⚡ Практическое занятие

Метрики производительности и CAP-теорема

ФИО студента: Иванов Иван Иванович

Группа: ИСТ-21

Дата: 20.10.2025

📊 Основные метрики производительности

Percentiles (Процентили)

- **P50 (медиана):** 50% запросов быстрее этого времени
- **P95:** 95% запросов быстрее
- **P99:** 99% запросов быстрее
- **P99.9:** 99.9% запросов быстрее

⚖️ CAP-теорема

Правило: В распределенной системе можно гарантировать только 2 из 3 свойств:

Практические компромиссы:

- **CP системы:** MongoDB, Redis Cluster (жертвуют доступностью)
- **AP системы:** Cassandra, DynamoDB (жертвуют согласованностью)
- **CA системы:** PostgreSQL, MySQL (не устойчивы к разделению)

🔒 ACID свойства

D - Durability (Долговечность)

Зафиксированные изменения сохраняются навсегда

Пример: После подтверждения транзакции данные не теряются даже при сбое

📈 Availability (Доступность)

Формула: Availability = (Uptime / Total Time) × 100%

SLA уровни:

🧪 Практические задания

Требования к выполнению:

- выполняются на ОС **Linux** (любой дистрибутив)
- пакеты устанавливаются в **виртуальное окружение venv**
- для создания серверов можно использовать **LLM** (желательно Claude Sonnet подключить в IDE)

Задание 1: Установка и использование Apache Bench

Установка Apache Bench на Linux:

```
# Ubuntu/Debian
sudo apt-get install apache2-utils

# CentOS/RHEL/Fedora
sudo yum install httpd-tools
# или
sudo dnf install httpd-tools

# Проверка установки
ab -V
```

Тестирование производительности:

```
# Запуск 1000 запросов с 10 одновременными соединениями
ab -n 1000 -c 10 http://httpbin.org/get
```

Результаты тестирования:

Время выполнения теста:

30.97 seconds

секунд

Requests per second:

32.29

запросов/сек

Time per request (mean):

309.7

мс

Статистика соединений (мс):

	min	mean	median	max
Connect:	45	67	65	123
Processing:	156	241	234	456

Перцентили времени ответа:

P50: 298 мс P90: 345 мс P95: 367 мс P99: 456 мс

Описание полученных результатов:

Опишите ваши наблюдения о производительности сервера...

📊 Подробный анализ результатов Apache Bench

Общая информация о тесте:

Утилита: ApacheBench 2.3
URL: http://httpbin.org/get
Количество запросов: 1000
Уровень параллелизма: 10 одновременных соединений

Параметры сервера:

```
Server Software: awselb/2.0 ← AWS Elastic Load Balancer
Server Hostname: httpbin.org
Server Port: 80 ← HTTP порт
Document Path: /get ← Тестируемый endpoint
Document Length: 162 bytes ← Размер ответа
```

Расшифровка статистики соединений:

- **Connect:** Время установки TCP-соединения
- **Processing:** Время обработки запроса сервером
- **Waiting:** Время от отправки запроса до получения первого байта
- **Total:** Общее время запроса

Анализ качества:

Failed requests: 0 (ошибок нет) Transfer rate: 15.2 Kbytes/sec

Выводы о производительности:

```
1. Стабильность: Низкое стандартное отклонение - стабильная работа
2. Производительность: запросов/секунду - умеренная нагрузка
3. Задержки: Среднее время ответа - типично для географически удаленного сервера
4. Надежность: 0 неудачных запросов - сервер стабилен
5. Масштабируемость: 99% запросов в пределах X мс - хорошая предсказуемость
```

Рекомендации для улучшения:

- Оптимизировать можно за счет кеширования
- Рассмотреть CDN для уменьшения задержек
- Увеличить параллелизм для проверки пределов сервера

⏱️ Задание 2: Измерение задержек (latency)

Простой HTTP сервер с задержками (Flask):

```
from flask import Flask
import time
import random

app = Flask(__name__)

@app.route('/fast')
def fast_endpoint():
    return {'response': 'fast', 'latency': '10ms'}

@app.route('/slow')
def slow_endpoint():
    time.sleep(random.uniform(0.1, 0.5))  # 100-500ms задержка
    return {'response': 'slow', 'latency': '100-500ms'}

if __name__ == '__main__':
    app.run(port=5000)
```

Измерение latency:

```
# Тестирование быстрого endpoint
ab -n 100 -c 5 http://localhost:5000/fast

# Тестирование медленного endpoint
ab -n 100 -c 5 http://localhost:5000/slow
```

Результаты /fast endpoint:

RPS:

Средняя задержка:

мс

Результаты /slow endpoint:

RPS:

Средняя задержка:

мс

Сравнение веб-фреймворков

FastAPI (порт 8000)

Код на Python:

```
Вставьте код FastAPI сервера...
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

Средняя задержка:

мс

Результаты /slow endpoint:

RPS:

Средняя задержка:

мс

Краткие выводы:

```
Результаты тестирования FastAPI...
```

Django (порт 8001)

Код на Python:

```
Вставьте код Django сервера...
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

Средняя задержка:

мс

Результаты /slow endpoint:

RPS:

Средняя задержка:

мс

Краткие выводы:

```
Результаты тестирования Django...
```

Django REST Framework (порт 8002)

Код на Python:

```
Вставьте код DRF сервера...
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

Средняя задержка:

мс

Результаты /slow endpoint:

RPS:

Средняя задержка:

мс

Краткие выводы:

```
Результаты тестирования DRF...
```

LiteStar (порт 8003)

Код на Python:

```
Вставьте код LiteStar сервера...
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

Средняя задержка:

мс

Результаты /slow endpoint:

RPS:

Средняя задержка:

мс

Краткие выводы:

некнке

Общие выводы о производительности веб-фреймворков:

нкенке

📝 Домашнее задание

Задача: Доделать задания

Срок сдачи: К следующему занятию

Что нужно сделать:

- Протестировать 3 разных сайта с помощью Apache Bench
- Сравнить их производительность
- Построить график зависимости RPS от количества одновременных соединений
- Создать HTTP серверы на всех указанных фреймворках
- Провести сравнительное тестирование производительности

📊 Критерии оценки

📚 Литература

- Клеппман М. "Высоконагруженные приложения"
- Фаулер М. "Архитектура корпоративных программных приложений"
- Документация Apache Bench
- Официальная документация веб-фреймворков