич — доктор техн. наук, профессор. Профессор кафедры «Информационные и вычислительные системы». Научные интересы: информационные системы, обработка больших данных, вероятностное моделирование геоинформационных систем, генетические алгоритмы, информационная безопасность. E-mail: khomon@mail.ru

Юдников Степан Сергеевич — магистр, аспирант кафедры «Информационные и вычислительные системы». Научные интересы: информационные системы, разработка и исследование характеристик приложений для работы с базами данных. E-mail: s.yudnikovv@gmail.com

Петербургский государственный университет путей сообщения Императора Александра I, Россия, 190031, Санкт-Петербург, Московский пр., 9

Для цитирования: Волошин Р. А., Хомоненко А. Д., Юдников С. С. Сравнительный анализ реализации модели «ключ — значение» в PostgreSQL и Redis: оценка задержек и пропускной способности // Интеллектуальные технологии на транспорте. 2026. № 3 (47). С. 44–55. DOI: 10.20295/2413-2527-2026-347-44-55

Аннотация. *Предлагается экспериментально обоснованный подход к сравнению производительности реализации модели «ключ — значение» в PostgreSQL и Redis.* ***Цель:*** *оценка задержек и пропускной способности операций чтения и записи при различной степени параллельной нагрузки.* ***Методы:*** *анализ архитектур PostgreSQL (MVCC, WAL, индексы B-tree, GIN, типы jsonb и hstore) и Redis (in-memory-хранение, строки и хэши, TTL, pipelining); создание экспериментального стенда; генерация тестовой таблицы объемом 1 000 000 записей; нагрузочное тестирование с использованием pgbench и redis-benchmark при параллельности от 1 до 128 клиентов; измерение пропускной способности, средней и хвостовой задержек (p95, p99).* ***Результаты:*** *Redis обеспечивает в 5–10 раз меньшую задержку чтения и в 3–5 раз более высокую пропускную способность по сравнению с PostgreSQL для точечных операций GET/SET; использование pipelining позволяет дополнительно повысить пропускную способность в 2–3 раза; PostgreSQL сохраняет преимущество в универсальности, поддерживая транзакции, сложные выборки и долговременное хранение.* ***Практическая значимость:*** *разработанная методика тестирования и полученные количественные оценки применимы при выборе архитектуры хранения для высоконагруженных информационных систем, включая системы мониторинга и трекинга на транспорте.*

Ключевые слова: *PostgreSQL, Redis, модель «ключ — значение», задержка, пропускная способность, высоконагруженные транзакции, jsonb, hstore, in-memory-хранилище*

2.3.5 — *математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей (технические науки);*
2.9.8 — *интеллектуальные транспортные системы (технические науки)*

Введение

Скорость доступа к данным напрямую влияет на работу современных информационных систем. Интернет-магазин проверяет корзину и остатки, банковский сервис читает параметры операции, система авторизации обращается к токенам, а игровой сервер постоянно обновляет счетчики. Во всех этих случаях снова и снова выполняются простые действия вроде «получить значение по ключу», «изменить значение» или «увеличить счетчик» [1]. Именно такая нагрузка характерна для модели «ключ — значение» (Key-Value, KV).

Сама модель устроена просто, но итоговая производительность сильно зависит от того, в какой СУБД она реализована. В PostgreSQL ключ и значение обычно размещаются в строке таблицы. Ключ индексируется, значение можно хранить в text, jsonb или hstore, а доступ выполняется через SQL-запросы [2]. Вместе с тем PostgreSQL обеспечивает транзакции, контроль целостности и долговременное хранение. За эти возможности приходится платить дополнительной работой WAL, MVCC и индексов [3].

Redis изначально создавался как хранилище структур данных в оперативной памяти [4]. Для GET, SET, HGET и HSET путь обработки короче, поскольку нужные структуры уже находятся в памяти, а команды передаются компактным протоколом [5]. Это дает низкую задержку. Но у такого подхода есть своя цена: нужно учитывать объем оперативной памяти, режим персистентности и сетевой RTT.

Цель исследования состоит в сравнении реализации KV-модели в PostgreSQL и Redis. Основное внимание уделяется скорости чтения и записи при росте параллельной нагрузки, после чего на основе измерений формируются рекомендации по выбору архитектуры хранения.

Для достижения цели решаются следующие задачи:

1. Сравнение вариантов хранения KV-данных в PostgreSQL с типами text, jsonb и hstore и в Redis с типами String и Hash.

2. Построение единой методики, по которой можно сопоставить чтение и запись в обеих системах.

3. Подготовка скриптов, позволяющих воспроизводить одинаковую нагрузку.

4. Обработка полученных результатов и подготовка практических рекомендаций.

Теоретические основы модели «ключ — значение»

Сущность модели и виды операций

В модели «ключ — значение» каждая запись определяется уникальным ключом, которому соответствует некоторое значение [6]. В роли значения могут выступать обычная строка, число, бинарный объект, JSON-документ или сериализованная структура приложения. Поэтому одна и та же схема подходит и для простого счетчика, и для более сложного пользовательского объекта.

Набор операций у KV-модели обычно невелик. Значение записывают через SET, читают через GET, обновляют или удаляют с помощью DEL. EXISTS проверяет наличие ключа, INCR увеличивает счетчик, TTL задает срок жизни записи. При большой нагрузке одной средней задержки уже недостаточно: p95 и p99 показывают, насколько медленными становятся запросы в хвосте распределения [7].

В [1] сравнивались шесть одинаковых профилей нагрузки. Redis в них показал самую высокую пропускную способность и самую низкую задержку, что связано с in-memory-архитектурой. PostgreSQL приближался к нему прежде всего в сценариях, где преобладало чтение, однако при этом сильнее загружал CPU.

Реализация Key-Value в PostgreSQL

PostgreSQL относится к объектно-реляционным СУБД [2]. Если в ней требуется хранить данные по модели Key-Value, обычно создают таблицу, где поле key становится первичным ключом или получает уникальный индекс [8]. Для самого значения можно выбрать несколько вариантов:

• text подходит для самого простого случая, когда значение хранится как строка, а поиск по ключу выполняется через B-tree-индекс;

• jsonb хранит JSON в бинарном виде и позволяет использовать GIN-индекс при поиске по вложенным полям [3];

• hstore предназначен для плоских текстовых пар «ключ — значение» внутри одной ячейки.

Главный плюс PostgreSQL здесь не в скорости одиночного GET, а в том, что KV-данные остаются частью реляционной СУБД. Их можно связывать внешними ключами, включать в транзакции и использовать в аналитических запросах. В [8] показано, что hstore и jsonb действительно позволяют воспроизводить поведение KV-хранилищ. При этом расходы на WAL и MVCC никуда не исчезают.

Реализация Key-Value в Redis

Redis работает как сервер структур данных в оперативной памяти [4]. Для обычной пары «ключ — значение» используется String с командами SET и GET. Когда объект состоит из нескольких полей, удобнее Hash с HSET и HGET [5]. В этом случае отдельное поле можно изменить без полной перезаписи всего объекта.

Низкая задержка Redis связана с тем, что команды выполняются над структурами, уже размещенными в памяти. Но реальная скорость зависит не только от этого. На нее влияют сетевой RTT, размер значения, число одновременно работающих клиентов и выбранный режим сохранности [9]. В [1] задержка Redis оставалась стабильной при изменении размера данных, тогда как у PostgreSQL результат заметнее зависел от сценария.

Сетевые задержки можно частично сократить с помощью pipelining. Redis позволяет отправить сразу несколько команд, не ожидая отдельного ответа после каждой. За счет этого уменьшается число циклов «запрос — ответ» и растет количество операций в секунду.

Сравнение архитектурных подходов

Архитектура PostgreSQL и Redis по-разному распределяет работу между скоростью и надежностью [10]. PostgreSQL выполняет запрос внутри транзакционной модели, пишет изменения в WAL, создает новые версии строк и обслуживает индексы. Все это добавляет задержку, зато дает гарантии ACID [2]. Redis меняет структуры прямо в памяти. Сохранность при этом зависит от RDB-снимков или AOF-журнала, поэтому настройки персистентности напрямую влияют на компромисс между быстродействием и надежностью [4].

Иногда KV-подход выигрывает даже на отдельных OLAP-сценариях, если данные заранее хранятся в денормализованном виде [11]. Но как только появляются сложные аналитические запросы, связи между сущностями и необходимость работать с несколькими таблицами, возможности PostgreSQL становятся важнее чистой скорости доступа по ключу [6].

Методика исследования производительности

Объект исследования и состав тестовых данных

Объектом исследования выбрано хранилище пользовательских параметров и счетчиков. Такой сценарий встречается и в веб-приложениях, и в транспортных информационных системах [12]. Тестовый набор состоит из 1 000 000 записей. Длина ключа меняется от 16 до 64 символов, а значение представляет собой JSON-объект размером от 256 байт до 4 Кбайт.

Чтобы не сводить эксперимент к одному типу нагрузки, использованы три профиля:

• R80/W20 моделирует систему, где 80 % операций приходится на чтение и 20 % — на запись;

• W80/R20, наоборот, создает нагрузку с 80 % операций записи и 20 % операций чтения;

• R50/W50 задает равное соотношение чтения и записи.

Число параллельных клиентов менялось от 1 до 128. Такой диапазон позволяет увидеть не только работу системы при одиночных запросах, но и момент, когда из-за конкуренции за ресурсы рост производительности начинает замедляться.

Сценарии хранения и скрипты эксперимента

Для PostgreSQL проверялись три способа хранения:

1. В PG-1 ключ задается как text PRIMARY KEY, а значение хранится в text.

2. В PG-2 ключ также задается как text PRIMARY KEY, значение хранится в jsonb, дополнительно используется GIN-индекс.

3. В PG-3 ключ имеет тип text PRIMARY KEY, атрибуты хранятся в hstore и индексируются через GIN.

Для Redis использовались два варианта:

1. RD-1 основан на String и командах SET/GET.

2. RD-2 использует Hash и команды HSET/HGET.

Ниже приведен фрагмент SQL-скрипта, который создает таблицы PostgreSQL.

```
sql
CREATE EXTENSION IF NOT EXISTS hstore;

CREATE TABLE kv_text (
  key_name text PRIMARY KEY,
  value_text text NOT NULL,
  updated_at timestamp NOT NULL DEFAULT
  now()
);

CREATE TABLE kv_jsonb (
  key_name text PRIMARY KEY,
  value_doc jsonb NOT NULL,
  updated_at timestamp NOT NULL DEFAULT
  now()
);

CREATE TABLE kv_hstore (
  key_name text PRIMARY KEY,
  attrs hstore NOT NULL,
  updated_at timestamp NOT NULL DEFAULT
  now()
);

CREATE INDEX idx_kv_jsonb_gin ON
kv_jsonb USING gin (value_doc);
CREATE INDEX idx_kv_hstore_gin ON
kv_hstore USING gin (attrs);
```

Примеры операций чтения и записи:

```
sql
-- Чтение
SELECT value_text FROM kv_text WHERE key_
name = 'user:100500: profile';

-- Запись с обновлением при конфликте
INSERT INTO kv_text(key_name, value_text)
VALUES ('user:100500: profile',
'{"plan":"premium"}')
ON CONFLICT (key_name) DO UPDATE
SET value_text = EXCLUDED.value_text,
updated_at = now();
```

Примеры команд Redis:

```
bash
SET user:100500: profile '{"plan":"premium
","region":"spb"}'
GET user:100500: profile
HSET user:100500 plan premium region spb
status active
HGET user:100500 plan
```

Чтобы сравнение PostgreSQL и Redis отражало не только лабораторный максимум, но и поведение под нагрузкой, одной метрики недостаточно. Нужны показатели, которые одновременно показывают объем выполненной работы и стабильность времени отклика. Для высоконагруженных OLTP-систем, в том числе транспортных и логистических приложений [12], это особенно важно. При конкуренции за ресурсы среднее значение может выглядеть приемлемо, хотя часть запросов уже выполняется заметно медленнее [13].

Другой проблеме — фрагментации данных в MySQL и ее влиянию на чтение и запись — посвящена [14]. Здесь сравниваются уже две разные архитектуры, дисковая реляционная СУБД и in-memory-хранилище структур данных. В KV-сценариях сами операции очень короткие, зато выполняются часто. Поэтому даже небольшие накладные расходы внутри СУБД заметно отражаются на итоговом числе операций в секунду [1].

Метрики оценки и порядок измерений

В эксперименте используются две основные метрики: throughput и latency. Throughput

показывает, сколько успешных операций система выполняет за секунду, то есть какой поток запросов она фактически выдерживает [10]. Latency показывает время одной операции. Среднее значение удобно для общего сравнения, но при росте нагрузки важнее становятся p95 и p99. Например, p99 показывает границу, быстрее которой завершились 99% запросов, а оставшийся 1% оказался медленнее [7].

Отдельно измеряется объем, занимаемый данными. Эта величина нужна не только для технического сравнения. PostgreSQL хранит основной массив на диске и использует буферный кэш, а Redis размещает данные в оперативной памяти. Поэтому одинаковое число мегабайт в этих системах имеет разную стоимость при масштабировании.

Каждый сценарий сначала проходит прогрев (warm-up), после чего запускается серия измерений. Первый результат в расчет не включается. Во время него PostgreSQL может заполнять буферный кэш, Redis создает внутренние структуры, а клиентские соединения еще только выходят на рабочий режим. В итоговые таблицы попадает медиана нескольких повторов.

При тестировании записи отдельно фиксируется режим долговечности. PostgreSQL работает в обычной конфигурации с WAL и подтверждением транзакции. Для Redis используется AOF everysec, то есть вариант, в котором скорость не ставится полностью выше сохранности данных [9].

Преимущество по пропускной способности рассчитывается по формуле K = Tredis / Tpg, где Tredis обозначает throughput Redis, а Tpg — throughput PostgreSQL. Для задержки отношение строится наоборот: Klat = Lpg / Lredis.

Анализ результатов эксперимента

Сравнение объема хранения

Табл. 1 и рис. 1 показывают, сколько памяти или дискового пространства занял один и тот же тестовый набор в разных вариантах хранения.

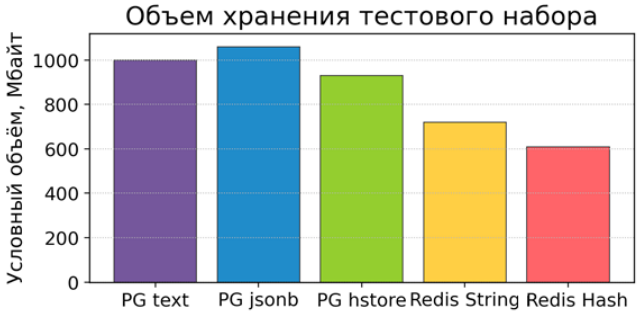


Рис. 1. Сравнение объема хранения тестового набора

По внутреннему объему Redis действительно выглядит компактнее. RD-2 Hash занимает 610 Мбайт, RD-1 String — 720 Мбайт. У PostgreSQL результат находится в диапазоне от 930 Мбайт для PG-3 до 1060 Мбайт для PG-2. Но эти цифры нельзя напрямую переводить в стоимость хранения: Redis требует оперативной памяти [4], тогда как PostgreSQL держит основной массив на диске и использует буферный кэш. Для большого объема холодных данных такой вариант может оказаться дешевле [2].

Таблица 1

Сравнение объема хранения тестового набора

Вариант	Данные, Мбайт	Служебные структуры, Мбайт	Итого, Мбайт	Комментарий
PG-1 key + text	780	220	1000	Базовая таблица и B-tree-индекс
PG-2 key + jsonb	820	240	1060	jsonb и GIN-индекс
PG-3 key + hstore	720	210	930	Компактнее для плоских атрибутов
RD-1 String	720	–	720	Ключи и строки в памяти
RD-2 Hash	610	–	610	Компактнее при группировке полей

Таблица 2

Сравнение оперативности чтения (GET)

Вариант	GET, ops/s	Avg, мс	p95, мс	p99, мс	Комментарий
PG-1 key + text	45 000	2,10	3,80	6,40	Стабильно при индексе
PG-2 key + jsonb	38 000	2,60	4,50	7,20	Дополнительная стоимость jsonb
PG-3 key + hstore	40 000	2,40	4,20	6,90	Близко к jsonb
RD-1 String	180 000	0,30	0,55	1,10	Минимальный путь чтения
RD-2 Hash	160 000	0,36	0,62	1,25	Чтение поля или объекта

Сравнение оперативности чтения

Численные результаты чтения по ключу собраны в табл. 2.

На чтении по ключу преимущество Redis видно особенно хорошо. RD-1 String достигает 180 000 GET ops/s при средней задержке 0,30 мс и p95 0,55 мс. Для PG-1 key + text получено 45 000 ops/s, 2,1 мс в среднем и 3,8 мс по p95. Причина различий связана с самим путем выполнения запроса. Redis получает значение непосредственно из памяти [5]. PostgreSQL выполняет SQL-запрос, проходит индекс, обращается к heap-строке и проверяет видимость ее версии [3]. Сходная картина наблюдалась и в [1].

Сравнение оперативности записи

Результаты для SET и UPDATE приведены в табл. 3.

На рис. 2 для чтения и записи сопоставлены задержки p95, рассчитанные по экспериментальным данным из табл. 2 и 3.

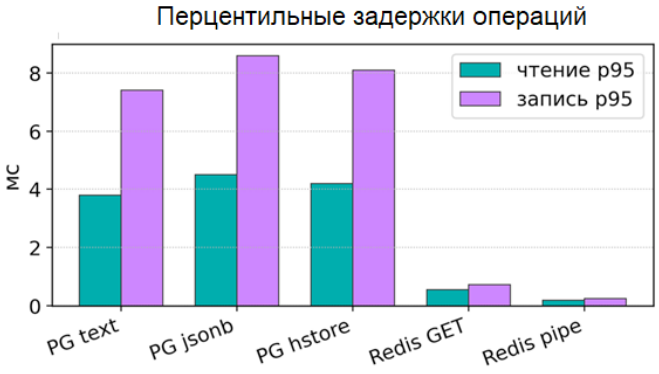


Рис. 2. Перцентильные задержки (p95) операций GET и SET/UPDATE

Запись сильнее нагружает PostgreSQL, чем чтение. Система пишет данные в WAL, обновляет

Таблица 3

Сравнение оперативности записи (SET/UPDATE)

Вариант	SET/UPDATE, ops/s	Avg, мс	p95, мс	p99, мс	Комментарий
PG-1 key + text	24 000	4,5	7,4	12,0	WAL и B-tree
PG-2 key + jsonb	20 000	5,2	8,6	14,5	Перезапись документа дороже
PG-3 key + hstore	22 000	4,9	8,1	13,2	Стоимость обновления атрибутов
RD-1 String AOF	135 000	0,42	0,72	1,60	Компромисс сохранности
RD-2 Hash AOF	120 000	0,50	0,85	1,85	Изменение поля

таблицу и индексы, а при UPDATE создает новую версию строки [2]. В PG-1 получено 24 000 операций в секунду при p95 7,4 мс. Redis в режиме RD-1 String AOF выполняет 135 000 операций в секунду при p95 0,72 мс. Он меняет структуру в памяти, а дополнительные расходы зависят от режима persistence [9]. Поэтому и в приводимых измерениях, и в [1] Redis показывает более низкую и более стабильную задержку.

Оценка поведения при росте параллельности

В табл. 4 показано, как меняется пропускная способность при увеличении числа параллельных клиентов.

При росте параллельности системы выходят на пределы по-разному [10]. Уже при одном клиенте Redis выполняет 38 000 GET ops/s против 7500 у PostgreSQL. При 32 клиентах значения достигают 180 000 и 45 000 ops/s соответственно. На 64 клиентах обе системы почти выходят на максимум: 190 000 GET ops/s у Redis и 47 000 у PostgreSQL. При 128 клиентах показатели снижаются до 185 000 и 44 000 ops/s. У PostgreSQL дополнительную конкуренцию создают индексы, WAL и блокировки [2, 3]. Redis Open Source традиционно выполняет команды в основном потоке, поэтому короткие операции обрабатываются быстро, но длинная команда способна задержать остальные запросы [4].

Еще заметнее разрыв становится при pipelining, поскольку клиент реже ждет отдельный ответ после каждой команды. В [9] Redis-кэширование сократило задержку чтения с 30 до 1,66 секунды, пропускная способность записи выросла на 50 %.

Рекомендации по выбору решения

Результаты эксперимента вместе с данными работ [1, 10] позволяют разделить сценарии применения PostgreSQL и Redis по характеру нагрузки и требованиям к данным.

PostgreSQL лучше подходит в следующих случаях:

- данные должны сохраняться долговременно и оставаться согласованными;
- в системе используются сложные SQL-запросы, соединения и аналитика [2];
- нужны транзакции и ограничения целостности [8];
- объем данных больше доступной оперативной памяти;
- необходимо хранить историю и проводить аудит изменений.

Redis лучше соответствует другому набору требований:

- критична задержка менее 1 мс [5];
- основная нагрузка состоит из обращений по ключу [4];
- хранятся временные данные, например, сессии, кэш или счетчики;
- система должна выполнять десятки тысяч операций в секунду [1];
- при выбранном режиме сохранности допустима потеря небольшого объема данных при сбое [9].

Таблица 4

Влияние параллельности на пропускную способность

Клиенты	PG GET, ops/s	Redis GET, ops/s	PG SET, ops/s	Redis SET, ops/s	Вывод
1	7500	38 000	4200	28 000	Redis быстрее без параллельности
8	32 000	120 000	18 000	92 000	Обе системы используют ресурсы
32	45 000	180 000	24 000	135 000	Redis сохраняет преимущество
64	47 000	190 000	24 500	138 000	Рост ограничен сервером и сетью
128	44 000	185 000	22 000	130 000	Появляются очереди и рост p99

Во многих прикладных системах разумнее не выбирать одну СУБД, а использовать PostgreSQL и Redis вместе:

• такой вариант подходит, когда одновременно есть горячие и холодные данные [12];

• часто запрашиваемые значения нужно отдавать быстро, но основной набор данных должен оставаться надежным;

• при этом архитектура допускает асинхронную синхронизацию.

Типичный сценарий выглядит так. Сначала приложение запрашивает ключ в Redis. Если его там нет, данные читаются из PostgreSQL, помещаются в Redis с TTL и затем возвращаются клиенту [15]. При изменении записи PostgreSQL остается источником истины, а соответствующий ключ в Redis удаляется или инвалидируется.

В [9] показано, что сочетание Redis-кэширования, пула соединений и пакетной обработки помогает масштабировать Node.js-приложения, работающие с PostgreSQL.

Заключение

В исследовании сравнивались два способа реализации модели «ключ — значение»: PostgreSQL и Redis. Сначала были рассмотрены особенности их архитектуры и типовые операции, затем обе системы проверялись под одинаковой нагрузкой [1–6, 10]. Это позволило связать различия в задержке и пропускной способности не только с результатами теста, но и с тем, как каждая СУБД обрабатывает запрос.

Эксперимент на наборе из 1 000 000 записей дал несколько практических результатов:

1. Для точечных GET/SET Redis обеспечивает в 5–10 раз меньшую задержку чтения и в 3–5 раз более высокую пропускную способность по сравнению с PostgreSQL [1].

2. Задержка Redis остается стабильнее при изменении размера данных. У PostgreSQL результат сильнее зависит от конкретного сценария работы [8].

3. При увеличении параллельности до 64 клиентов Redis сохраняет заметное преимущество. На 128 клиентах рост прекращается и начинает проявляться ограничение однопоточной архитектуры [4].

4. Pipelining дает Redis еще один прирост. В рассмотренных условиях пропускная способность повышается в 2–3 раза [5].

5. PostgreSQL медленнее при точечном доступе, но его сильная сторона заключается в другом. СУБД поддерживает транзакции, сложные запросы и долговременное хранение, поэтому лучше подходит для основных бизнес-данных [2, 3].

На практике эти свойства дополняют друг друга. PostgreSQL целесообразно оставить основной базой данных и источником истины, а Redis использовать для горячих ключей, временных записей и операций с высокой частотой [9, 11]. Выбор схемы при этом зависит от профиля нагрузки, требований к сохранности и бюджета сопровождения.

Полученные результаты можно использовать при проектировании высоконагруженных информационных систем. Они полезны администраторам баз данных, разработчикам и техническим архитекторам, в том числе при работе с транспортными и логистическими системами [12, 15].

Для интеллектуальных транспортных систем эти различия имеют вполне прикладной смысл. В железнодорожной инфраструктуре постоянно обрабатываются данные о подвижном составе; телеметрия бортовых датчиков; сведения о скорости, температуре, вибрации и нагрузке на ось. К этому добавляются сессионные данные диспетчеров и авторизационные токены. Во многих запросах объект определяется одним идентификатором, например номером вагона, локомотива, станции или маршрута. При пиковой нагрузке таких обращений одновременно приходят десятки и сотни. В этих условиях Redis с задержкой чтения в 5–10 раз ниже и стабильными p95/p99 можно использовать для горячих данных о движении поездов и состоянии инфраструктуры. PostgreSQL при этом остается источником долговременных записей, журналов событий, нормативно-справочной информации, планов-графиков движения и отчетов о техническом обслуживании.

Та же методика нагрузочного тестирования подходит для предварительной проверки аппаратно-программных решений на транспортных полигонах.

Здесь важны не только средняя скорость, но и хвостовые задержки, поскольку несколько медленных запросов способны затормозить критичную операцию. Это актуально, например, для систем предиктивной диагностики рельсов и подвижного состава [16] и для цифровых двойников железнодорожной инфраструктуры [17]. Выбор схемы кэширования и распределения нагрузки в таких системах влияет на время, за которое диспетчер получает нужные данные. Поэтому результаты исследования можно использовать при подготовке регламентов испытаний производительности для центров управления перевозками и решений класса «умная железная дорога».

СПИСОК ИСТОЧНИКОВ

1. Performance Comparison of Redis, Memcached, MySQL, and PostgreSQL: A Study on Key-Value and Relational Databases / D. Almeida [et al.] // Proceedings of the 2023 Second International Conference on Smart Technologies for Smart Nation (SmartTechCon) (Singapore, Singapore, 18–19 August 2023). Institute of Electrical and Electronics Engineers, 2023. Pp. 902–907. DOI: 10.1109/SmartTechCon57526.2023.10391649

2. Хомоненко А. Д., Цыганков В. М., Мальцев М. Г. Базы данных: учебник для вузов / под ред. А. Д. Хомоненко. 6-е изд., доп. СПб.: Корона-Век, 2009. 736 с.

3. Коннолли Т., Бегг К. Базы данных: проектирование, реализация и сопровождение: теория и практика. 3-е издание = Database Systems: A Practical Approach to Design, Implementation, and Management. Third Edition / пер. с англ. Р. Г. Имамутдиновой и К. А. Птицына; под ред. К. А. Птицына. М.: Вильямс, 2017. 1440 с.

4. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. СПб.: Питер, 2018. 640 с.

5. Gembalczyk D., Schuhknecht F. M., Dittrich J. An Experimental Analysis of Different Key-Value Stores and Relational Databases // Datenbanksysteme für Business, Technologie und Web (BTW 2017): Proceedings 17 Fachtagung des GI-Fachbereichs “Datenbanken und Informationssysteme” (DBIS) (Stuttgart, Germany, 06–10 March 2017) / B. Mitschang [et al.] (eds). Lecture Notes in Informatics. Vol. 265. Bonn: Gesellschaft für Informatik, 2017. Pp. 351–360.

6. Salunke S. V., Ouda A. A Performance Benchmark for the PostgreSQL and MySQL Databases // Future Internet. 2024. Vol. 16, iss. 10. Art. no. 382. 22 p. DOI: 10.3390/fi16100382

7. Dean J., Barroso L. A. The Tail at Scale // Communications of the ACM. 2013. Vol. 56, no. 2. Pp. 74–80. DOI: 10.1145/2408776.2408794

8. Sikh A., Axén J. High-Performing Cloud Native SW Using Key-Value Storage or Database for Externalized States. Bachelor’s Thesis. Department of Computer and Information Science, Linköping University. Linköping, 2023. 18 p. URL: http://liu.diva-portal.org/smash/get/diva2:1768552/FULLTEXT01.pdf (дата обращения: 03.08.2026).

9. Optimizing Asynchronous Performance in Node.js with Express and PostgreSQL / M. D. Imam Sakti [et al.] // Procedia Computer Science. 2025. Vol. 269. Pp. 172–181. DOI: 10.1016/j.procs.2025.08.270

10. High-Throughput Real-Time Leaderboard a Comparative Performance Analysis of PostgreSQL and Redis / S. Khanum [et al.] // Proceedings of International Conference on Computational Technologies for Research in Data Analytics (ICCTRDA 2025) (Da Nang, Vietnam, 21–22 November 2025). Volume 3 / N. G. Nhu [et al.] (eds). Lecture Notes in Networks and Systems. Vol. 1988. Cham: Springer, 2026. Pp. 114–126. DOI: 10.1007/978-3-032-26898-3_11

11. Adaptive and Scalable Database Management with Machine Learning Integration: A PostgreSQL Case Study / M. Abbasi [et al.] // Information. 2024. Vol. 15, iss. 9. Art. no. 574. 25 p. DOI: 10.3390/info15090574

12. Малыгин Д. С. Проблемы производительности реляционных баз данных в распределенных архитектурах и стратегии их решения // Современные наукоемкие технологии. 2024. № 10. С. 61–71. DOI: 10.17513/snt.40173

13. Privalov M. V., Stupina M. V. Improving Web-Oriented Information Systems Efficiency Using Redis Caching Mechanisms // Indonesian Journal of Electrical Engineering and Computer Science. 2024. Vol. 33, no. 3. Pp. 1667–1675. DOI: 10.11591/ijeecs.v33.i3.pp1667-1675

14. Черкасов С. Д., Хомоненко А. Д. Исследование влияния фрагментации данных на производительность чтения/записи и методы ее устранения в MySQL (InnoDB) // Интеллектуальные технологии на транспорте. 2026. № 2 (46). С. 64–74. DOI: 10.20295/2413-2527-2026-246-64-74

15. Zulfa M. I., Fadli A., Wardhana A. W. Application Caching Strategy Based on In-Memory Using Redis Server to Accelerate Relational Data Access // Jurnal Teknologi dan Sistem Komputer. 2020. Vol. 8, no. 2. Pp. 157–163. DOI: 10.14710/jtsiskom.8.2.2020.157-163

16. Implementation of an AI-Based Predictive Structural Health Monitoring Strategy for Bonded Insulated Rail Joints Using Digital Twins Under Varied Bolt Conditions / G. Bianchi [et al.] // Railway Engineering Science. 2025. Vol. 33. No. 4. Pp. 703–720. DOI: 10.1007/s40534-024-00371-3

17. Ariyachandra M. R. M. F., Wen Y., Yu J. Advancing Rail Infrastructure: Integrating Digital Twins and Cyber-Physical Systems for Predictive Maintenance // Proceedings of the 2025 European Conference on Computing in Construction and 42nd International CIB W78 Conference on Information Technology in Construction (Porto, Portugal, 14–17 July 2025) / Petrova E. [et al.] (eds). European Council on Computing in Construction, 2025. Pp. 1266–1273. DOI: 10.35490/EC3.2025.272

Дата поступления: 10.08.2026
Решение о публикации: 27.08.2026

Investigation of Key-Value Model Performance in PostgreSQL and Redis: A Comparative Analysis of Read/Write Operations and Parallel Load Impact

Ratmir A. Voloshin — 3rd-year Bachelor's Degree Student in 38.03.05 Business Informatics. Research interests: information systems, development and research of characteristics of applications for working with databases. E-mail: vratmir@yandex.ru

Anatoly D. Khomonenko — Dr. Sci. in Engineering, Full Professor. Professor of the "Information and Computing Systems" Department. Research interests: information systems, big data processing, probabilistic modeling of geographic information systems, genetic algorithms, information security. E-mail: khomon@mail.ru

Stepan S. Yudnikov — M. S., Postgraduate Student of the "Information and Computing Systems" Department. Research interests: information systems, development and performance analysis of database applications. E-mail: s.yudnikovv@gmail.com

Emperor Alexander I St. Petersburg State Transport University, 9 Moskovsky ave., Saint Petersburg, 190031, Russia

For citation: Voloshin R. A., Khomonenko A. D., Yudnikov S. S. Investigation of Key-Value Model Performance in PostgreSQL and Redis: A Comparative Analysis of Read/Write Operations and Parallel Load Impact, *Intellectual Technologies on Transport*, 2026, no. 3 (47), pp. 44–55. DOI: 10.20295/2413-2527-2026-347-44-55 (In Russian)

Abstract. *An experimentally substantiated approach to comparing the performance of the key-value model implementation in PostgreSQL and Redis is proposed.* ***Purpose:*** *to evaluate the latency and throughput of read and write operations under varying degrees of parallel load.* ***Methods:*** *analysis of PostgreSQL architecture (MVCC, WAL, B-tree and GIN indexes, jsonb and hstore types) and Redis architecture (in-memory storage, strings and hashes, TTL, pipelining); creation of an experimental testbed; generation of a test table with 1,000,000 records; load testing using pgbench and redis-benchmark with parallelism from 1 to 128 clients;*

measurement of throughput, average and tail latencies (p95, p99). ***Results:*** *Redis provides 5–10 times lower read latency and 3–5 times higher throughput compared to PostgreSQL for point GET/SET operations; the use of pipelining additionally increases throughput by 2–3 times; PostgreSQL retains advantages in versatility, supporting transactions, complex queries, and long-term storage.* ***Practical significance:*** *the developed testing methodology and obtained quantitative estimates are applicable when choosing a storage architecture for high-load information systems, including monitoring and tracking systems in transport.*

Keywords: *PostgreSQL, Redis, key-value model, latency, throughput, high-load transactions, jsonb, hstore, in-memory storage*

REFERENCES

1. Almeida D., et al. Performance Comparison of Redis, Memcached, MySQL, and PostgreSQL: A Study on Key-Value and Relational Databases, *Proceedings of the 2023 Second International Conference on Smart Technologies for Smart Nation (SmartTechCon)*, Singapore, Singapore, August 18–19, 2023. Institute of Electrical and Electronics Engineers, 2023, pp. 902–907. DOI: 10.1109/SmartTechCon57526.2023.10391649

2. Khomonenko A. D., Tsygankov V. M., Maltsev M. G. Bazy dannykh: uchebnik dlya vuzov [Databases: A Textbook for Universities]. Saint Petersburg, Korona-Vek Publishing House, 2009, 736 p. (In Russian)

3. Connolly T. M., Begg C. E. Bazy dannykh: proektirovanie, realizatsiya i soprovozhdenie: teoriya i praktika. Tretye izdanie [Database Systems: A Practical Approach to Design, Implementation, and Management. Third Edition]. Moscow, Williams Publishing House, 2017, 1440 p. (In Russian)

4. Kleppmann M. Vysokonagruzhennye prilozheniya. Programmirovanie, masshtabirovanie, podderzhka [Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems]. Saint Petersburg, Piter Publishing House, 2018, 640 p. (In Russian)

5. Gembalczyk D., Schuhknecht F. M., Dittrich J. An Experimental Analysis of Different Key-Value Stores and Relational Databases. In: *Mitschang B., et al. (eds) Datenbanksysteme für Business, Technologie und Web (BTW 2017): Proceedings 17 Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS)*, Stuttgart, Germany, March 06–10, 2017. Lecture Notes in Informatics, vol. 265. Bonn, Gesellschaft für Informatik, 2017, pp. 351–360.

6. Salunke S. V., Ouda A. A Performance Benchmark for the PostgreSQL and MySQL Databases, *Future Internet*, 2024, vol. 16, iss. 10, art. no. 382, 22 p. DOI: 10.3390/fi16100382

7. Dean J., Barroso L. A. The Tail at Scale, *Communications of the ACM*, 2013, vol. 56, no. 2, pp. 74–80. DOI: 10.1145/2408776.2408794

8. Sikh A., Axén J. High-Performing Cloud Native SW Using Key-Value Storage or Database for Externalized States. Bachelor's Thesis. Department of Computer and Information Science, Linköping University. Linköping, 2023, 18 p. Available at: http://liu.diva-portal.org/smash/get/diva2:1768552/ FULLTEXT01.pdf (accessed: August 03, 2026).

9. Imam Sakti M. D., et al. Optimizing Asynchronous Performance in Node.js with Express and PostgreSQL, *Procedia Computer Science*, 2025, vol. 269, pp. 172–181. DOI: 10.1016/j.procs.2025.08.270

10. Khanum S., et al. High-Throughput Real-Time Leaderboard a Comparative Performance Analysis of PostgreSQL and Redis. In: *Nhu N. G., et al. (eds) Proceedings of International Conference on Computational Technologies for Research in Data Analytics (ICCTRDA 2025)*, Da Nang, Vietnam, 21–22 November 2025. Volume 3. Lecture Notes in Networks and Systems, vol. 1988. Cham, Springer, 2026, pp. 114–126. DOI: 10.1007/978-3-032-26898-3_11

11. Abbasi M., et al. Adaptive and Scalable Database Management with Machine Learning Integration: A PostgreSQL Case Study, *Information*, 2024, vol. 15, iss. 9, art. no. 574, 25 p. DOI: 10.3390/info15090574

12. Malygin D. S. Problemy proizvoditelnosti relyatsionnykh baz dannykh v raspredelennykh arkhitekturakh i strategii ikh resheniya [Performance Issues of Relational Databases in Distributed Architectures and Strategies for Resolution], *Sovremennye naukoemkie tekhnologii [Modern High Technologies]*, 2024, no. 10, pp. 61–71. DOI: 10.17513/snt.40173 (In Russian)

13. Privalov M. V., Stupina M. V. Improving Web-Oriented Information Systems Efficiency Using Redis Caching Mechanisms, *Indonesian Journal of Electrical Engineering and Computer Science*, 2024, vol. 33, no. 3, pp. 1667–1675. DOI: 10.11591/ijeecs.v33.i3.pp1667-1675

14. Cherkasov S. D., Khomonenko A. D. Issledovanie vliyaniya fragmentatsii dannykh na proizvoditelnost chteniya/zapisi i metody ee ustraneniya v MySQL (InnoDB) [Investigation of Data Fragmentation Impact on Read/Write Performance and Methods for Its Elimination in MySQL (InnoDB)], *Intellektualnye tekhnologii na transporte [Intellectual Technologies on Transport]*, 2026, no. 2 (46), pp. 64–74. DOI: 10.20295/2413-2527-2026-246-64-74 (In Russian)

15. Zulfa M. I., Fadli A., Wardhana A. W. Application Caching Strategy Based on In-Memory Using Redis Server to Accelerate Relational Data Access, *Jurnal Teknologi dan Sistem Komputer*, 2020, vol. 8, no. 2, pp. 157–163. DOI: 10.14710/jtsiskom.8.2.2020.157-163

16. Bianchi G., et al. Implementation of an AI-Based Predictive Structural Health Monitoring Strategy for Bonded Insulated Rail Joints Using Digital Twins Under Varied Bolt Conditions, *Railway Engineering Science*, 2025, vol. 33, no. 4, pp. 703–720. DOI: 10.1007/s40534-024-00371-3

17. Ariyachandra M. R. M. F., Wen Y., Yu J. Advancing Rail Infrastructure: Integrating Digital Twins and Cyber-Physical Systems for Predictive Maintenance. In: *Petrova E., et al. (eds) Proceedings of the 2025 European Conference on Computing in Construction and 42nd International CIB W78 Conference on Information Technology in Construction*, Porto, Portugal, July 14–17, 2025. European Council on Computing in Construction, 2025, pp. 1266–1273. DOI: 10.35490/EC3.2025.272

Received: August 10, 2026
Accepted: August 27, 2026