⚡ Практическое занятие

Метрики производительности и CAP-теорема

ФИО студента:

Уральцева Александра Витальевна

Группа:

ИМО-426

Дата:

22.10.2025

📊 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:

192.513 секунд

Requests per second:

5.19 запросов/сек

Time per request (mean):

1925.13 мс

Статистика соединений (мс):

	min	mean	median	max
Connect:	148	192	177	15200
Processing:	164	1722	1594	16649

Перцентили времени ответа:

P50: 1770 мс P90: 1845 мс P95: 1875 мс P99: 16767 мс

Описание полученных результатов:

Тестирование показало среднюю производительность сервера на уровне 5.19 запросов в секунду при 10 одновременных подключениях. Большинство запросов (95%) обрабатывались в пределах 1875 мс. Однако наблюдаются значительные задержки для некоторых запросов, на что указывают максимальные значения времени соединения (15200

📊 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: 1.68 Kbytes/sec

Выводы о производительности:

Стабильность: Высокое стандартное отклонение (475.3 для соединения и 1425.1 для обработки) указывает на нестабильную работу.
Производительность: 5.19 запросов/секунду - умеренная нагрузка.
Задержки: Среднее время ответа - 1914 мс, что может быть типично для географически удаленного сервера.
Надежность: 0 неудачных запросов - сервер надежен.
Масштабируемость: 90% запросов в пределах 1845 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):

```python
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:

```bash
# 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:

143.05

Средняя задержка:

34.953 мс

Результаты /slow endpoint:

RPS:

15.10

Средняя задержка:

331.153 мс

Сравнение веб-фреймворков

FastAPI (порт 8000)

Код на Python:

```python
import asyncio
import random
from fastapi import FastAPI

app = FastAPI()

@app.get("/fast")
async def fast_endpoint():
    return {'response': 'fast', 'latency': '0ms'}

@app.get("/slow")
async def slow_endpoint():
    delay = random.uniform(0.1, 0.5)
    await asyncio.sleep(delay)
    return {'response': 'slow', 'latency': f'{int(delay * 1000)}ms'}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

74.73

Средняя задержка:

66.909 мс

Результаты /slow endpoint:

RPS:

14.17

Средняя задержка:

352.805 мс

Краткие выводы:

```
Server Port:            8000

Document Path:          /slow
Document Length:        37 bytes

Concurrency Level:      5
Time taken for tests:   7.056 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      18100 bytes
HTML transferred:       3700 bytes
Requests per second:    14.17 [#/sec] (mean)
Time per request:       352.805 [ms] (mean)
Time per request:       70.561 [ms] (mean, across all concurrent
requests)
Transfer rate:          2.51 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       2
Processing:   121  333 127.0    330     645
Waiting:      121  331 127.2    328     645
Total:        122  333 127.0    331     645

Percentage of the requests served within a certain time (ms)
```

Django (порт 8001)

Код на Python:

```python
import sys
import time
import random
from django.conf import settings
from django.urls import path
from django.core.management import execute_from_command_line
from django.http import JsonResponse
from django.shortcuts import HttpResponse

settings.configure(
    DEBUG=True,
    SECRET_KEY='a-secret-key-for-demonstration-purposes',
    ROOT_URLCONF=__name__,
)

def fast_endpoint(request):
    return JsonResponse({'response': 'fast', 'latency': '0ms'})


def slow_endpoint(request):
    delay = random.uniform(0.1, 0.5)
    time.sleep(delay)
    return JsonResponse({'response': 'slow', 'latency': f'{int(delay * 1000)}ms'})
```

```python
def index(request):
    return HttpResponse("Django server is running. Try /fast or /slow endpoints.")

urlpatterns = [
    path('', index),
    path('fast', fast_endpoint),
    path('slow', slow_endpoint),
]

if __name__ == "__main__":
    execute_from_command_line([sys.argv[0], 'runserver', '0.0.0.0:8001'])
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

66

Средняя задержка:

75.753 мс

Результаты /slow endpoint:

RPS:

13.02

Средняя задержка:

384.013 мс

Краткие выводы:

```
Time taken for tests:   7.680 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      18700 bytes
HTML transferred:       4000 bytes
Requests per second:    13.02 [#/sec] (mean)
Time per request:       384.013 [ms] (mean)
Time per request:       76.803 [ms] (mean, across all concurrent
requests)
Transfer rate:          2.38 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       1
Processing:   129  354 113.6    364     564
Waiting:      122  343 111.5    360     560
Total:        130  354 113.6    364     565

Percentage of the requests served within a certain time (ms)
  50%    364
  66%    418
  75%    454
  80%    462
  90%    502
  95%    517
```

Django REST Framework (порт 8002)

Код на Python:

```
import sys
import time
import random
from django.conf import settings

settings.configure(
    DEBUG=True,
    SECRET_KEY='a-very-secret-key-for-drf-testing',
    ROOT_URLCONF=__name__,
    INSTALLED_APPS=[
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'rest_framework',
    ],
)

from django.urls import path
from django.core.management import execute_from_command_line
from django.core.wsgi import get_wsgi_application
from rest_framework.views import APIView
from rest_framework.response import Response

class FastEndpoint(APIView):
    def get(self, request, format=None):
        return Response({'response': 'fast', 'latency': '0ms'})


class SlowEndpoint(APIView):
    def get(self, request, format=None):
        delay = random.uniform(0.1, 0.5)
        time.sleep(delay)
        return Response({'response': 'slow', 'latency': f'{int(delay * 1000)}ms'})


class IndexView(APIView):
    def get(self, request, format=None):
        return Response({"message": "Django REST Framework server is running. Try
/fast or /slow endpoints."})

urlpatterns = [
    path('', IndexView.as_view()),
    path('fast', FastEndpoint.as_view()),
    path('slow', SlowEndpoint.as_view()),
]

application = get_wsgi_application()

if __name__ == "__main__":
    execute_from_command_line([sys.argv[0], 'runserver', '0.0.0.0:8002'])
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

69.25

Средняя задержка:

72.206 мс

Результаты /slow endpoint:

RPS:

14.26

Средняя задержка:

350.737 мс

Краткие выводы:

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       1
Processing:    16   71 121.9     35     658
Waiting:        8   58 109.6     29     651
Total:         17   71 122.0     36     658

Percentage of the requests served within a certain time (ms)
  50%     36
  66%     43
  75%     53
  80%     58
  90%     99
  95%    435
  98%    618
  99%    658
 100%    658 (longest request)

---

Server Software:        WSGIServer/0.2
Server Hostname:        localhost
```

LiteStar (порт 8003)

Код на Python:

```python
import asyncio
import random
import uvicorn
from litestar import Litestar, get

@get("/fast")
async def fast_endpoint() -> dict[str, str]:
    return {"response": "fast", "latency": "0ms"}


@get("/slow")
async def slow_endpoint() -> dict[str, str]:
```

```
    delay = random.uniform(0.1, 0.5)
    await asyncio.sleep(delay)
    return {"response": "slow", "latency": f"{int(delay * 1000)}ms"}

app = Litestar(route_handlers=[fast_endpoint, slow_endpoint])

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8003)
```

Результаты тестирования:

Результаты /fast endpoint:

RPS:

74.51

Средняя задержка:

67.109 мс

Результаты /slow endpoint:

RPS:

14.52

Средняя задержка:

344.238 мс

Краткие выводы:

```
  66%    396
  75%    430
  80%    453
  90%    493
  95%    516
```

Общие выводы о производительности веб-фреймворков:

На основе проведенных тестов Flask показал себя как самый быстрый фреймворк для простых, быстрых задач, обработав около 143 запросов в секунду (RPS) на эндпоинте без задержки.

Асинхронные фреймворки FastAPI и Litestar продемонстрировали практически идентичные и очень высокие результаты (около 75 RPS). Это подтверждает их эффективность для современных приложений, особенно тех, где много операций ожидания (например, запросы к базам данных или другим API).

Django и Django REST Framework оказались немного медленнее в этих тестах (примерно 66-69 RPS), что ожидаемо из-за их более сложной внутренней структуры и большого количества встроенных функций. Их основное преимущество не в максимальной скорости на простых запросах, а в скорости и удобстве разработки больших, комплексных проектов.

В итоге, для легковесных задач Flask — лидер по скорости. Для сложных асинхронных API отлично подойдут FastAPI или Litestar. Для крупных проектов, где важна экосистема и наличие готовых инструментов, Django и DRF остаются прекрасным выбором.

📝 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