УДК 681.3.06

Ступень П. В.,
доцент, канд. техн. наук, Одесский национальный политехнический университет, г. Одесса, Украина
Нгуен Д. Х.,
Одесский национальный политехнический университет, г. Одесса, Украина

АНАЛИЗ МЕТОДОВ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

Современные веб-приложения уже сопоставимы по своим возможностям с классическими приложениями, но при этом могут быть доступны в любом месте и в любое время на компьютере, планшете или мобильном устройстве и зачастую имеют меньшую совокупную стоимость владения. Эти особенности делают веб-технологии очень привлекательными для решения широкого спектра бизнес задач. В свою очередь, это выдвигает высокие требования к их производительности. Для достижения максимальной производительности веб-приложения необходимо минимизировать объём передаваемых данных и число запросов к веб-серверу. По мере увеличения сложности веб-приложений неизменно возрастает вероятность нарушения их функциональности.

В статье рассматриваются методы повышения скорости загрузки веб-страницы: кэширование данных на стороне сервера; кэширование веб-страниц (на стороне сервера либо на стороне клиента); использование многоуровневой архитектуры FrontEnd-BackEnd; использование веб-сервера, построенного по FSM (Finite State Machine), сжатие передаваемых данных средствами протокола HTTP. Все они направлены на повышение производительности веб-приложения и могут применяться как по отдельности, так и комбинироваться.

Ключевые слова: *веб-приложение; оптимизация; кэширование; динамическая страница; статический файл; производительность.*

Большое влияние на скорость загрузки оказывает качество программного кода: применение оптимизированных конструкций кода может дать ощутимый прирост производительности веб-приложения, но на производительность Веб-приложения влияют и технические характеристики среды выполнения (объем оперативной памяти, быстродействие жестких дисков и процессоров, пропускная способность каналов связи). Основными же характеристиками применимости и использования реализуемых веб-приложений (ожидаемыми результатами) являются количество одновременно обслуживаемых пользователей и время реакции на их запросы. Влияние указанных факторов на производительность веб-приложений очевидно.

На практике даже секундная задержка может испортить впечатление от просмотра. Быстрая загрузка представляет особую проблему для страниц, которые должны просматриваться с помощью мобильных устройств. Посетители могут пользоваться различными сетями стандартов. В этом случае задержки будут гораздо больше, чем при высокоскоростном подключении.

При большом количестве пользователей может наступить такой момент, когда количество входящих запросов превысит вычислительные возможности сервера, соответственно время для генерации одной страницы будет гораздо больше.

Существуют несколько методов повышения эффективности работы Веб-приложений. Рассмотрим следующие методы:

1. Кэширование данных на стороне сервера (DB cache). Кэширование запроса на уровне базы данных это один из способов оптимизации Веб-приложений. В любом приложении встречаются медленные SQL запросы, результаты которых можно сохранить на некоторое время. Это позволит выполнить меньше таких операций, а большинству пользователей показать заранее сохраненные данные.

2. Кэширование страниц на стороне сервера (Page cache). Суть кэширования страниц на стороне сервера состоит в том, чтобы записать все, что происходит на сервере, в файл, сохранить его и при последующем обращении пользователя к этой странице выдать ему статическую копию. В результате будет получено не только ускорение загрузки страниц, но и снижена нагрузка на сервер и базу данных.

3. Кэширование страниц на стороне клиента (Client cache). Есть различные способы для кэширования страниц на стороне клиента. Самые распространённый способ это кэширование через HTTP-заголовки, **Expires –** указывает срок годности полученных данных (в формате RFC 1123). **Last-Modified –** дата и время последнего изменения запрашиваемого контента. Получив такой заголовок, клиент кэширует стра-

ницу и при следующих запросах добавляет заголовок If-Modified-Since, чтобы проверить не изменился ли документ с указанного момента. На стороне сервера можно выполнить сравнение дат и вернуть либо заголовок 304 Not Modified в случае, если метка даты свежая, либо весь контент.

4. Предварительная генерация содержимого Веб-страниц в статические файлы. (Pre-generate page).

Статические страницы (сайты) загружаются быстрее динамических. Это происходит потому, что статический сайт выдает пользователю в браузер себя «сразу как он есть», в то время как динамический сайт предварительно обрабатывается на сервере, делает запросы к базе данных.

5. Использование архитектуры Frontend-Backend [1]. На крупных проектах программную серверную часть обычно представляют 2 веб-сервера – Apache и Nginx. Nginx принимает запросы и, в случае статического файла, (изображение, файл css, js или xml) сразу же отдаёт его содержимое, а в случае PHP-скрипта, отправляет его к серверу Apache, который уже умеет обрабатывать PHP. Тут nginx – это Frontend, а Apache – Backend. При таком подходе значительно экономится память и процессорные ресурсы т. к. сервер Nginx большую часть трудоемких операций (отправка файлов, ожидание ответа, считывание данных с диска и т. п.) осуществляет асинхронно с помощью функций ядра операционной системы, получая только сигналы об их завершении. Например, в Apache каждый процесс функционирует в бесконечном цикле, ожидая завершения трудоемких операций.

6. Использование Веб-сервера, построенного по FSM (Finite State Machine) [2]. (FSM Server)

Для того, чтобы отказаться от использования ресурсоемкого сервера Apache, в качестве Frontend-сервера, выполненного по архитектуре FSM можно использовать Nginx. В качестве Backend используется FastCGI сервер. FastCGI, вместо того, чтобы создавать новые процессы для каждого нового запроса, использует постоянно запущенные процессы для обработки множества запросов. Это позволяет исключить накладные расходы на загрузку и выгрузку интерпретаторов в случае обычного CGI режима сервера Apache.

7. Сжатие передаваемых данных средствами HTTP протокола (Apache cache, Nginx cache). Данные перед тем как будут отправлены на браузер пользователя, могут быть сжаты сервером. Например, на сервере Apache используются модуль mode_deflate, позволяющие сжимать ответ методами gzip или deflate. GZIP лучше всего сжимает текстовые ресурсы, часто достигая коэффициента сжатия 70–90 % при работе с большими файлами. Современные браузеры поддерживают и автоматически применяют сжатие GZIP для всех HTTP-запросов.

Для имитационных экспериментов были использованы следующие программные средства: Apache 2.4 – в качестве Веб-сервера; Nginx 1.8.0 – как Веб-сервер с FSM архитектурой; PHP 5.5.9 – язык выполнения серверных сценариев; MySQL 5 – в качестве сервера СУБД.

Для проведения эксперимента используем рабочий проект, который был написан на PHP фреймворке Codeigniter [3]. С помощью генератора были созданы примерно 100 000 типовых записей, каждая из которых содержала заголовок, ссылку на графический файл, мета данные (используются для поисковых систем) и случайный текст на 10 000 символов. Тестовая страница содержит в себя дополнительно 1 css файл, около 10–15 графических файлов (в основном это иконки) и 7 файлов js-скриптов.

Таким образом, тестовое задание эмулирует нагрузку типичной динамически формируемой Веб-страницы средней трудоемкости.

Для имитации клиентских обращений была выбрана программа siege [4], это утилита для нагрузочного тестирования веб-серверов. Она была создана для того, чтоб дать разработчикам возможность проверить ресурсоёмкость своего кода в условиях, максимально приближенных к реальным. Так же Siege может имитировать обращения к сайту сразу нескольких пользователей. Пример типовой команды для имитации 350 пользователей, обращающихся к серверу в течение 30 минут без задержек, показан ниже:

siege –c 350 –d -0 –f 168.urls –i –t 30M –H

Все эксперименты были проведены на сервере под управлением Ubuntu Linux. Технические характеристики: центральный процессор Intel i5 2.2 ГГц, объем оперативной памяти 4 ГБ, жесткий диск объемом 500 ГБ (частота вращения шпинделя 5400 об/мин). Для исключения накладных сетевых расходов на трафик, эксперимент был проведён локально.

Так как тестовый проект был написан на фреймворке Codeigniter, то этапы работы Веб-приложения будут следующие:

1. Веб-браузер пользователя отправляет HTTP-запрос определенной Веб-страницы.

2. Веб-сервер принимает запрос (в нашем случае это Nginx или Apache), получает файл и передает его механизму PHP на обработку.

3. Механизм PHP начинает синтаксический анализ сценария. Маршрутизатор отправляет запрос к определённому контроллеру для обработки запроса. Контроллер вызывает соответствующий модель, который делает запрос на получение данных из базы данных MySQL.

4. База данных MySQL принимает запрос, обрабатывает его, а затем отправляет результаты, обратно в механизм PHP.

5. Далее данные будут отработаны и отправлены обратно к контроллеру, а контролер в свою очередь отправит данные на View (представление), где формируется HTML страница, после чего отправляет результаты в HTML-формате Веб-серверу.

6. Веб-сервер пересылает HTML в браузер, с помощью которого пользователь получает необходимый результат.

Этапы работы Веб-приложения показаны на рис. 1.

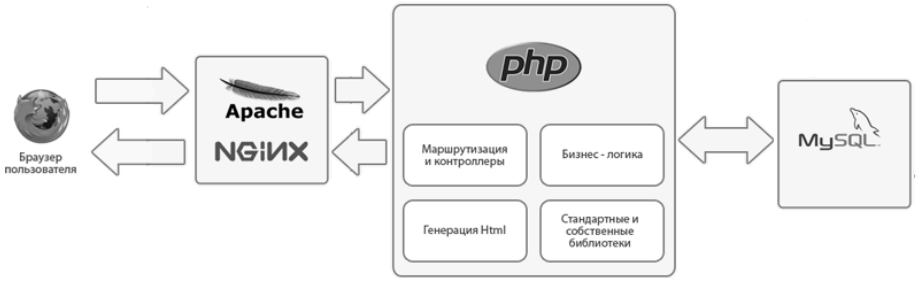

Рис. 1. Этапы работы Веб-приложения

Результат проведения эксперимента для динамических Веб-страниц

Количество запросов, обрабатываемых сервером за 1 секунду (рис. 2). При количестве пользователей от 10 до 100 количество запросов, которое может обработать система не зависит от метода увеличения производительности и составляет примерно от 20 до 200 запросов в секунду. Сервер без использования каких-либо методов повышения производительности (эталонный вариант) не позволяет обрабатывать более 200 запросов в секунду. Наилучшие результаты показал метод кэширования страниц на стороне сервера при котором количество запросов обрабатываемых в секунду растет пропорционально количеству пользователей. Остальные методы показали близкие результаты, при увеличении числа пользователей от 100 до 350 количество запросов обрабатываемых в секунду увеличивается по сравнению с эталонным вариантом, на величину от 10 до 50 %.

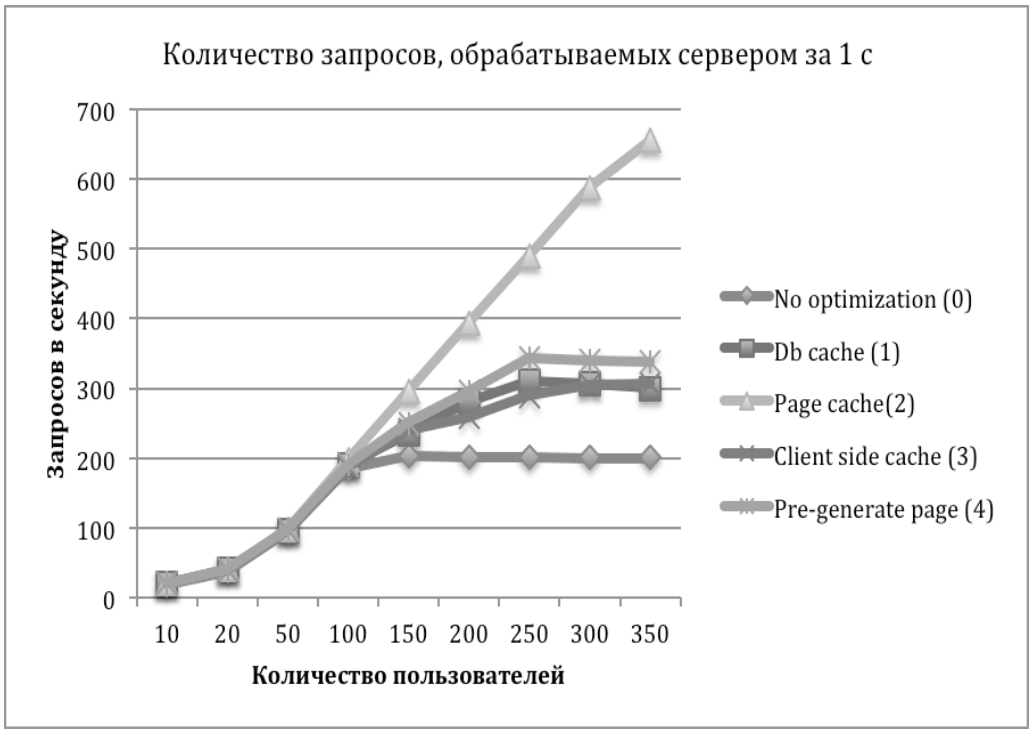

Рис. 2. Количество запросов, обрабатываемых сервером за 1 с

Сетевой трафик тестового приложения (рис. 3). Суммарный объем трафика от сервера к клиенту при увеличении числа пользователей остался неизменным для эталонного метода и при кэшировании на уровне базы данных и составил величину менее 1,5 MB/c, причем объем трафика практически не менялся при увеличении пользователей от 100 до 350. Наибольший объем трафика при количестве пользователей 350 показал метод кэширования страниц, при котором величина трафика достигла 4,5 MB/c. При использовании метода кэширования страниц наблюдается линейный рост объема трафика при росте количества пользователей. Методы кэширования страниц на стороне клиента и предварительная генерация содержимого Веб-страницы показали резкий рост трафика при увеличении пользователей от 10 до 150, дальнейшее увеличение пользователей приводило к незначительному изменению трафика в пределах 14 % при предварительной генерации содержимого Веб-страниц и 7 % для метода кэширования страниц на стороне клиента.

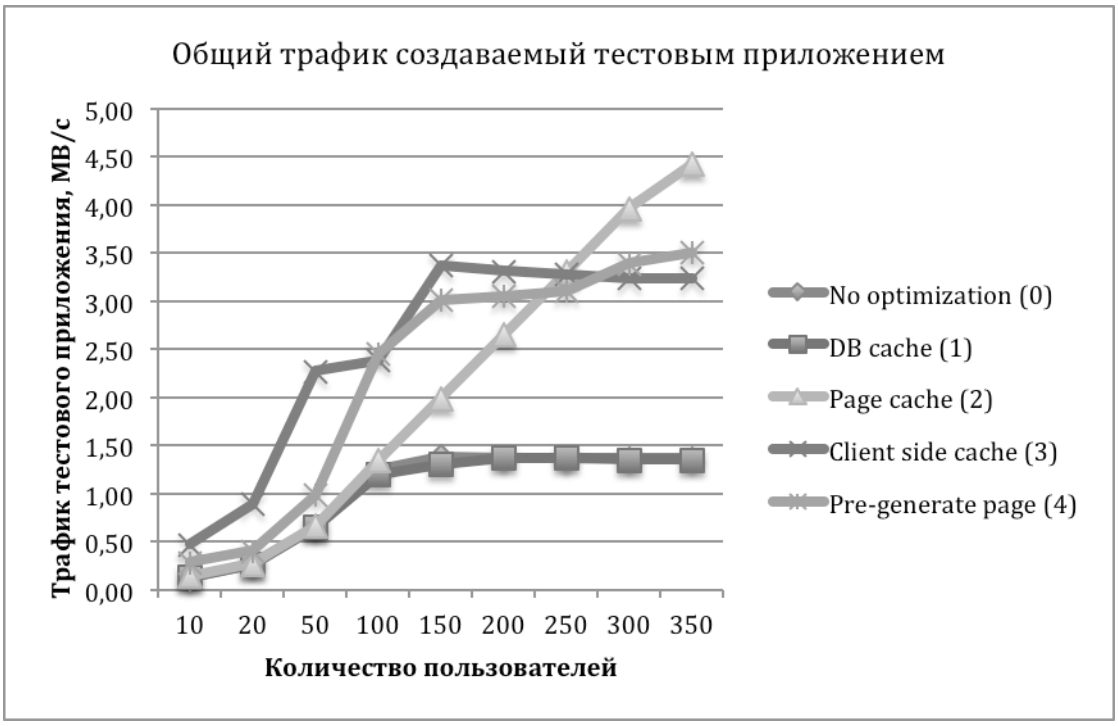

Рис. 3. Общий трафик создаваемый тестовым приложением

Объем памяти для выполнения тестового приложения (рис. 4). Рост объема используемой памяти при увеличении числа пользователей наблюдался при использовании каждого метода. Кэширование данных из БД, кэширование страниц на сервере и кэширование страниц на стороне клиента показали примерно одинаковые результаты, максимальный объем памяти составил 500 МБ при количестве пользователей 350. Наибольший рост объема памяти наблюдается при эталонном варианте и для 350 пользователей составляет 800 МБ. Промежуточный результат показал метод предварительной генерации страниц, который при количестве пользователей до 100 позволял использовать до 400 МБ памяти, а при увеличении пользователей до 350 объем используемой памяти вырос и составил 650 МБ.

Рис. 4. Объём памяти для выполнения тестового приложения

Среднее время отклика (рис. 5). Для 5-ти методов среднее время отклика является примерно одинаковым при количестве пользователей до 100 и не превышает 0,1 секунду. Начиная с количества пользователей 150, среднее время отклика линейно растет с увеличением пользователей для методов эталонного,

кэширования на уровне БД, кэширования страниц на стороне клиента и предварительной генерации страниц. Время отклика совпадает для эталонного метода и метода кэширования на уровне БД и составляет наибольшее значение 1,2 секунды для 350 пользователей. При использовании кэширования на стороне клиента среднее время отклика уменьшается по сравнению с кэшированием на уровне БД примерно на 10–15 %. Метод предварительной генерации страниц дает выигрыш еще примерно на 10 %. Наилучший результат показал метод кэширования страниц на стороне сервера, при котором среднее время отклика не превысило 0,2 секунды.

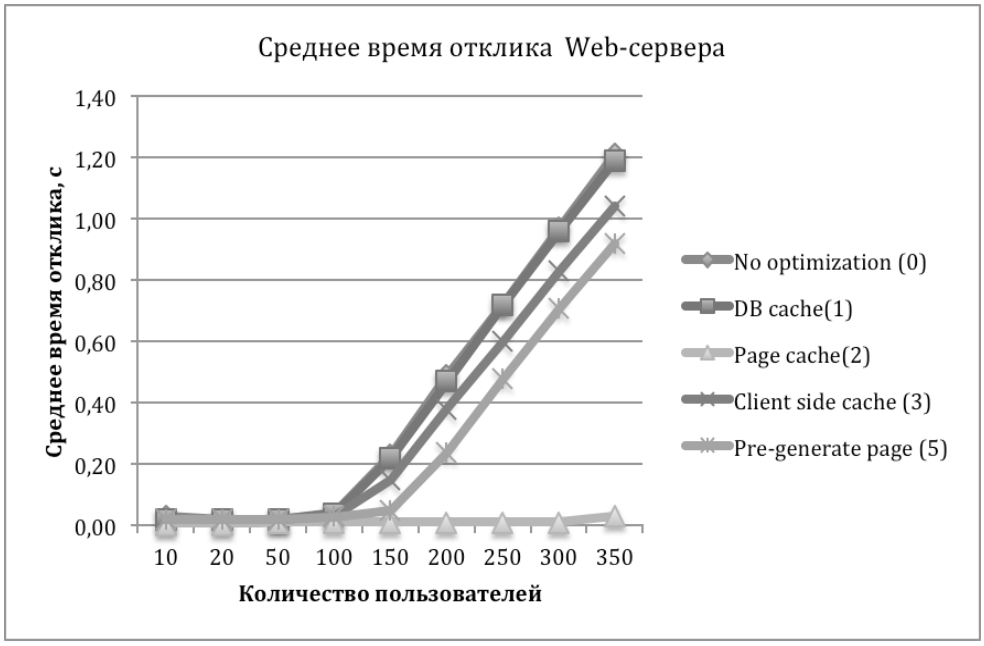

Рис. 5. Среднее время отклика Веб-сервера

Результат проведения эксперимента для статических Веб-страниц

Количество запросов, обрабатываемых сервером за 1 секунду (рис. 6). Вариант без оптимизации и использование сжатия передаваемых данных средства-и Apache показали одинаковый результат, причем количество запросов обрабатываемых в секунду меняется незначительно до 260 при увеличении числа пользователей от 150 до 350. Остальные три метода Frontend-Backend, FSM Веб-server, сжатие средствами Nginx также показали примерно одинаковые результаты между собой, при увеличении числа пользователей количество запросов возрастает более чем в 2 раза по сравнению с первыми двумя методами.

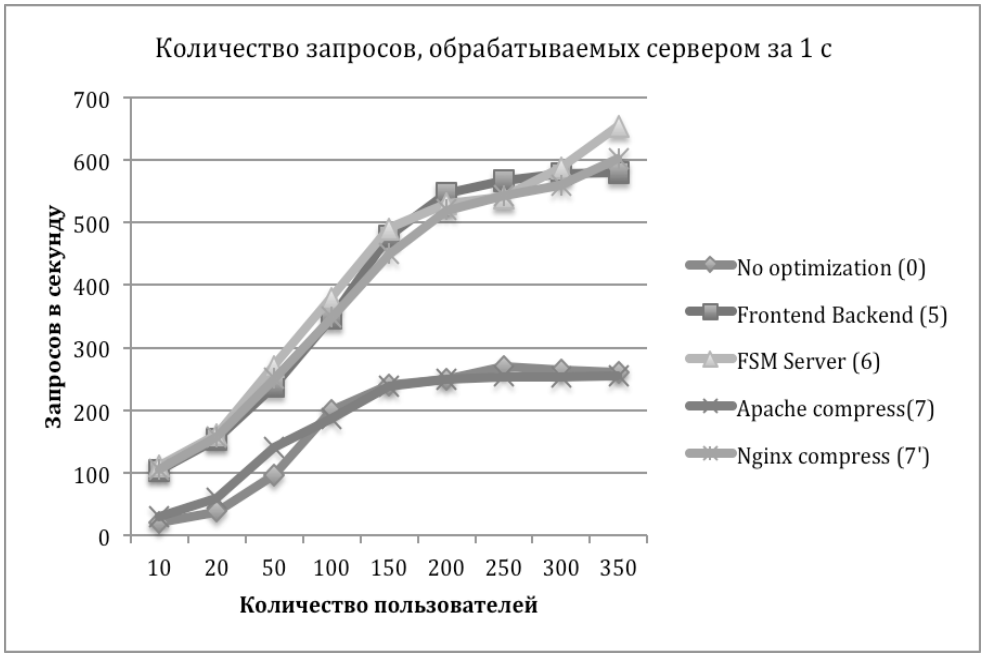

Рис. 6. Количество запросов, обрабатываемых сервером за 1 с

Общий трафик создаваемый тестовым приложением (рис. 7). Величина трафика существенно меняется в зависимости от применяемого метода. Так веб-сервер на FSM архитектуре генерирует наибольший объем трафика и только при количестве пользователей от 250 объемы трафика совпадают с методом Frontend-Backend. Для обоих методов объемы трафика возрастают при увеличении количества пользователей и достигают 7 MB/c для 350 пользователей. Наименьший объем трафика генерируется при сжатии передаваемых данных средствами Apache. Объем трафика остается неизменным при количестве пользователей более 150 и приближается к 2,2 MB/c. Объем трафика при использовании сжатия средствами Nginx совпадает с методом сжатия средствами Apache, при количестве пользователей менее 100, затем объем трафика увеличивается примерно на 30 %. Без оптимизации сервер показал средний результат, объем трафика в среднем на 20 % больше по сравнению с методом сжатия средствами Nginx.

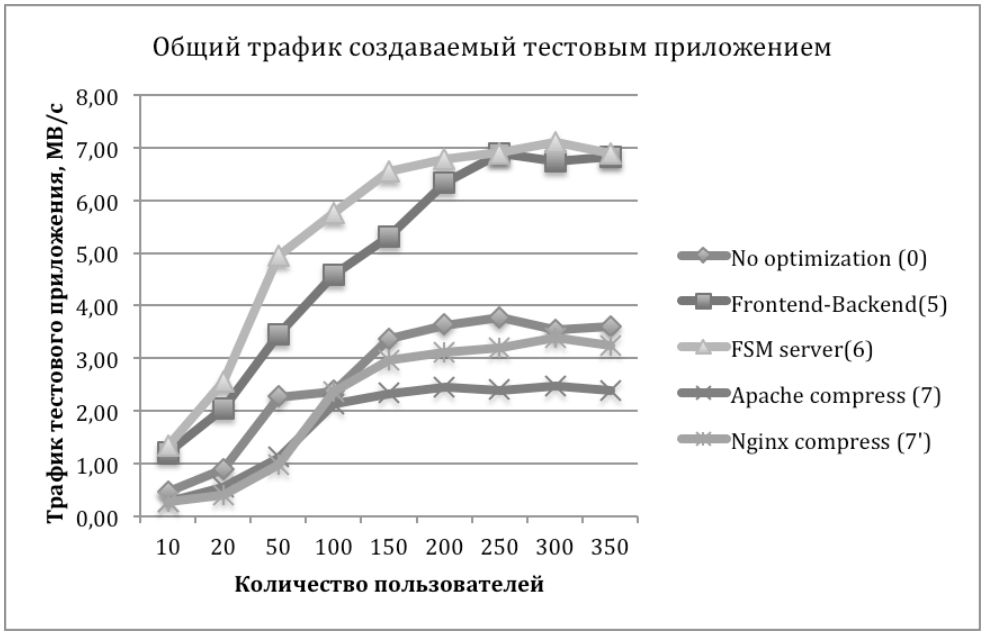

Рис 7. Общий трафик создаваемый тестовым приложением

Объем памяти для выполнения тестового приложения (рис. 8). Наибольший объем памяти используется при сжатии передаваемых данных средствами Apache, причем объем памяти возрастает при увеличении числа пользователей. Максимальный объем памяти 830 МБ при количестве пользователей 350. Наименьший объем памяти используется при использовании веб-сервера на FSM архитектуре и сжатии средствами Nginx от 250 до 350 МБ приросте числа пользователей от 10 до 350. Лучший показатель при использовании веб-сервера на FSM архитектуре. Для всех методов наблюдается рост объемов памяти при увеличении числа пользователей, но при использовании веб-сервера на FSM рост самый медленный.

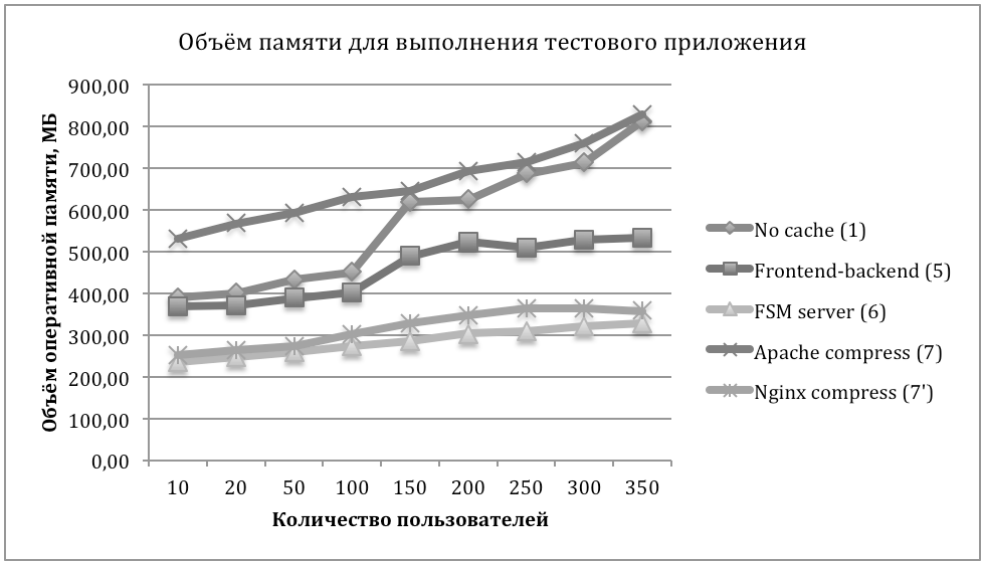

Рис. 8. Объём памяти для выполнения тестового приложения

Среднее время отклика (рис. 9). Среднее время отклика растет практически линейно для всех методов оптимизации. Наиболее резкий рост наблюдается при сжатии передаваемых данных средствами Apache, среднее время отклика превысило 1,4 секунды при количестве пользователей 350. Наименьшее время отклика и наименьший угол наклона кривой наблюдается при использовании веб-сервера на FSM архитектуре, хотя в большинстве случаев среднее время отклика совпадает с методом сжатия средствами Nginx. При использовании метода Frontend-Backend среднее время отклика в среднем на 35 % больше в диапазоне пользователей от 100 до 250 по сравнению с веб-сервером на FSM архитектуре. При количестве пользователей 350 среднее время отклика для трех методов примерно одинаково и не превышает 0,6 секунд.

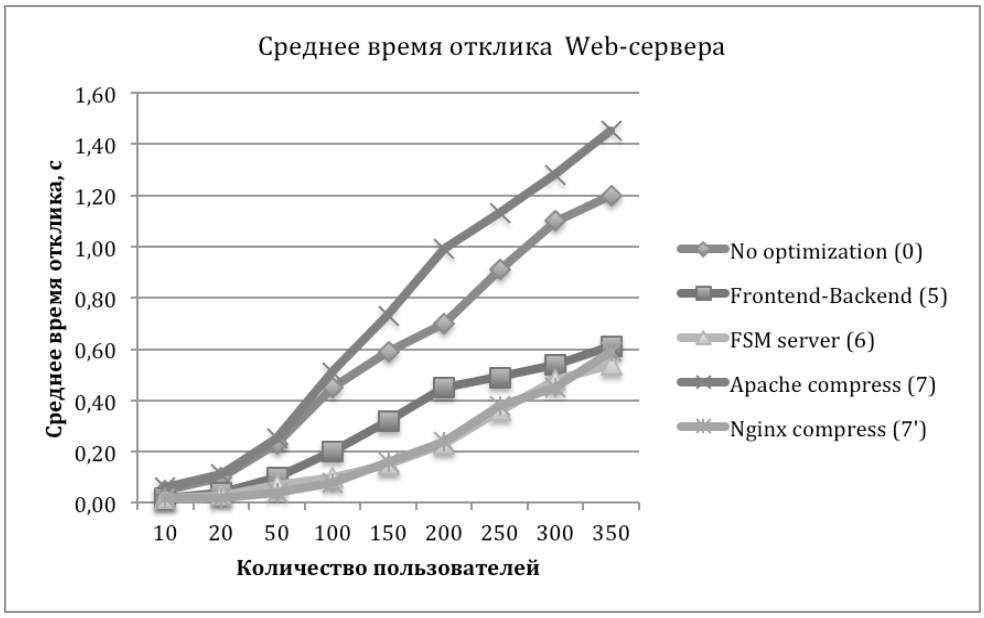

Рис. 9. Среднее время отклика Веб-сервера

Проведенные эксперименты по исследованию скорости загрузки веб-страниц показали, что наиболее предпочтительным с точки зрения практического применения являются следующие методы увеличения производительности: кэширование динамических страниц на стороне клиента, использование архитектуры Frontend-Backend и сжатие передаваемых данных средствами HTTP протокола.

ЛИТЕРАТУРА

1. Frontend and Backend [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/Front_and_back_ends. – Загл. с экрана.
2. FAQ appendix 1: как писать сервера [Электронный ресурс]. – Режим доступа : https://groups.google.com/forum/#!topic/fido7.ru.unix.prog/Mu2xfgP-NuU. – Загл. с экрана.
3. Codeigniter Web Framework [Электронный ресурс]. – Режим доступа : http://www.codeigniter.com. – Загл. с экрана.
4. Seige [Электронный ресурс]. – Режим доступа : http://habrahabr.ru/post/65128/. – Загл. с экрана.

Ступень П. В., Нгуен Д. Х.,
Одеський національний політехнічний університет, м. Одеса, Україна

Аналіз методів збільшення продуктивності веб-додатків

Сучасні веб-додатки вже співставні за своїми можливостями з класичними додатками, але при цьому можуть бути доступні в будь-якому місці і в будь-який час на комп'ютері, планшеті або мобільному пристрою і часто мають меншу сукупну вартості володіння. Ці особливості роблять веб-технології дуже привабливими для вирішення широкого спектра бізнес завдань. У свою чергу, це висуває високі вимоги до їх продуктивності. Для досягнення максимальної продуктивності веб-додатку необхідно мінімізувати обсяг переданих даних і число запитів до веб-серверу. По мірі збільшення складності веб-додатків незмінно зростає ймовірність порушення їх функціональності.

У статті розглянуто набір методів для підвищення швидкості завантаження веб-сторінки: кешування даних на стороні сервера; кешування веб-сторінок (на стороні сервера або на стороні клієнта); використання багаторівневої архітектури FrontEnd-BackEnd; використання веб-сервера, побудованого за FSM (Finite State

Machine), стиснення переданих даних засобами протоколу HTTP. Усі вони спрямовані на підвищення продуктивності веб-додатків і можуть застосовуватися як окремо, так і комбінуватися.

Ключові слова: *Веб-додаток; оптимізація; кешування; динамічна сторінка; статичний файл; продуктивність.*

Stupen P. V., Nguyen D. H.,
Odessa National Polytechnic University, Odessa, Ukraine

Analysis methods of increasing performance of Web-applications

Modern Web applications are already comparable in its capabilities with classical applications, but it can be accessed anywhere and at any time on a computer, tablet or mobile device, and often have a lower total cost of ownership. These features make web technology very attractive for a wide range of business problems. In turn, this puts high demands on their performance. For maximum performance web-application is necessary to minimize the amount of data transmitted and the number of requests to the web-server. As the complexity of the web-based application always increases the likelihood of violations of their functionality.

The article discusses a set of methods to increase the speed of loading web-pages: data caching on the server side; caching of web-pages (on the server side or the client side); Use a layered architecture FrontEnd-BackEnd; Use the web-server, built on FSM (Finite State Machine), the compression of data transmitted by means of the protocol HTTP. All are aimed at increasing the productivity of web-applications and can be used either singly or combined.

Key words: *Web-application; optimization; caching; dynamic page; static file; performance.*

 Дата надходження статті до редколегії 07.11.2015