Принципы построения высоконагруженных систем
Институт прикладных компьютерных наук ИТМО

Домашнее задание 2. Масштабируемость и балансировка нагрузки

Георгий Семенов
georgii.v.semenov@mail.ru
Мягкий дедлайн: Сб, 06.12.2025, 23:59 МСК
Жесткий дедлайн: Сб, 13.12.2025, 23:59 МСК

Имя Фамилия

DDIA25-HW2-NameSurname.pdf

November 28, 2025

---

1 Проектирование высоконагруженных систем

В ходе выполнения упражнений в этом домашнем задании вам предстоит помочь корпорации *«Репликон»* разобраться с вопросами масштабирования и балансировки.

1.1 Масштабирование

Корпорация «Репликон» запускает свой видеохостинг и решила подготовить серверное оборудование для зон облака, в которых планируется разместить его сервисы CDN. В рамках этого процесса Cloud-команде требуется подобрать характеристики для «стойки мечты».

Стажер предложил использовать для всех новых стоек одинаковые машины с максимально возможными характеристиками CPU, RAM, Disk и Network.

Эта идея не очень понравилась опытным инженерам *Репликона*, которые на это возразили, что дорогие вертикальные улучшения серверов не всегда приводят к пропорциональному росту производительности из-за аппаратных ограничений, но совершенно не пояснили, о каких именно ограничениях идет речь.

Также, инженеры предложили не использовать отдельные серверные стойки для видеохостинга, а разместить CDN-сервисы, просто горизонтально масштабируя уже существующие сервера облака *Репликона*.

Задание:

1. (0.75 балл) Помогите стажеру понять – какие аппаратные ограничения имеют в виду опытные инженеры? *(Указание: рассмотрите, как процессор, память, диск и сеть с точки зрения материнской платы взаимодействуют друг с другом; приведите* ≥ 3 *ограничения и поясните их)*
2. (0.75 балл) Почему при горизонтальном масштабировании на всех машинах стараются использовать одни и те же hardware-характеристики (что такое vCPU)?

Решение:

□

1.2 Тонкости CAP-теоремы

Ведущий разработчик корпорации *Репликон* является огромным поклонником CAP-теоремы[1] и исследователя в области распределенных систем Мартина Клеппмана. В очередной раз перелистывая любимого «кабанчика»[2], ведущий разработчик неожиданно для себя обнаружил в книге своего кумира «махровый популизм» – Мартин назвал CAP-теорему «бесполезной»!

Ведущий разработчик был в ярости! Такого предательства он не мог простить. Он решил написать гневное письмо Мартину Клеппману, в котором изложил бы все свои аргументы в пользу важности CAP-теоремы. Но перед этим он решил подробнее изучить вопрос и изучить, почему же исследователь так считает.

Задание:

1. (1.5 балл) «Докажите» (или покажите) CAP-теорему, показывая несовместность каждой двойки свойств с оставшимся третьим свойством. Приведите по примеру соответствующих систем и соответствующий тип consistency.

2. (0.75 балл) Приведите ≥ 3 критических аргумента Мартина Клеппмана о CAP-теореме из книги (раздел «Теорема CAP»).

3. (0.75 балл) Наряду с CAP-теоремой существует также PACELC-теорема. Сформулируйте PACELC-теорему и объясните, как она расширяет CAP-теорему.

4. (2 балл) (*) Свойство SEC (strong eventual consistency) считается «разрешением» CAP-теоремы. Объясните, что это за свойство и почему рассматривая набор свойств {SEC, A, P} мы избегаем противоречия CAP-теоремы.

5. (3 балла) (*) В статье «A Critique of the CAP Theorem» Мартин Клеппман более подробно критикует классическую формулировку CAP-теоремы, а также неявно предлагает альтернативную формулировку на основе «delay-sensitivity framework». Сформулируйте ее.

Решение:

□

[1] CAP-теорема (Brewer, 2000): в распределённых системах невозможно одновременно обеспечить все три свойства: согласованность (consistency), доступность (availability) и устойчивость к разделению (partition tolerance).

[2] «Кабанчик» – это книга Мартина Клеппмана «Высоконагруженные приложения: программирование, масштабирование, поддержка». Для работы над заданием можно воспользоваться текстом на русском или английском языке

1.3 Модели репликации

Для хранения пользовательских данных в «Репликоне» решили воспользоваться платформой хранения больших данных YTsaurus, а точнее – ее механизмом реплицированных динамических таблиц, которые представляют собой key-value хранилища в реляционном стиле.

Если коротко, применение **модифицирующих операций** сводится к добавлению соответствующей записи в очередь операций на master-кластере. Затем каждое изменение из лога последовательно применяется на всех slave-кластерах синхронно и/или асинхронно (это может приводить к рассогласованию данных между master- и slave-кластерами в течение некоторого времени). Пока операция модификации не применена на всех slave-кластерах, она не очищается из очереди на master-кластере. **Операции чтения** производятся на slave-кластерах.

Задание:

1. (1.25 балл)
   - Сформулируйте общее и различия между понятиями *федерация* и *шардирование* в контексте баз данных.
   - Как эти подходы помогают масштабировать системы хранения данных?
   - Что делать, если мы уперлись в пределы вертикального масштабирования внутри одной СУБД (при федерации) / партиции (при шардировании)?
2. (1.25 балла)
   - Опишите с точки зрения CAP-теоремы, к какому классу систем можно отнести реплицированные динамические таблицы в YTsaurus.
   - Предположим, что один из slave-кластеров отказал. Почему в этот момент система все еще может считаться доступной?
   - Какой тип согласованности (consistency) обеспечивается реплицированными динамическими таблицами? Это модель ACID или BASE?
3. (2 балла) (*) Утверждается, что CRDT-типы [3] способны решить задачу master-master репликации. Объясните, что это за типы данных и почему им не требуется механизм консенсуса для обеспечения согласованности.

Решение:

□

[3] Хабр: Наивное введение в CRDT-типы

1.4 Балансировка нагрузки

«Дореплицировались!» – подумал лид Конвергентий Смоуктуновский одной из команд «Репликона», когда его поисковая система картинок с котиками «Котоморфизм» сложилась карточным домиком под напором пользователей.

СТО «Репликона» не остался в долгу и поручил Конвергентию разобраться с проблемой, хотя и не очень обрадовался тому, что лид в его подчинении не догадался заранее позаботиться о масштабировании и балансировке нагрузки в критически важном сервисе.

Чтобы быстро исправить техдолг перед внедрением полноценного кэша на основе Redis, Конвергентий решил выполнить простые шаги:

1. Переехать из одной зоны в три георазпределенные зоны с балансировкой нагрузки между ними с помощью **nginx**, чтобы «Котоморфизм» спокойно переживал отказ одной из зон.

2. Добавить **fallback** на уровне балансировщика, чтобы отображать типовые картинки с котиками, если «Котоморфизм» снова начнет «мяукать» под нагрузкой.

Помогите Конвергентию выполнить поставленные задачи, чтобы «Котоморфизм» снова начал радовать пользователей. В папке `homeworks/cotomorphism` находится тестовый стенд из нескольких узлов «Котоморфизма», которые возвращают различные картинки с котиками. С помощью эндпоинтов `/degrade/on` и `/degrade/off` можно включать и выключать «режим деградации», при котором сервер начинает возвращать ошибку 503 вместо картинки.

В вашем распоряжении baseline-конфигурация `default.conf`, которая балансирует нагрузку между тремя узлами «в разных зонах» «Котоморфизма» с помощью round-robin.

% % % %

```
upstream app_backend {
    server cat1:80;
    server cat2:80;
    server cat3:80;
}

upstream app_fallback {
    server cat_cached:80;
}

server {
    listen 80;
```

```
    server_name _;

    location / {
        proxy_pass http://app_backend;
    }

    location @fallback {
        proxy_pass http://app_fallback;
    }
}
```

Задание:

1. (0.75 балл) Сделайте так, чтобы при возникновении ошибки 503 от одного из узлов без каких-либо дальнейших попыток балансировщик переадресовывал запрос на upstream `app_fallback`.

2. (0.75 балл) Настройте балансировщик так, чтобы он пытался повторить запрос на другой узел при возникновении ошибок 503. Если `cat1`, `cat2`, `cat3` лежат, то необходимо спроксировать запрос на upstream `app_fallback`.

3. (0.75 балл) Добавьте кэширование запроса картинок на уровне балансировщика с временем жизни кэша 5 секунд.

4. (0.75 балл) Добавьте rate limiting на уровне балансировщика: не более 1 запроса в секунду от одного клиента. Убедитесь, что при превышении лимита вы падаете на `app_fallback`.

Указание: оформите ваше решение как минимальный набор команд, которые надо дописать в baseline-конфигурацию. Воспользуйтесь блоком minted..

Решение:

□

1.5 Кэширование

Уверенный мидл разработчик Шардимир в свободное время от работы в «Репликоне» играет в Minecraft и больше всего на свете любит мод ComputerCraft, который добавляет в игру компьютеры, которые работают на операционной системе и поддерживают программы в виде скриптов на языке Lua. Единственное, что огорчает Шардимира в этом моде, – это необходимость писать на совершенно бесполезном в жизни языке Lua.

Шардимир получил задание от своего тимлида Конвергентия – реализовать кэш для сервиса «Котоморфизм» на основе Redis. Какое же было удивление Шардимира, когда он осознал, что для этого ему придется написать пару Lua-скриптов для Redis!

```
return 'Hello World'
```

```
EVAL "return 'Hello World'" 0
```

Помогите Шардимиру реализовать **cache-aside** кэширование запросов к сервису «Котоморфизм» с помощью Lua-скриптов для Redis. Поднять консоль можно с помощью папки `homeworks/redis`.

Задание:

1. (1 балл) Напишите два скрипта: один принимает в качестве аргументов поисковый запрос (строка), TTL (в секундах) и путь к файлу (строка); скрипт должен сохранить в Redis значение из файла по ключу поискового запроса. Второй должен принимать в качестве аргумента поисковый запрос (строка) и возвращать значение по этому ключу из Redis. Вам могут пригодиться команды `EVAL`, `GET`, `SETEX`.
2. (1 балл) Реализуйте rate limiter для пользователя как соответствующие Lua скрипты для Redis. Вам могут пригодиться команды `INCR`, `EXPIRE`.

Решение:

□