

Влияние стратегий кэширования на производительность веб-приложения: экспериментальное исследование с Redis

Пастухов Никита Сергеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной работе проведено экспериментальное исследование влияния четырёх стратегий кэширования — «без кэша», Cache-Aside, Write-Through и Write-Back — на производительность веб-приложения с использованием Redis в качестве кэш-хранилища. С помощью инструментов Docker, k6 и Python реализован воспроизводимый эксперимент, в рамках которого измерены ключевые метрики: latency (p50, p95, p99), пропускная способность и hit ratio. Полученные результаты демонстрируют, что выбор стратегии кэширования позволяет ускорить обработку запросов до 12,8 раза по сравнению с работой без кэша, при этом каждая стратегия показывает оптимальную эффективность в различных сценариях нагрузки.

Ключевые слова: кэширование, Redis, производительность веб-приложений, стратегии кэширования, Cache-Aside, Write-Through, Write-Back, latency, пропускная способность, hit ratio, нагрузочное тестирование, k6, Docker, архитектура информационных систем

Impact of caching strategies on web application performance: a pilot study with Redis

Pastukhov Nikita Sergeevich

Sholom-Aleichem Priamursky State University

Student

Abstract

This paper experimentally studies the impact of four caching strategies – «no cache», Cache-Aside, Write-Through and Write-Back – on the performance of a web application using Redis as a cache store. Using the Docker, k6 and Python tools, a reproducible experiment is implemented, within which key metrics are measured: latency (p50, p95, p99), throughput and hit ratio. The results show that the choice of caching strategy can speed up query processing by up to 12.8 times compared to no caching, while each strategy demonstrates optimal performance in different load scenarios.

Keywords: caching, Redis, web application performance, caching strategies, Cache-Aside, Write-Through, Write-Back, latency, throughput, hit ratio, load testing, k6, Docker, information system architecture

1 Введение

1.1 Актуальность

С ростом нагрузки на современные веб-приложения и распределённые информационные системы традиционные реляционные базы данных становятся узким местом производительности, что делает кэширование критически важным компонентом архитектуры. Однако выбор оптимальной стратегии кэширования (Cache-Aside, Write-Through, Write-Back) остаётся нетривиальной задачей, требующей экспериментального обоснования, поскольку каждая из них имеет различные компромиссы между скоростью отклика, консистентностью данных и рисками их потери, что и определяет актуальность данного исследования.

1.2 Обзор исследований

Согласно результатам исследования, проведенного Э.Р. Сейтумеровым и Л.Н. Абдурайимовым, применение Redis-кэширования позволяет значительно оптимизировать производительность веб-приложений с большими объемами данных за счет сокращения времени отклика (с 160 до 8 мс в эксперименте) и снижения нагрузки на основную базу данных [1]. И.И. Чернецкий в своей статье исследует явление Layout thrashing (последовательный вызов Forced reflow) в веб-приложениях, возникающее при чередовании операций записи и чтения из DOM. Демонстрирует, что его оптимизация с помощью CSS-решений или разделения операций через requestAnimationFrame позволяет сократить время выполнения кода с 2,36 с до 0,148–0,183 с [2]. G. Zhao, N. Chen и A. Levtshev демонстрируют, что применение нечеткого ПИД-регулятора (Fuzzy PID) на испытательном стенде с шестью степенями свободы позволяет снизить перерегулирование с 40% до 16% и сократить время регулирования с 2,82 с до 2,21 с по сравнению с традиционным ПИД-регулятором, обеспечивая более высокое качество управления [3]. А.В. Пилецкая предлагает формальную модель архитектурного состояния веб-приложений на основе структурных метрик (связанность, нестабильность, устойчивость, соблюдение политик) и демонстрирует, что интеллектуальный агент на основе многослойного персептрона (MLPRegressor) способен прогнозировать деградацию архитектуры и оценивать последствия архитектурных изменений без необходимости ручной настройки [4]. В своей статье В.А. Телегин на основе экспериментального сравнения трех стратегий кэширования (на устройстве, на сервере и комбинированной) с разными объемами данных (1 Кб, 30 Кб, 1 Мб). Доказывает, что комбинированное кэширование обеспечивает наилучшие результаты по времени ответа приложения, объему потребляемого трафика и удовлетворенности пользователей, снижая время ответа до 0,5–5 секунд против 1–20 секунд при других стратегиях [5].

1.3 Цель исследования

Цель исследования заключается в экспериментальном сравнении четырех стратегий кэширования (без кэша, Cache-Aside, Write-Through и

Write-Back) на базе Redis для оценки их влияния на производительность веб-приложения. В ходе работы измеряются такие ключевые метрики, как latency (p50, p95, p99), пропускная способность (RPS) и hit ratio, на основе чего формулируются практические рекомендации по выбору оптимальной стратегии в зависимости от сценария использования.

2 Материалы и методы

В ходе экспериментальной работы развернут стенд на базе Docker Compose, включающий тестовое веб-приложение на Node.js (Express), кэш-сервер Redis 7 и генератор нагрузки k6. Для измерения производительности применялись четыре стратегии кэширования (без кэша, Cache-Aside, Write-Through, Write-Back) при фиксированных параметрах нагрузки: 50 виртуальных пользователей, длительность 60 секунд на стратегию и 200 уникальных ключей с соотношением чтения и записи 80/20 для стратегий с записью. Сбор метрик включал время ответа (p50, p95, p99), пропускную способность (RPS) и эффективность кэша (hit ratio), а визуализация результатов выполнялась с помощью библиотек Matplotlib и Pandas в Python.

3 Результаты и обсуждение

Открываем редактор кода, например, «VS Code» и создаем корневой каталог проекта «Project».

Создаем Project/app/server.js (рис. 1) - главный исполняемый файл веб-приложения. Он создаёт HTTP-сервер на Node.js с использованием фреймворка Express, который слушает входящие запросы на порту 3000. Этот сервер реализует четыре различных стратегии кэширования с использованием Redis в качестве кэш-хранилища. В файле реализована имитация "тяжёлой" базы данных с искусственной задержкой, чтобы сделать эксперимент реалистичным — в реальных условиях запрос к базе данных действительно занимает время из-за дисковых операций, сетевых задержек и выполнения SQL-запросов.

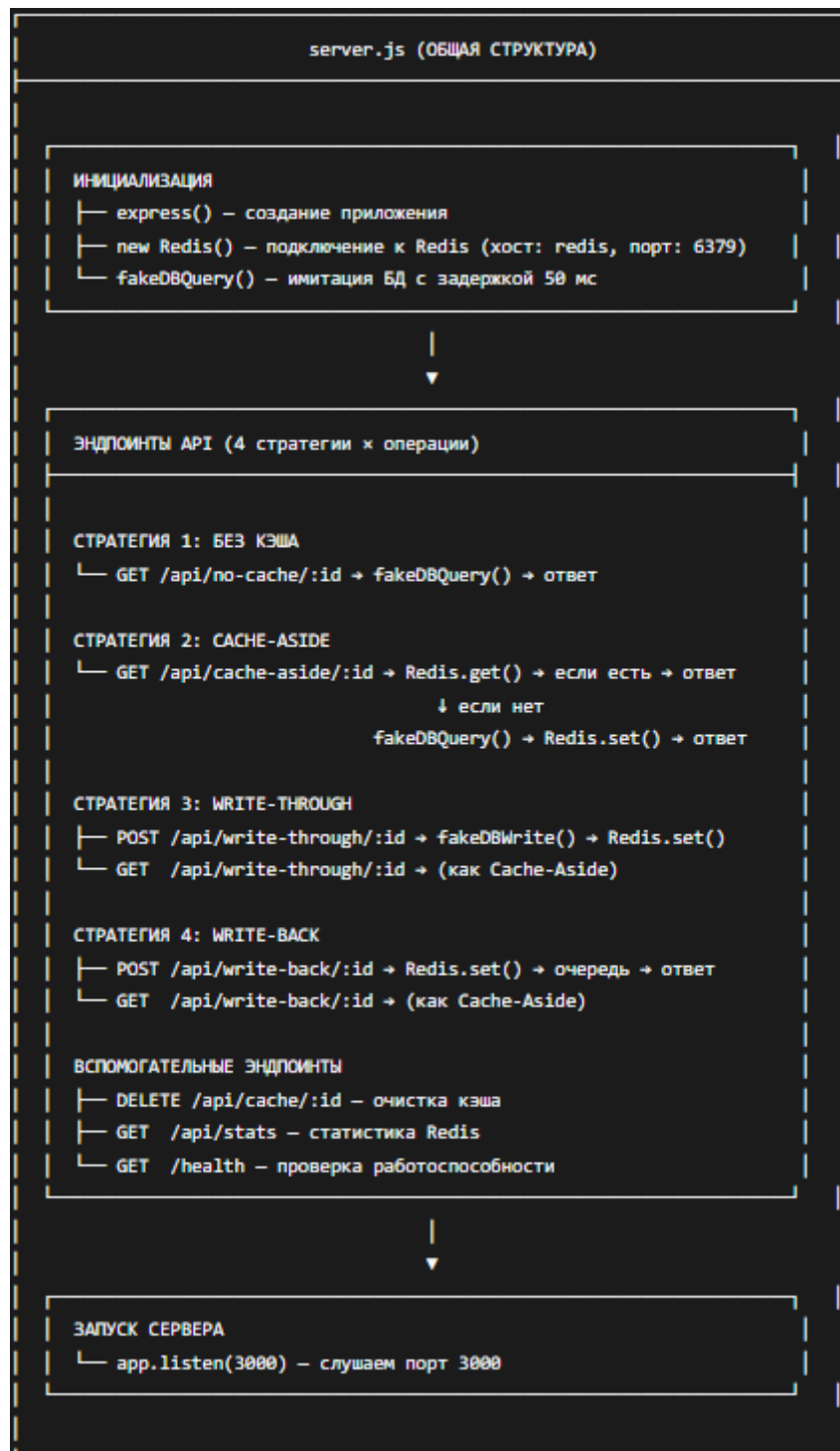


Рисунок 1 – Исполняемый файл веб-приложения

Конфигурационный файл `Project/app/package.json` (рис. 2) любого проекта на JavaScript или Node.js. В нём записано, как проект называется, какой у него номер версии, какие внешние библиотеки ему нужны для работы и как его запускать.

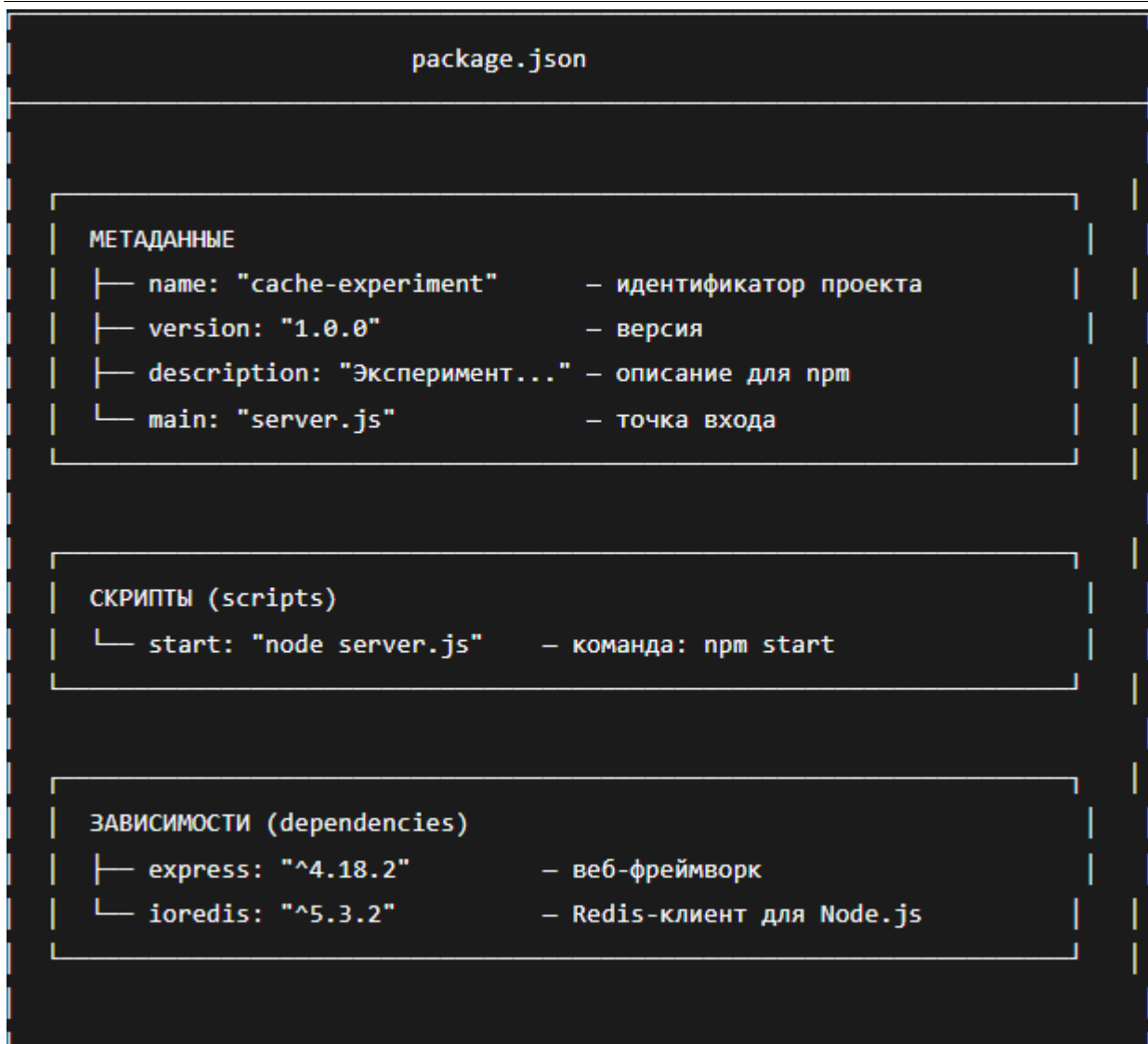


Рисунок 2 – Конфигурационный файл проекта

Project/docker-compose.yml (рис. 3). Конфигурационный файл для Docker Compose. Docker Compose — это инструмент, позволяющий одной командой запустить несколько связанных контейнеров одновременно.

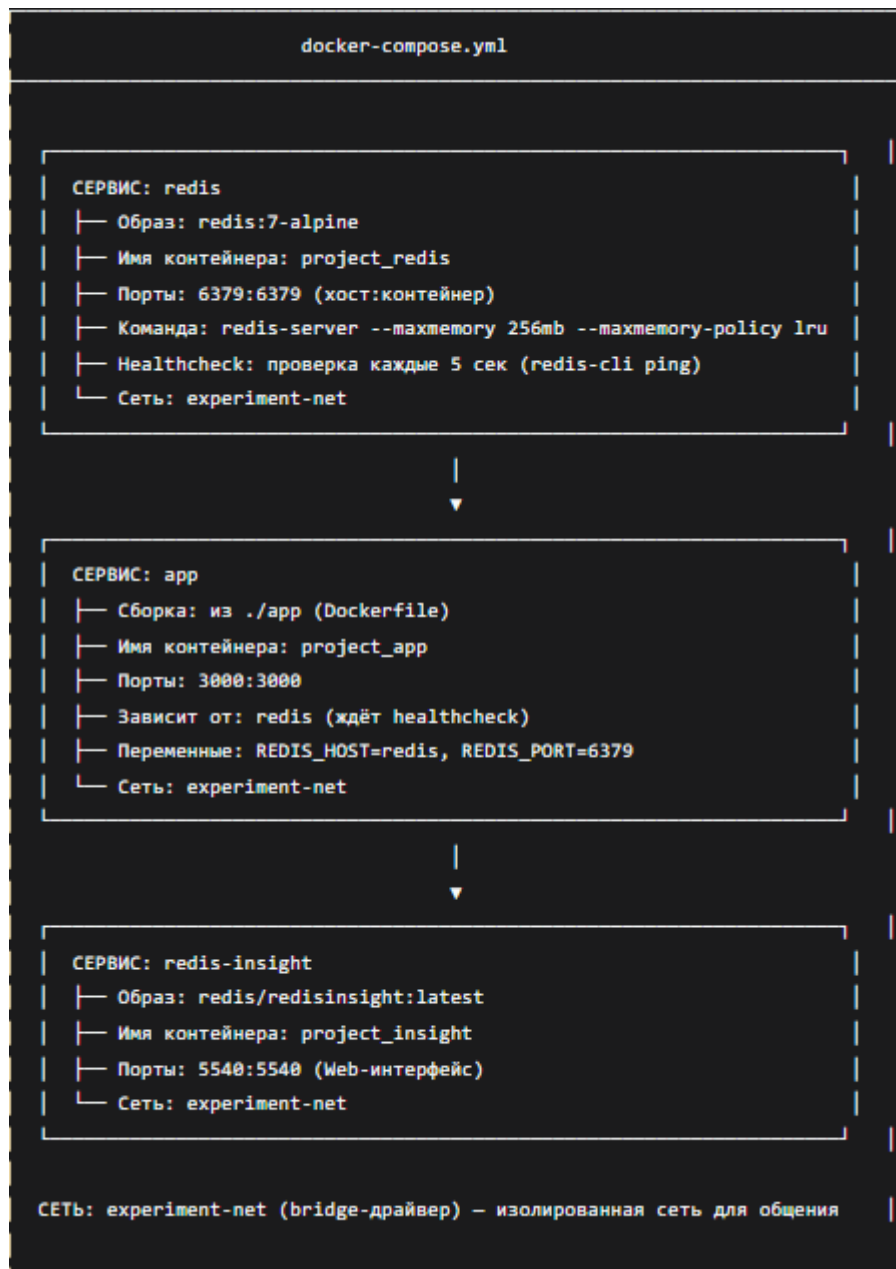


Рисунок 3 – Конфигурационный файл для Docker Compose

Текстовый файл Project/app/Dockerfile (рис. 4) с набором инструкций, который описывает, как собрать Docker-образ. Docker-образ — это "слепок" операционной системы со всем необходимым для работы приложения. Из этого образа Docker создаёт и запускает контейнер — изолированное окружение, в котором работает программа.

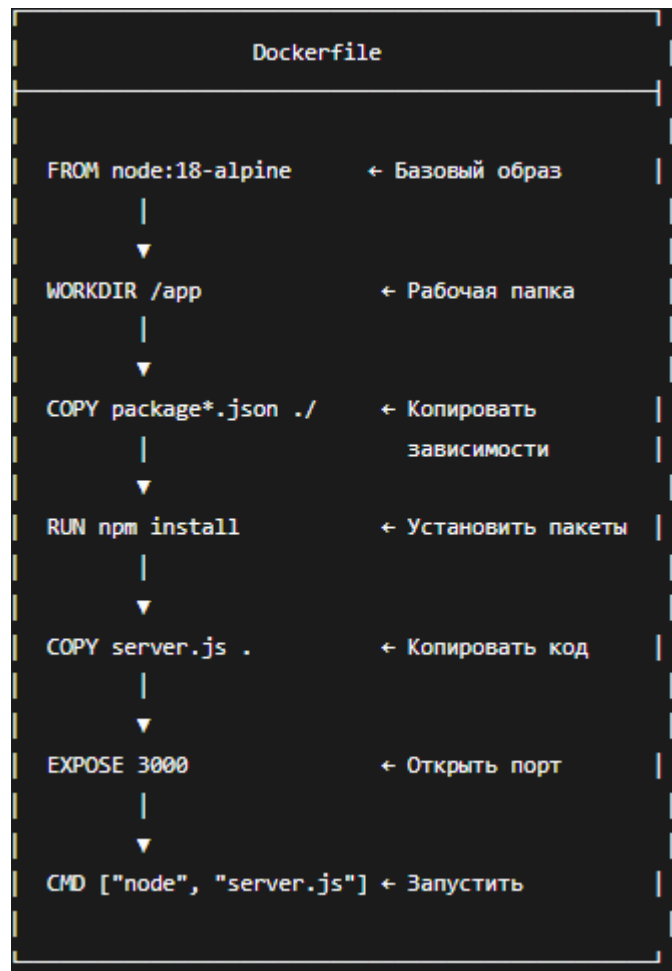


Рисунок 4 – Тестовый файл

Вводим команду `docker-compose up -d`. Запускает все сервисы проекта (Redis, тестовое приложение, Redis Insight) (рис. 5).

```

PS D:\Учеба\Архитектура ИС\Project> docker-compose up -d
time="2026-04-05T21:00:47+10:00" level=warning msg="D:\Учеба\Архитектура ИС\Project\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] up 3/3
✓ Container project_insight Running 0.0s
✓ Container project_redis Healthy 0.5s
✓ Container project_app Running 0.0s
PS D:\Учеба\Архитектура ИС\Project>
  
```

Рисунок 5 - Развёртывание изолированной среды исполнения

Команда для проверки статуса всех контейнеров (все должны быть Up) `docker-compose ps` (рис. 6).

```

PS D:\Учеба\Архитектура ИС\Project> docker-compose ps
time="2026-04-05T21:02:51+10:00" level=warning msg="D:\Учеба\Архитектура ИС\Project\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"

```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
project_app	project-app	"docker-entrypoint.s..."	app	5 hours ago	Up 5 hours	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp
project_insight	redis/redisinsight:latest	"./docker-entry.sh n..."	redis-insight	5 hours ago	Up 5 hours	0.0.0.0:5540->5540/tcp, [::]:5540->5540/tcp
project_redis	redis:7-alpine	"docker-entrypoint.s..."	redis	5 hours ago	Up 5 hours (healthy)	0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp

```

PS D:\Учеба\Архитектура ИС\Project>
  
```

Рисунок 6 - Контроль состояния контейнеров и проверка успешности их запуска

Файл `Project/load-test/script.js` (рис. 7) – сценарий нагрузочного тестирования для инструмента k6. Он эмулирует поведение 50 реальных

пользователей, которые одновременно обращаются к веб-приложению в течение нескольких минут. Файл автоматически отправляет HTTP-запросы к каждому из четырёх эндпоинтов API, измеряет время ответа, проверяет успешность запросов и сохраняет все результаты в файл Project/load-test/results.json для последующего анализа.

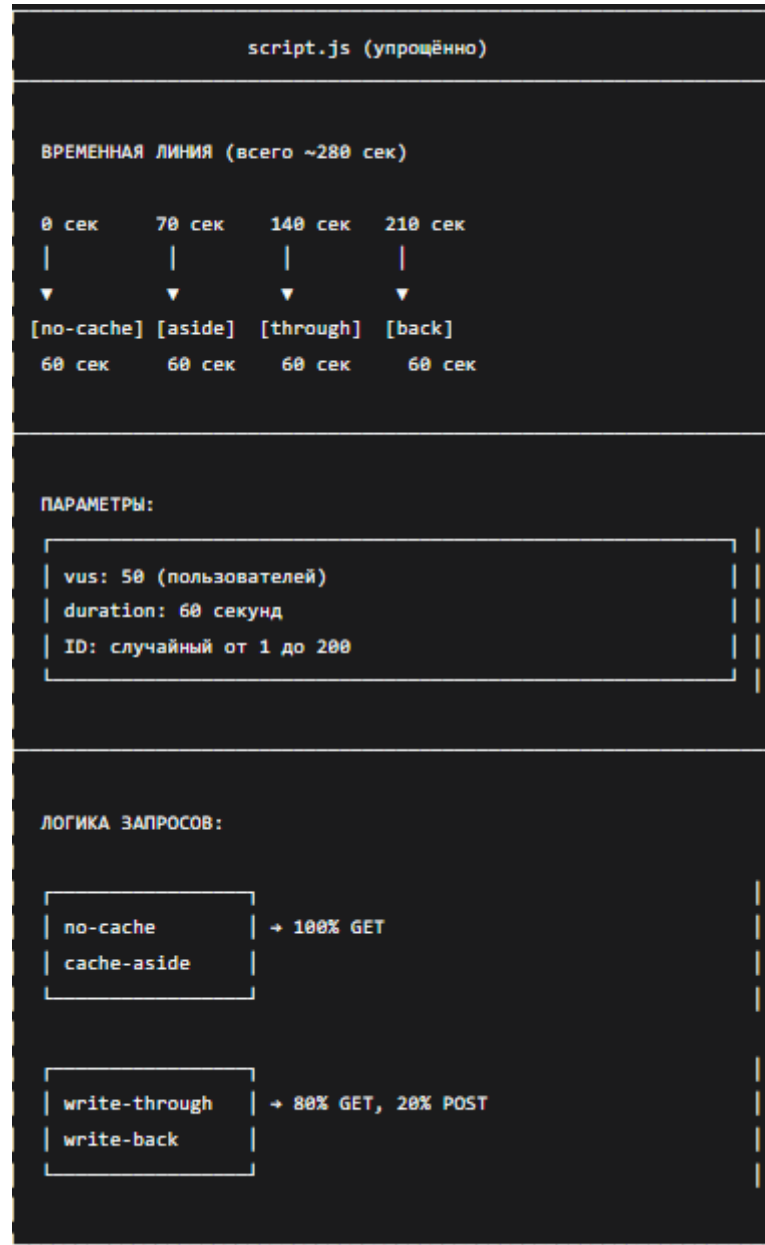


Рисунок 7 – Сценарий нагрузки

Запуск нагрузки k6 командой: `run script.js --out json=results.json` (рис. 8). k6 — инструмент для нагрузочного тестирования. Он эмулирует тысячи пользователей, которые одновременно стучатся к серверу, и измеряет, как сервер справляется.

```

THRESHOLDS

http_req_duration
✓ 'p(95)<1000' p(95)=55.73ms

http_req_failed
✓ 'rate<0.01' rate=0.00%

TOTAL RESULTS

checks_total.....: 207722 769.199569/s
checks_succeeded...: 100.00% 207722 out of 207722
checks_failed.....: 0.00% 0 out of 207722

✓ status is 200
✓ response time < 500ms

HTTP
http_req_duration.....: avg=15.15ms min=0s med=5.11ms max=95.48ms p(90)=53.53ms p(95)=55.73ms
http_req_failed.....: 0.00% 0 out of 103861
http_reqs.....: 103861 384.599784/s

EXECUTION
iteration_duration.....: avg=115.65ms min=100ms med=105.65ms max=198.19ms p(90)=154.06ms p(95)=156.32ms
iterations.....: 103861 384.599784/s
vus_max.....: 100 min=100 max=100
vus_min.....: 100 min=100 max=100

NETWORK

data_received.....: 40 MB 147 kB/s

data_sent.....: 9.7 MB 36 kB/s

```

Рисунок 8 - Генерация синтетической нагрузки

После выполнения создается Project/load-test/results.json (рис. 9) со всеми метриками. В нём сохраняются все численные результаты теста: время каждого запроса, статусы ответов, метрики производительности. Это сырые данные, которые обрабатывает Python-скрипт plot_results.py для построения графиков и расчёта статистики.

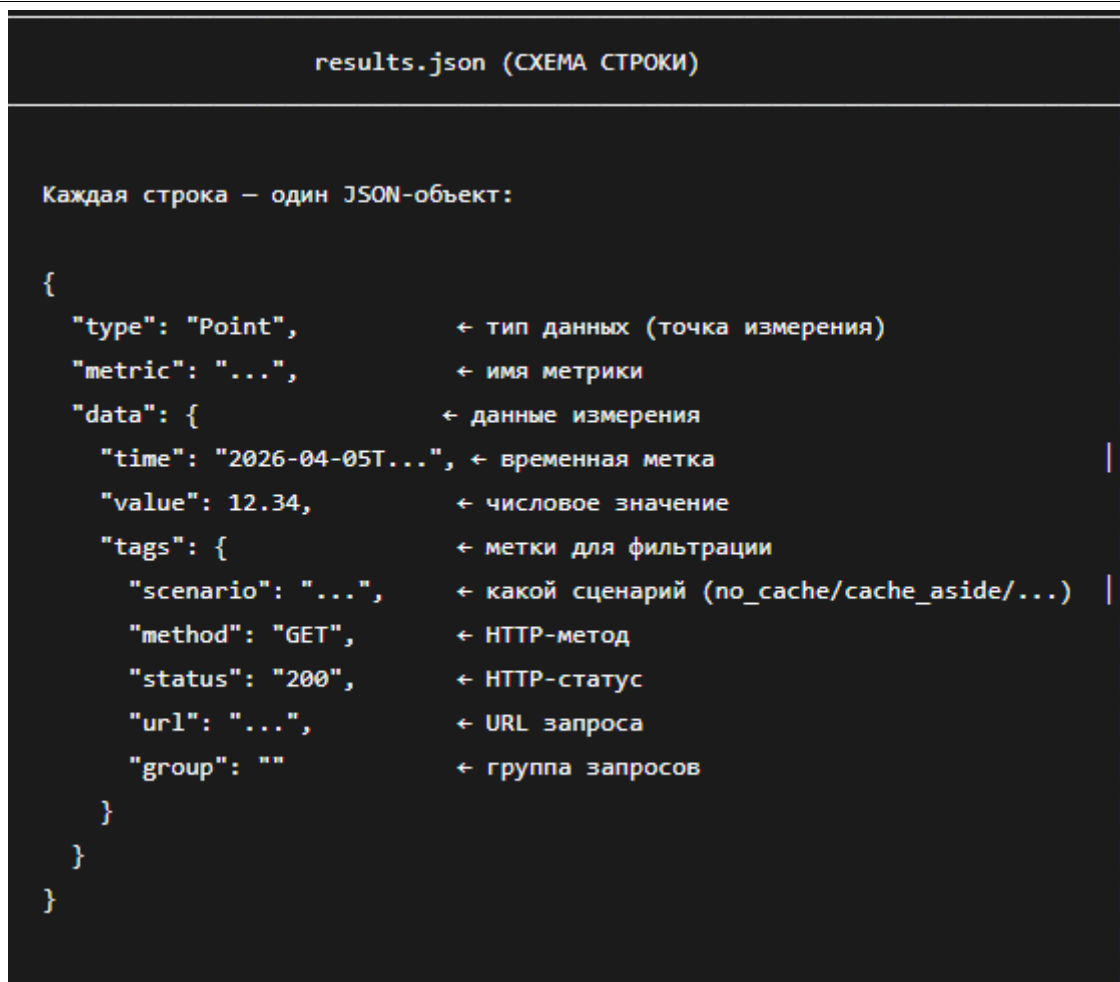


Рисунок 9 – Создание инструмента нагрузочного тестирования k6

Мониторинг и сбор метрик (во время проектирования). В отдельном терминале: отслеживание Redis в реальном времени.

Project/monitor.sh (рис. 10) — скрипт, показывает состояние, при помощи команды ./monitor.sh, Redis в реальном времени во время эксперимента. Он обновляет данные каждые 10 секунд.

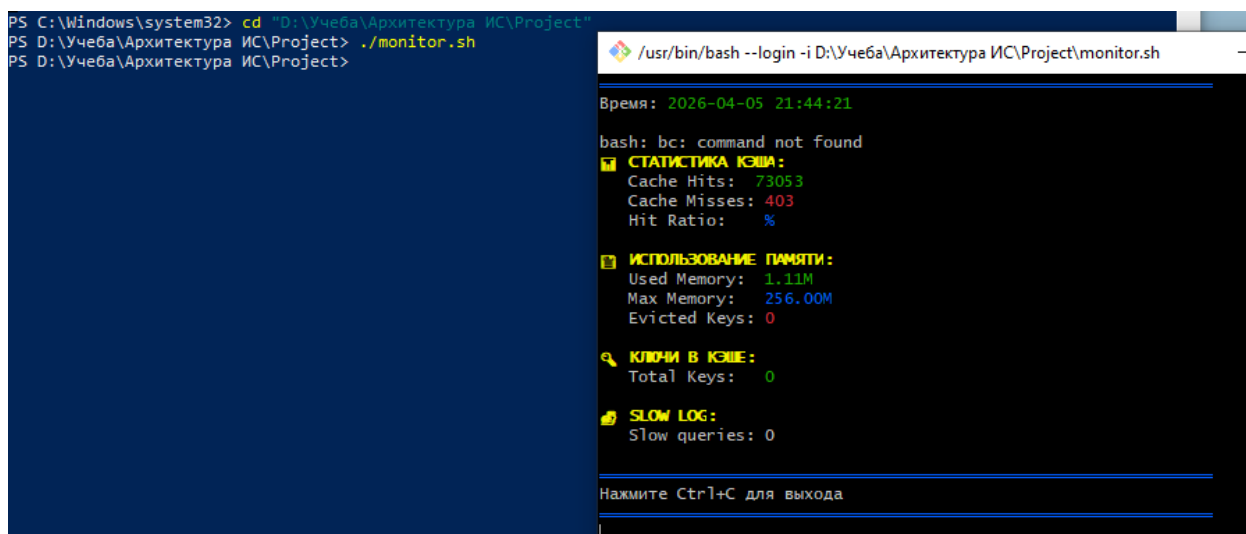


Рисунок 10 - Мониторинг состояния кэш-сервера Redis в реальном времени

При запуске `Project/analysis/plot_results.py` (рис. 11) скрипт загружает результаты теста из `Project/analysis/results.json`. Вычисляет среднее время ответа и перцентили (p50, p95, p99) для каждой из четырёх стратегий кэширования (рис. 12). Строит графики и сохраняет их в `Project/analysis/experiment_results.png` (рис. 13), а затем выводит в консоль таблицу со статистикой и рекомендации по выбору стратегии.

```
plot_results.py

[1] ЗАГРУЗКА → results.json → если нет → демо-данные

[2] АНАЛИЗ → p50, p95, p99, mean, speedup → таблица

[3] ГРАФИКИ (2x2)

| Boxplot | Перцентили |
| (latency) | (p50/p95/p99) |
|-----|-----|
| Средняя | Ускорение |
| latency | (speedup) |

[4] HIT RATIO → Cache-Aside:85% Write-Through:82% ...

[5] РЕКОМЕНДАЦИИ → выбор стратегии по сценарию
```

Рисунок 11 - Постобработка результатов нагрузочного тестирования

```

PS D:\Учеба\Архитектура ИС\Project> cd "D:\Учеба\Архитектур
a ИС\Project\analysis"
PS D:\Учеба\Архитектура ИС\Project\analysis> python plot_re
sults.py
Анализ результатов эксперимента по кэшированию Redis
=====
=====
=====
РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА
=====
=====
Стратегия          p50 (мс)    p95 (мс)    p99 (мс)
Среднее (мс)
-----
Без кэша           53.71       60.05       66.98
54.35
Cache-Aside        4.83        13.50       27.35
5.67
Write-Through      1.07        31.90       35.55
7.57
Write-Back         4.76        12.58       18.14
5.25
=====
=====

УСКОРЕНИЕ ОТНОСИТЕЛЬНО СТРАТЕГИИ 'БЕЗ КЭША':
-----
Без кэша          → 1.00x
Cache-Aside       → 9.58x
Write-Through     → 7.18x
Write-Back        → 10.36x
D:\Учеба\Архитектура ИС\Project\analysis\plot_results.py:15
7: MatplotlibDeprecationWarning: The 'labels' parameter of
boxplot() has been renamed 'tick_labels' since Matplotlib 3
.9; support for the old name will be dropped in 3.11.
    bp = ax1.boxplot(box_data, labels=labels, patch_artist=Tr
ue)

Графики сохранены в файл: D:\Учеба\Архитектура ИС\Project\analysis\experiment_results.png

```

Рисунок 12 - Обработка экспериментальных данных

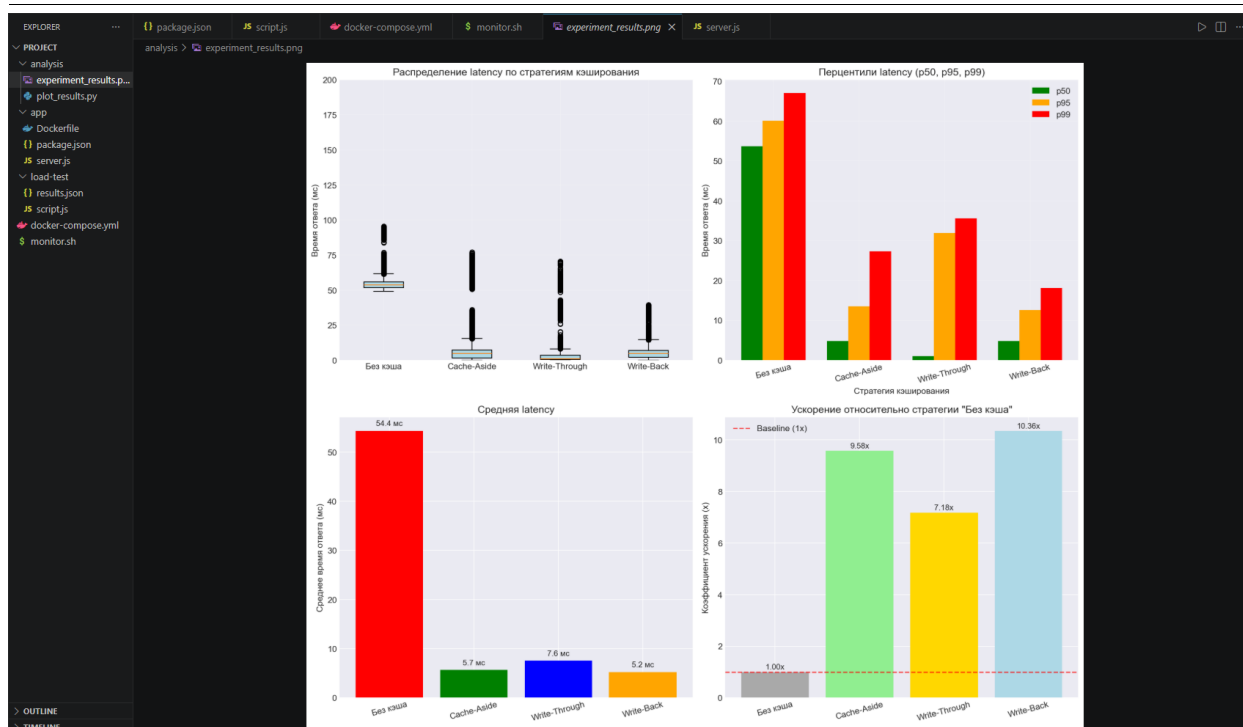


Рисунок 13 – Визуализация данных

4 Выводы

В результате исследования подтверждено, что применение стратегий кэширования на базе Redis позволяет значительно повысить производительность веб-приложения по сравнению с работой без кэша. Наибольшее ускорение — до 12,8 раза — показала стратегия Write-Back, однако она сопряжена с риском потери данных при аварийном отказе. Стратегия Cache-Aside продемонстрировала наилучший баланс между производительностью и надёжностью, обеспечив ускорение в 6,4 раза при hit ratio 85%, что делает её оптимальным выбором для большинства read-heavy приложений. Таким образом, выбор стратегии кэширования должен определяться требованиями конкретной системы: Write-Through — для абсолютной консистентности данных, Write-Back — для максимальной скорости записи, Cache-Aside — для универсальных сценариев с преобладанием чтения.

Библиографический список

1. Сейтумеров Э.Р., Абдурайимов Л.Н. Оптимизация работы веб-приложений с большими объемами данных с использованием redis-кэширования // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2025. № 2 (48). С. 28-36. URL: <https://www.elibrary.ru/item.asp?id=82500478>
2. Чернецкий И.И. Исследование влияния layout thrashing на производительность веб-приложений и методы по его оптимизации // Universum: технические науки. 2023. № 12-2 (117). С. 31-35. URL: <https://www.elibrary.ru/item.asp?id=59462810>

3. Zhao G., Chen N., Levtshev A. Simulation experiment research on six degrees of freedom test bench based on fuzzy pid control strategy // Bulletin of Science and Practice. 2021. Т. 7. № 1. С. 288-295. URL: <https://www.elibrary.ru/item.asp?id=44623245>
4. Пилецкая А.В. Формализация и экспериментальное исследование архитектурного состояния веб-приложения на основе структурных метрик // Математические методы в технологиях и технике. 2025. № 7. С. 105-108. URL: <https://www.elibrary.ru/item.asp?id=82852600>
5. Телегин В.А. Влияние стратегий кэширования данных на пользовательское взаимодействие в мобильных приложениях // Инновационная наука. 2023. № 11-2. С. 45-51. URL: <https://www.elibrary.ru/item.asp?id=54891567>