



ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование

Не забыть включить запись!





Меня хорошо видно && слышно?

Ставьте ☐+, если все хорошо
Напишите в чат, если есть проблемы

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в Slack #канал группы или #general



Вопросы вижу в чате, могу ответить не сразу



Паттерны кэширования и основные принципы



Тюменцев Евгений

Генеральный директор

HWdTech LLC

etyumentcev@gmail.com

Преподаватель



Тюменцев Евгений

- 9 лет руковожу компаний по разработке ПО
- в прошлом занимался разработкой многопоточных кросс-платформенных приложений на C++, серверных приложений на C#
- 20 преподаю ООП, паттерны, C++, C#, Kotlin

Цели вебинара | После занятия вы сможете

1

Использовать основные
паттерны кэширования

2

Решать типичные проблемы,
связанные с кэшированием

3

Выбирать инструмент кэширования
под задачу

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. The diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the slide. Centered within this band is the text "01. Кэширование" in a large, white, sans-serif font.

01. Кэширование

Кэширование

- Кэш – промежуточный буфер с более быстрым доступом, чем основное хранилище

Кэширование

- Кэш – промежуточный буфер с более быстрым доступом, чем основное хранилище
- Кэширование – способ оптимизации, использующий кэш

Кэширование

- Кэш – промежуточный буфер с более быстрым доступом, чем основное хранилище
- Кэширование – способ оптимизации, использующий кэш
- Почему не хранить всё в кэше?

Кэширование

- Кэш – промежуточный буфер с более быстрым доступом, чем основное хранилище
- Кэширование – способ оптимизации, использующий кэш
- Почему не хранить всё в кэше?
 - Дороже

Кэширование

- Кэш – промежуточный буфер с более быстрым доступом, чем основное хранилище
- Кэширование – способ оптимизации, использующий кэш
- Почему не хранить всё в кэше?
 - Дороже
 - Меньший объём данных

Кэширование

- Кэш – промежуточный буфер с более быстрым доступом, чем основное хранилище
- Кэширование – способ оптимизации, использующий кэш
- Почему не хранить всё в кэше?
 - Дороже
 - Меньший объём данных
 - Обычно «энергозависимый»

Принцип локальности

- В конкретные моменты времени используется лишь небольшое подмножество данных («активная зона»)

Принцип локальности

- В конкретные моменты времени используется лишь небольшое подмножество данных («активная зона»)
- Примеры:
 - Чаты – последние сообщения

Принцип локальности

- В конкретные моменты времени используется лишь небольшое подмножество данных («активная зона»)
- Примеры:
 - Чаты – последние сообщения
 - Сервис такси – последние заказы

Принцип локальности

- В конкретные моменты времени используется лишь небольшое подмножество данных («активная зона»)
- Примеры:
 - Чаты – последние сообщения
 - Сервис такси – последние заказы
- В СУБД активная зона «поднимается» в память

Алгоритмы вытеснения

- Алгоритм Беладии (идеальный, теоретический)

Алгоритмы вытеснения

- Алгоритм Беладди (идеальный, теоретический)
- Least Recently Used (LRU)

Алгоритмы вытеснения

- Алгоритм Беладди (идеальный, теоретический)
- Least Recently Used (LRU)
- Most Recently Used (MRU)

Алгоритмы вытеснения

- Алгоритм Беладди (идеальный, теоретический)
- Least Recently Used (LRU)
- Most Recently Used (MRU)
- Псевдо-LRU (PLRU)

Алгоритмы вытеснения

- Алгоритм Беладди (идеальный, теоретический)
- Least Recently Used (LRU)
- Most Recently Used (MRU)
- Псевдо-LRU (PLRU)
- Least Frequently Used (LFU)

Алгоритмы вытеснения

- Алгоритм Беладди (идеальный, теоретический)
- Least Recently Used (LRU)
- Most Recently Used (MRU)
- Псевдо-LRU (PLRU)
- Least Frequently Used (LFU)
- Adaptive Replacement (балансировка между LRU и LFU)

Максима

- Кэш **не должен** держать нагрузку

Максима

- Кэш **не должен** держать нагрузку
- Кэш нужен только для ускорения ответа

Максима

- Кэш **не должен** держать нагрузку
- Кэш нужен только для ускорения ответа
- Для проверки этого утверждения нужно проводить нагрузочное тестирование

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. The diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the slide. In the center of this band, the text '02. Виды кэширования' is written in a large, white, sans-serif font.

02. Виды кэширования

Что кэшируем?

- Результаты запросов к БД

Что кэшируем?

- Результаты запросов к БД
- Ответы внешних сервисов

Что кэшируем?

- Результаты запросов к БД
- Ответы внешних сервисов
- Динамически сгенерированные страницы

Что кэшируем?

- Результаты запросов к БД
- Ответы внешних сервисов
- Динамически сгенерированные страницы
- Данные для обмена между сервисами

Классификация кэшей: по ценности данных

- Легко вычисляемые

Классификация кэшей: по ценности данных

- Легко вычисляемые
- Длительно вычисляемые

Классификация кэшей: по ценности данных

- Легко вычисляемые
- Длительно вычисляемые
- Невосстанавливаемые

Классификация кэшей: по типу доступа

- Локальный

Классификация кэшей: по типу доступа

- Локальный
- Приватный

Классификация кэшей: по типу доступа

- Локальный
- Приватный
- Общий

Кэширование: вопросы

- Вы разрабатываете веб-приложение. Когда нужно начинать думать о кэшировании?

Кэширование: вопросы

- Вы разрабатываете веб-приложение. Когда нужно начинать думать о кэшировании?
- Где граница «легкой вычислимости» данных, когда уже требуется кэширование?

Кэширование: вопросы

- Вы разрабатываете веб-приложение. Когда нужно начинать думать о кэшировании?
- Где граница «легкой вычислимости» данных, когда уже требуется кэширование?
- Какой основной минус кэширования?

Кэширование: вопросы

- Вы разрабатываете веб-приложение. Когда нужно начинать думать о кэшировании?
- Где граница «легкой вычислимости» данных, когда уже требуется кэширование?
- Какой основной минус кэширования?
- Какие данные нельзя хранить в кэше? Почему?

The background of the slide features an aerial view of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that contains a white network pattern of interconnected dots and lines, suggesting a digital or technological theme.

03. Встроенные инструменты

Кэширование в браузере

- Кэшировать можем только GET (идемпотентный)

Кэширование в браузере

- Кэшировать можем только GET (идемпотентный)
- Информация о кэшировании передаётся в заголовках
 - ETag

Кэширование в браузере

- Кэшировать можем только GET (идемпотентный)
- Информация о кэшировании передаётся в заголовках
 - ETag
 - If-Modified-Since

Кэширование в браузере

- Кэшировать можем только GET (идемпотентный)
- Информация о кэшировании передаётся в заголовках
 - ETag
 - If-Modified-Since
 - Cache-Control

Кэширование в браузере

- Кэшировать можем только GET (идемпотентный)
- Информация о кэшировании передаётся в заголовках
 - ETag
 - If-Modified-Since
 - Cache-Control
- LocalStorage

Кэширование в браузере – инвалидация

- Указывать версию в GET-параметрах

Кэширование в браузере – инвалидация

- Указывать версию в GET-параметрах
- Добавлять хэш (например, md5) в имена файлов

Кэширование в nginx

```
proxy_cache_path /data/nginx/cache keys_zone=cache_zone:10m;

map $request_method $purge_method {
    PURGE      1;
    default    0;
}

server {
    ...
    location / {
        proxy_pass http://backend;
        proxy_cache cache_zone;
        proxy_cache_key $uri;
        proxy_cache_purge $purge_method;
    }
}
```


Кэширование на сервере

- Хранение данных в ОЗУ

Кэширование на сервере

- Хранение данных в ОЗУ
- Memcache

Кэширование на сервере

- Хранение данных в ОЗУ
- Memcache
- Redis

The background of the slide is a blue-tinted aerial photograph of a dense city skyline, likely New York City. Overlaid on this image is a semi-transparent network diagram consisting of numerous small blue dots connected by thin, light blue lines, creating a web-like pattern across the center of the slide.

04. Memcached

Memcached: преимущества и недостатки

- Преимущества:
 - Многопоточная архитектура

Memcached: преимущества и недостатки

- Преимущества:
 - Многопоточная архитектура
 - Производительность

Memcached: преимущества и недостатки

- Преимущества:
 - Многопоточная архитектура
 - Производительность
- Недостатки:
 - Нет persistence «из коробки»

Memcached: преимущества и недостатки

- Преимущества:
 - Многопоточная архитектура
 - Производительность
- Недостатки:
 - Нет persistence «из коробки»
 - Не поддерживает типизацию

Memcached: преимущества и недостатки

- Преимущества:
 - Многопоточная архитектура
 - Производительность
- Недостатки:
 - Нет persistence «из коробки»
 - Не поддерживает типизацию
 - Безопасность

Memcached

- Внутри – хэш-таблица

Memcached

- Внутри – хэш-таблица
- Алгоритм вытеснения - <https://memcached.org/blog/modern-lru/>

Memcached

- Внутри – хэш-таблица
- Алгоритм вытеснения - <https://memcached.org/blog/modern-lru/>
- Основные операции: get, set, delete

Memcached

- Внутри – хэш-таблица
- Алгоритм вытеснения - <https://memcached.org/blog/modern-lru/>
- Основные операции: get, set, delete
- Атомарные операции: incr, decr, append, prepend, cas, add

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. This diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the slide. Centered within this band is the text "05. Redis" in a large, white, sans-serif font.

05. Redis

Redis: преимущества и недостатки

- Преимущества:
 - Поддерживает типизацию

Redis: преимущества и недостатки

- Преимущества:
 - Поддерживает типизацию
 - Поддерживает очереди и транзакции

Redis: преимущества и недостатки

- Преимущества:
 - Поддерживает типизацию
 - Поддерживает очереди и транзакции
 - Возможна master-slave репликация

Redis: преимущества и недостатки

- Преимущества:
 - Поддерживает типизацию
 - Поддерживает очереди и транзакции
 - Возможна master-slave репликация
 - Есть persistence «из коробки»

Redis: преимущества и недостатки

- Преимущества:
 - Поддерживает типизацию
 - Поддерживает очереди и транзакции
 - Возможна master-slave репликация
 - Есть persistence «из коробки»
- Недостатки:
 - Однопоточная архитектура

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent network diagram consisting of numerous small blue dots connected by thin, light-blue lines, creating a web-like pattern across the center of the slide.

06. Tarantool

Tarantool: преимущества и недостатки

- Преимущества:
 - ACID, репликация, персистентность

Tarantool: преимущества и недостатки

- Преимущества:
 - ACID, репликация, персистентность
 - Хранимые процедуры на Lua

Tarantool: преимущества и недостатки

- Преимущества:
 - ACID, репликация, персистентность
 - Хранимые процедуры на Lua
 - Может реплицировать MySQL

Tarantool: преимущества и недостатки

- Преимущества:
 - ACID, репликация, персистентность
 - Хранимые процедуры на Lua
 - Может реплицировать MySQL
- Недостатки:
 - Дорогое хранение холодных данных

Tarantool: преимущества и недостатки

- Преимущества:
 - ACID, репликация, персистентность
 - Хранимые процедуры на Lua
 - Может реплицировать MySQL
- Недостатки:
 - Дорогое хранение холодных данных
 - Нет полной поддержки SQL



07. Проблемы и пути решения



Cache-miss

$$\text{AverageTime} = \text{DbAccessTime} * \text{CacheMissRate} + \text{CacheAccessTime}$$

Cache-miss

$$\text{AverageTime} = \text{DbAccessTime} * \text{CacheMissRate} + \text{CacheAccessTime}$$

Пусть:

- $\text{DbAccessTime} = 100\text{ms}$
- $\text{CacheAccessTime} = 20\text{ms}$

Cache-miss

$$\text{AverageTime} = \text{DbAccessTime} * \text{CacheMissRate} + \text{CacheAccessTime}$$

Пусть:

- $\text{DbAccessTime} = 100\text{ms}$
- $\text{CacheAccessTime} = 20\text{ms}$

Тогда при $\text{CacheMissRate} > 0.8$ получаем, что $\text{AverageTime} > \text{DbAccessTime}$

Cache-miss

$$\text{AverageTime} = \text{DbAccessTime} * \text{CacheMissRate} + \text{CacheAccessTime}$$

Пусть:

- $\text{DbAccessTime} = 100\text{ms}$
- $\text{CacheAccessTime} = 20\text{ms}$

Тогда при $\text{CacheMissRate} > 0.8$ получаем, что $\text{AverageTime} > \text{DbAccessTime}$

Кэш замедляет среднее время ответа!

Cache-miss

$$\text{AverageTime} = \text{DbAccessTime} * \text{CacheMissRate} + \text{CacheAccessTime}$$

Пусть:

- $\text{DbAccessTime} = 100\text{ms}$
- $\text{CacheAccessTime} = 20\text{ms}$

Тогда при $\text{CacheMissRate} > 0.8$ получаем, что $\text{AverageTime} > \text{DbAccessTime}$

Кэш замедляет среднее время ответа!

Вывод: минимальный $\text{CacheHitRate} = \text{CacheAccessTime} / \text{DbAccessTime}$

Кэширование тяжёлых запросов

Задача:

Есть тяжёлый запрос, который выполняется несколько секунд.
Обращение к результатам запроса может быть сотни раз в минуту.

Хотим закэшировать данные с интервалом обновления, например, 1 час.

Крон недоступен.

Базовое решение

```
$result = $memcache->get('heavy_request');  
  
if ($result !== false) {  
    return $result;  
}  
  
$result = $someRepository->getSomeHeavyData();  
  
$memcache->set('heavy_request', $result, 60 * 60);
```

Методика дублирования

```
$result = $memcache->get('heavy_request');
```

```
if ($result !== false) {  
    return $result;  
}
```

```
// временно восстанавливаем ключ из копии
```

```
$copy = $memcache->get('heavy_request_copy');  
$memcache->set('heavy_request', $copy, 60 * 60);
```

```
$result = $someRepository->getSomeHeavyData();
```

```
$memcache->set('heavy_request', $result, 60 * 60);
```

```
// дополнительно копируем значение с большим TTL
```

```
$memcache->set('heavy_request_copy', $result, 60 * 60 + 30);
```


Вероятностное обновление

```
$result = $memcache->get('heavy_request');

// определяем необходимость обновления
$shouldExpire = $memcache->get('heavy_request_expiration');
$needUpdate = ($result === false) ||
               (time() + $delta * $beta * log(rand()) > $shouldExpire);

if (!$needUpdate) {
    return $result;
}

$result = $someRepository->getSomeHeavyData();

$memcache->set('heavy_request', $result, 60*60);

// дополнительно сохраняем «момент экспирации»
$memcache->set('heavy_request_expiration', time() + 60 * 60, 60 * 60);
```

Параллельные обновления (concurrency)

- Оптимистичный подход

Параллельные обновления (concurrency)

- Оптимистичный подход
 - Нужна дополнительная обработка неудачных попыток

Параллельные обновления (concurrency)

- Оптимистичный подход
 - Нужна дополнительная обработка неудачных попыток
- Пессимистичный подход

Параллельные обновления (concurrency)

- Оптимистичный подход
 - Нужна дополнительная обработка неудачных попыток
- Пессимистичный подход
 - Параллельная блокировка

Параллельные обновления (concurrency)

- Оптимистичный подход
 - Нужна дополнительная обработка неудачных попыток
- Пессимистичный подход
 - Параллельная блокировка
 - «Вечная» блокировка

Параллельные обновления (concurrency)

- Оптимистичный подход
 - Нужна дополнительная обработка неудачных попыток
- Пессимистичный подход
 - Параллельная блокировка
 - «Вечная» блокировка
 - Взаимная блокировка

Инвалидация кэша

- TTL

Инвалидация кэша

- TTL
- Прямая инвалидация
 - Очереди

Инвалидация кэша

- TTL
- Прямая инвалидация
 - Очереди
 - Тэги

The background of the slide is a high-angle, blue-tinted aerial photograph of a city skyline, likely New York City, showing numerous skyscrapers and buildings. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. This diagram consists of numerous small dots (nodes) connected by thin white lines, forming a complex web that suggests data connectivity or a network structure. The text '08. Кластеризация' is centered within this blue band in a large, white, sans-serif font.

08. Кластеризация

Кластеризация

- Алгоритмы определения сервера

Кластеризация

- Алгоритмы определения сервера
 - Хеширование по модулю

Кластеризация

- Алгоритмы определения сервера
 - Хеширование по модулю
 - Согласованное хеширование

Кластеризация

- Алгоритмы определения сервера
 - Хеширование по модулю
 - Согласованное хеширование
- Холодный старт

Кластеризация

- Алгоритмы определения сервера
 - Хеширование по модулю
 - Согласованное хеширование
- Холодный старт
 - Плавный запуск пользователей

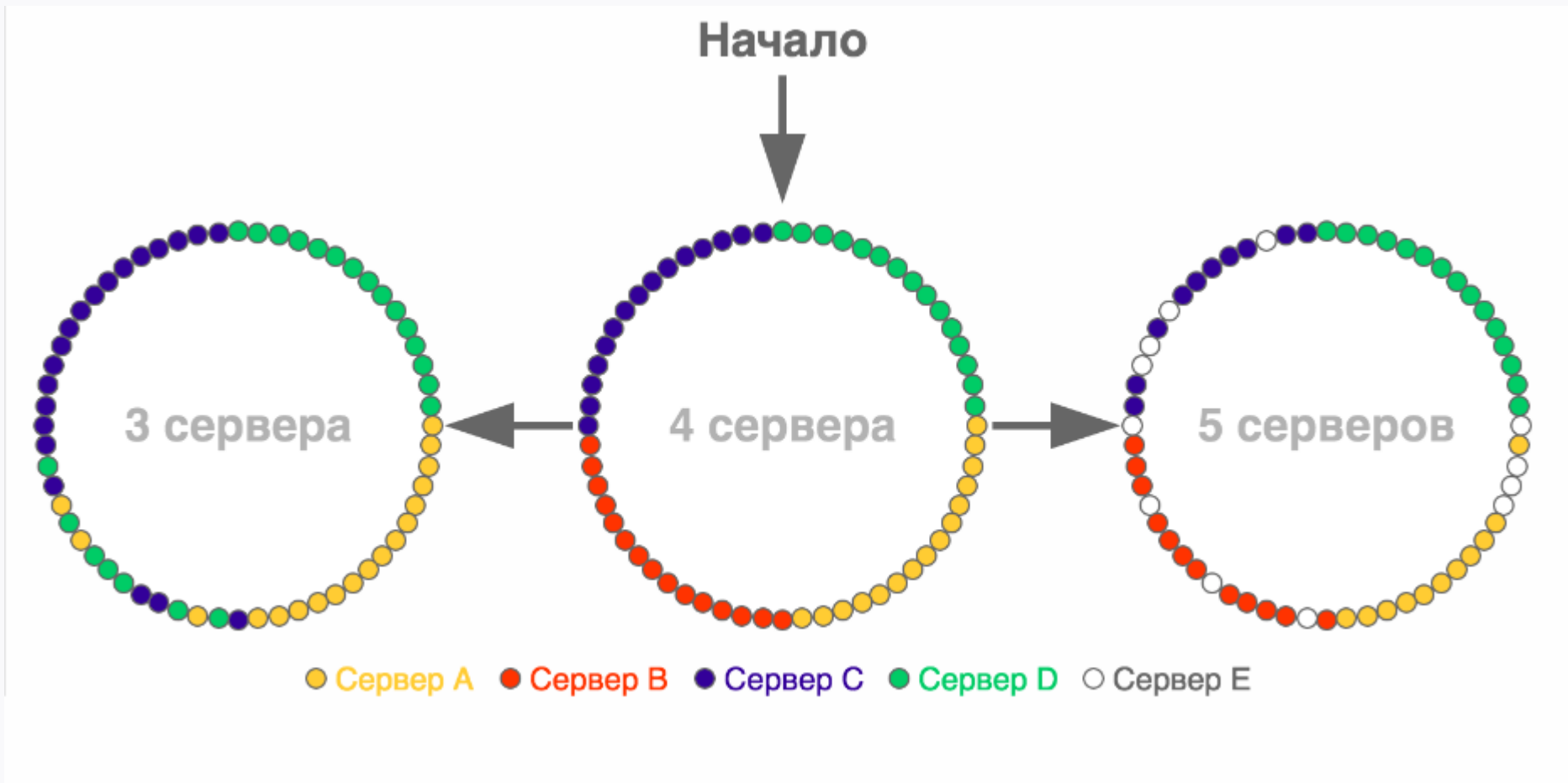
Кластеризация

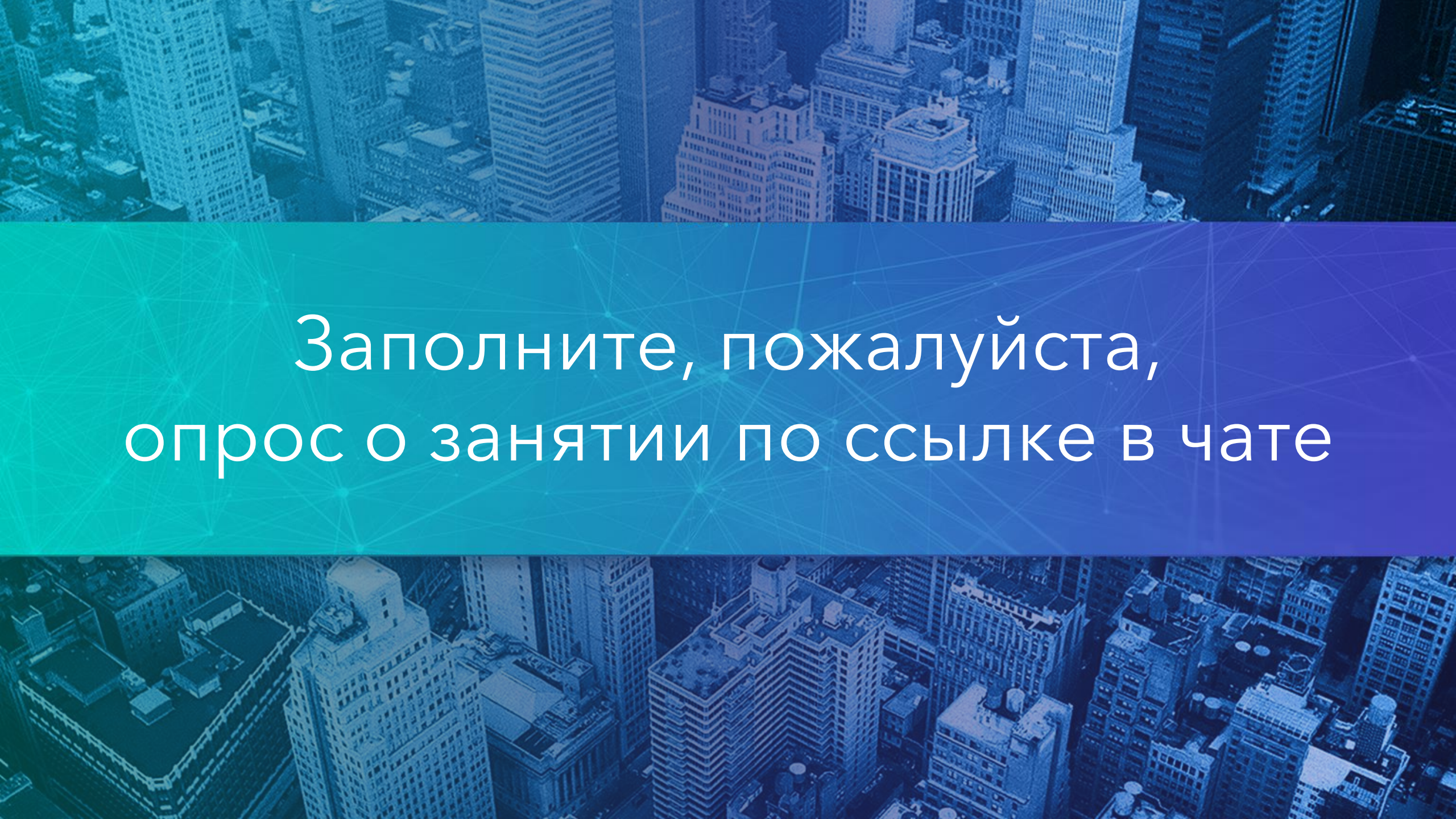
- Алгоритмы определения сервера
 - Хеширование по модулю
 - Согласованное хеширование
- Холодный старт
 - Плавный запуск пользователей
 - «Прогрев» со случайным TTL

Кластеризация

- Алгоритмы определения сервера
 - Хеширование по модулю
 - Согласованное хеширование
- Холодный старт
 - Плавный запуск пользователей
 - «Прогрев» со случайным TTL
- «Горячие» ключи

Консистентное хэширование



The background of the image is an aerial photograph of a city with many skyscrapers, overlaid with a semi-transparent blue layer. A network of thin, light-blue lines connects various points across the blue area, creating a digital or technological aesthetic. The text is centered in this blue area.

Заполните, пожалуйста,
опрос о занятии по ссылке в чате