Оптимизация WEB-приложений

Лекция 4

Оптимизация серверной части веб-приложений

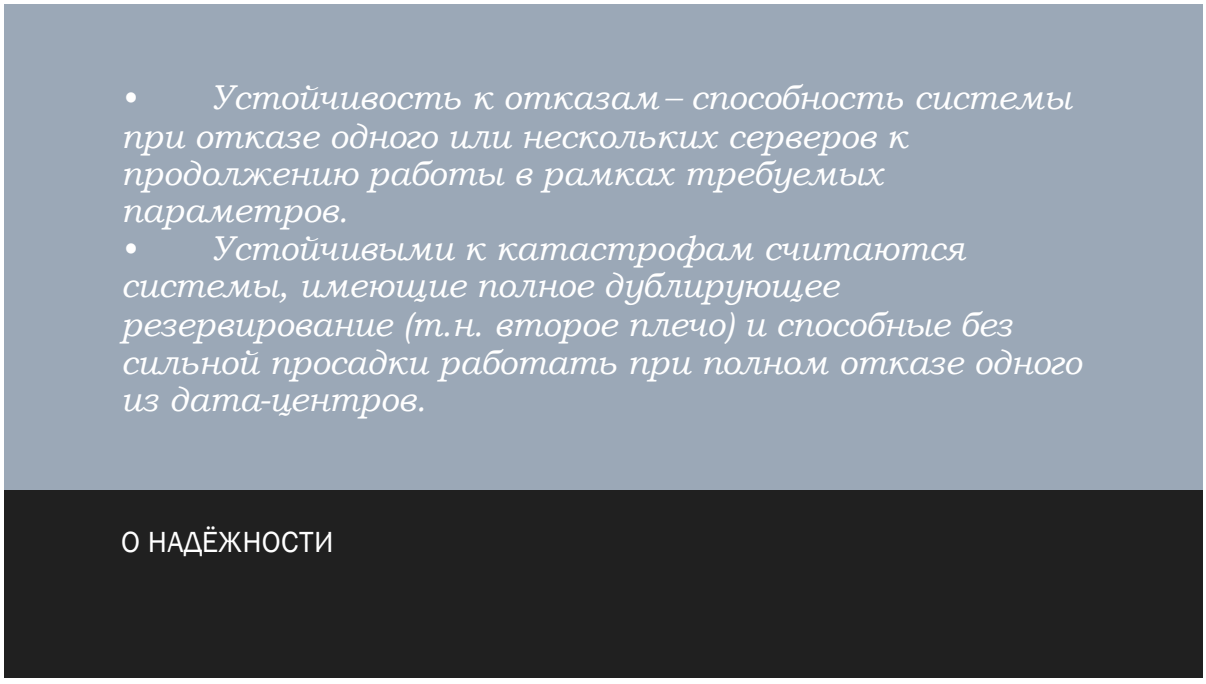
Речь пойдет о том, как оптимизировать веб-приложения, которые страдают от хронических проблем с масштабируемостью, производительностью или надежностью.

Терминология

Давайте для начала разберемся в терминологии. Говоря о производительности веб-проектов или веб-систем, я в первую очередь имею в виду back-end и серверную составляющую. Мерилом **производительности** приложения у нас будет являться количество обрабатываемых запросов в секунду (RPS) и скорость их выполнения (TTFB – Time to First Byte).

- Соответственно, под **масштабируемостью** системы мы будем понимать пул возможностей для увеличения RPS.

Теперь о надежности. Здесь обязательно нужно разделять два п онятия: отказоустойчивость и катастрофоустойчивость.

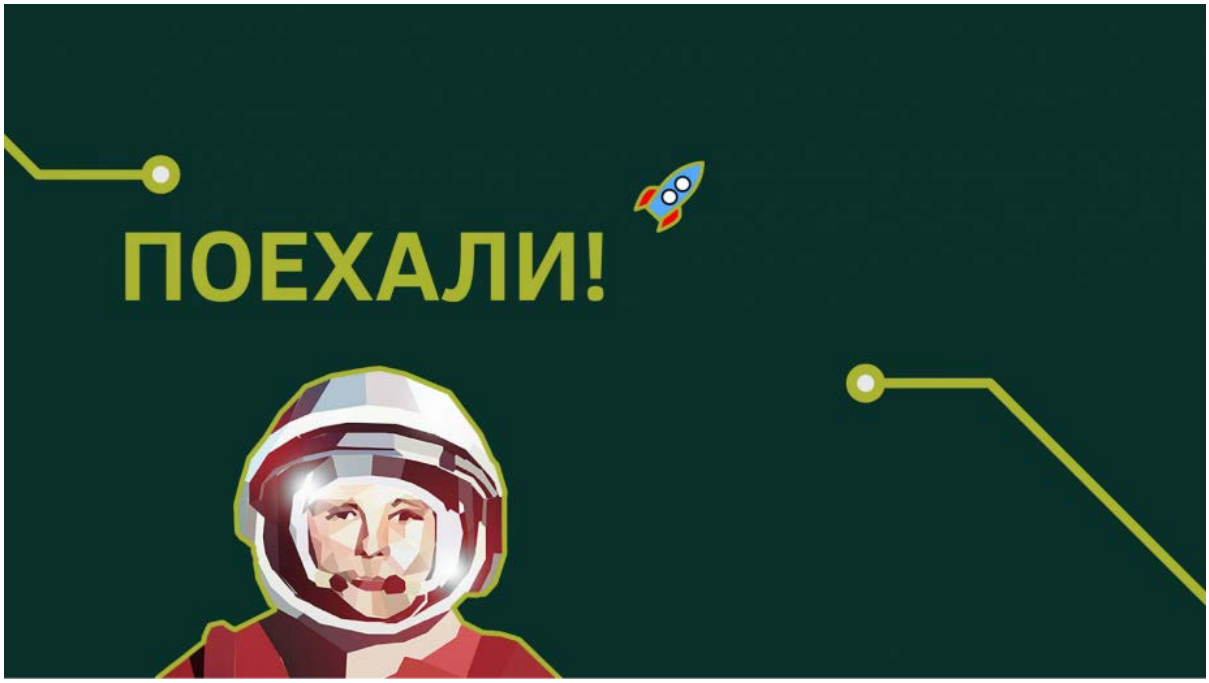


- **Устойчивость к отказам** – способность системы при отказе одного или нескольких серверов к продолжению работы в рамках требуемых параметров.
- **Устойчивыми к катастрофам** считаются системы, имеющие полное дублирующее резервирование (т.н. второе плечо) и способные без сильной просадки работать при полном отказе одного из дата-центров.

При этом катастрофоустойчивая система ≠ отказоустойчивая система. Ситуация, в которой катастрофоустойчивая, но не отказоустойчивая система продолжает работать только на одном «плече», вполне нормальна. Но если откажет один из серверов, система также выйдет из строя.

Теперь, когда мы разобрались с ключевыми понятиями и освежили актуальную терминологию, пора переходить непосредственно к азам оптимизации и лайфхакам.

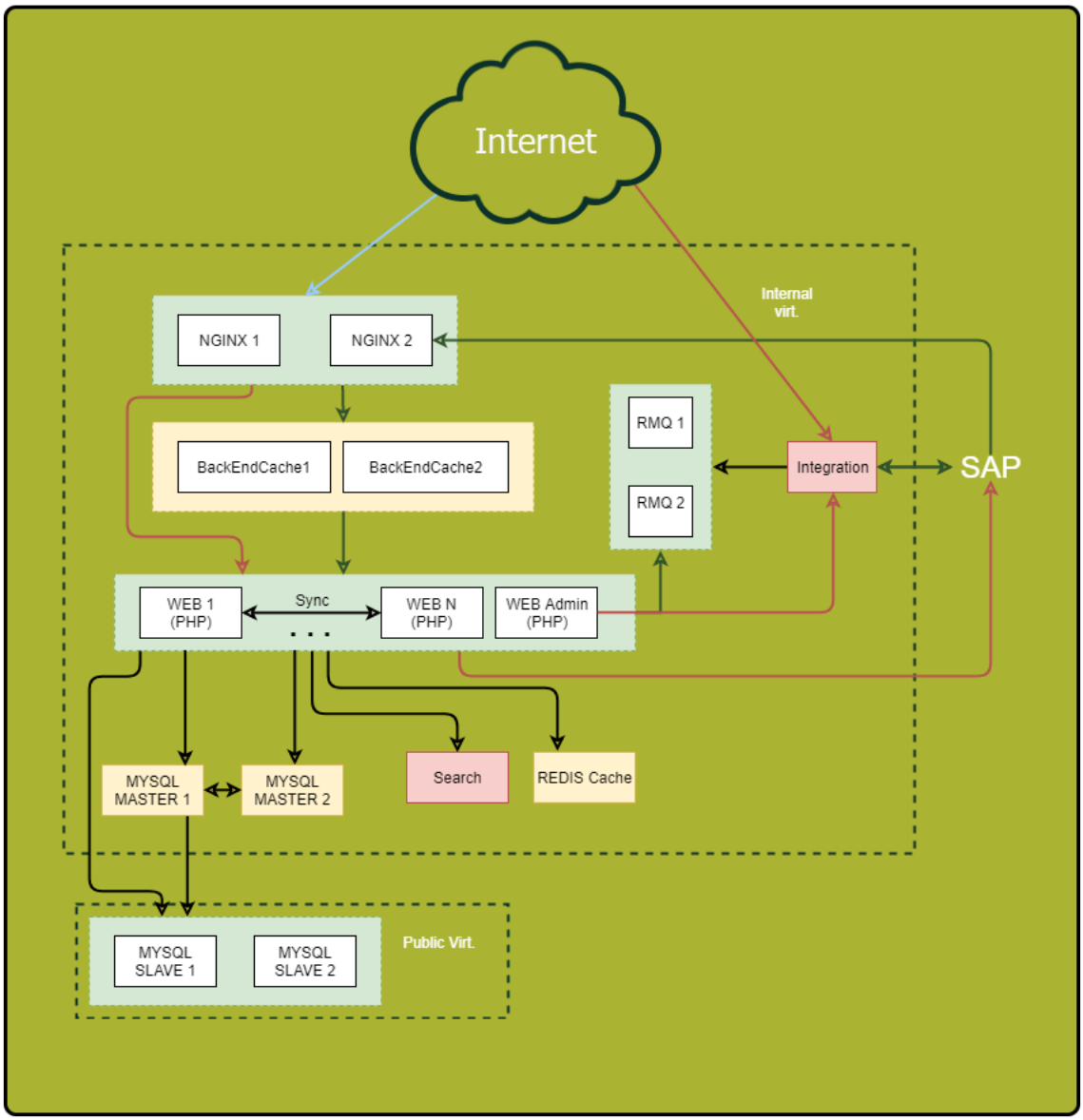
С чего начать оптимизацию?



Как понять, с чего начать оптимизацию? Прежде, чем вы броситесь оптимизировать, сделайте глубокий вдох и потратьте время на исследование работы приложения.

Обязательно нарисуйте подробную схему. Отобразите на ней все компоненты приложения и их взаимосвязи. Изучив эту схему, вы

сможете обнаружить ранее неприметные уязвимости и потенциальные точки отказа.



«Что? Где? Когда?» — оптимизируем запросы

Особое внимание уделите синхронным запросам. Напомню, это такие запросы, когда в одном и том же потоке мы отправляем запрос и ждем по нему ответа. Тут как раз кроются причины серьезных тормозов, когда на другой стороне что-то идет не так. Поэтому, если можете сократить число синхронных запросов или заменить их на асинхронные, сделайте это.

Вот маленькие хитрости, которые помогут вам отследить запросы:

- **Присваивайте каждому входящему запросу уникальный идентификатор.** В Nginx для этого есть встроенная переменная *$request_id*. Передавайте идентификатор в заголовках на back-end и пишите во все логи. Так вы сможете удобно трассировать запросы.
- **Логируйте не только конец запроса к внешнему компоненту, но и его начало.** Так вы измерите реальную продолжительность отработки внешнего вызова. Она может существенно отличаться от того, что вы видите в удаленной системе, например, из-за проблем с сетью или тормозов DNS.

Итак, данные собраны. Теперь разберем проблемные точки. Определите:

- Где тратится больше всего времени?
- Куда приходит наибольшее количество запросов?
- Куда приходят самые «долгие» запросы?

В итоге вы получите список наиболее интересных для оптимизации участков системы.

Совет: если какая-либо точка «собирает» множество мелких запросов, попробуйте объединить их в один большой запрос для сокращения накладных расходов. Результаты долгих запросов часто имеет смысл сохранить в кэш.

Кэшируем с умом

Существуют общие правила кеширования, на которые стоит опираться при оптимизации:

- **Чем ближе кэш к потребителю, тем быстрее работа.** Для приложения “ближайшим" местом будет оперативная память. Для пользователя — его браузер.
- **Кэширование ускоряет получение данных и снижает нагрузку на источник.**

Если десять веб-серверов делают одинаковые запросы к базе данных, централизованный промежуточный кэш, например в Redis, даст более высокий процент попаданий (по сравнению с

локальным кэшем) и снизит общую нагрузку на БД, что существенно улучшит общую картину.

Совет 1: Делайте компонентное кеширование готовой странички на стороне Nginx с помощью Edge Side Includes. Оно хорошо ложится на микросервисную/SOA архитектуру и разгружает систему в целом, значительно улучшая скорость отклика.

Совет 2: Следите за размером объектов в кэше, показателем hit ratio и объемами записи/чтения. Чем больше объект, тем дольше он будет обрабатываться. Если вы пишете в кэш чаще или больше, чем читаете, такой кэш — вам не товарищ. Его стоит или убрать, или подумать над повышением его эффективности.

Совет 3: Используйте собственные кэши баз данных там, где это возможно. Их правильное конфигурирование может качественно ускорить работу.

Профили нагрузки

Переходим к профилям нагрузки. Как вы знаете, есть два основных типа: OLAP и OLTP.

- **Для OLAP** (Online Analytical Processing) важно количество отработанного трафика в секунду.
- **Для OLTP** (Online Transaction Processing) ключевой показатель — скорость отклика, миллисекундные тайминги.

Чаще всего бывает эффективно разделить эти два вида нагрузки. Как минимум, вам понадобится раздельный тюнинг базы данных и, возможно, других компонентов системы.

Совет: Запросы на чтение из админки, как правило, обрабатываются по типу OLAP. Создайте под эту задачу отдельную копию БД и веб-сервер, чтобы разгрузить основную систему.

Базы данных

Итак, мы закономерно подошли к одному из самых сложных этапов оптимизации — а именно, к оптимизации базы данных.

Напомню вам общее правило: чем меньше объем базы, тем быстрее она работает. Сама организация базы данных имеет решающее значение, когда дело касается скорости.

По возможности храните **исторические данные**, логи приложения и **часто используемые** данные в разных базах данных. Еще лучше — разнесите их на разные сервера. Это не только облегчит жизнь основной БД, но и даст больше пространства для дальнейшей оптимизации, к примеру в ряде случаев позволит использовать разные индексы под разную нагрузку. Также “однотипность” нагрузки упрощает жизнь планировщику и оптимизатору запросов сервера БД.

И снова о важности планирования

Чтобы не ломать голову над оптимизацией там, где она не сильно нужна, выбирайте железо, исходя из задач.

- Под мелкие, но частые запросы лучше взять больше ядер процессора.
- Под тяжелые запросы – меньше ядер с более высокой тактовой частотой.

Постарайтесь поместить рабочий объем базы данных в оперативную память. Если это невозможно или имеет место большое количество запросов на запись пора посмотреть в сторону перенести базы данных на SSD-диски. Они дадут существенный прирост скорости работы с диском.

Масштабирование

Выше я описал ключевые механики повышения производительности приложения без увеличения его физических ресурсов.

Теперь мы поговорим о том, как выбрать стратегию масштабирования и повысить отказоустойчивость.

Существует два вида масштабирования системы:

- **Вертикальное** — рост объема ресурсов при сохранении количества сущностей;
- **Горизонтальное** — рост количества сущностей.

Растём в высоту

Начнем с выбора стратегии вертикального масштабирования.

Для начала рассмотрим **увеличение мощности системы**. Если ваша система работает в рамках одного сервера, придется сделать

выбор между повышением мощности текущего сервера или покупкой еще одного.

Может показаться, что первый вариант проще и безопаснее. Но дальновиднее будет докупить еще один сервер и бонусом к производительности получить большую отказоустойчивость. Об этом я говорил в начале статьи.

Если в вашей системе несколько серверов и стоит выбор — увеличить мощность существующих или докупить еще несколько, обратите внимание на финансовую сторону. Например, один мощный сервер может оказаться дороже, чем два на 50% «слабее». Поэтому резонно будет остановиться на втором компромиссном варианте. В то же время, при большом количестве серверов решающее значение имеет соотношение производительности, энергопотребления и стоимости полной стойки.

Растём в ширину

Горизонтальное масштабирование системы — это история про отказоустойчивость и кластеризацию. В общем случае, чем больше экземпляров одной сущности мы имеем, тем выше отказоустойчивость целого решения.

Вероятно, первое, что вам захочется масштабировать – это **серверы приложений**. Первое препятствие на этом пути – организация работы с централизованными источниками данных. Помимо баз данных, это еще и сессионные данные, и статический контент. Вот что я советую сделать:

- **Для хранения сессий** используйте Couchbase, а не привычный Memcached, так как он работает с тем же протоколом, но, в отличие от memcached, поддерживает кластеризацию.
- **Всю статику**, особенно большие объемы изображений и документов, храните отдельно и отдавайте с помощью Nginx, а не из кода приложения. Так вы сэкономите на потоках и облегчите управление инфраструктурой.

«Подтягиваем» базы данных

Сложнее всего масштабировать базы данных. Для этого есть две основные техники: шардирование и тиражирование. Рассмотрим их.

При **тиражировании** мы добавляем в систему полностью идентичные копии базы данных, при **шардировании** – логически разделенные части, шарды. При этом, шардирование крайне желательно проводить параллельно с репликацией (тиражированием) каждого шарда, чтобы не потерять отказоустойчивость.

Помните: зачастую кластер БД состоит из одной master-ноды, принимающей на себя поток записи, и нескольких slave-нод, используемых для чтения. С точки зрения отказоустойчивости, это немногим лучше одиночного сервера, так как общая отказоустойчивость определяется наименее устойчивым элементом системы.

Схемы с более, чем двумя мастерами баз данных (топология «кольцо») без подтверждения записи на каждом из серверов, очень часто страдают от неконсистентности. В случае сбоя одного из серверов восстановить логическую целостность данных в кластере будет крайне затруднительно.

Совет: Если в вашем случае не рационально иметь несколько мастер-серверов, предусмотрите архитектурную возможность работы системы без мастера хотя бы в течение часа. В случае аварии это даст вам время на замену сервера без простоя всей системы.

Совет: Если есть необходимость держать более 2-х мастеров баз данных, рекомендую вам рассмотреть NoSQL-решения, так как многие из них имеют встроенные механизмы приведения данных в консистентное состояние.

В погоне за отказоустойчивостью ни в коем случае не забывайте, что репликация страхует вас **только от физического отказа сервера**. Она не спасет от логической порчи данных из-за ошибки пользователя.

Помните: Любые важные данные необходимо бэкапить и хранить в виде независимой не редактируемой копии.

Как увеличить скорость работы веб-ресурса — EdgeЦентр на vc.ru

EdgeЦентр

Разбираемся, от чего зависит скорость работы веб-приложений и как её сократить.

Скорость работы веб-ресурса — один из самых важных факторов и с точки зрения клиентского опыта, и с точки зрения SEO-продвижения.

По многочисленным исследованиям, более 50% пользователей закрывают страницу, если она грузится дольше 3 секунд. Помимо ухудшения клиентского опыта, это увеличивает показатель отказов и понижает позиции сайта в поисковой выдаче. Кроме того, многие поисковики сейчас учитывают скорость загрузки веб-ресурса и отдают предпочтение более быстрым сайтам.

О чём пойдёт речь:

- От чего зависит скорость работы веб-ресурсов
- Как сократить время ответа сервера
- Как увеличить скорость загрузки контента
- Как оптимизировать отрисовку страницы

От чего зависит скорость работы веб-ресурсов

Скорость загрузки сайта зависит от множества разных факторов. В этой статье мы разберём три основных:

- Время ответа сервера.
- Особенности контента и скорость его загрузки.
- Отрисовка страницы.

Время ответа сервера

Когда мы заходим на какой-то ресурс, перед тем как отобразить контент, браузер посылает HTTP-запрос на сервер, а тот

отправляет ответ. Время между отправкой запроса и получением ответа и называется временем ответа сервера.

Что такое время ответа сервера

Ещё одно название для этого понятия — время получения первого байта (Time To First Byte, или TTFB).

Если говорить с позиции клиента, то это время между моментом, как он зашёл на ресурс, и моментом, когда ему начала загружаться страница.

От чего зависит время ответа сервера:

- **Нагрузка на сервер.** Сколько пользователей заходит на ваш ресурс и какое количество запросов приходит одновременно. Чем их больше, тем медленнее сервер будет их обрабатывать.
- **Характеристики сервера.** Достаточно ли он мощный, чтобы справляться с нагрузкой.
- **Расположение сервера относительно ваших пользователей.** Чем дальше он от них, тем дольше будут идти запрос и ответ.
- **Работа с базами данных.** Если ваше приложение взаимодействует с базой данных, БД тоже может быть причиной задержек, если она расположена на сервере, у которого недостаточно ресурсов, или имеет не подходящие для вашей нагрузки настройки.

Особенности контента и скорость его загрузки

Здесь всё просто: чем тяжелее ваш контент, тем дольше он будет загружаться. Поэтому чем меньший размер имеют элементы вашего веб-ресурса, тем лучше.

Но здесь важно найти баланс между размером файла и его качеством. Картинка, которая весит несколько килобайт, конечно, будет загружаться быстрее. Но если при этом она будет выглядеть размытой или элементы на ней будет трудно различить, это может

испортить впечатление о вашем сервисе и снизить количество продаж.

Отрисовка страницы

Ваше веб-приложение — это картинки, видео, текстовый контент, различная графика и так далее. Когда ресурс загружается, все эти элементы собираются воедино и отображаются на экране у пользователя. Это и называется отрисовкой страницы.

Обычно именно скорость отрисовки страницы проверяют различные сервисы для проверки скорости загрузки — например, популярный Page Speed Insight.

От чего зависит скорость отрисовки страницы:

- Количество элементов.
- В каком порядке они загружаются (самые важные должны отображаться в первую очередь).
- Вёрстка страницы.

Как сократить время ответа сервера

Оптимизировать работу с базами данных

Чтобы отобразить нужный контент на странице, сервер обращается к базе данных, на которой этот контент хранится. Если работа с БД не оптимизирована, это может приводить к увеличению времени загрузки.

Это особенно актуально, если речь идёт о динамическом контенте (соцсети, форумы, список рекомендаций в интернет-магазинах и любой другой контент, который не хранится в готовом виде, а создаётся после получения запроса). В этом случае нужно не только «забрать» файлы, но и сформировать ту информацию, которую должен увидеть клиент.

Увеличить скорость работы с БД — задача для опытных программистов. Самое основное, что нужно сделать в первую очередь:

- **Оптимизировать запросы к базе данных.** Используйте команду EXPLAIN в своих БД, чтобы понять, какой запрос работает медленно, и придумать, как его можно ускорить.
- **Кешировать ответы на самые частые, одинаковые запросы.** Это особенно нужно в случае с динамическими сайтами. Вместо того чтобы каждый раз формировать контент заново, БД может отдавать уже сформированный материал из кеша.
- **Настроить индексы базы данных.** Поиск данных по таблицам должен идти по индексируемым полям.

Подключить CDN

Чем дальше сервер находится от пользователей, тем дольше будут идти запрос и ответ. Если все ваши клиенты сосредоточены в одном месте, никакой проблемы нет — вы просто покупаете сервер в том же районе. Например, если вся ваша аудитория живёт в Новосибирске, ваш сервер тоже должен быть в Новосибирске.

Но если мы говорим об онлайн-бизнесе, то чаще всего бывает так, что аудитория рассредоточена по разным городам и странам. И поставить серверы в каждый город просто невозможно.

Эту проблему решает CDN (Content Delivery Network) — сеть доставки контента. Это множество связанных между собой серверов (точек присутствия), которые забирают информацию с сервера-источника, кешируют её и отдают пользователям.

CDN-серверы обычно располагаются как можно ближе к конечным пользователям. И когда кто-то из них заходит на ваш ресурс, запрос идёт не к серверу-источнику, а к ближайшей точке присутствия. Путь запроса сокращается, и время ответа сервера уменьшается.

При использовании CDN информация кешируется в точках присутствия после первого запроса:

- Допустим, ваш сервер-источник располагается в Санкт-Петербурге, а клиенты живут по всей России и в странах СНГ. Вы подключаете CDN с точками присутствия во всех этих регионах.

- Когда кто-то из пользователей, например, из Хабаровска первый раз заходит на ваш ресурс, запрос идёт к серверу-источнику.
- Но информация, полученная от источника, сохраняется на ближайшем к пользователю CDN-сервере.
- Когда следующий юзер из Хабаровска зайдёт на ваш ресурс, запрос пойдёт уже не к источнику, а к ближайшей точке присутствия, в которой есть требуемая информация.

CDN влияет не только на уменьшение времени ответа сервера, но и на его отказоустойчивость. При всплесках трафика запросы равномерно распределяются между ближайшими серверами, а не идут всем скопом на источник. Это помогает балансировать нагрузку и делает работу ресурса более стабильной.

Если у вас мировой проект, для быстрой работы сайтов CDN должна быть по-настоящему глобальной. Обязательно учитывайте количество точек присутствия и их расположение. В первую очередь они должны быть там, где живут ваши клиенты. И чем их больше, тем меньше будет время ответа и тем более стабильной будет система.

У EdgeCDN десятки точек присутствия по всей России и в СНГ, и огромная партнёрская сеть на всех континентах, кроме Антарктиды. Благодаря отличному покрытию мы обеспечиваем среднее время отклика по миру в пределах 30 мс. Суммарная ёмкость сети составляет уже более 110 Тбит/с.

Мощная, распределённая по миру CDN может значительно сократить время ответа сервера. Но учитывайте, что поможет она только в случае, если ваши клиенты действительно живут далеко от сервера. Если это не так, а показатель времени ответа сервера оставляет желать лучшего, проблема в чём-то другом, и CDN не спасёт ситуацию.

Уменьшить нагрузку на сервер

Чтобы уменьшить нагрузку, первым делом можно попробовать сократить количество запросов.

При загрузке сайта браузер делает запросы на каждый его элемент. Получается, если мы уменьшим количество этих элементов, мы сократим и количество запросов.

Чтобы при этом внешних вид страницы не изменился, сделать можно следующее:

- Использовать CSS-спрайты — комбинированные изображения, содержащие в себе несколько маленьких картинок. Обычно в один спрайт объединяют фоновые изображения, элементы интерфейса (например, иконки, кнопки), картинки, имеющие одну цветовую палитру.
- Использовать inline-изображения — изображения, которые используют схему data: URL (изображение включается в сам HTML-файл).
- Объединить несколько файлов в один: например, несколько CSS-файлов или JS-файлов.

CSS-спрайты в некоторых случаях могут даже уменьшить объём картинок и ускорить загрузку контента. Но вот inline-изображения увеличивают объём HTML-файла. Объединённые файлы тоже получаются большими. Поэтому здесь выходит палка о двух концах: вы сокращаете количество запросов и уменьшаете время ответа сервера, но вместе с тем утяжеляете контент.

Если у вас много тяжёлых файлов, с помощью объединения вы уменьшите нагрузку, но существенного ускорения работы ресурса добиться не удастся.

Есть другой, более эффективный способ уменьшить нагрузку на сервер — подключить шилдинг источника. Это дополнительный прекеш-сервер, который располагается между точками присутствия и источником.

Основная функция шилдинга — защита вашего сервера от слишком большого количества запросов. Все запросы от CDN-серверов приходят сначала на шилдинг и только потом идут на источник. Получается, последний взаимодействует только с одним прекеш-сервером, а не со всей сетью.

Количество запросов снижается, они обрабатываются быстрее, и время ответа сервера сокращается.

Шилдинг источника предоставляют многие продвинутые CDN-провайдеры. У EdgeCDN он тоже есть.

Разместить доменные имена на ближайших к клиентам DNS-серверах

Чтобы отправить запрос к серверу, на котором расположен ваш ресурс, браузеру сначала нужно получить его IP-адрес. Информация о доменных именах и соответствующих им IP-адресах хранится на DNS-серверах.

Получается, перед тем как отправить запрос на сервер, браузер сначала посылает запрос на DNS-сервер, чтобы выяснить нужный ему IP. Подробнее о том, как всё это работает, можно почитать в нашей статье «DNS-сервер: что это и как работает?».

Чем дальше от ваших пользователей находится DNS-сервер, на котором размещены ваши доменные имена и их IP, тем дольше будет идти запрос. А следовательно, и время ответа сервера будет больше.

Как лучше поступить? Разместить свои доменные имена на DNS-серверах как можно ближе к вашим клиентам. И выбрать надежный DNS-хостинг с большим количеством локаций и возможностью балансировать нагрузку.

Сменить сервер

Если вы перепробовали всё, а время ответа сервера по-прежнему оставляет желать лучшего, возможно, причина в самом сервере. Может быть, он просто не справляется с нагрузкой и вам нужно больше вычислительных мощностей.

Посчитайте, какое количество дискового пространства и оперативной памяти требуется для нормальной работы вашего ресурса и соответствует ли сервер этим параметрам.

Если нет, смените тариф хостинга на более мощный или перейдите к другому провайдеру.

А если ваш проект разросся до действительно больших масштабов, возможно, ресурсов обычного хостинга вам будет уже недостаточно и пора мигрировать в облако. О различиях этих двух

сервисов вы можете почитать в нашем блоге, в статье «Облако или хостинг: что выбрать?».

Как увеличить скорость загрузки контента

Чем меньше весят элементы вашего приложения, тем быстрее оно будет загружаться. Чтобы добиться ускорения ресурса, нужно по максимуму уменьшить размер всех файлов, из которых он состоит.

Но при уменьшении размера не должно сильно ухудшаться качество.

Чтобы ускорить загрузку и при этом сохранить высокое качество контента, используют современные алгоритмы сжатия — Gzip, Brotli и WebP.

Это алгоритмы сжатия без потерь. Они существенно уменьшают размер элементов, но при этом никак не изменяют их внешний вид. Алгоритмы умеют сжимать файлы «на лету», то есть прямо в процессе отдачи контента пользователям.

Как работает сжатие «на лету»

Gzip умеет сжимать только текстовые элементы. Алгоритм находит в файле одинаковые строки и объединяет их. За счёт этого страница уменьшается на 60–70%.

Brotli работает с любым контентом. Он использует уже встроенный в браузеры пользователей словарь. В этом словаре хранится почти 100 000 фраз и фрагментов, которые чаще всего встречаются в интернете. Алгоритм вычисляет новые фрагменты в файлах и передаёт только их, а всё остальное заменяет элементами из словаря.

Такая технология позволяет сжимать контент на 20–25% эффективнее, чем Gzip.

WebP — алгоритм для сжатия изображений. Сейчас его используют как эффективный аналог PNG и JPEG. WebP сжимает изображения без потерь на 26% лучше, чем PNG, и на 25–34% эффективнее JPEG.

Сжатие происходит в два этапа:

1. Алгоритм пытается предсказать содержание блока по уже декодированным.
2. Кодирует и исправляет ошибки предсказания.

Все три алгоритма поддерживаются всеми популярными браузерами. Их использование — отличный способ уменьшения времени загрузки.

EdgeCDN не только поддерживает эти алгоритмы, но и умеет сжимать файлы. В процессе отдачи контента Gzip работает на серверах, а Brotli — на шилдинге источника. Это значит, что вам не нужно настраивать сжатие и прописывать код на своей стороне. Вы можете загрузить исходники на источник, а при доставке контент будет сжат автоматически.

Как оптимизировать отрисовку страницы

Поместите javascript-файлы в конец страницы. Мы уже упоминали, что при загрузке страницы самые важные элементы должны загружаться в первую очередь. Так вот, JS-файлы к важным элементам не относятся.

В первую очередь у пользователя должен отображаться контент, чтобы он видел содержание страницы ещё до того, как она загрузилась полностью. А javascript-файлы могут весить довольно много. Поэтому лучше всего сделать так, чтобы они загружались в самом конце.

Скройте элементы, не нужные для мобильной версии сайта. На многие веб-ресурсы пользователи сейчас заходят со смартфонов. Мобильные устройства слабее стационарных ПК. А если к этому прибавляется ещё и нестабильный мобильный интернет, время загрузки может сильно увеличиться.

Чтобы не заставлять пользователей ждать, максимально оптимизируйте мобильную версию вашего ресурса и удалите из неё всё ненужное, что может замедлять работу.

Подведём итоги

- Более 50% пользователей закрывают страницу, если она грузится дольше 3 секунд, а поисковики в выдаче отдают предпочтение быстрым сайтам. Поэтому очень важно сделать так, чтобы ваш веб-ресурс работал быстро.
- Скорость работы зависит от времени ответа сервера, скорости загрузки контента, отрисовки страницы и других факторов.
- Для сокращения времени ответа сервера можно оптимизировать работу с базами данных, сократить количество запросов, разместить доменные имена на DNS-серверах, близких к вашим клиентам, и подключить CDN.
- Чтобы контент грузился быстрее, нужно максимально уменьшить размер файлов. Отличное средство для этого — алгоритмы сжатия Brotli, Gzip и WebP.
- Чтобы увеличить скорость отрисовки страницы, поместите JS-файлы в конец страницы и скройте элементы, не нужные для мобильной версии.

CDN влияет на ускорение сайта и его отказоустойчивость. Это отличный способ оптимизировать работу ваших ресурсов, особенно если сеть поддерживает удобные функции: например, шилдинг источника и сжатие Brotli, Gzip и WebP. Такая CDN ещё и уменьшит нагрузку на сервер и поможет оптимизировать контент.

Все эти функции есть у EdgeCDN.