



1С-БИТРИКС

Компания «1С-Битрикс» Системы управления веб-проектами

Тел.: (495) 363-37-53; (4012) 51-05-64; e-mail: info@1c-bitrix.ru, <http://www.1c-bitrix.ru>

1С-Битрикс: Управление сайтом

Курс для хостеров

Конфигурирование веб-систем



1С-БИТРИКС



Содержание

Глава 1. Конфигурирование веб-систем для оптимальной работы.....	3
ОСОБЕННОСТИ ВЕБ-ПРИЛОЖЕНИЙ	3
PHP-приложения	4
Базы данных	5
Какие проекты можно назвать большими?	7
ПОЧЕМУ УМИРАЮТ САЙТЫ?	8
Передача данных клиенту	9
Производительность PHP, базы данных и статика	11
Причины "умирания" сайтов	11
ДВУХУРОВНЕВАЯ КОНФИГУРАЦИЯ ВЕБ-СЕРВЕРА FRONT-END И BACK-END.....	13
Front-end.....	13
Back-end и порядок взаимодействия.....	15
Стабилизируем Back-end по расходу оперативной памяти	17
Производительность PHP.....	20
Сжатие страниц	22
Дополнительные рекомендации для двухуровневой конфигурации.....	23
Достигнутые результаты.....	24
ПРИМЕР НАСТРОЙКИ ДВУХУРОВНЕВОЙ АРХИТЕКТУРЫ	25
Настройка веб-сервера Apache	25
Пример: число процессов веб-сервера.....	26
Настройка Front-end NGINX	27
Отдача графики напрямую NGINX	34
ОПТИМИЗАЦИЯ БАЗЫ ДАННЫХ.....	34
Основные принципы.....	35
Постоянное соединение с базой данных	35
Настройка базы данных MySQL.....	36
Настройка базы данных Oracle	40
Пример-упражнение	42



Глава 1. Конфигурирование веб-систем для оптимальной работы

Типовые настройки серверного программного обеспечения рассчитаны на минимальное оборудование и статические HTML-приложения. Внесение некоторых конфигурационных изменений в серверное ПО позволяет в несколько раз увеличить производительность системы в целом, сократить время генерации страниц, увеличить устойчивость системы к пиковым нагрузкам.

В этой главе приведены рекомендации по настройке серверного ПО, которые выполняются сотрудниками компании «Битрикс» при конфигурировании проектов с посещаемостью более 3-10 тысяч уникальных пользователей в день, либо при недостаточности системных ресурсов для обработки меньшей нагрузки.

Все рекомендации относятся в основном к UNIX-системам или Windows-системам, использующим веб-сервер *Apache*.

Рекомендованные решения не являются единственными возможными. Предполагается, что данные инструкции послужат руководством для создания и доработки данных рекомендаций в соответствии с имеющимися ресурсами, оборудованием, конфигурациями из нескольких серверов.

⚠ Внимание! Все рекомендации подразумевают, что выполнены все обязательные требования и, по возможности, рекомендуемые установки на странице **Проверка сайта** (Инструменты > Проверка сайта) в Административном разделе продукта.

⚠ Примечание: Помимо изучения данной главы рекомендуем обратиться к опыту разработчиков сайтов на платформе *Bitrix Framework*. В частности можно использовать методики описанные в группе [Оптимизация веб-проектов](#).

Особенности веб-приложений

Веб-приложения отличаются от программ, работающих под управлением операционной системы, например, *Windows*.

Основное отличие – в задачах. Windows-приложение ожидает действий пользователя и работает постоянно, пока оно запущено. В случае веб-приложения посетитель сайта видит уже результат выполнения программы, которая, к этому времени, уже завершила свою работу и освободила ресурсы сервера для обслуживания других посетителей. При открытии другой (или этой же) страницы выполняется новый экземпляр программы, который, после выдачи данных пользователю, также завершается.



Таким образом, в операционной системе вам принадлежат все ресурсы компьютера, и почти неограниченное время, а в Веб вам принадлежит лишь совсем чуть-чуть памяти (обычно 32-256 мб) и немного времени (обычно не более 30-90 секунд). Кроме того, спецификой любого более-менее посещаемого проекта является необходимость обслуживать различное число посетителей в разное время суток, и простым увеличением ресурсов сервера добиться необходимого качества очень сложно.

В связи с этим конфигурирование веб-систем для работы с проектами на платформе *Bitrix Framework* имеет свои особенности.

PHP-приложения

Раньше HTML-проекты представляли собой обычный статический документ, который содержал специальные теги разметки. С точки зрения администратора веб-сайта, данные документы считываются веб-сервером и передаются по TCP/IP протоколу. Не выполняется никаких дополнительных приложений, не потребляется дополнительная память, не используется база данных. Это очень просто и удобно, но этого уже недостаточно для современных проектов.

Наличие программной среды в системе управления сайтом позволяет создавать динамические интернет-проекты, оперативно и легко управлять информацией, анализировать эффективность проектов, менять содержимое вашего сайта в зависимости от тех или иных потоков посетителей и многое другое. Можно сказать, что все современные веб-проекты создаются с использованием того или иного языка программирования.

Программный продукт *"1С-Битрикс: Управление сайтом"* разработан на языке программирования PHP. На данный момент поддерживается PHP версии 5.0.0 и выше.

PHP: (англ. PHP: Hypertext Preprocessor — «PHP: препроцессор гипертекста») — скриптовый язык программирования общего назначения с открытым исходным кодом, применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков программирования, применяющихся для создания динамических веб-сайтов.

Обратите внимание на отличие PHP от скриптов, написанных на других языках, например, на Perl или C: вместо того, чтобы создавать программу, которая занимается формированием HTML-кода и содержит бесчисленное множество предназначенных для этого команд, создается HTML-код с несколькими внедренными командами PHP. Код PHP отделяется специальными начальным и конечным тегами, которые позволяют процессору PHP определять начало и конец участка HTML-кода, содержащего PHP-скрипт.

Значительным отличием PHP от какого-либо кода, выполняющегося на стороне клиента, например, JavaScript, является то, что PHP-скрипты выполняются на сервере. Если бы у



вас на сервере был размещен PHP-скрипт, клиент получил бы только результат выполнения скрипта, причем он не смог бы выяснить, какой именно код выполняется. Вы даже можете сконфигурировать свой сервер таким образом, чтобы HTML-файлы обрабатывались процессором PHP, так что клиенты даже не смогут узнать, получают ли они обычный HTML-файл или результат выполнения скрипта.

С точки зрения администратора, принципиально важен тот факт, что PHP исполняется на сервере и является интерпретируемым языком. Интерпретируемый язык - это значит, что сколько бы раз вы не запрашивали страницу на языке программирования PHP, она будет каждый раз обрабатываться на сервере специальным интерпретатором PHP, будет проверяться синтаксис языка, правильность конструкций и вызовы функций, и только после этого, код PHP будет исполняться.

Таким образом, в отличие от обычных HTML-страниц, сайты на PHP потребляют больше оперативной памяти на один процесс веб-сервера. Веб-сервер может приступить к передаче страницы клиенту в большинстве случаев только после того, как страница готова и PHP выполнил свои инструкции. Это приводит к некоторому замедлению по сравнению с HTML-страницами в выдаче содержимого клиенту.

⚠ Внимание! *Задача администратора сервера – настроить программное обеспечение для минимизации нагрузки на сервер, организации стабильной работы сервера и ускорения работы сайтов.*

Базы данных

Дополнительным фактором, влияющим на производительность системы, является правильное использование баз данных. Все современные проекты для отображения динамической информации (новостей, каталога товаров и др.) используют СУБД (системы управления базами данных), а СУБД, в свою очередь, требует правильной настройки своих параметров.

Соединение с базой данных

База данных представляет собой независимое клиент-серверное приложение, которое запускается и работает в операционной системе. Внешние приложения соединяются с базой данных через TCP/IP или внутренние потоки, направляют SQL-запросы и получают в ответ от базы данных необходимые данные.

Возможно два типа соединения с базой данных для веб-приложения:

- обычное соединение;
- постоянное соединение (Persistent).



Обычное соединение устанавливается каждый раз во время выполнения страницы при первом обращении к базе данных. Установленное соединение освобождается (в большинстве случаев и закрывается) после завершения страницы.

Постоянное соединение (функции PHP обычно называются `*_pconnect`) устанавливается один раз при первом обращении к базе данных и при повторных обращениях, даже из других страниц, используются уже открытые соединения к базе данных.

Учитывая, что открытие соединения к базе данных - процесс относительно дорогой по ресурсам и времени (происходит установление TCP/IP соединения, выделяются буферы памяти, проводится проверка и авторизация, настраиваются перекодировщики и выполняется целый ряд других функций), использование постоянных соединений является предпочтительным, если это не приводит к превышению числа одновременных соединений с базой данных.

⚠ Примечание: Устанавливая соединение с базой данных, при указании параметру `server` значения `localhost` или `localhost:port`, PHP в большинстве своем будет пытаться соединиться с локальным сокетом без использования TCP/IP. Если вы все же хотите использовать TCP/IP, используйте адрес `127.0.0.1` вместо `localhost`.

Соединение без использования TCP/IP дает наивысшие результаты по производительности, особенно при передаче больших объемов информации из базы данных. Но если база данных расположена на другой машине и избежать соединения по TCP/IP не удастся, использование постоянного соединения становится еще более выгодным.

В большинстве случаев база данных работает на той же машине, что и PHP-приложение и веб-сервер. Но вполне возможна ситуация, когда база данных установлена на соседней машине или даже у другого провайдера. "1С-Битрикс: Управление сайтом" устанавливает соединение с сервером базы данных через стандартные библиотеки, которые идут в поставке языка программирования PHP.

В "1С-Битрикс: Управление сайтом" тип соединения с базой данных устанавливается в файле: `/bitrix/php_interface/dbconn.php`.

Пример содержимого файла:

```
define("DBPersistent", true);

$DBType = "mysql";

$DBHost = "localhost:31006";

$DBLogin = "root";

$DBPassword = "";
```



```
$DBName = "bsm_demo";  
$DBDebug = false;  
$DBDebugToFile = false;
```

Используйте константу **DBPersistent**, чтобы установить тип соединения с базой данных.

Для отключения постоянного соединения используйте:

```
define("DBPersistent", false);
```

Использование постоянных соединений может потребовать некоторой настройки Apache и MySQL. Убедитесь, что вы не превысите максимальное число дозволенных соединений. Читайте дополнительную информацию в документации PHP относительно используемой вами базы данных.

Используемые базы данных

"1С-Битрикс: Управление сайтом" поддерживает следующие типы баз данных:

- Бесплатные
 - [MySQL](#)
 - [MSSQL Express 2005](#)
 - [OracleXE](#)
- Коммерческие
 - [Oracle](#)
 - [MSSQL 2005](#)

Какие проекты можно назвать большими?

Комплексное сочетание целого ряда факторов может привести к тому, что проект получает название «большой» и требует определенного конфигурирования и отношения:

- большая посещаемость проекта в среднесуточном выражении;
- высокие пиковые нагрузки;
- невозможность кэшировать страницы в силу сложной бизнес-логики;
- большие интерактивные проекты: форумы, блоги, журналы;
- индивидуальные страницы для отдельных пользователей;
- большие объемы данных;



- недостаточность аппаратных ресурсов по отношению к предыдущим факторам.

⚠ Примечание: даже если у вас сравнительно небольшой проект, лучше настроить веб-сервер правильно, чтобы в случае неожиданного роста посещаемости, например, при рекламной кампании, сайт стабильно работал.

Почему умирают сайты?

Прежде чем переходить к выбору рекомендаций по конфигурированию, необходимо внимательнее изучить основные причины, которые приводят к нестабильной работе веб-сервера или даже к полному отказу в обслуживании. Четкое понимание причин позволит вам вдумчиво подходить к рекомендациям и максимально эффективно использовать все имеющиеся у вас аппаратные ресурсы.

Как работает веб-сервер

Рассмотрим типичную схему работы веб-сервера:



При запросе страницы сайта происходит обращение к веб-серверу, который запускает интерпретатор **PHP** для выполнения скрипта. Далее программа выполняется, взаимодействует с СУБД и отдает результат выполнения клиенту. Кроме того, веб-сервер отдает клиенту сопутствующие файлы – картинки, документы, **css** файлы и другую статическую информацию.

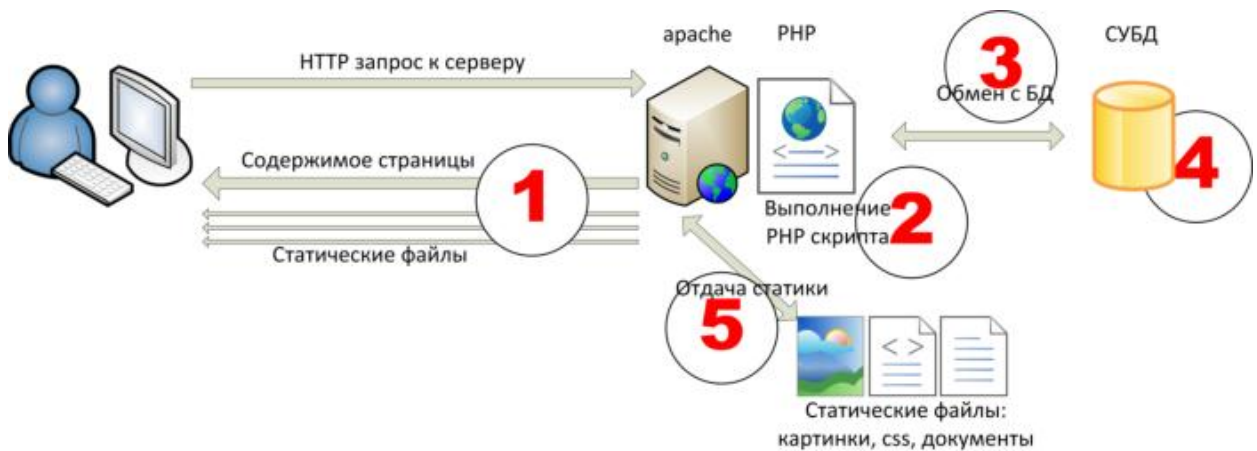
В современных сайтах при открытии каждой страницы клиенту отдается несколько десятков файлов – от действительно результата выполнения **PHP** программы до статических картинок.

Важно отметить, что для отдачи каждого файла используется, как правило, отдельный процесс **apache**, который занимает память веб-сервера. В 2012 году средний размер процесса **apache** в памяти – от 64 до 500 мб, и очень легко занять всю оперативную память процессами веб-сервера.

Узкие места



Выделим несколько узких мест в приведенной схеме:



1. Передача данных клиенту. Решение проблем медленных каналов
2. Производительность PHP. Уменьшение времени выполнения скрипта.
3. Обмен с базой данных.
4. Настройка СУБД на максимальную производительность
5. Отдача статики. Решение проблемы медленных каналов.

Передача данных клиенту

Работа веб-сервера *Apache* организована таким образом, что процесс веб-сервера занимает память системы, пока полностью не завершит отдачу файла. Таким образом, в случае если у клиента медленный канал связи, то даже при быстром выполнении PHP скрипта сервер будет не справляться даже с маленькой нагрузкой.

Дополнительно, при отдаче статических файлов с помощью *Apache* ценная память веб-сервера используется расточительно – под выдачу статики, для обработки которой не требуется выполнять программы на PHP.

В течение всего времени передачи страницы клиенту веб-сервер будет держать в памяти практически бездействующий процесс *Apache*, который будет только дожидаться завершения передачи данных, но не сможет высвободить память и высвободиться сам, чтобы обработать другой запрос.

Очень часто администраторы не отдают себе отчет в том, насколько данный фактор влияет на стабильность системы и эффективное использование оперативной памяти.

Давайте сделаем простой расчет. Рассмотрим две системы: **А** и **Б**.



В системе **А** время генерации страниц составит 0.1 секунды, а время передачи страницы клиенту в среднем будет составлять всего 5 секунд (в реальной жизни среднее значение окажется еще больше). В системе **Б** мы будем считать время генерации страниц равным 0.1, а время передачи страницы пользователю равным нулю. Предположим, что на каждый сайт поступает по 100 запросов в секунду.

Система А

Обработка 100 запросов в секунду потребует одновременной работы 500 самостоятельных процессов веб-сервера. "Почему?" - спросите вы. А как же иначе, если даже обработав запрос за 0.1 с., наши процессы, получается, еще не способны обрабатывать другие запросы, а будут висеть в памяти и просто дожидаться, пока пользователи в течение 5 секунд будут получать страницу. На четвертой секунде веб-сервер получит еще 100 запросов и должен будет запустить еще 100 процессов. Соответственно, на пятой секунде в памяти должно находиться 500 процессов и только с этого момента процессы первой секунды начнут высвобождаться и обрабатывать новые запросы.

Таким образом, система А для нормальной работы будет запускать порядка 500 процессов, что потребует от нас в лучшем случае 32 Гб оперативной памяти. Обратите внимание, что даже если бы время генерации страниц было равно 0.001 секунды, это бы не увеличило производительность системы, так как процессы ожидают передачи данных пользователям на медленных каналах, а не времени генерации страниц. Т.е. **производительность системы А никак не связана с производительностью РНР и продукта.**

Система Б

За первую секунду на сервер поступит 100 запросов. Для обработки 100 запросов нам потребуется всего 10 процессов. Один процесс обрабатывает один запрос за 0.1 секунды. Как мы договорились для системы Б, время передачи страницы пользователю будет равно нулю. Т.е. за 1 секунду, один процесс веб-сервера способен обработать 10 запросов пользователей! К завершению первой секунды, все запросы будут обработаны всего 10 процессами и ко второй секунде все эти процессы будут свободны и готовы обрабатывать следующие запросы. Так же случится и на третьей секунде, и через час.

Таким образом, система Б для нормальной работы будет запускать всего 10 процессов, что потребует от нас порядка 640М оперативной памяти. И очень важно отметить, что уменьшение времени генерации страниц до 0.01 секунды позволит реально увеличить производительность системы в 10 раз, и нам будет достаточно уже только 1 процесса для обработки всех 100 запросов в секунду. **Производительность системы Б зависит только от производительности РНР и продукта и не зависит от медленных каналов.**

Это очень наглядный пример, который демонстрирует, насколько велико влияние медленных каналов пользователей на общую производительность веб-системы. Веб-сервер очень неэффективно расходует оперативную память на медленных каналах.



Немного забегаая вперед отметим, что существуют методы, которые позволяют построить систему очень близкую к модели Б и полностью снять зависимость веб-системы от медленных каналов и увеличить производительность и устойчивость в несколько раз.

Производительность PHP, базы данных и статика

Рассмотрим остальные "узкие места".

Производительность PHP. Уменьшение времени выполнения скрипта.

До 60% рабочего времени веб-сервера тратят на повторную компиляцию PHP-кода перед исполнением. Ключевой способ снизить нагрузку на процессор – использовать прекомпиляторы PHP-кода, которые снижают нагрузку на систему за счет перевода PHP из интерпретируемого в частично компилируемый язык.

Обмен с базой данных

Если ваш проект работает на выделенном сервере, вы можете настроить взаимодействие с СУБД для уменьшения затрат времени на установку соединений. Для этого используется постоянные соединения с базой данных, что позволяет уменьшить общее время выполнения скрипта.

Настройка СУБД на максимальную производительность

Установки по умолчанию для большинства СУБД рассчитаны на некие средние величины. Для повышения производительности необходимо оптимизировать эти настройки.

Отдача статика. Решение проблемы медленных каналов.

Также как и при [передаче данных клиенту](#), для повышения производительности хостинга необходимо решить вопрос с медленными каналами связи, но уже применительно к статической информации. Все сказанное для первого пункта применимо и для отдачи статика.

Причины "умирания" сайтов

Таким образом, при больших нагрузках проект может прекратить нормальное функционирование по следующим причинам:

- **Нехватка оперативной памяти для нормальной одновременной работы процессов веб-сервера и базы данных.**



В большинстве систем на каждый запрос к сайту открывается отдельный процесс веб-сервера. Обычный размер процесса *Apache* с подключенным PHP-модулем и работающим приложением может составить порядка 64-500 мегабайт. В результате пиковых нагрузок происходит одновременный запуск очень большого числа процессов (иногда больше нескольких сотен процессов). И как следствие, начинается свопирование процессов, а это неизбежно сказывается на производительности базы данных, и производительность всей системы в целом резко снижается.

- **Нехватка процессорных ресурсов для одновременного выполнения процессов и обеспечения адекватного для пользователя времени реакции.**

Данная ситуация может возникнуть в том случае, если в результате большого числа запросов к вашему серверу число одновременно выполняемых запросов превысит процессорные мощности сервера. И даже если у вас достаточно оперативной памяти и первая проблема не проявила себя, вы можете обнаружить, что система перестала адекватно отвечать на запросы, время выполнения страниц увеличилось в несколько раз, база данных перегружена очень большим числом запросов. Все это может привести к тому, что все без исключения пользователи не смогут работать с сервером.

- **Недостаточная производительность базы данных при одновременных конкурентных запросах, невозможность полностью использовать ресурсы сервера.**

Эта ситуация очень часто возникает при работе с MySQL. Надо отметить, что обычно MySQL использует формат таблиц MyISAM. Это очень простой и эффективный вариант работы, но, к сожалению, при большом числе одновременных запросов такая база данных становится критически узким местом в производительности системы в целом. Во время вставки данных, обновления и некоторых других запросах, происходит эксклюзивное блокирование таблиц и, как следствие, все запросы выполняются только последовательно, а не одновременно. В результате, при росте нагрузки, время генерации страниц возрастает необоснованно резко и в итоге становится неприемлемо большим. Менее всего подвержены подобным проблемам проекты, использующие Oracle или MSSQL, MySQL с форматом таблиц InnoDB.

- **Общая несбалансированность веб-системы при пиковых нагрузках и быстрая регрессия производительности даже при незначительных стрессах.**

При пиковых нагрузках вся система испытывает перегрузки. В дополнение к перечисленным проблемам возможно возникновение проблем в дисковой подсистеме. В итоге, если под одной из составляющих начинается потеря производительности, то существенно падает производительность всей системы, начинается падение производительности в других частях и, в итоге, еще больше снижается работоспособность, производительность системы регрессирует и иногда наступает полная остановка в работе.



Двухуровневая конфигурация веб-сервера Front-End и Back-End

Наилучшим способом для устранения перечисленных проблем является создание двухуровневой системы **Front-End + Back-End** для обработки запросов.

Front-End – публичная часть проекта, обеспечивающая прием запросов от пользователей, трансляцию запросов к Back-End и выдачу непосредственного содержимого пользователю.

Back-End – исполнительная часть системы, которая обеспечивает выполнение PHP-скриптов, формирование контентных страниц и работу бизнес-логики приложений.

Рассмотрим пример конфигурации более детально. После применения оптимизации у нас должна получиться система примерно следующего вида:



Front-end

Начнем с построения **Front-end** системы и определения целей и задач, которые будет решать данная часть двухуровневой архитектуры.

В качестве **Front-End** сервера можно использовать [NGINX](#), [SQUID](#), [OOPS](#) или любой аналогичный продукт.

NGINX представляет собой очень компактный и быстрый веб-сервер (HTTP-сервер). Он потребляет очень мало оперативной памяти, умеет самостоятельно обслуживать статические запросы и выполнять акселерированное проксирование без кэширования статических объектов. Например, если запрашивается графический объект, NGINX самостоятельно выполняет считывание данных с диска и передает файл пользователю.

SQUID и *OOPS* - это классические прокси-сервера, которые выполняют проксирование запросов и чаще всего кэшируют статические запросы, сохраняя в кеше или у себя на



диске копии статических запрашиваемых объектов в течение определенного интервала времени.

Наилучшие практические результаты получены при использовании NGINX. Но использование кэширующих прокси-серверов также возможно с получением отличных результатов.

Создавая двухуровневую архитектуру мы выставляем перед пользователем **Front-end** систему - легкий веб-сервер или прокси-сервер, который принимает все запросы от пользователей, исполняет все запросы, которые возможно обработать самостоятельно без обращения к **Back-End**. Если используется NGINX или аналогичный продукт, все статические объекты напрямую считываются с диска и передаются клиенту. Если используется кэширующий прокси-сервер, статические объекты, графические файлы и таблицы стилей запрашиваются с **Back-end** только при первом обращении к ним. После этого файлы хранятся в кэш **Front-end** в соответствии с политикой кэширования и отдаются пользователям без обращения к **Back-end**.

Основные цели, которых мы добиваемся созданием первого **Front-end** уровня:

- **минимизация числа запросов, поступающих к Back-end веб-серверу.** Надо добиться ситуации, когда **Front-end** будет обращаться к **Back-end** процессам только для получения содержимого PHP-страницы. Все запросы к статическим объектам должны обрабатываться легкими процессами **Front-end** самостоятельно или при использовании кэширующего прокси-сервера на всех запросах, кроме первого.

⚠ Совет: проверьте лог **Back-end** веб-сервера, чтобы убедиться, что вы настроили все правильно и действительно исключили лишние запросы. В логе должны быть представлены только страницы PHP, другие запросы не должны проходить к **Back-end** системе и не будут отражаться в лог файле.

- **минимальное потребление оперативной памяти при обработке статических запросов.** Число статических запросов существенно превосходит число запросов к PHP-страницам. Процессы **Front-end** потребляют в среднем 2-5М оперативной памяти и очень незначительно увеличиваются при обработке статических документов. Это позволяет в разы уменьшить потребление оперативной памяти всей системой в целом.
- **защита системы от фактора медленных каналов.** Как для статических запросов, так и для HTML-страниц, полученных от PHP-процессов **Back-end** веб-сервера, процессы **Front-end** могут передавать страницу пользователю достаточно долго, потребляя при этом очень мало оперативной памяти. Таким образом, **Front-end**, транслируя запрос к **Back-end**, получает ответ, высвобождает процессы **Back-end** для обработки других запросов, а сам передает страницу пользователю, снимая фактор медленных каналов. Система приближается к идеальной системе Б в примере, приведенном в уроке [Передача данных клиенту](#).

⚠ Внимание! Убедитесь, что буферы **Front-end** достаточны, чтобы без ожидания на передаче принять от **Back-end** всю страницу. Т.е. фактически



буфер в идеале должен быть равен размеру самой большой страницы у вас на сайте.

- **механизм защиты Back-end от большого числа запросов** за счет ожидания свободных процессов **Back-end** для продолжения работы. Проверьте и убедитесь, что **Front-end** будет ожидать 5-10-15 минут, пока не высвободятся процессы **Back-end**. Наличие такого механизма позволит нам настроить **Back-end** так, чтобы полностью стабилизировать систему и подготовиться к стрессовым нагрузкам.

Как вы видите, появление **Front-end** позволяет нам снять целую категорию рисков и подготовить систему к дальнейшей работе

⚠ Внимание! Если вы используете кэширующий прокси-сервер в качестве **Front-End**, обязательно настраивайте время кэширование документов. Графические файлы и таблицы стилей, XML-файлы и другие статические объекты с веб-сервера должны запрашиваются только в соответствии с политикой кэширования. После этого файлы хранятся в прокси-сервере и отдаются пользователям без обращения к **Back-End** и Apache. Рекомендуется настраивать время кэширования для графических файлов на 3-5 дней. Пример настройки кэширования через файл `.htaccess` в корне веб-сервера:

```
ExpiresActive on
ExpiresByType image/jpeg "access plus 3 day"
ExpiresByType image/gif "access plus 3 day"
```

Для работы этого примера необходимо, чтобы веб-сервер позволял переопределение переменных через файл `.htaccess` и модуль `mod_expires` был установлен. В некоторых случаях на **Front-end** политика кеширования настраивается независимо от настроек **Back-end**.

Таким образом, **Front-end** будет кешировать все графические изображения. Запросы к контентным страницам не будут кешироваться и будут перенаправляться к **Back-end**.

Back-end и порядок взаимодействия

Back-end представляет собой обычный веб-сервер *Apache*, который исполняет PHP-приложения.

Back-end готов исполнять запросы на графические и статические документы, если вы используете кэширующий прокси-сервер для **Front-end**. Но очень важно, чтобы число запросов к статическим элементам через **Back-end** было минимальным и 99% запросов приходилось на выполнение именно PHP-страниц. Не забывайте, что использование **Back-end** для обработки статических запросов обходится очень дорого.



Конфигурируя **Back-end**, можно добиться значительного выигрыша в производительности и стабилизировать систему по расходу памяти. В большинстве случаев **Back-end** представляет собой обычный веб-сервер Apache, работающий на нестандартном порту, к примеру, на порту 88 и отвечающий только на запросы с **localhost** или IP адреса **Front-end**.

⚠ Совет администратору: *Лучше использовать несколько внутренних IP адресов типа 127.0.0.2, 127.0.0.3 и т.д. с 80-м портом, иначе возможны нежелательные редиректы на неработающий порт у **Front-end**.*

Процесс взаимодействия

Рассмотрим процесс взаимодействия **Front-End** и **Back-End** при обработке запроса пользователя к обычной странице сайта.

- Запрос от пользователей принимается **Front-end**, например, по адресу <http://www.1c-bitrix.ru/> на 80 порту. На нашем сайте мы используем NGINX в качестве **Front-end**.
- Запрос принимается и транслируется к **Back-end** (веб-сервер Apache с PHP), который обрабатывает запросы по адресу <http://127.0.0.2:80/>.
- Запрос исполняется **Back-end** веб-сервером, обрабатывает программный продукт на PHP и генерируется HTML-текст страницы для пользователя.
- Подготовленная HTML-страница от **Back-end** передается **Front-end** как ответ на запрос пользователя, соединение между **Front-end** и **Back-end** закрывается (желательно не использовать KeepAlive на **Back-end**), и процесс **Back-end** высвобождает оперативную память или начинает обработку другого запроса.
- **Front-end** передает готовую сформированную страницу посетителю столько времени, сколько требуется пользователю, даже если он работает на медленном канале. Потребление памяти **Front-end** для передачи страницы пользователю минимально.
- Получив страницу, браузер посетителя посылает последовательно серию запросов на графические элементы и таблицу стилей. Все запросы принимаются **Front-end** и обрабатываются без обращений к **Back-end**, т.е. все статически документы самостоятельно вычитываются с диска без использования медленных и дорогих процессов **Back-end**.

Пример уменьшения объема памяти

Пример уменьшения объема занимаемой оперативной памяти и числа процессов при подключении Front-end сервера (без настройки на отдачу статических файлов).

В этом примере был подключен *NGINX*, который все запросы, в том числе и запросы к статике, передавал back-end (*Apache*). Никаких дополнительных настроек на Back-end не выполнялось. Время подключения *NGINX* – 12:50.

Используемая веб-сервером память:



Число одновременно выполняемых процессов **apache**:



Как показала практика, двухуровневая конфигурация **Front-end + Back-end** существенно разгружает машину, уменьшает объемы потребляемой памяти, значительно ускоряет время обработки запросов и позволяет больше памяти выделить для работы базы данных. Такая конфигурация также позволяет значительно разгрузить сервер при обработке большого числа статических файлов, например, музыки, дистрибутивов программных продуктов, презентаций и других схожих объектов. Например, простым подключением *NGINX* удалось снизить нагрузку на веб-сервер в 5 раз.

Стабилизируем Back-end по расходу оперативной памяти

Даже создав двухуровневую конфигурацию, очень важно стабилизировать системы по расходу памяти независимо от нагрузки и защитить сервер от перегрузки.

Для стабилизации системы по расходу памяти и минимизации числа запущенных процессов **Back-end** мы рекомендуем в настройках сервера *Apache* установить параметр **MaxClients** в значении от 5 до 50, в зависимости от объема оперативной памяти.



Значение этого параметра не должно превышать 80% от объема доступной памяти сервера за вычетом памяти, выделенной под СУБД. Устанавливая **MaxClients**, вы ограничиваете возможное число одновременно запущенных процессов **Back-end**. Тем самым, удастся поставить жесткий лимит по потреблению памяти и исключить выход машины из строя при стрессовых нагрузках.

Пример поведения сервера при неправильно установленном MaxClients

Зависание сервера при слишком большом значении **MaxClients**:

```
top - 15:47:00 up 1 day, 4 min, 2 users, load average: 40.00, 31.94, 16.80
Tasks: 185 total, 37 running, 147 sleeping, 0 stopped, 1 zombie
Cpu(s): 0.0%us,100.0%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 5188748k total, 5012204k used, 176544k free, 560k buffers
Swap: 4096564k total, 4096564k used, 0k free, 2128k cached
PID to renice: ms`H`H
  PID USER      PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  COMMAND
 300  root       20  -5   0     0   0   R 100.1  0.0   13:05.65 kswapd0
24273 apache    25   0 593m 252m 288   R 23.3  5.0    3:20.15 httpd
24231 apache    25   0 585m 242m 292   R 22.3  4.8    4:08.31 httpd
24230 apache    25   0 560m 237m 288   R 21.9  4.7    3:59.39 httpd
 3440  root       21   0 13892 408 296   S 20.3  0.0    1:20.52 automount
25287 apache    25   0 572m 240m 412   R 20.3  4.8    3:19.70 httpd
 3480  root       22   0 32356 372 260   S 19.9  0.0    3:01.10 qlremote
 3692  root       25   0 9068 420 224   R 19.9  0.0    1:56.24 sendmail
 3734  root       25   0 115m 3664 264   R 19.9  0.1    2:11.30 httpd
 3840  root       25   0 2264 236 196   R 19.9  0.0    0:55.69 atd
 4013  haldaemo  25   0 5952 768 312   R 19.9  0.0    1:59.26 hald
 4091  nobody    25   0 21016 4492 196   R 19.9  0.1    4:00.63 nginx
24115 apache    25   0 562m 233m 276   R 19.9  4.6    4:08.01 httpd
24125 apache    25   0 576m 244m 284   R 19.9  4.8    4:00.97 httpd
24238 apache    25   0 561m 239m 288   R 19.9  4.7    4:06.51 httpd
24254 apache    25   0 573m 265m 348   R 19.9  5.2    3:54.63 httpd
24257 apache    25   0 578m 248m 288   R 19.9  4.9    3:35.79 httpd
24266 apache    25   0 583m 257m 288   R 19.9  5.1    3:32.39 httpd
```

На приведенном скриншоте произошло следующее:

Запущена команда **top** в *Linux*, показывающая объем занятой и свободной памяти, число выполняющихся процессов и объем памяти, занимаемый ими. В данном примере достаточно посещаемый веб-сайт работал на мощном сервере с двумя 4-х ядерными процессорами и 5 Гб оперативной памяти. Размер swar-файла – 4 Гб.

MaxClients был установлен в 100. Каждый процесс **apache** занимал около 250 Мб, что привело к полному исчерпанию и оперативной памяти, и файла подкачки на 40 процессах **apache**. В связи с отсутствием доступной памяти процессоры сервера не смогли справиться с нагрузкой и сервер зависал.



Таким образом, число **MaxClients** необходимо подбирать, исходя из системных ресурсов и нагрузки.

Методика подбора параметра

Методика подбора может быть следующей. Посчитайте, сколько памяти у вас занимает один **Back-end** процесс. Например, 50 Мб. Если мы установим **MaxClients** равным 4, значит максимальное потребление памяти может составить 200 Мб. Для машины с 512 Мб оперативной памяти **MaxClients** желательно выбрать между 5-10. Для правильной конфигурации **Front-end**, когда вся статика обрабатывается без участия **Back-end**, это позволит вполне комфортно обрабатывать порядка 50 тысяч хитов в сутки или примерно 10-20 тысяч уникальных пользователей.

Для крупных проектов с конфигурациями из двух машин или при наличии больших объемов памяти рекомендуется производить выбор значения **MaxClients** в процессе нагрузочного тестирования.

⚠ Важно! Подбирайте **MaxClients** так, чтобы система при стрессовых нагрузках потребляла не более 90% процессорных ресурсов и никогда не доходила до 100%. Это позволит вам стабилизировать использование процессорных ресурсов и быть уверенным, что не начнется общая регрессия по производительности во время пиковых нагрузок.

Обратите внимание, что очень важно настраивать время ожидания для **Front-end** таким образом, чтобы при отсутствии свободных процессов в **Back-end**, **Front-end** ожидал освобождения ресурсов. Тем самым организуется очередь запросов, и **Back-end** защищается от перегрузки.

Так же рекомендуется подбирать параметры управления процессами **Back-end** в соответствии с установленным лимитом **MaxClients**. Например, если **MaxClients = 5**, тогда рекомендуется установить:

```
MinSpareServers 5
StartServers 5
MaxClients 5
```

Т.е. это означает, что при старте сервера сразу будет запущено столько процессов **Back-end**, сколько максимально возможно соединений. Процессы никогда не будут выгружаться из памяти и будут готовы в любой момент принять и обработать запрос от **Front-end**. При запуске системы мы с вами сразу увидим объем используемой **Back-end** оперативной памяти, что позволит остальную память распределить для базы данных.

⚠ Совет! Рекомендуется для начала выбирать минимальное значение **MaxClients**, например, 5. В процессе работы проекта проверять время исполнения страницы. Если на быстром канале наблюдается ситуация, когда время выполнения страницы (параметр `?show_page_exec_time=Y`) в пиковые моменты показывает стабильно



минимальное значение, а визуально мы наблюдаем существенную задержку перед открытием страницы, то это может свидетельствовать о нехватке процессов **Back-end**. Иными словами о том, что запросы принимаются **Front-end** и долго удерживаются в ожидании высвобождения **Back-end** процессов.

В этом случае можно рекомендовать увеличить **MaxClients**, но обязательно с учетом общего баланса системы по расходу памяти.

MaxClients и базы данных

Еще одно преимущество использования **MaxClients** связано с базами данных.

Наличие лимита позволяет включить постоянное соединение к базе данных (только если у вас свой сервер с 1-2-я проектами) и уменьшить время соединения с базой данных и число работающих процессов базы данных. Если величина параметра **MaxClients** меньше или равна максимальному числу соединений с базой данных, то это гарантирует, что никогда не будет запущено больше этого числа процессов **Back-end**. Следовательно к базе данных не будет открыто больше соединений. Фактически, всегда в памяти будут находиться **MaxClients** загруженных процессов **Back-end** с открытыми соединениями к базе данных, готовые к обработке запросов.

⚠ Внимание! Практика показывает, что администраторы поверхностно относятся к данной рекомендации и не устанавливают лимиты или устанавливают их необоснованно большими. Это приводит к полному дисбалансу двухуровневой системы и потере стабильности при больших нагрузках (см. рисунок выше).

Еще раз подчеркнем, что соблюдение этих рекомендаций позволяет:

- Стабилизировать конфигурацию по расходу памяти при любых нагрузках;
- Защитить сервер от процессорных перегрузок и общей регрессии производительности;
- Безопасно использовать постоянное соединение к базе данных.

Производительность PHP

⚠ Важно! До 60% рабочего времени веб-сервера тратят на повторную компиляцию PHP-кода перед исполнением.

Ключевой способ снизить нагрузку на процессор – использовать прекомпиляторы PHP-кода.

PHP прекомпиляторы:

- [Zend Performance Suite](#)



- [Alternative PHP Cache \(APC\)](#)
- [eAccelerator](#)
- [XCache](#)
- [PHP Accelerator](#)
- [AfterBurner Cache](#)

Лучшие результаты производительности и кэширования PHP достигаются с использованием прекомпилятора *Zend optimizer+* из пакета *Zend Server (CE)* или *APC (Alternative PHP Cache)*, как более стабильная, но менее производительная альтернатива. Не забывайте выделять достаточный объем оперативной памяти для хранения разделяемого кэша скомпилированных PHP-файлов. Обычно бывает достаточно 32-64 Мб, но для уверенности можно увеличить объем выделяемой памяти до 128 Мб, в расчете на файлы административного раздела. Прекомпиляторы используют разделяемый кэш для хранения скомпилированного PHP кода в оперативной памяти, который доступен всем рабочим процессами веб-сервера, при этом скомпилированный PHP код хранится в кэше в единственном экземпляре (без дублирования).

Для уменьшения потребляемой памяти процессами веб-сервера, в котором запускается PHP, желательно исключить из компиляции или динамической загрузки все неиспользуемые модули.

При этом очень важно, чтобы в кэш прекомпилятора помещалось достаточное количество скриптов на PHP. Одна из самых часто встречающихся ошибок - это отсутствие каталога для сохранения откомпилированного кода.

Для ускорения работы с PHP-сессиями рекомендуется сохранять файлы сессий в каталоге, который представляет собой виртуальный диск в памяти или использовать установку `session.save_handler=mm` в `php.ini`. Если есть возможность, рекомендуется использовать системный **RAM** диск.

Панель производительности

Важным инструментом по настройке производительности PHP является модуль **Монитор производительности**, входящий в комплект всех продуктов "1С-Битрикс". Протестировать настройки системы можно в административной части на странице Панель производительности ([Настройки](#) > [Производительность](#) > [Панель производительности](#)). Неоптимальная конфигурация PHP:



Подсистема	Оценка	Эталон	Примечание
Конфигурация	27.22	30	
Среднее время отклика	0.0367	0.0330	секунд
Процессор (CPU)	5.5	9.0	миллионов операций в секунду
Файловая система	14 300.4	10 000	файловых операций в секунду
Почтовая система	0.0757	0.0100	время отправки одного письма (в секундах)
Время старта сессии	0.0002	0.0002	секунд
Конфигурация PHP	не оптимально	оптимально	рекомендации
База данных MySQL (запись)	2 631	5 600	количество запросов на запись в секунду
База данных MySQL (чтение)	13 604	7 800	количество запросов на чтение в секунду
База данных MySQL (изменение)	4 254	5 800	количество запросов на изменение в секунду

Как правило, выполнение рекомендаций позволяет увеличить производительность системы до достаточных величин. Численное значение параметра **Конфигурация** показывает основную характеристику сайта – скорость отдачи страниц клиенту. Чем больше число, тем лучше.

Некоторые типовые ошибки

Ошибка **Segmentation fault** может произойти:

- В результате "падения" PHP при использовании отложенной загрузки классов;
- При использовании Zend server могут "упасть" скрипты в cron или консоли.

В первом случае необходимо определить в **dbconn.php**:

```
define("NO_BITRIX_AUTOLOAD", true)
```

Во втором случае надо использовать другую версию PHP без подключения Zend optimizer+ или в "падающем" скрипте определить:

```
define('BX_NO_ACCELERATOR_RESET', true)
```

Сжатие страниц

Компрессия при передаче данных между сервером и клиентом обеспечивает сжатие страниц для ускорения вывода содержания сайта пользователям.

Сжатие в несколько раз уменьшает объем передаваемых HTML-данных между сайтом и браузером клиента, что существенно увеличивает скорость работы как для посетителей, так и для администраторов сайта.



Предпочтительнее сжимать страницы на **Front-End**. При отсутствии такой возможности можно сжимать данные на **Back-End**.

Возможные способы сжатия:

- [mod_deflate](#);
- модуль **Компрессии**, входящий в продукты "1С-Битрикс";
- стандартные модули PHP ;
- стандартные модули *Apache*.

Для исключения излишнего расходования процессорных ресурсов рекомендуется применять сжатие только на одном уровне. Например, при использовании сжатия **Front-end** сервером рекомендуется не использовать сжатие на уровне *Apache* и отключить модуль компрессии "*1С-Битрикс: Управление сайтом*".

Дополнительные рекомендации для двухуровневой конфигурации

По умолчанию, при работе в двухуровневой конфигурации, в качестве адреса клиента будет указываться адрес, на котором работает *NGINX* или другой акселератор. Для правильной работы модуля статистики необходимо обеспечить передачу реального IP адреса с **Front-end** в **Back-end**.

Например, для *NGINX* используется следующая технология: сервер *NGINX* устанавливает специальный заголовок в запросе, а специальный модуль *apache* (**rfaf** или **real_ip**) учитывает этот заголовок вместо стандартного.

Если же такой модуль не установлен, то вы можете сами изменить адрес клиента. Например, если адрес клиента передается в переменной `HTTP_X_FORWARDED_FOR` (так делает прокси-сервер *SQUID*) или `HTTP_X_REAL_IP`, то для замены переменной в продукте необходимо в файле `/bitrix/php_interface/dbconn.php` вставить подобный пример кода:

```
if(isset($_SERVER['HTTP_X_FORWARDED_FOR']) ||
isset($_SERVER['HTTP_X_REAL_IP'])) {

    foreach(array('HTTP_X_FORWARDED_FOR', 'HTTP_X_REAL_IP') as $key =>
$value) {

        if(

            isset($_SERVER[$value])

            && strlen($_SERVER[$value]) > 0

            && strpos($_SERVER[$value], "127.") !== 0
```



```
    ) {  
  
        if($p = strrpos($_SERVER[$value], ","))  
  
        {  
  
            $_SERVER["REMOTE_ADDR"] = $REMOTE_ADDR =  
trim(substr($_SERVER[$value], $p+1));  
  
            $_SERVER["HTTP_X_FORWARDED_FOR"] =  
substr($_SERVER[$value], 0, $p);  
  
        }  
  
        else  
  
            $_SERVER["REMOTE_ADDR"] = $REMOTE_ADDR =  
$_SERVER[$value];  
  
  
            break;  
  
        }  
  
    }  
  
}
```

Кроме того, в конфигурации Apache на **Back-end** желательно отключить KeepAlive. Поскольку **Front-end** находится или на этой машине, или "рядом", более быстрое высвобождение ресурсов предпочтительнее.

Достигнутые результаты

В результате построения двухуровневой архитектуры и выполнения ряда рекомендаций мы должны получить следующие результаты:

- система стабилизирована по расходу памяти; **Front-End** и **Back-End** занимают заранее отведенный объем памяти, который не будет расти даже при увеличении нагрузки;
- в стрессовой ситуации система будет стабильно и равномерно обрабатывать запросы, **Back-end** не будет увеличивать число одновременно выполняемых процессов выше установленного лимита **MaxClients**, **Front-end** будет принимать все запросы от пользователей и ожидать освобождения процессов **Back-end**;
- использование процессорных ресурсов ограничено числом одновременно работающих процессов **Back-end** в соответствии с **MaxClients**, не начнется регрессия производительности;



- возможно безопасное использование постоянного соединения с базой данных без опасения превысить число возможных соединений; в памяти все время находится установленное число **Back-end** процессов, готовых к обработке запросов и с установленным соединением с базой данных;
- процессорные ресурсы существенно высвобождены за счет прекомпиляции PHP-кода;
- пользователи комфортно работают со сжатыми страницами.

Пример настройки двухуровневой архитектуры

В качестве примера рассмотрим конфигурацию виртуальной машины **VMBitrix**. В состав конфигурации входят:

- Веб-сервер Apache в качестве Back-end
- NGINX в качестве Front-end
- Zend Server CE в качестве настроенного модуля PHP (с прекомпилятором)
- Дополнительные настройки СУБД MySQL и других параметров сервера.

Для рассмотрения и управления настройками перейдите в консоль виртуальной машины или подключитесь к ней через **ssh** или **sftp**.

Настройка веб-сервера Apache

Основным конфигурационным файлом веб-сервера является `/etc/apache2/apache2.conf` (в других системах файл может называться `/etc/httpd/httpd.conf`). Далее, подключается файл с настройкой портов прослушивания для веб-сервера и другие файлы. Иногда это все размещается в одном файле, иногда (как в виртуальной машине – в разных).

Рассмотрим основные параметры этих файлов.

`/etc/apache2/apache2.conf`:

```
Timeout 300 #если 300 секунд не происходит никаких операций, завершить процесс

KeepAlive Off #все запросы у нас короткоживущие

User ${APACHE_RUN_USER} #пользователь, под которым работает веб-сервер

Group ${APACHE_RUN_GROUP} #группа, под которой работает веб-сервер
```

`/etc/apache2/ports.conf`:



```
Listen *:8888 #веб-сервер работает на порту 8888
```

/etc/apache2/envvars

Осуществляется установка переменных окружения – пользователя и группу.

```
export APACHE_RUN_USER=bitrix
export APACHE_RUN_GROUP=bitrix
```

/etc/apache2/conf.d/prefork

Осуществляется настройка числа процессов сервера.

```
#работает с этим расширением
StartServers 4 #4 одновременных сервера
MinSpareServers 4
MaxSpareServers 4
MaxClients 4 #4 одновременных клиента
MaxRequestsPerChild 200 #после 200 запросов перезапускать процесс
```

Таким образом, настройки веб-сервера достаточно простые. Фактически, в стандартной конфигурации достаточно изменить только порт, на котором работает веб-сервер и параметр **MaxClients** и связанные с ним.

Пример: число процессов веб-сервера

Как определить максимальное число процессов веб-сервера? Естественно, только опытным путем. При этом начальное значение можно подобрать достаточно просто – нужно посмотреть, сколько занимает процесс **apache** во время типового обращения к сайту:

- Зайдите в *Linux* и запустите команду **free**. Она покажет вам общий размер оперативной памяти и свободный размер:

```
# free
              total        used        free      shared    buffers
cached
Mem:          255676      224340      31336           0       33468
67964
```



-/+ buffers/cache:	122908	132768	
Swap:	530136	51800	478336

В примере показано, что в системе ~256 Мб памяти, ~500 Мб swap (виртуальная память на диске). Системной памяти занято ~120 мб, вся неиспользуемая память отдана под файловый кэш

- Наберите команду **top**.
- Откройте любую страницу сайта, параллельно смотря значение в программе **top**. Вверху появится процесс с названием **apache2**:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
14687	bitrix	20	0	153m	45m	28m	R	6.7	18.2	0:05.34
apache2										

В этом примере столбец **RES** как раз и показывает примерное количество памяти, выделяемое на один процесс. Таким образом, для типового сайта на "1С-Битрикс: Управление сайтом" – это около 50 Мб. Соответственно, параметр **MaxClients** выбран 4 исходя из соображения, что должна остаться память под операционную систему и работу СУБД.

Из этого примера видно, что для достаточно посещаемого сайта (не интернет-магазина) минимальные требования по памяти для веб-сервера (или VPS) – от 512 Мб. Для интернет-магазина с обменом с "1С" минимальный объем памяти на сервере или VPS должен быть не менее 1 Гб.

Настройка Front-end NGINX

Каталог с конфигурацией *NGINX* - */etc/nginx*. Рассмотрим его конфигурационный файл */etc/nginx/nginx.conf*.

user	bitrix;	#пользователь, под которым работает nginx.
Желательно совпадение с пользователем apache		
worker_processes	8;	#8 одновременных процессов
error_log	/var/log/nginx/error.log	warn;
pid	/var/run/nginx.pid;	
worker_rlimit_nofile	10240;	#максимальное число открытых файлов
events	{	



```
        use epoll;

        worker_connections      10240;    #максимальное число
соединений с одним процессом. Система может одновременно работать с
max_clients = worker_processes * worker_connections, т.е. с 81920
соединений, в том числе статических файлов

    }

    http {

        include      /etc/nginx/mime.types;

        default_type application/octet-stream;

#формат логов

        log_format   main      '$remote_addr - $remote_user
[$time_local] $status'

                                '$request'          $body_bytes_sent
"$http_referer" '

                                '$http_user_agent'

"$http_x_forwarded_for";

        log_format   common    '$remote_addr - -
[$time_local] "$request" $status $bytes_sent "$http_referer"
"$http_user_agent" $msec';

        access_log   /var/log/nginx/access.log  common;

        sendfile     on;

        tcp_nopush   on;

        tcp_nodelay  on;

        client_max_body_size      10m;    # максимально
допустимый размер тела запроса клиента, указываемый в строке "Content-
Length" в заголовке запроса

        client_body_buffer_size   128k;

        proxy_connect_timeout     300;    #время на ожидание
соединения

        proxy_send_timeout        300;

        proxy_read_timeout        300;
```



```
proxy_buffer_size          64k;
proxy_buffers              8 64k;
proxy_busy_buffers_size    64k;
proxy_temp_file_write_size 10m;
gzip on; #сжимать передаваемые данные
gzip_proxied any;
gzip_types application/x-javascript text/css;

server { #виртуальный хост
    listen 80; #порт 80
    server_name bitrix; #адрес узла. Если узел всего один
- можно написать любой
    server_name_in_redirect off; #лучше поставить в ON -
передавать запрошенное имя сайту
    access_log /var/log/nginx/access.log common;
    index index.php;
    error_page 500 502 503 504 /500.html;
    error_page 404 = /404.php;
#установить дополнительные заголовки для определения адреса клиента в
статистике сайта
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header                                X-Forwarded-For
$proxy_add_x_forwarded_for;
    proxy_set_header Host $host:80;
    client_max_body_size 1024M; #максимальный размер
передаваемого файла
    client_body_buffer_size 4M;
    root /var/www; #корневая папка сайта

#включить https режим при нахождении в корне сайта файла .htsecure
```



```
if (-f /home/bitrix/www/.htsecure) {  
    rewrite ^(.*)$ https://$host$1 permanent;  
}  
  
#выбрать, какие данные пересылать backend серверу, а какие -  
показывать напрямую  
  
#в данной конфигурации все пересылается backend серверу  
  
#обилие вариантов потребовалось компании Битрикс для того, чтобы при  
добавлении в конфигурацию отдачу статических файлов напрямую через  
NGINX динамические запросы на псевдостатические файлы все-таки  
перенаправлялись на backend  
  
    location / { expires 3d;  
  
        if ($request_method = OPTIONS ) {  
            proxy_pass  
http://127.0.0.1:8888;  
        }  
  
        if ($request_method = PROPFIND ) {  
            proxy_pass  
http://127.0.0.1:8888;  
        }  
  
        if ($request_method = PROPPATCH ) {  
            proxy_pass  
http://127.0.0.1:8888;  
        }  
  
        if ($request_method = MKCOL ) {  
            proxy_pass  
http://127.0.0.1:8888;  
        }  
  
        if ($request_method = COPY ) {  
            proxy_pass  
http://127.0.0.1:8888;  
        }  
    }  
}
```



```
        if ( $request_method = MOVE ) {
            proxy_pass
http://127.0.0.1:8888;
        }
        if ( $request_method = LOCK ) {
            proxy_pass
http://127.0.0.1:8888;
        }
        if ( $request_method = UNLOCK ) {
            proxy_pass
http://127.0.0.1:8888;
        }
    }

    location ~
^ (/extranet/docs|/docs|/workgroups|/company/profile|/bitrix/tools|/com
pany/personal/user) .*/$ {
        proxy_pass      http://127.0.0.1:8888;
    }

    location ~
^ (/extranet/docs|/docs|/workgroups|/company/profile|/bitrix/tools|/com
pany/personal/user) {
        if (-d $request_filename) {
            rewrite ^(.*) (/*)$ $1/ last;
        }
        proxy_pass      http://127.0.0.1:8888;
    }

    location ~ ^ (/bitrix/html_pages)
    {
```



```
root /var/www;

index index@.html;

if (!-f $request_filename)
    {
        rewrite
^/bitrix/html_pages(.*)@(.*)\.html$ $1.php?$2 break;
        rewrite ^/bitrix/html_pages(.*)\.html$
$1\.php break;

        proxy_pass http://127.0.0.1:8888;
    }
}

location ~ \.php$ {
    root /var/www;

    if ($request_method = POST ) {
        break;

        proxy_pass
http://127.0.0.1:8888;
    }

    if ($http_cookie !~ "PHPSESSID=" ) {
        rewrite ^(.*)\.php$
/bitrix/html_pages$1@$args.html? last;
    }

    proxy_pass http://127.0.0.1:8888;
}

location ~ /$ {
    root /var/www;

    if ($request_method = POST ) {
        break;
```



```

        proxy_pass
http://127.0.0.1:8888;
    }
    if ($http_cookie !~ "PHPSESSID=" ) {
        rewrite                ^(.*)/$
/bitrix/html_pages$1/index@$args.html? last;
    }
    proxy_pass http://127.0.0.1:8888;
}

location ~ (/\.\php|\.\asmx)$ {
    proxy_pass            http://127.0.0.1:8888;
}

location ~ /\.ht {
    deny  all;
}

location ~ /favicon.ico {
    proxy_pass            http://127.0.0.1:8888;
}

location ~ ^(/bitrixsetup\.\php)$ {
    proxy_pass            http://127.0.0.1:8888;
    proxy_buffering off;
}

}

#аналогичная конфигурация для https (удалена)
}
```



В данной конфигурации настройка *NGINX* проведена так, что все обращения к серверу будут перенаправляться на **Back-end** сервер. Однако, можно избранные файлы отдавать через *NGINX*. В этом случае будет достигнуто дополнительное увеличение производительности.

Дополнительную информацию по настройке *NGINX* вы можете получить на сайте www.nginx.ru и wiki.nginx.org.

Отдача графики напрямую NGINX

Для того, чтобы графические файлы отдавались напрямую NGINX, вам необходимо добавить в конфигурацию сервера строки, подобные этим:

```
location ~* \.(jpg|jpeg|gif)$ {  
  
    root          /var/www;  
  
    access_log    off;  
  
    expires       3d;  
  
}
```

В приведенном примере все файлы с расширением **jpg**, **jpeg** или **gif** будут считываться напрямую *NGINX*, причем корневой папкой для хранения файлов будет */var/www*.

Оптимизация базы данных

Оптимизация работы с базой данных для *MySQL*, *Oracle*, *MSSQL* является одной из важнейших стратегий в оптимизации системы в целом.

У каждого производителя базы данных есть целый раздел в документации, который посвящен вопросам производительности и оптимизации работы базы данных.

Вы должны понимать, что типовая поставка базы данных обычно подразумевает минимальное серверное оборудование, минимальную память и диски. Т.е. стандартные конфигурации не учитывают возможности вашего оборудования и вашего приложения. Вы обязательно должны настраивать базу данных для обеспечения оптимальной работы.



Основные принципы

Если кратко попробовать сформулировать стратегию оптимизации, то прозвучит это примерно так:

- **как можно меньше дисковых операций чтения данных;** старайтесь увеличивать размер буфера кэширования, чтобы база данных как можно меньше читала данные с диска;
- **как можно меньше сортировок данных на диске;** старайтесь увеличить буфер сортировки таким образом, чтобы избежать двойных сортировок и сортировок на диске.
- **увеличивайте параллельность исполнения запросов** за счет запуска большого числа процессов базы данных, выбора форматов хранения данных, обеспечивающих параллельность работы;
- **отложенная фиксация транзакций на диске;** очень желательно настроить базу данных так, чтобы при изменении данных или их вставке не производилась немедленная запись на диск, а изменения собирались и фиксировались с некоторым интервалом; это позволяет значительно быстрее возвращать управление продукту после выполнения SQL запросов.

⚠ Примечание: *Надо отметить, что при этом несколько снижается надежность. В случае сбоя системы те данные, которые еще не "сброшены" на диск, будут потеряны.*

Давать конкретные рекомендации по настройке базы данных достаточно сложно, так как сервер базы данных - это сложное приложение и вам обязательно нужно проконсультироваться с документацией поставщика по вопросам настройки.

Но стоит отметить особое преимущество, которое мы получили в результате создания двухуровневой конфигурации. Созданная нами система сбалансирована по расходу оперативной памяти, и мы точно знаем свободный объем памяти и можем безопасно для системы использовать эту память для конфигурирования базы данных. В большинстве систем порядка 60-80% оперативной памяти удается выделить для базы данных и это позволяет значительно ускорить работу всей системы в целом.

Постоянное соединение с базой данных

Важным параметром, влияющим на потребление памяти базами данных (MySQL\Oracle), является максимальное число одновременных соединений (определяемое параметрами **max_connections** и **processes** соответственно).

При использовании двухуровневой архитектуры Front-end/Back-end можно в несколько раз уменьшить число одновременных соединений и высвободить больше памяти для сортировки данных в памяти и буферизации.



Если установлено число **MaxClients** в настройках веб-сервера **Back-end**, можно считать, что максимальное число соединений к базе данных будет соответствовать максимальному числу одновременных соединений.

Рекомендуется подбирать параметр **max_connections** таким образом, чтобы иметь резерв в 10-20% свободных соединений от максимального значения.

Настройка базы данных MySQL

Оптимизация работы с базой данных для MySQL-версии продукта является одной из важнейших стратегий в оптимизации системы в целом, так как продукт активно работает с базой данных.

Стандартно *MySQL* работает с форматом данных **MyISAM**. Простой формат данных хранит каждую таблицу с данными или индекс в отдельном файле. В целом, на небольших по нагрузке сайтах данный формат является наиболее быстрым, хотя и не обеспечивает полной целостности и надежного хранения данных за счет отсутствия транзакций.

Основным недостатком **MyISAM** с точки зрения производительности является блокировка на уровне таблицы при выполнении тех или иных операций. В результате, при большой нагрузке *MySQL* именно **MyISAM** таблицы становятся основным узким местом в системе, мешая увеличивать утилизацию машины и число обрабатываемых запросов. Это также приводит к увеличению времени работы страницы за счет ожидания используемых таблиц на уровне *MySQL*.

Рекомендуется переводить все таблицы проекта в формат данных [InnoDB](#). Формат **InnoDB**, начиная с версии *MySQL 4.0*, входит в стандартную поставку продукта и обеспечивает надежное хранение данных, транзакционность и блокирование данных на уровне строки.

Два важных момента, которые дают основание предпочесть таблицы **InnoDB** перед **MyISAM**:

- **Надежность.** В **MyISAM** высока вероятность сбоя таблиц, особенно больших, особенно при высокой посещаемости, особенно часто изменяемых. Есть риск потерять несколько (десятков, сотен) записей и целостность данных. В **InnoDB** чинить отдельные таблицы не придется. Если упадет, так все сразу. Но на практике это - исключительное явление, практически не встречаемое.

Благодаря транзакционности, риск нарушения целостности минимальный. Недостатки **InnoDB**: нужно внимательно следить за свободным местом на диске; накапливающаяся фрагментация данных (лечится периодическим переводом таблиц из **InnoDB** в **MyISAM** и обратно).



- **Скорость.** На невысокой посещаемости **MyISAM** ведет себя быстрее, как на модификацию, так и на чтение. Однако, при росте посещаемости достаточно быстро сказывается отсутствие транзакций и блокировка на уровне таблиц. При некоторой величине посещаемости проект просто реально умирает.

В **InnoDB** запись будет медленнее (транзакции же), зато при высокой посещаемости блокировки наступят намного, намного позже, чем для **MyISAM**.

Поменять тип таблиц на **InnoDB** можно следующим образом:

1. В административном меню [Настройки системы](#) > [Инструменты](#) > [SQL запрос](#) выполнить команду:

```
SHOW TABLES
```

2. В результате вы получите список всех текущих таблиц продукта. Для каждой таблицы необходимо выполнить команду:

```
ALTER TABLE <ИМЯ ТАБЛИЦЫ>, type=InnoDB
```

В FAQ приведен [пример](#) для создания скрипта для перевода таблиц в **InnoDB**.

3. После перевода таблиц вашей базы в **InnoDB** надо добавить в файл `/bitrix/php_interface/dbconn.php` нижеследующий код:

```
define("MYSQL_TABLE_TYPE", "InnoDB");
```

Переход на тип таблиц **InnoDB** позволяет избежать возникновения узкого участка в производительности при работе с базой данных и в полном объеме использовать системные ресурсы.

⚠ Внимание! Обязательно конфигурируйте **InnoDB**. Для лучшей производительности базы данных при работе с **InnoDB** рекомендуется настроить `my.cnf` для MySQL в разделе [параметров для InnoDB innodb_*](#).

Наибольшее внимание следует обратить на следующие параметры и примеры:

```
set-variable = innodb_buffer_pool_size=250M
set-variable = innodb_additional_mem_pool_size=50M
set-variable = innodb_file_io_threads=8
set-variable = innodb_lock_wait_timeout=50
set-variable = innodb_log_buffer_size=8M
set-variable = innodb_flush_log_at_trx_commit=0
```



⚠️ Рекомендуется: Для сокращения времени ответа сервера можно использовать отложенные транзакции и, в частности, устанавливать переменную `set-variable = innodb_flush_log_at_trx_commit=0`.

Если **MyISAM** уже не используется активно, можно высвободить память в пользу **InnoDB** параметров.

Желательно, чтобы кэш данных вмещал в себя основной объем данных, используемых продуктом в работе. Обычно для работы базы данных выделяется порядка 60-80% свободной памяти в системе.

⚠️ Рекомендуется: Производить многопоточную (*multithreading*) сборку MySQL для улучшения производительности системы и возможностей по параллельной обработке запросов.

Пример рекомендуемых настроек для сервера с 2 Гб оперативной памяти, работающего с операционной системой FreeBSD/Linux:

```
set-variable = table_cache=4096
```

В составе продукта около 250 таблиц, поэтому рекомендуется увеличивать кэш для заголовков таблиц.

```
set-variable = key_buffer_size=16M
set-variable = sort_buffer=8M
set-variable = read_buffer_size=16M
```

Эти параметры используются только для **MyISAM**. Если в базе нет таблиц **MyISAM**, то лучше установить минимальные значения.

```
set-variable = query_cache_size=64M
set-variable = query_cache_type=1
```

Кэширование результатов запросов. Обычно бывает достаточно 32 Мб (смотреть на статус **Qcache_lowmem_prunes**). Максимальный размер результата по умолчанию - 1 Мб, его можно регулировать.

```
set-variable = innodb_buffer_pool_size=780M
```

Основной буфер - чем больше, тем лучше.

```
set-variable = innodb_additional_mem_pool_size=20M
```

Вспомогательный буфер на внутренние структуры, большой делать не имеет смысла.

```
set-variable = innodb_log_file_size=100M
```



```
set-variable = innodb_log_buffer_size=16M
```

Чем больше размер лог-файла, тем реже надо будет записывать в основной файл данных. Суммарный размер лог-файла может быть сопоставим с величиной `innodb_buffer_pool_size` (по умолчанию ведется два лога).

```
set-variable = innodb_flush_log_at_trx_commit=0
```

Отложенная фиксация транзакций, раз в секунду

```
set-variable = tmp_table_size=32m
```

Размер временных таблиц рекомендуется увеличивать до 32 Мб.

Рекомендуется так же увеличивать `join_buffer_size` до 2 Мб, это существенно влияет на скорость выполнения ряда запросов.

```
set-variable = join_buffer_size = 2M
```

⚠ Совет администратору Переход на **InnoDB** может привести к значительному замедлению некоторых масштабных операций записи и обновления данных. Это связано с тем, что все операции по изменению данных начинают выполняться с использованием транзакций. Кроме того, в отличие от **MyISAM** для кэширования таблиц **InnoDB** не используется кэш операционной системы, а все кэшированные данные хранятся в кэше БД, определяемом параметром `innodb_buffer_pool_size`. Вы должны сами определить оптимальность перехода вашего проекта на формат данных **InnoDB**.

⚠ Внимание! Если по каким-то причинам вы решили продолжить работу с типом данных **MyISAM**, обязательно проведите конфигурирование MySQL для увеличения объемов кэшируемой информации, областей сортировки и минимизации числа дисковых операций. Использование для базы данных 60-80% оперативной памяти может ускорить работу стандартного проекта в несколько раз.

Временная папка

При наличии достаточного количества ОЗУ рекомендуется выносить временную папку MySQL на **ramdisk** в памяти.

Для этого:

- Убедитесь, что в ядре реализована поддержка **tmpfs**.
- Создайте новую точку монтирования и дайте все права на использование:

```
# mkdir /mnt/tmpfs/  
  
# chmod 777 /mnt/tmpfs/
```



- Дайте команду (от рута или через sudo):

```
# mount -t tmpfs -o size=1024M tmpfs /mnt/tmpfs/
```

или

```
$ sudo mount -t tmpfs -o size=1024M tmpfs /mnt/tmpfs/
```

где 1024M есть размер RAMdisk в Мегабайтах.

⚠ Внимание! К размеру папки нужно подходить осторожно: если вы попросите создать **ramdisk** больше, чем имеете оперативной памяти, система начнёт сгружать всё в swap-файл и реальный результат подключения временной папки может быть отрицательным.

Список ссылок по теме:

- [Частный опыт настройки MySQL](#)

Настройка базы данных Oracle

Продукт «1С-Битрикс: Управление сайтом» протестирован и успешно работает с *Oracle 9i* и *10g*.

Использование *Oracle* в качестве базы данных позволяет значительно увеличить надежность системы и производительности при большой нагрузке. Архитектура *Oracle* позволяет практически полностью использовать ресурсы серверов, добиваясь при этом прекрасной интерактивности приложений, даже при работе с большими объемами данных, построении больших отчетов или большом числе одновременных соединений. Работа с *Oracle* также позволяет использовать самые разные варианты масштабирования интернет-проекта.

Общие рекомендации по настройке *Oracle* совпадают с рекомендациями *Oracle* по конфигурированию системы для уменьшения дисковых операций чтений, сортировки и перестроения.

Стоит обратить внимание на системные переменные управления памятью. Рекомендуется использовать до 60-80% оперативной памяти для кеширования данных за счет управления переменными:

```
db_cache_size  
shared_pool_size  
pga_aggregate_target
```



Начиная с версии *Oracle 10*, рекомендуется использовать **Automatic Shared Memory Management**:

- при установке БД, выбрать % ОЗУ, которая будет доступна БД в пределах 60-80% от ОЗУ сервера с помощью параметра **SGA_MAX_SIZE**
- во время эксплуатации БД количество памяти, доступное *Oracle* можно откорректировать (в большую или меньшую стороны) с помощью параметра **SGA_TARGET** без перезапуска БД. Индикатором правильности установки параметров управления памятью будет отсутствие дисковых операций **swap**, также имеет смысл контролировать общесистемный размер используемого **swap** пространства - не более 100 MB.

Для экономии места SGA (shared_pool) и уменьшения расходов процессорных ресурсов на разбор SQL-запросов, отличающихся только значениями констант, рекомендуется установить параметр

```
cursor_sharing = FORCE
```

либо

```
cursor_sharing = SIMILAR
```

отключив расчёт гистограмм для столбцов таблиц в параметрах сбора статистики:

```
begin dbms_stats.set_param( 'METHOD_OPT', 'FOR ALL COLUMNS SIZE 1' );  
end;
```

Если *Oracle* установлен на той же машине, что и веб-сервер, рекомендуется использовать протокол IPC (`PROTOCOL = IPC`) и (`KEY = EXTPROC`) для соединения с базой для исключения работы через IP-стек.

Если реализована двухуровневая схема обработки запросов с Front-end и Back-end и установлен параметр **MaxClients**, то можно без опасений использовать постоянные соединения между PHP и Oracle (Persistent connection), выполнив следующую установку в файле `/bitrix/php_interface/dbconn.php`:

```
define("DBPersistent",true);
```

⚠ Примечание: Начиная с релиза *Oracle 10g R2*, возможно использование отложенных транзакций ([Enhanced COMMIT](#)) в случаях если бизнес-требования к проекту допускают возможную потерю данных нескольких последних транзакций. Использование отложенных транзакций существенно ускоряет проект, позволяет снять прямую зависимость от производительности дисковой подсистемы и существенно уменьшить время генерации страниц. Для использования этого механизма, необходимо установить параметр: `COMMIT_WRITE='BATCH,NOWAIT'`.



Пример-упражнение

Настройки MySQL для виртуальной машины

В качестве примера рассмотрим как настроена база данных *MySQL* в виртуальной машине *VMBitrix*.

Перейдите в папку */etc/mysql* и посмотрите настройки MySQL для виртуальной машины. Выключите виртуальную машину и установите ей большее значение ОЗУ (например, 512 мб). Посмотрите, как изменились настройки в файле */etc/mysql/conf.d/bvat.cnf*.

Для 256 мб:

```
[mysqld]
query_cache_size=32M
innodb_buffer_pool_size=32M
```

для 512 мб:

```
[mysqld]
query_cache_size=48M
innodb_buffer_pool_size=96M
```

Кроме того, при 512 мб система чувствует себя гораздо свободнее:

Доступная память при 256 мб:

```
# free
      total          used          free      shared    buffers
cached
Mem:   255676      224340      31336           0      33468
67964
-/+ buffers/cache:  122908      132768
Swap:   530136       51800      478336
```

Доступная память при 512 мб:

```
# free
```



	total	used	free	shared	buffers
cached					
Mem:	515572	299208	216364	0	6944
186336					
-/+ buffers/cache:		105928	409644		
Swap:	530136	0	530136		

Связано это с тем, что виртуальная машина *VMBitrix* содержит скрипты, активизирующиеся при загрузке и устанавливающие необходимые параметры системы. Ключевым параметром является объем оперативной памяти, установленный в системе.

⚠ Примечание: *Кастомизацию настроек можно производить без отключения виртуальной машины. Для этого достаточно вносить изменения в файл `/etc/mysql/conf.d/2_bx_custom.cnf`.*