ПРОБЛЕМЫ СКОРОСТИ ЗАГРУЗКИ ВЕБ-РЕСУРСОВ НА СТОРОНЕ КЛИЕНТА: КЛАССИФИКАЦИЯ И МЕТОДЫ РЕШЕНИЯ

Н.С. Мациевский

Московский физико-технический институт (государственный университет)
141700, Московская область, г. Долгопрудный, Институтский пер., д. 9

Аннотация. В данной статье рассматриваются вопросы, связанные с текущим состоянием и производительностью веб-ресурсов в современном Интернете. Делается аспект на вопросах производительности, связанных с особенностями работы пользовательских агентов (браузеров), а также на существующих методах решения определенного ряда проблем, возникающих вследствие использования браузерами тех или иных ограничений при загрузке веб-ресурсов. В статье предлагается простой алгоритм, позволяющий принять решение относительно любого аспекта клиентской производительности загрузки веб-ресурсов.

Annotation. This article is concerned about client side issues of web resources load process related to user agents (browsers) behavior. A lot of modern problems with current load algorithms are investigated and all known solutions with their area or efficiency are compared. Also a simple way to make decision about every aspect of client side productivity of web resources is offered with detailed explanations.

Введение

В современном мире Интернет играет, пожалуй, роль основной среды для распространения информации. Отправка электронных писем, поиск информации на интересующие темы, чтение новостных сводок, общение с друзьями и коллегами – вот лишь малая доля того, ради чего мы используем Интернет.

Каждое наше действие в Интернете затрагивает многочисленные технические аспекты сетевых соединений и передачи данных. При текущих скоростях доступа в Интернет можно подумать, что любой веб-ресурс работает быстро или что скорость его работы зависит только от скорости подключения (самого ресурса или конечного пользователя). Однако, по данным последних исследований [1] рост размера страницы среднего веб-ресурса лишь немногим уступает росту пропускной способности каналов доступа. А если учесть, что с каждым годом расслоение пользователей по скорости подключения к Интернету только усиливается, то ситуация принимает уже критический характер: ведь для обеспечения высокой скорости загрузки для 90% пользователей нужно использовать более прогрессивные и технологичные методы.

Клиентская архитектура и ее отличия от серверной

Важность клиентской архитектуры в настоящее время невозможно переоценить, потому что подавляющая часть вопросов по ускорению загрузки веб-ресурсов связана именно с клиентской частью. В стремлении создать удобный, быстрый и кроссбраузерный веб-ресурс современный архитектор клиентской части должен решить массу проблем, согласовать видение заказчика с удобством для пользователей и обязательно учесть, как веб-ресурс (или целый портал) будет развиваться в дальнейшем.

Согласно прошлогоднему исследованию [2], проведенному инженерами Yahoo! в области пользовательских интерфейсов, 95% времени при загрузке веб-ресурса связано с задержками на стороне конечного пользователя. И только 5% приходится на «серверную» составляющую (которая включает, кроме ожидания ответа, собственно, от сервера, еще и время на DNS-запрос, время на установление TCP/IP-соединения и ряд

других издержек). Именно поэтому оптимизация времени загрузки страницы является одной из первостепенных задач

При этом каждый проблемный вопрос, будь то использование стандартов при верстке страницы, комбинирование фоновых изображений для уменьшения числа запросов к серверу, применение JavaScript-логики на странице, должен решаться, в первую очередь, исходя не только из фактического технического задания, но и максимальной производительности на стороне браузера.

Давайте рассмотрим, с какими проблемами можно столкнуться при создании высокопроизводительных веб-ресурсов, и каким образом их лучше всего решать.

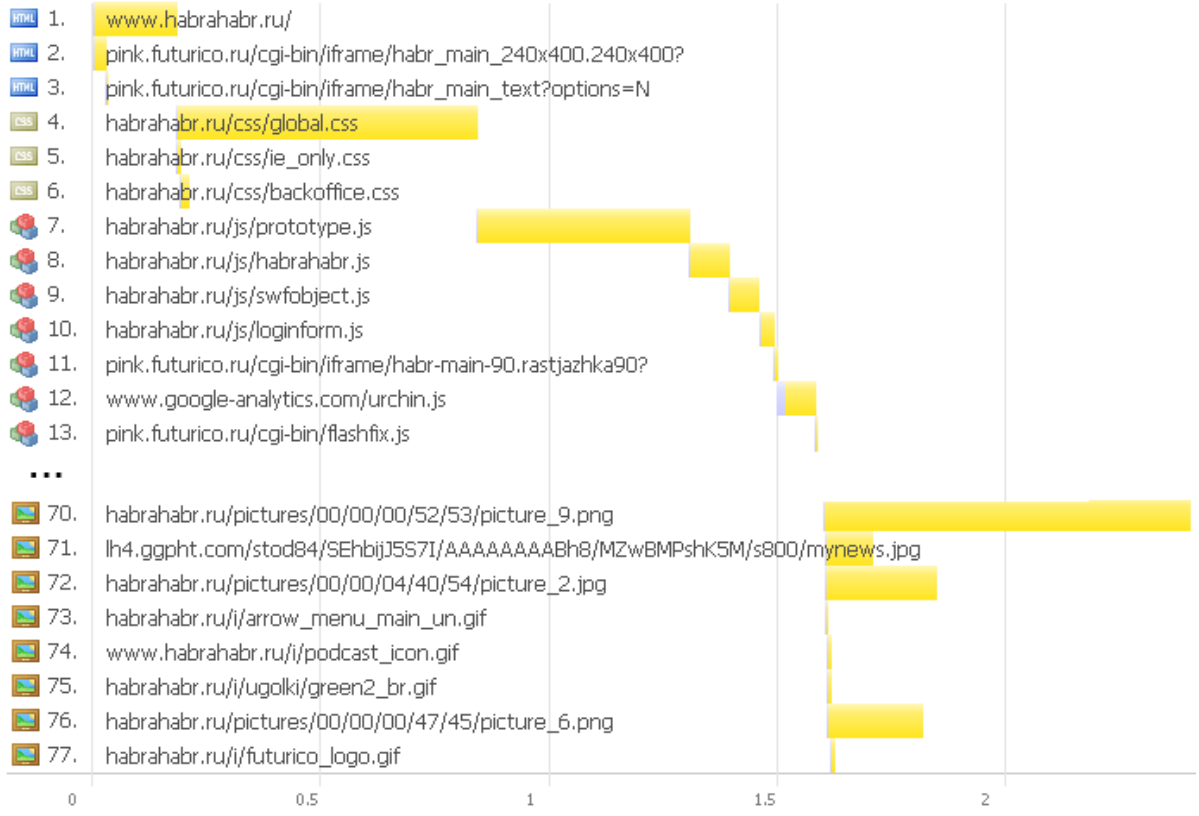

Рисунок 1. Ожидание и загрузка HTML-файла составляет малую долю от времени загрузки всей страницы

Компромиссы во всем

Если говорить о проблемах, связанных с клиентской производительностью, то стоит сразу упомянуть о неоднозначности подхода к решению любой проблемы, лежащей в этой области. При создании любого внешнего модуля (страницы или ее части) клиентскому архитектору приходится делать выбор. Либо на этой странице будет использовать своя таблица стилей (тогда для ускорения загрузки возможно ее включить в итоговый HTML-документ). Либо на ней будет использован общий стилевой файл, тогда нужно не забыть о кэшировании и оценить число постоянных пользователей.

Компромиссы преследуют архитектора повсюду: объединять все файлы в один или разбить на несколько независимых модулей? Кэшировать ли отдельные ресурсные файлы, необходимые для отображения страницы, или включить их в сам документ? Какие набор браузеров необходимо поддерживать, и какие приемы при этом использовать? Какую палитру, формат и степень сжатия использовать для изображений, и как наиболее оптимально сложный рисунок разбить на несколько составляющих? Для каких страниц возможно использование методов, осуществляющих загрузку ресурсов после загрузки самой страницы, а для каких нужна предварительная загрузка?

Вопросов очень много, и бóльшая их часть завязана на знании основ клиентской оптимизации.

Психологические аспекты загрузки веб-страницы

Соответствующее исследование продемонстрировало, что пользовательское раздражение сильно возрастает, если скорость загрузки страницы превышает 8-10 секунд безо всякого уведомления пользователя о процессе загрузки [3]. Последние работы в этой области показали, что пользователи с широкополосным доступом еще менее терпимы к задержкам при загрузке веб-страниц по сравнению с пользователями с более узким каналом. В опросе, проведенном JupiterResearch [4], было установлено, что

33% пользователя скоростного соединения не хотят ждать более 4 секунд при загрузке страницы, при этом 43% пользователей не ждут более 6 секунд.

В исследовании, проведенном в 2004, Fiona Nah установила, что терпимое время ожидания (ТВО) для неработающих ссылок (без обратной связи) находилось между 5 и 8 секундами [5]. С добавлением уведомления пользователя о процессе загрузки (обратной связи), например, индикатора загрузки, ТВО увеличилось до 38 секунд. Распределение ТВО для повторных попыток зайти на неработающие ссылки имело максимум в районе 2-3 секунд (без обратной связи). Nah заключила, что ТВО для веб-пользователей имеет максимум около 2 секунд. Если учесть стремление пользователя посетить веб-ресурс повторно, то Dennis Galletta и др. показали, что кривая сглаживается при 4 и более секундах и уходит в нуль в районе 8 и более секунд [6].

Основные задачи клиентской оптимизации

В качестве основных проблемных мест при загрузке страницы любого веб-ресурса можно выделить четыре ключевых момента:

1. **Предзагрузка** – появление страницы в браузере пользователя. После некоторого момента ожидания загрузки при заходе на веб-ресурс у пользователя в браузере возникает нарисованная страница. В этот момент, скорее всего, на странице отсутствуют рисунки и, возможно, не полностью функционирует JavaScript-логика.
2. **Интерактивная загрузка** – появление интерактивности у загруженной веб-страницы. Обычно вся клиентская логика взаимодействия доступна сразу после первоначальной загрузки страницы (стадия 1), однако, в некоторых случаях (о них речь пойдет чуть дальше) поддержка этой логики немного запаздывает по времени от появления основной картинки в браузере пользователя.
3. **Полная загрузка** страницы. Страница веб-ресурса полностью появилась в браузере, на ней представлена вся заявленная информация, и она готова к дальнейшим действиям пользователя.
4. **Пост-загрузка** страницы. На данной стадии полностью загруженная страница может (в невидимом для пользователя режиме) осуществлять загрузку и

кэширование некоторых ресурсов или компонентов. Они могут потребоваться пользователю при переходе на других страницы данного веб-узла или для отображения (но не функционирования логики) каких-либо интерактивных эффектов.

Для большинства веб-ресурсов на данный момент стоит различать только предзагрузку (в которую по умолчанию включается интерактивная загрузка) и полную загрузку страницы. Пост-загрузка, к несчастью, сейчас используется крайне мало.

Оптимизация скорости загрузки веб-страницы сосредоточена на двух ключевых аспектах: ускорение предзагрузки и ускорение основной загрузки. Все основные методы сфокусированы именно на этом, потому что именно эти две стадии воспринимаются пользователем как «загрузка» веб-страницы.

Эффективность основных методов

Если описать все оптимизационные методы, то они разбиваются на 4 основные группы:

1. Уменьшение объема представляемых данных (сюда относится использование алгоритмов сжатия и соответствующих форматов представления изображений).
2. Кэширование (использование парных клиент-серверных заголовков для уменьшения времени передачи информации при ее сохранении ее актуальности).
3. Уменьшение числа ресурсов (различные методы объединения загружаемых файлов)
4. «Визуальная» оптимизация (к которой относятся алгоритмы разделения процесса загрузки на 4 стадии для максимального ускорения загрузки основных стадий, а также ряд методов, связанных с параллельными потоками загрузки).

При уменьшении объема данных наиболее действенных будет архивирование (gzip/deflate) на сервере. Практически все современные гиганты Интернет-индустрии сейчас отдают текстовые файлы в gzip-формате (это и GoogLe, и Yahoo!, и Yandex).

Существуют определенные проблемы с восприятием таких файлов некоторыми браузерами, однако, почти все они на данный момент преодолимы. Данный подход наиболее простой для применения, и поэтому имеет наибольшую эффективность: минимум действий приводит к максимуму результата

Кэширование также не требует глубоких знаний сетевых протоколов и тонкостей верстки, но способно (при больших количествах постоянных посетителей) в значительной степени повлиять на скорость загрузки страницы конкретно для них.

Далее по эффективности обычно используют объединение как текстовых (.html, .css, .js) файлов, так и графических, используемых в оформительских целях. Текстовые файлы объединять очень просто, при этом мы можем сэкономить значительное время, уходящее на дополнительные запросы к серверу. При объединении и графических файлов обычно используют технологии CSS Sprites (Image Map), которую не так легко автоматизировать, но при больших количествах иконок на странице она способна значительно ускорить загрузку.

К методам «визуальной» оптимизации можно отнести как экстремальную оптимизацию: когда все сопутствующие файлы включены в итоговый – так и распределение нагрузки по загрузке файлов по нескольким серверам. На реальном времени отображения страницы эти действия сказываются не сильно, в некоторых случаях их внедрение сопряжено со значительными трудностями. Однако, в случае высоконагруженных проектов, даже несколько миллисекунд могут сказаться на значительных (в абсолютном смысле) приростах прибылей [7].

Основным критерием, который должен определять, какие методы и в каком объеме стоит применять, естественно, будет аудитория ресурса. Для каждого веб-ресурса можно выделить несколько характерных групп страниц, которые посещаются тем или иным типом аудитории. В первую группу войдут веб-страницы, на которые заходят всякий раз новые пользователи. Это и специальные рекламные страницы, задачей которых является прямая продажа продукта. Это и страницы объявлений, которые пользователи должны увидеть один, максимум, два раза. И т.д. Аудитория таких страниц на 99,9% состоит из новых пользователей. Следовательно, для них нужно применять методы, снижающие, в первую очередь, число обращений к серверу для показа страницы: объединение файлов и экстремальную оптимизацию.

Ко второй группе можно отнести страницы, аудитория которых также часто меняется, однако, часть ее может просматривать страницы неограниченное число раз. Для таких страниц можно выделить характерное ядро постоянных посетителей, однако, оно составляет не более 30-40% от общего числа. Большинство веб-ресурсов, которые «живут» на поисковом трафике, являются замечательным примеро, полностью подходящих под эту группу. Для таких страниц, в первую очередь, стоит рассмотреть методы уменьшения числа запросов (CSS Sprites), также возможную минимизацию всех текстовых файлов (HTML, CSS, JavaScript). Однако, применение кэширование в данном случае оправдано в меньшей степени, так как уменьшит время загрузки страницы не так сильно (если брать средневзвешенное значение), чем, например, параллельные запросы.

Наконец, к третьей группе относятся все остальные страницы, а именно те, аудитория которых в большей степени постоянна (конкретные числа стоит рассматривать из параметров бизнес-эффективности различных групп аудитории, однако, характерные значения здесь – это 30% постоянных пользователей ресурса). В этой группе наиболее действенным будет, конечно же, кэширование и оптимизация скорости работы JavaScript и Flash-анимации – ведь именно она будет «съедать» больше всего времени.

Методы сжатия

Основными инструментами для уменьшения объема данных являются разнообразные минимизаторы и обфускаторы (для JavaScript-файлов), архивирование и также ряд утилит для уменьшения размера изображений. Давайте рассмотрим их по порядку. Как показало проведенное тестирование средств сжатия CSS, лучше всего с этой задачей справляется проект CSS Tidy [8] (примерно на одном уровне с ним идет YUI Compressor [9]), который вместе с дополнительным архивированием файлов позволяет получить выигрыш до 85% [10].

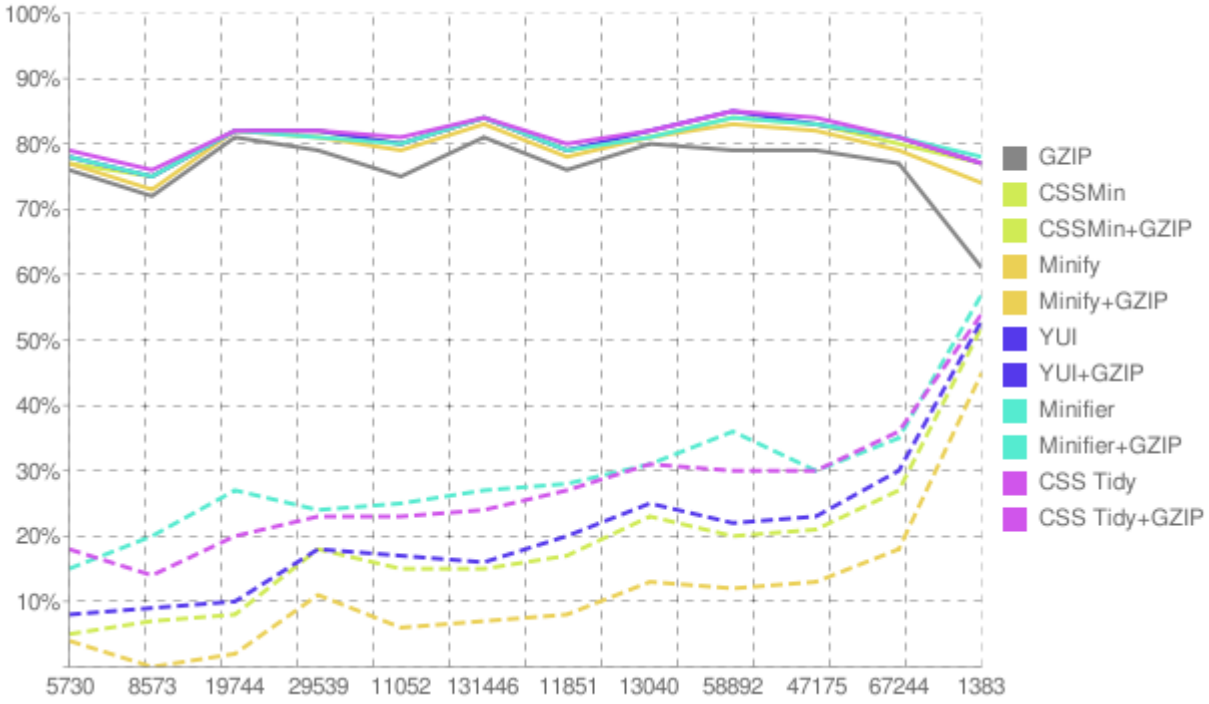

Рисунок 2. Зависимость выигрыша для сжатия CSS-файлов при использовании различных инструментов и архивирования

Для JavaScript-файлов ситуация несколько интереснее [11]. Если применять архивирование, то лучше всего использовать YUI Compressor [9], так как он, в среднем, сжимает в этом случае лучше. Если архивирование для JavaScript-файлов применять нельзя, то лидером в сжатии является Dean Edwards Packer [12], однако, он вносит дополнительные издержки на свою «распаковку». Как показали исследования, для пользователей, которые будут, в основном, загружать JavaScript из кэша, лучше использовать сжатие без обфускации.

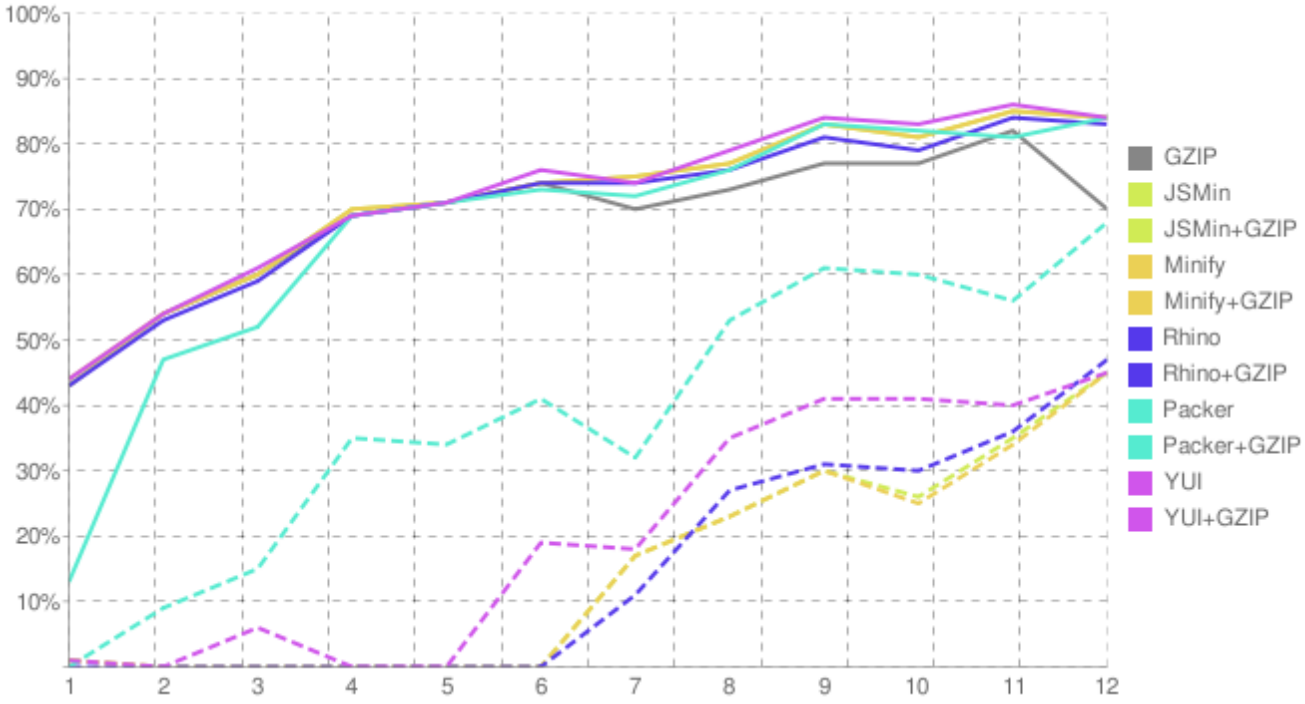

Рисунок 3. Зависимость выигрыша для сжатия JavaScript-файлов при использовании различных инструментов и архивирования

Использование архивирования через mod_gzip или mod_deflate для веб-сервера Apache (и соответствующих модулей для других веб-серверов) способно существенно уменьшить размер загружаемых файлов. Однако, в случае очень быстрого канала у пользователей (например, локальный ресурс) и ограниченных ресурсов сервера (высокая удельная нагрузка на создание страницы) будет разумнее сжатие не использовать [13].

Также стоит для архивированных файлов добавить соответствующие заголовки (Cache-Control:private), чтобы избежать ряда проблем с локальными кэширующими прокси-серверами. Для архивирования CSS- и JavaScript-файлов также нужно исключить Safari (для Windows-платформ) и Konqueror из числа тех браузеров, которым можно отправлять gzip-файлы: эти браузеры до последнего времени не умели их корректно распознавать.

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    AddEncoding gzip .gz
    RewriteCond %{HTTP:Accept-encoding} !gzip [OR]
    RewriteCond %{HTTP_USER_AGENT} Safari [OR]
    RewriteCond %{HTTP_USER_AGENT} Konqueror
    RewriteRule ^(.*)\.gz(\?.+)?$ $1 [QSA,L]
</IfModule>

<IfModule mod_headers.c>
    Header append Vary User-Agent
    <FilesMatch .*\.js.gz$>
      ForceType text/javascript
      Header set Content-Encoding: gzip
      Header set Cache-control: private
    </FilesMatch>
    <FilesMatch .*\.css.gz$>
      ForceType text/css
      Header set Content-Encoding: gzip
      Header set Cache-control: private
    </FilesMatch>
</IfModule>
```

Рисунок 4. Пример конфигурации веб-сервера Apache для включения сжатия CSS-и JavaScript-файлов

Для большинства графических элементов рекомендуется использовать формат PNG, так как он более экономичен, чем GIF [14] в представлении графиков и рисунков с ограниченной цветовой палитрой. Однако, для небольших изображений GIF-формат может оказаться лучше. На данный момент PNG-изображения поддерживаются, практически, всеми браузерами.

Сейчас существует проблема с поддержкой альфа-канала в Internet Explorer (которую обещают исправить в 8 версии), однако, в 6 и 7 версии она решается через фильтр ImageAlphaLoader, что позволяет использовать полупрозрачность, практически, в полном объеме.

В случае проблем с совпадением цветом (снова в Internet Explorer) рекомендуется удалить из изображения gAMA-чанков, отвечающих за прозрачность (в таком случае получить прозрачное изображение с совпадающими цветами в Internet

Explorer не получится). В этом также может помочь ряд утилит, уменьшающих размер PNG-изображений: например, pngcrush. Для уменьшения размера JPEG-изображений можно применить утилиту jpegtran, которая не затрагивает цветовые данные изображения, а удаляет комментарии и другие мета-данные.

Для анимированных изображений стоит использовать либо GIF-изображения с несколькими кадрами, либо DHTML-анимацию (применяя JavaScript0логику для смены PNG- или JPEG-картинок). Анимированные PNG-изображения пока не поддерживаются браузерами [14].

После применения всех методов конечный размер страницы может уменьшиться на 30-80%. Однако, если веб-ресурс загружается более 10 секунд, этого может оказаться недостаточно.

Методы объединения файлов

Каждый запрос от пользовательского браузера к серверу является достаточно ресурсоемкой операцией. Даже при наличии открытого соединения с сервером браузер все равно должен передать (и получить) соответствующие FTP- или HTTP-заголовки (которые только увеличивают фактический объем передаваемой информации). Кроме того, по спецификации HTTP/1.1 браузер не имеет права открыть более 4 соединений с одним сервером (в прошлом это было связано с большой нагрузкой на серверы при большом количестве одновременных запросов, однако, на данный момент браузеры, конечно, увеличивают это число до 6-10). Поэтому каждый следующий запрос должен дождаться своей очереди в общем потоке, прежде чем он будет обработан и передан серверу.

При значительном количестве файлов, которые требуются для отображения страницы на веб-ресурсе, это может вылиться в существенное время ожидания для пользователя. Для рассмотрения методов ускорения процесса в этой области стоит обратиться к упомянутым выше стадиям загрузки и сделать акцент на переносе всех возможных файлов из первой стадии (предзагрузки) во вторую, третью или четвертую стадию.

К сожалению, для корректного отображения страницы браузеру необходимо загрузить все файлы стилей (CSS), которые на ней указаны. Для ускорения процесса предзагрузки необходимо объединить все имеющиеся файлы (файлы для различных устройств можно объединить при помощи селектора @media) или даже (при небольшом их объеме или непостоянной аудитории веб-ресурса) расположить их прямо в HTML-файле. Кроме того, вызов CSS-файла на странице должен находиться перед вызовами любых других файлов (например, favicon.ico).

Как показало проведенное исследование [15], для ускорения загрузки HTML- и JavaScript-файлов наиболее оптимально будет их объединить в один-единственный файл (на установление множества соединений расходуется слишком много ресурсов по сравнению со скоростью передачи информации).

Наиболее популярной техникой для объединения изображений является CSS Sprites [16] (или CSS Image Map), когда для отображения нескольких (десятков или даже сотен) изображений используется один ресурсный файл. Она дает ощутимый выигрыш в скорости загрузки при использовании анимационных эффектов (например, смене изображения при наведении мыши), а также при большом количестве иконок на странице. При использовании данной техники фоновое изображение позиционируется с помощью стилевых правил, которые фактически «вырезают» его из общего ресурсного файла.

В качестве основных рекомендаций при создании ресурсных файлов для CSS Sprites можно назвать следующие:

1. Разбиение всех изображений на 5 основных групп (не рекомендуется использовать в одном файле изображений из более, чем 2 групп):
    1. Изображения, повторяющиеся по всем направлениям (соответствующие CSS-правилу repeat).
    2. Изображения, повторяющиеся по горизонтали (правило repeat-x).
    3. Изображения, повторяющиеся по вертикали (правило repeat-y).
    4. Изображения, которые не повторяются (правило no-repeat).
    5. И анимированные изображения.
2. Размер файлов должен быть не более 10-20 Кб
3. Следует проводить объединение близких по цветовой гамме изображений

Если у вас на странице выводится много небольших изображений, возможно, стоит воспользоваться техникой Image Map, чтобы сократить их количество.

Кэширование

Основной техникой для ускорения загрузки страницы для постоянных посетителей является кэширование, которое может свести число запросов к серверу для отображения страницы к минимуму (в идеале, к нулю). Здесь стоит помнить о корректной настройке заголовка Cache-Control. Для сброса кэша всегда можно воспользоваться дополнительным параметром в GET-запросе к ресурсу, который заставит сервер взять тот же физический файл, а клиентский браузер запросить файл и сохранить его под новым именем. Для статических ресурсов стоит выставлять достаточно большой срок кэша (можно экспериментировать со значения от месяца), для остальных файлов это значение должно быть равно среднему времени изменения либо вообще отсутствовать (для HTML-файлов, например).

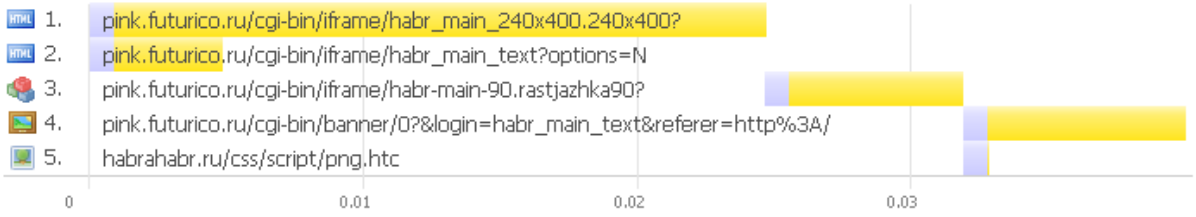

Рисунок 5. Добавление кэширующих заголовков для статических файлов способно в десятки раз уменьшить их число запросов при повторных загрузках страницы

В качестве дополнительного кэширующего аргумента можно использовать уникальные идентификаторы ресурсов (ETag). Они позволяет серверу не отдавать файл заново даже при закончившемся времени кэширования, а просто его продлять. Однако, существует ряд проблем с распределением файлов по разным физическим серверам и настройке на них идентичных ETag, но это, скорее, относится к уже очень большим системам.

В качестве альтернативы ETag можно использовать заголовок Last-Modified.

Параллельные загрузки

Для уменьшения удельного времени ответа от сервера при загрузке большого числа файлов можно разделить загрузку на несколько потоков (серверов) [17]. Сами сервера для такой цели (быстрой отдачи статических ресурсов) лучше настраивать под «легким» окружением (например, nginx). В качестве балансирующего параметра можно рассматривать как распределение по географическому принципу (например, кластеры в США, Европе, Азии), так и по нагрузке (пул свободных серверов определяется каждый раз при загрузке страницы). Также возможно использование балансировки на клиенте для достижения того же эффекта.

В качестве основных проблем стоит отметить необходимости создания хеш-функции от имени файла, чтобы один и тот же файл загружался только с одного сервера, иначе браузер будет запрашивать его с серверов-зеркал, пока не забьет кэш всеми его копиями со всех распределенных серверов. Также стоит ограничиться 4 хостами (для большого числа файлов), для небольшого их числа (15-25) стоит использовать не более 3. 2 хоста разумно использовать, только если число файлов превосходит 10 из-за дополнительных расходов на распознавание имени в DNS-таблице [2].

Оптимизация JavaScript

Существует возможность загружать необходимые для отображения страницы JavaScript-файлы, фактически, после ее загрузки. Однако, есть еще и пара нюансов. Например, веб-ресурс при этом должен полностью функционировать и без JavaScript (должны осуществляться переходы по ссылкам, первоначальный вид страницы должен формироваться на сервере). Это позволит повысить индексируемость ресурса поисковыми машинами и обезопасит тех пользователей, у которых ваши JavaScript-файлы не работают по тем или иным причинам (мобильные или устаревшие браузеры, отключенный JavaScript и др.).

Стоит больше полагаться на возможности CSS при создании анимационных эффектов. Вы не сможете отобразить страницу в браузере лучше, чем сам браузер, поэтому стоит оставить всю тяжелую работу по рисованию страницу для внутреннего движка (это относится к анимационным эффектам при наведении на кнопки, изменению разметки при изменении размеров окна и т.д.). CSS-движок, в среднем, работает быстрее, чем JavaScript. Также стоит избегать использования CSS-выражений (CSS expressions) либо оптимизировать их, чтобы они исполнялись только один раз на странице [18].

При оптимизации взаимодействия JavaScript-машины с браузером также следует обновлять DOM-дерево большими кусками. Все DOM-обращения ресурсоемки, это является полной аналогией базы данных для серверных приложений. Чем меньше будет произведено работы с DOM, тем быстрее будет выполняться JavaScript.

Здесь стоит сделать отдельное замечание по поводу обработчиков событий: их количество необходимо сводить к минимуму. В идеале, стоит использовать один-единственный onclick для body и обрабатывать уже источник произведенного действия. Если требуются менее глобальные эффекты – можно просто ограничиться одним обработчиков на блоке, в котором заключена требуемая область. Большое количество обработчиков событий (которые иногда забывают убирать при изменении содержащего их HTML-кода) приводит к утечкам памяти в Internet Explorer.

Также можно посоветовать кэшировать глобальные переменные в локальные (однако, тут могут быть нюансы, особенно с цепочками вызовов функций), избегать использования eval и setTimeout / setInterval (которые выполняют eval на передаваемую в качестве аргумента строку). Вместо этого можно использовать анонимные функции.

При осуществлении тяжелых вычислений (например, дополнительная загрузка данных с сервера или сортировка больших массивов) стоит обновлять интерфейс пользователя, чтобы он знал, что выполняются какие-то действия, и мог немного подождать. Однако, самое основное здесь тут главное не переусердствовать с уведомлениями: ведь каждое обновление страницы занимает некоторое время, и накладные издержки из-за него могут оказаться много больше «полезного» времени.

Предзагрузка, интерактивная и полная загрузка

Рассмотренные выше методы помогут в значительной степени ускорить стадию предзагрузки. Однако, при наличии значительного числа JavaScript-файлов, обеспечивающих анимационные эффекты или взаимодействие с пользователей, полная загрузка может быть значительно задержана по времени.

Для предотвращения этого эффекта наиболее распространенным подходом является вынесение интерактивной загрузки (фактически, всех внешних JavaScript-файлов) в область предзагрузки (при небольшом объеме кода можно его полностью включить в исходный HTML) или пост-загрузки (используя комбинированное событие window.onload [19]).

Как обычно главным критерием будет аудитория веб-ресурса: следует оптимизировать загрузку страницы, ориентируясь именно на ее характеристики (средняя скорость доступа, типичный браузер и др.). При этом стоит руководствоваться следующими подходами:

1. Для постоянно обновляющейся аудитории наиболее оптимальным будет включение всей файлов в исходный HTML (изображений – с помощью комбинированного data:URL подхода [20]).
2. Если аудитория смешанная, то рекомендуется примерно половину размера страница оставить в HTML-файле, а оставшуюся половину разделить на несколько (4-8) файлов, которые затем можно будет кэшировать.
3. При постоянной аудитории стоит сократить размер HTML-файла до минимума и настроить кэширующие заголовки, ориентированные на продолжительное время.
4. Загрузку всех JavaScript-файлы следует вынести в четвертую стадию (пост-загрузку), ускорив тем самым отображение страницы в третьей стадии. В идеале, второй стадии (интерактивной загрузки) не должно существовать: пользователи некоторой время после загрузки страницы «осваиваются» на ней, привыкают к элементам навигации, поэтому в это время ожидания можно невидимо для них подгрузить все необходимые интерактивные элементы.

5. Для ускорения загрузки необходимых фоновых изображений, можно форсировать их вызовы через динамическое создание соответствующих картинок в head области страницы (метод new Image в JavaScript).

Неблокирующая загрузка JavaScript

Как уже было отмечено выше, загрузку JavaScript-кода на странице следует убрать из числа факторов, влияющих на ход основной (до 3 стадии) загрузки. Зачем это делается? JavaScript-код может содержать вызовы document.write (которые изменяют структуру DOM-дерева документа) или location.href (которые осуществляют перенаправление на другую страницу). Браузер не имеет права отображать страницу, не проанализировав весь JavaScript-код, поэтому большое количество вызовов таких файлов, происходящих во второй стадии, может существенно замедлить загрузку [21].

Если общий размер JavaScript-кода менее 5% от общего размера HTML-/CSS-кода, то его следует включить в последний (расположив максимально близко к закрывающему тегу body). Если JavaScript представляет собой один монолитный блок, обеспечивающий интерактивное функционирование всей страницы, при этом его размер достаточно велик для первого случая, то стоит вынести загрузку этого кода (как одного-единственного внешнего файла) в пост-загрузку.

При наличии нескольких не связанных друг с другом JavaScript-частей их можно вызывать в пост-загрузке независимо (в несколько потоков через создания динамического узла script в head-области страницы), что увеличит скорость их загрузки. Это относится, в частности, к различным счетчикам [22].

Если JavaScript на странице представляет собой некоторую библиотеку, которая затем используется различными приложениями, и при этом библиотека используется на большинстве страниц веб-ресурса, а использующие ее приложения различаются от странице к страницы, то в этом случае стоит организовать загрузку по цепочке. Вначале в пост-загрузке вызывается сам библиотечный файл, затем он подгружает все необходимые на данной странице приложения. Это относится к использованию большинства современных JavaScript-фреймворков – jQuery, Prototype, Mootools, Ext2, Dojo, YUI.

Все вышеописанные методы помогут избежать задержек в загрузке, связанных с использованием любого JavaScript-кода.

Заключение

Подводя итоги всему вышесказанному, можно с твердостью заявить: по-настоящему быстрые веб-ресурсы существуют, и создавать их не настолько сложно, как может показаться на первый взгляд. Самое главное в использовании подходов клиентской оптимизации – это понимать, на какой стадии загрузки отразятся те или иные действия, стремясь максимально ускорить предзагрузку страницы и ее основную загрузку.

Описанные выше алгоритмы применимы, практически, в любой ситуации. Это показало обширное практическое применение их для широкого круга задач: оптимизации высоконагруженных страниц [23], ускорение работы JavaScript-логики на странице [24] и оптимизационный анализ неоднородных веб-ресурсов [25].

Как показывает практика, загрузку среднего веб-ресурса можно ускорить примерно в 2-3 раза, и все это достигается за счет очень простых действий: фактически, все, что не нужно пользователю прямо сейчас (сразу после предзагрузки) можно загрузить при отображении страницы или даже после ее отображения (в течение первых 100-500 миллисекунд), пока пользователь еще не совершил никаких активных действий.

Список литературы

1 “Average Web Page Size Triples Since 2003” // Web Site Optimization. - http://www.websiteoptimization.com/speed/tweak/average-web-page/.

2 “Best Practices for Speeding Up Your Web Site” // Yahoo!. - http://developer.yahoo.com/performance/rules.html.

3 Bouch, A., Kuchinsky, A., Bhatti, N. , “Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service” // CHI. - The Hague, The Netherlands : [б.н.], 2000 г..

4 “Retail Web Site Performance: Consumer Reaction to a Poor Online Shopping Experience” // Akamai. - 2006 г..

5 Nah, F., “A study on tolerable waiting time: how long are Web users willing to wait?” // Behaviour. - [б.м.] : Information Technology, 2004 г.. - 23 : Т. 3.

6 Galletta, D., Henry, R., McCoy, S., Polak, P., “Web Site Delays: How Tolerant are Users?” // Journal of the Association for Information Systems. - 5 : Т. 1.

7 “The Psychology of Web Performance” // Web Site Optimization. - http://www.websiteoptimization.com/speed/tweak/psychology-web-performance/.

8 “CSS Tidy” // Soureforge. - http://csstidy.sourceforge.net/.

9 “YUI Compressor” // Yahoo!. - http://developer.yahoo.com/compressor/.

10 «CSS: все о сжатии» // Web Optimizator. - http://webo.in/articles/habrahabr/14-minifing-css/.

11 «Javascript: жать или не жать?» // Web Optimizator. - http://webo.in/articles/habrahabr/11-minifing-javascript/.

12 “Dean Edwards Packer” // Dean Edwards. - http://dean.edwards.name/packer/.

13 «Влияние уровня gzip-сжатия на производительность сервера» // Web Optimizator. - http://webo.in/articles/habrahabr/33-gzip-level-tool/.

14 “Replace GIF with PNG Images” // Web Site Optimization. - http://www.websiteoptimization.com/speed/tweak/png/.

15 «Аккуратно нарезаем поток» // Web Optimizator. - http://webo.in/articles/habrahabr/48-flow-slices-optimization/.

16 «CSS Sprites: все, что вы знали, но боялись спросить» // Web Optimizator. - http://webo.in/articles/habrahabr/08-all-about-css-sprites/.

17 “Optimize Parallel Downloads to Minimize Object Overhead” // Web Site Optimization. - http://www.websiteoptimization.com/speed/tweak/parallel/.

18 «Практический JS: оптимизируем CSS expressions» // Web Optimizator. - http://webo.in/articles/habrahabr/10-css-expressions-optimization/.

19 «Практический JS: “отложенная” загрузка» // Web Optimizator. - http://webo.in/articles/habrahabr/05-delayed-loading/.

20 «Кроссбраузерное использование data:URL» // Web Optimizator. - http://webo.in/articles/habrahabr/46-cross-browser-data-url/.

21 “Non-blocking JavaScript Downloads” // Yahoo! User Interfacses Blog. - http://yuiblog.com/blog/2008/07/22/non-blocking-scripts/.

22 «Разгоняем счетчики: от мифов к реальности» // Web Optimizator. - http://webo.in/articles/habrahabr/61-counters-optimization/.

23 «Оптимизация часто показываемых страниц» // Web Optimizator. - http://webo.in/articles/clientside2007/common-pages-optimization/.

24 “High Performance Ajax Applications” // Julien Lecompte Blog. - http://www.julienlecomte.net/blog/2007/12/39/.

25 «Результаты оптимизации» // Web Optimizator. - http://webo.in/about/results/.