УДК 681.3.06

# ИССЛЕДОВАНИЕ МЕТОДОВ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ WEB-ПРИЛОЖЕНИЙ

И.А. Ботыгин, К.А. Каликин

Томский политехнический университет
E-mail: bia@tpu.ru

*Рассмотрены основные методы увеличения производительности серверных приложений и проведены имитационные программные эксперименты, направленные на оценку количества запросов обрабатываемых сервером, объемов трафика создаваемого приложениями; требуемого объема оперативной памяти, среднего времени отклика. Показано, что наиболее предпочтительным с точки зрения практического применения является вариант с кэшированием динамических страниц на стороне клиента, запуском PHP в режиме FastCGI и обработкой статических файлов FrontEnd-сервером Nginx.*

## Введение

В настоящее время информационные сервисы в глобальных телекоммуникациях привлекают миллиарды пользователей не только как источники информации, общения, развлечения, но и как средство организации электронного бизнеса и управления различными объектами. Наиболее распространенным вариантом практической реализации подобных сервисов являются WEB-приложения. Заметим, что среда WEB-приложений обладает рядом специфических особенностей. *Во-первых*, взаимодействие пользователя с сервером осуществляется как совокупность кратковременных часто повторяющихся запросов (обращений). *Во-вторых*, помимо данных и команд пользователя по глобальным телекоммуникациям передаются дополнительные файлы или данные, содержащие в себе элементы дизайна, иллюстрации, медиа-контент, исполняемый код, справочную информацию и др., что порождает множество параллельных запросов, значительно повышающих загрузку серверов и каналов связи. *В-третьих*, пропускная способность каналов связи между пользователями и сервером может быть ограничена, что значительно увеличивает время обслуживания каждого клиента по сравнению со временем выполнения программных сценариев (часто в несколько десятков раз). *В-четвертых*, количество одновременно работающих пользователей может быть очень большим, что требует принятия специфических мер при реализации серверных приложений.

Таким образом, актуально исследование различных способов и методов рационального использования имеющихся аппаратных ресурсов путем разработки и внедрения дополнительных программных средств, модификацией существующего программного кода, улучшающих качество информационно-вычислительной среды (производительность, требуемый объем оперативной памяти, загруженность каналов связи и т. д.).

## Методы увеличения производительности WEB-приложений

В основном, факторами, ограничивающими производительность WEB-приложения, являются техническими характеристиками среды выполнения (объем оперативной памяти, быстродействие жестких дисков и процессоров, пропускная способность каналов связи). Основными же характеристиками применимости и использования реализуемых WEB-приложений (ожидаемыми результатами) являются количество одновременно обслуживаемых пользователей и время реакции на их запросы. Влияние указанных факторов на производительность WEB-приложений очевидно.

Основным ограничивающим фактором увеличения количества одновременно обслуживаемых пользователей является объем оперативной памяти, ввиду того что, все процессы делят общее ограниченное адресное пространство. Скорость выполнения других процессов существенно замедляется, поскольку для хранения swap-файла, эмулирующего недостающую оперативную память, используется жесткий диск.

Пропускная способность и время отклика используемого канала связи существенно влияет на производительность WEB-приложений. Низкая пропускная способность канала увеличивает время доставки страниц пользователям. Это повышает количество процессов, одновременно находящихся в оперативной памяти и занятых передачей уже сформированных данных. Таким образом, процессам, порождаемым запросами новых пользователей, не хватает оперативной памяти и быстродействия процессора для обеспечения нормального (удобного пользователю) времени отклика.

Кроме того, при низком быстродействии процессора, малой производительности жестких дисков, малом объеме оперативной памяти и высокой сложности запросов к БД время обработки этих запросов становится критическим. При большом количестве пользователей может наступить такой момент, когда суммарное время выполнения всех запросов, необходимых для формирования динамических страниц, превысит время, выделенное на обслуживание, и очередь входящих запросов превысит вычислительные возможности сервера БД.

Существует множество подходов, позволяющих улучшить производительность серверных приложений. В настоящей работе исследовались следующие методы:

1. Кэширование данных на стороне сервера.

2. Кэширование страниц на стороне сервера.
3. Кэширование страниц на стороне клиента.
4. Предварительная генерация содержимого WEB-страниц в статические файлы.
5. Использование многоуровневой архитектуры FrontEnd-BackEnd [1].
6. Использование WEB-сервера, построенного по FSM (Finite State Machine) [2].
7. Сжатие передаваемых данных средствами HTTP протокола.

Заметим, что в практической реализации эти методы могут быть использованы отдельно и совместно.

Целью исследований являлась количественная оценка производительности серверных приложений по следующим критериям: количество запросов обрабатываемых сервером за одну секунду; объем трафика, создаваемого тестовыми приложениями; объем памяти, необходимый для выполнения тестовых приложений; среднее время отклика WEB-сервера.

#### Схема проведения исследования методов

Для имитационных экспериментов были использованы следующие программные средства: Apache 2.2.6 [3, 4] – в качестве process ориентированного WEB-сервера; Nginx 0.6.25 [5] – как WEB-сервер с FSM архитектурой; PHP 5.2.5 [6, 7] – язык выполнения серверных сценариев; MySQL 5.0.44 [8, 9] – в качестве сервера СУБД.

Тестовая страница формировалась PHP-сценарием и обладала следующими характеристиками:

1. Отображение 30-ти произвольных записей из таблицы тестовой БД, содержащей 100000 типовых записей. Каждая запись содержала ссылку на графический файл, заголовок и случайный текст из 1000 символов.
2. Ссылки на два CSS-файла, пять графических файлов, два файла JS-скриптов.
3. С вероятностью 0,1 при загрузке страницы происходит одно из следующих событий: создается новая запись с текстом и ссылкой на случайный графический файл, удаляется или изменяется одна запись из базы данных (БД).

Таким образом, тестовое задание эмулирует нагрузку типичной динамически формируемой WEB-страницы средней трудоемкости. При исследовании различных методов повышения производительности выполнения самих программных сценариев, из теста были исключены все статические файлы.

Для имитации клиентских обращений была выбрана программа *siege* [10], позволяющая имитировать одновременно обращение к серверу нескольких пользователей. Пример типовой команды для имитации 200 пользователей, обращающихся к серверу в течение одной минуты без задержек, показан ниже:

*siege -c 200 -d 0 -f 168.urls -i -t 1M -H «If-Modified-Since: Fri, 28 Mar 2008 08:10:30 GMT»*

Все эксперименты проводились на сервере под управлением Gentoo Linux. Технические характеристики: центральный процессор Intel Celeron 2,6 ГГц, объем оперативной памяти 2 ГБ, жесткий диск объемом 200 ГБ (частота вращения шпинделя 7200 об/мин). Тестовое приложение запускалось локально, поэтому были исключены накладные сетевые расходы и трафик, обеспечиваемый тестовым приложением, был максимально возможным.

Этапы работы типичного WEB-приложения показаны на рис. 1.

Для этапа «оформления результатов запроса в виде HTML-страницы» использовался «шаблонизатор» Smarty [11].

#### Варианты повышения производительности динамических WEB–страниц

**Без оптимизации (вариант 0).** Для получения базового варианта (эталона) производительности тестового приложения, был проведен имитационный эксперимент в стандартных условиях (без использования каких-либо методов повышения производительности).

**Кэширование данных на стороне сервера (вариант 1).** Производилось кэширование результатов запросов к БД в файлах. В результате количество обращений к базе данных уменьшилось на 70 %. Данные кэша не имели времени актуальности и уничтожались при изменении записей БД. При добавлении или удалении одной или нескольких записей очищался весь кэш. При изменении одной или нескольких записей БД удалялись только страницы, содержащие изменяемые записи. Если серверный скрипт при обращении в кэш не находил нужной ему записи, то производился запрос к БД и его результаты сохранялись в кэше.

В реальных условиях при таком подходе возникли коллизии, вызванные тем, что за время генерации новой записи кэша ее данные запрашивали еще нескольких серверных скриптов и все они параллельно пытались создать ее заново, генерируя множество параллельных одинаковых запросов к

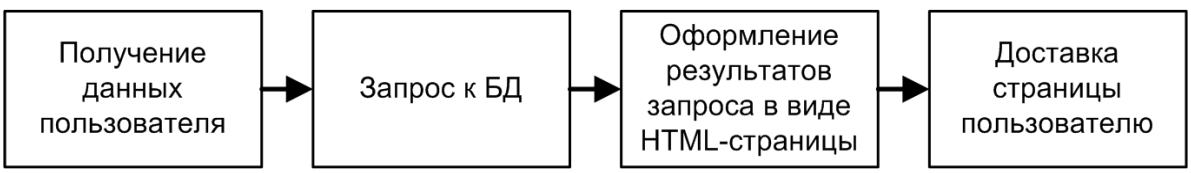


***Рис. 1.*** *Этапы работы WEB-приложения*

БД. Это приводило к повышенной нагрузке на сервер БД, т. к. все обращения к одной таблице размещались в очереди и при большом количестве запросов сервер просто не успевал выполнять весь поток обращений к БД. Это создавало значительные задержки генерации страниц клиента и как результат отказ в обслуживании при трудоемких запросах или большом количестве пользователей (для современных браузеров максимальное время ожидания данных страницы равно 30 с).

Для устранения выявленного недостатка был разработан способ, заключающийся в том, что устаревшие записи кэша не удалялись, а маркировались определенным флагом. Если серверный скрипт при обращении к соответствующей записи кэша обнаруживал необходимость обновления данных, то запись маркировалась уникальным флагом и осуществлялось обновление. Другие скрипты, при обращении к указанной записи, обнаружив, что происходит обновление, использовали для работы существующие, но устаревшие данные. Время использования устаревших данных было минимальным, т. к. каждая запись кэша при сотнях пользовательских обращений в секунду многократно обновится (более 10 раз в секунду). Подобным способом удалось уменьшить количество обращений к БД в три раза.

**Кэширование страниц на стороне сервера (вариант 2).** Если в варианте 1 кэшировался результат выполнения блока 2, то здесь кэшировалась готовая к отправке пользователю страница, полученная после блока 3. Если при обращении пользователя страница находилась в кэше, то ее содержимое отдавалось пользователю, минуя выполнение блоков 2 и 3.

**Кэширование страниц на стороне клиента (вариант 3).** Все современные броузеры и Proxy-сервера сохраняют запрашиваемые страницы в лоакльном кэше [12], и при обращении к серверу отправляют следующие заголовки: **IfModified-Since** – время последнего изменения страницы, находящейся в локальном кэше броузера, **If-None-Match –** уникальный идентификатор переданный сервером при получении к сохраненной в кэше странице. По этим заголовкам серверный скрипт определял: изменялась ли страница со времени последнего обращения данного пользователя. И если страница не изменилась, то формировался HTTP-код 304 и соединение закрывалось без отправки содержимого страницы. Подобный поход позволил значительно сократить передаваемый по сети трафик, и освободить ресурсы сервера для обслуживания вновь обратившихся пользователей.

**Предварительная генерация содержимого WEB-страниц (вариант 4).** Практика показывает, что WEB-запросы пользователей к статическим страницам обрабатываются быстрее и требуют меньше накладных расходов (память, использование БД и т. п.), чем к динамически формируемым серверными скриптами страницам. В эксперименте пользователи обращались к статическим документам *.html, а если такие файлы отсутствовали, то они формировались с помощью специального обработчика страницы ошибок (HTTP-код 404 – Документ не найден) сервера Apache [13].

Результаты выполнения тестовых приложений, реализующих все вышеприведенные варианты, показаны на рис. 2–5. Для указания вариантов на графиках используются следующие обозначения вариантов:


Общий трафик, создаваемый тестовыми приложениями – это суммарная скорость передачи данных от сервера к клиенту данных в МБ/с. Объем памяти, необходимой для выполнения тестовых приложений вычислялся суммированием объемов оперативной памяти, занимаемой всеми процессами сервера Apache и MySQL. При отсутствии пользовательских запросов объем оперативной памяти, занимаемой процессами Apache, принимался равным 102 МБ, MySQL – 240 МБ. Среднее время отклика определялось суммированием времени запросов страниц и делением на их общее количество. В некоторых случаях время формирования отдельных страниц достигало 30 с, что является максимальным временем ожидания большинства браузеров. Максимальное количество одновременно имитируемых пользователей было равно 350, т. к. объем оперативной памяти, необходимый каждому процессу *siege* для имитации отдельного пользователя, составлял порядка 8 МБ.

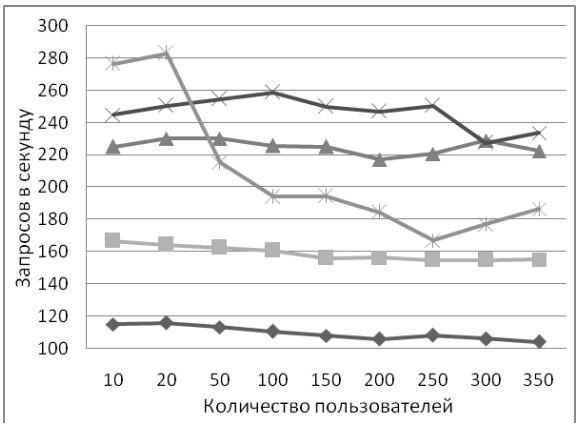


***Рис. 2.*** *Количество запросов, обрабатываемых сервером за 1 с*

Вариант 1 оказался самым медленным из-за высокой нагрузки на БД. Вариант 2 оказался несколько более быстрым, сократив частоту запросов к БД более чем в три раза, освобождая ресурсы процессора и оперативную память для обслуживания большего числа пользователей. Также проводился дополнительный эксперимент, в котором данные файлового кэша хранились на виртуальном диске. В этом случае не было достигнуто значительного преимущества перед использованием файлов на жестком диске. Это можно объяснить особенностью работы с файлами в ОС Linux, которая производит самостоятельное кэширование всех операций с жестким диском в оперативной памяти. Использование специализированного распределен-

ного кэша Memcached [14] понизило производительность на 20 %, что объясняется необходимостью взаимодействия с ним по сети. Подобное решение может быть применимо для больших проектов с общим кэшем, расположенным на отдельных серверах с большим объемом оперативной памяти. Это позволяет снизить объем требуемой памяти, т. к. исключается дублирование данных в локальных кэшах. Вариант 3 обеспечил 30-ти процентный прирост производительности благодаря исключению операций «шаблонизатора» Smarty.

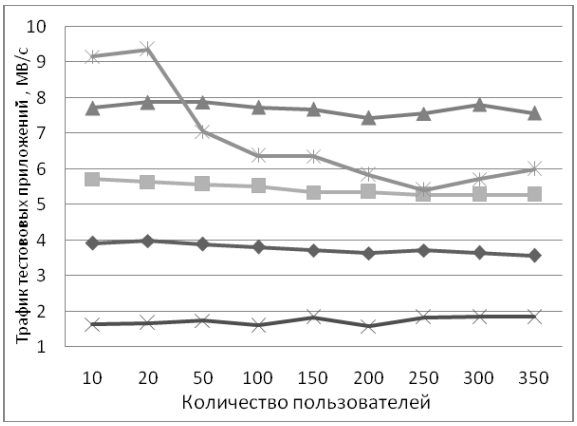


***Рис. 3.*** *Общий трафик, создаваемый тестовыми приложениями*

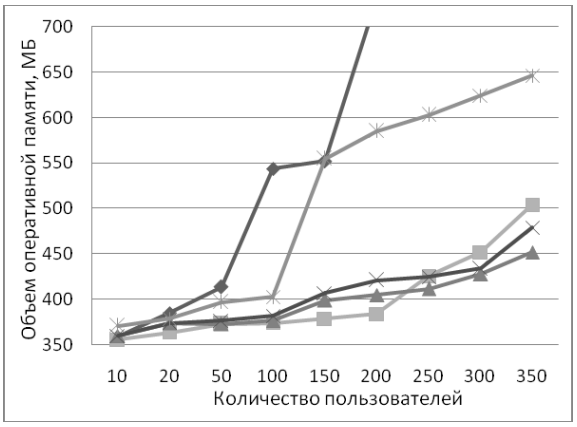


***Рис. 4.*** *Объем памяти, необходимый для выполнения тестовых приложений*

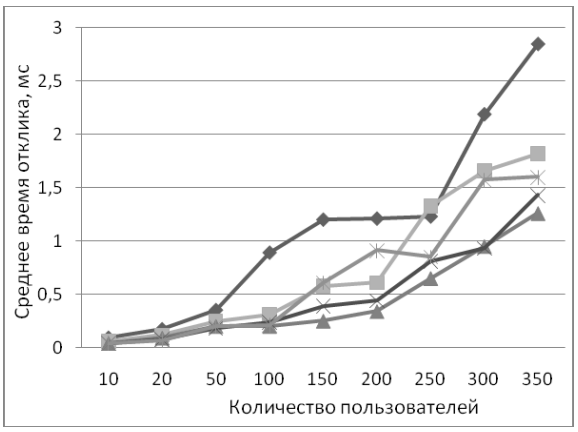


***Рис. 5.*** *Серднее время отклика WEB-сервера*

Наилучшим можно считать вариант 4. Обеспечивается значительная экономия передаваемого трафика и высокое быстродействие при минимальных накладных расходах. Вариант 5 показал наилучшие показатели при небольшом количестве пользователей, но при повышении нагрузки быстродействие стало уменьшаться из-за низкого быстродействия файловой системы при частых операциях перезаписи файлов статических страниц.

#### Варианты повышения производительности статических WEB-страниц

Все вышеперечисленные имитационные эксперименты были направлены на выявление способов формирования и передачи динамических WEB-страниц. В реальных проектах более 70 % трафика и запросов к WEB-серверу уходит на обработку статических страницы, графических и CSS-файлов, JS-скриптов и т. п. Поэтому представляет интерес оценки быстродействия WEB-сервера и с учетом передачи всех вспомогательных статических файлов.

**Без оптимизации (вариант 0′).** Все запросы обрабатываются сервером Apache. Скрипт генерации WEB-страницы работает в режиме кэширование страниц на стороне клиента (вариант 3).

**Использование многоуровневой архитектуры FrontEnd-BackEnd (вариант 5).** При многоуровневой архитектуре FrontEnd-BackEnd все запросы пользователя принимает FrontEnd-сервер, реализованный по схеме FSM Nginx. Если пользователь обращается к статическому файлу, то запрос FrontEnd-сервером обрабатывается самостоятельно. Если пользователь запрашивает динамически формируемые WEB-страницы, то Nginx формирует запросы к BackEnd-серверу Apache и, получив от него данные WEB-страниц, возвращает их пользователю. 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.

**Использование WEB-сервера построенного по FSM (вариант 6).** В качестве сервера, выполненного по архитектуре FSM, использовался Nginx. Архитектура сервера Nginx не позволяет самостоятельно осуществлять вызов интерпретатора PHP для формирования содержимого WEB-страниц, поэтому он используется как FrontEnd, а в качестве BackEnd используется Fast-CGI сервер [15]. Данный подход позволяет отказаться от использования ресурсоемкого сервера Apache, заменяя его запуском всего нескольких процессов PHP. При этом исключаются накладные расходы на загрузку и выгрузку интерпретаторов в случае обычного CGI режима сервера Apache.

**Сжатие передаваемых данных средствами Apache (вариант 7).** Для сжатия передаваемых данных сервера Apache применяется модуль mod_deflate. Сжатию подлежат результаты выполнения PHP-сценариев и CSS- и JS-файлы, что позволяет уменьшить их размер от 30 до 90 %.

**Сжатие передаваемых данных средствами Nginx (вариант 7′).** Вариант 7 модифицируется заменой Apache сервером Nginx, выполненным по FSM-архитектуре, который дополнительно сжимает передаваемые данные, используя метод gzip.

Результаты выполнения тестовых приложений, реализующих все вышеприведенные варианты, показаны на рис. 6–9. Для указания вариантов на графиках используются следующие обозначения вариантов:

– 0′, – 5, – 6, – 7, – 7′.

Заметим, что на рис. 6 показано общее число запросов к серверу для всех файлов, а каждый пользователь запрашивает по 5 файлов при обращении к серверу. Поэтому количество обслуживаемых пользователей будет, примерно, на 30 % ниже, чем количество обрабатываемых файлов. Прирост производительности для сервера Nginx обусловлен более эффективной передачей именно статических файлов. Время генерации динамических WEB-страниц во всех случаях практически одинаково.

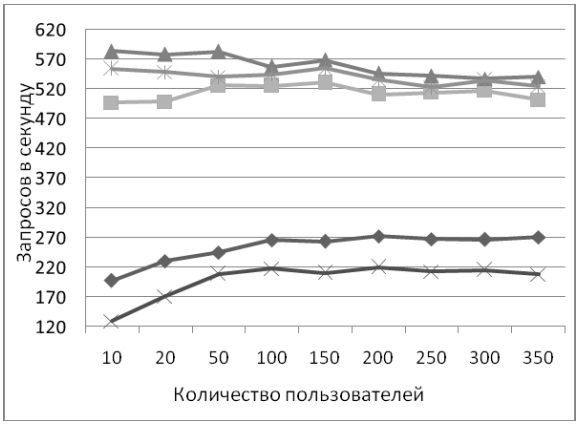


***Рис. 6.*** *Количество запросов, обрабатываемых сервером за 1 с*

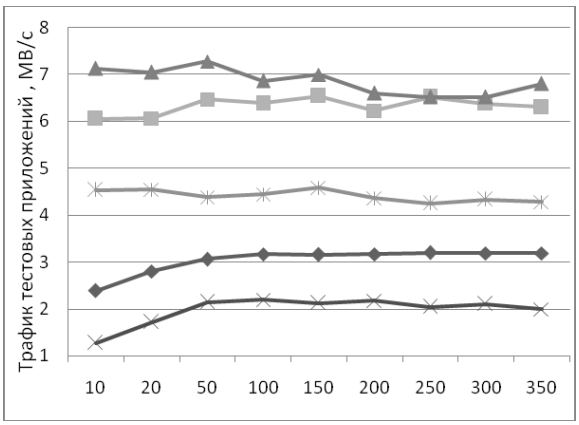


***Рис. 7.*** *Общий трафик, создаваемый тестовыми приложениями*

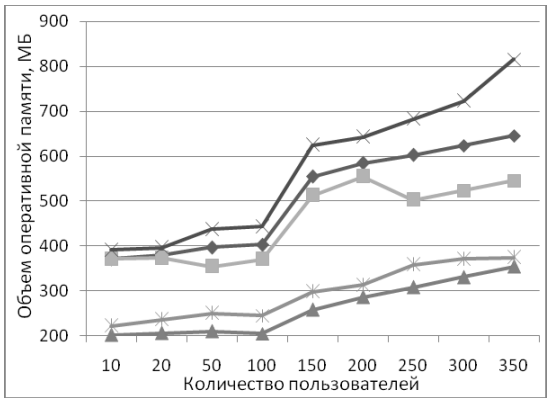


***Рис. 8.*** *Объем памяти, необходимый для выполнения тестовых приложений*

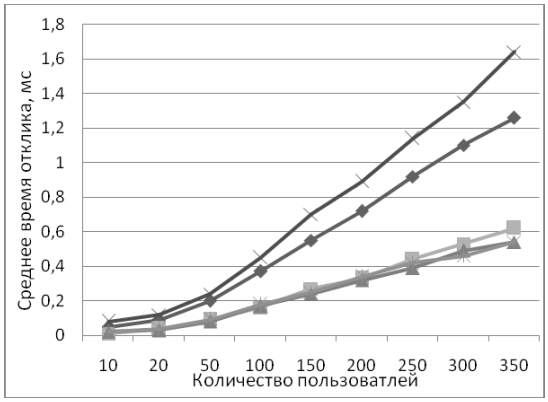


***Рис. 9.*** *Серднее время отклика WEB-сервера*

Использование двухуровневой схемы обработки пользовательских запросов позволяет повысить количество одновременно обслуживаемых пользователей более, чем в два раза. Отказавшись от сервера Apache в пользу сервера Nginx и Fast-CGI, можно значительно снизить расходы оперативной памяти. Сжатие передаваемых пользователю данных при использовании сервера Apache снижает количество одновременно обслуживаемых пользователей на 10...20 %. При использовании Nginx потери производительности практически незаметны. Таким образом, оптимальным и рациональным с точки зрения практического применения можно считать вариант 8, обеспечивающий самую высокую производительность, низкое потребление системных ресурсов и хорошее сжатие передаваемых данных.

#### Заключение

Проведены имитационные эксперименты по исследованию быстродействия WEB-приложений при различных вариантах организации взаимодействия пользователя с сервером, сервера с интерпретатором PHP и статическими файлами, FrontEnd- и BackEnd-серверов. Показано, что наиболее предпочтительным с точки зрения практического применения является вариант реализации WEB-приложений с кэшированием динамических страниц на стороне клиента, запуском PHP в режиме FastCGI и обработкой статических файлов FrontEnd-сервером Nginx.

**СПИСОК ЛИТЕРАТУРЫ**

1. Front-end and back-end – Режим доступа: http://en.wikipedia.org/wiki/Front-end_and_back-end – Загл. с экрана.
2. FAQ appendix 1: как писать сервера – fido7.ru.unix.prog. Группы Google – Режим доступа: http://groups.google.ru/group/fido7.ru.unix.prog/browse_thread/thread/e8f8edf4f2f2447b/ – Загл. с экрана.
3. The Apache HTTP Server Project – Режим доступа: http://httpd.apache.org/ – Загл. с экрана.
4. Уэйнрайт П. Apache для профессионалов. – М.: Wrox Press Ltd, 2001. – 474 с.
5. Nginx – Режим доступа: http://sysoev.ru/nginx/ – Загл. с экрана.
6. PHP: Hypertext Preprocessor – Режим доступа: http://www.php.net/ – Загл. с экрана.
7. Веллинг Л., Томсон Л. Разработка Web-приложений с помощью PHP и MySQL. – М.: Вильямс, 2007. – 880 с.
8. The world's most popular open source database – Режим доступа: http://www.mysql.com/ – Загл. с экрана.
9. Дюбуа П. MySQL. – М.: Вильямс, 2007. – 1168 с.
10. Siege – Режим доступа: http://www.joedog.org/JoeDog/Siege – Загл. с экрана.
11. Smarty : Template Engine – Режим доступа: http://www.smarty.net/ – Загл. с экрана.
12. Caching Tutorial for Web Authors and Webmasters – Режим доступа: http://www.mnot.net/cache_docs/ – Загл. с экрана.
13. Обработка ошибки 404 – Режим доступа: http://lekx.ru/modules/myarticles/article.php?storyid=515 – Загл. с экрана.
14. Memcached: a distributed memory object caching system – Режим доступа: http://www.danga.com/memcached/ – Загл. с экрана.
15. FastCGI – Режим доступа: http://ru.wikipedia.org/wiki/FastCGI – Загл. с экрана.

*Поступила 23.04.2008 г.*

**Ключевые слова:**

WEB-приложение, быстродействие, кэширование, динамическая страница, статический файл, производительность.

УДК 002.53:004.89

# АВТОМАТИЗАЦИЯ СБОРА ОНТОЛОГИЧЕСКОЙ ИНФОРМАЦИИ ОБ ИНТЕРНЕТ-РЕСУРСАХ ДЛЯ ПОРТАЛА НАУЧНЫХ ЗНАНИЙ

Ю.А. Загорулько

Институт систем информатики им. А.П. Ершова СО РАН, г. Новосибирск
E-mail: zagor@iis.nsk.su

*Предлагается подход к автоматизации сбора онтологической информации об Интернет-ресурсах, релевантных предметной области портала научных знаний. Специальная подсистема выполняет поиск ресурсов (документов), оценку их релевантности, содержательный анализ, индексирование и классификацию с использованием предметного словаря и онтологии предметной области.*

## Введение

Для решения задачи сведения ресурсов, относящихся к одной области знаний в единое информационное пространство, обеспечения возможности открытого и удобного доступа к ним, а также поддержки их целостности нами была предложена концепция специализированных Интернет-порталов знаний [1]. Основу портала знаний составляет онтология и соотнесенное с ней описание соответствующих сетевых ресурсов.

Особенность предложенной концепции состоит в том, что портал знаний обеспечивает доступ не только к собственным информационным ресурсам, но и поддерживает навигацию по заранее размеченным (проиндексированным) ресурсам, размещенным в сети Интернет. При этом информация о ресурсах накапливается коллекционером онтологической информации, т. е. специальной подсистемой портала знаний, осуществляющей сбор, анализ, оценку релевантности Интернет-ресурсов, а также их автоматическое индексирование и классификацию.

Коллекционер онтологической информации о ресурсах фактически выполняет функцию извлечения знаний и данных из сети Интернет [2].

В этой статье подход к автоматизации сбора онтологической информации об Интернет-ресурсах рассматривается на примере портала знаний, служащего для поддержки научных исследований.

## Система знаний портала

Базис системы знаний портала (см. рис. 1) составляет онтология, которая не только обеспечивает формальное представление системы понятий предметной области (ПО) портала, но и интеграцию в его информационное пространство релевантных информационных ресурсов.

Формально онтология портала знаний может быть описана семеркой вида:

$$O = \langle C, A, R_C, T, D, R_A, F \rangle,$$

где $C$ – множество классов, описывающих понятия некоторой предметной или проблемной области;