HTTP/1.1, веб-клиенты указывают, какие типы сжатия они поддерживают, устанавливая заголовок Accept-Encoding в HTTP-запросе. Если веб-сервер видит такой заголовок в запросе, он может применить сжатие ответа одним из методов, перечисленных клиентом. При выдаче ответа посредством заголовка Content-Encoding сервер уведомляет клиента о том, каким методом сжимался ответ. Переданные таким образом данные меньше первоначальных примерно в 5 раз, и это существенно ускоряет их доставку. Однако здесь есть один недостаток: увеличивается нагрузка на веб-сервер. Но вопрос с сервером всегда можно решить[2].

В последние годы мир веб-разработки стал существенно тяготеть к клиентским приложениям. Браузеры стали настолько быстрыми и приобрели такое феноменальное количество возможностей, что клиентская сторона стала не проще, а даже иногда намного сложнее, чем серверная составляющая. Именно на фоне увеличившегося внимания к клиентской составляющей веб-сайтов и зародилась ее оптимизация, оптимизация скорости загрузки, отображения и функционирования веб-сайтов в браузерах конечных пользователей. На данный момент направление это новое и весьма перспективное для изучения и прикладного использования.

ЛИТЕРАТУРА

1. Мациевский, Н.С. Реактивные веб-сайты. Клиентская оптимизация в алгоритмах и примерах : учеб. пособие / Н.С. Мациевский, Е.В. Степанищев, Г.И. Кондратенко. – М. : Интернет-Университет Информационных Технологий : БИНОМ. Лаборатория знаний, 2010. – 336 с. – Режим доступа: http://inethub.olvi.net.ua/ftp/pub/books/programming/web/html/reactivewebsites.v1.4.pdf. – Дата доступа: 29.09.2015.
2. Хабрахабр [Электронный ресурс] / Топ-10 советов о том, как увеличить скорость загрузки страницы. – Режим доступа: http://habrahabr.ru/post/137239/. – Дата доступа: 29.09.2015.

УДК 004.75

СПОСОБЫ ПОСТРОЕНИЯ РАСПРЕДЕЛЕННЫХ ВЕБ-СИСТЕМ

Ю.В. ЛАПТЕВ
(Представлено: канд. техн. наук, доц. Р.П. БОГУШ)

Рассматривается некоторые ключевые вопросы, которые следует учитывать при проектировании больших веб-сайтов, а также некоторые базовые компоненты, используемые для достижения этих целей. Основное внимание в статье уделяется анализу веб-систем.

Открытое программное обеспечение стало основным элементом при создании некоторых крупнейших веб-сайтов в последнее время. Создание и управление масштабируемого веб-сайта на примитивном уровне является простым соединением пользователей с удаленными ресурсами через Интернет. А доступ к ресурсам, которые рассредоточены на множестве серверов и являются звеном, обеспечивающим масштабируемость веб-сайта. Время, потраченное на планирование построения веб-службы на начальном этапе может помочь в дальнейшем. На проектирование крупномасштабных веб-систем могут влиять следующие принципы: производительность, стоимость, надежность, доступность, масштабируемость и управляемость.

Скорость работы веб-сайта влияет на удовлетворенность пользователей, а также ранжирование поисковыми системами. В результате нужно стремится к созданию системы, которая оптимизирована для быстрых ответов. Система должна быть надежной. Это означает, что определенный запрос на получение данных должен возвращал эти данные, а в случае изменения данных должен возвращать новые данные. Пользователи должны быть уверенным, что данные не будут потеряны. Разработка доступных к отказу систем является фундаментальным технологическим требованием. Высокая доступность в распределенных системах требует внимательного рассмотрения избыточности для ключевых компонентов, а также быстрого восстановления после системных отказов. Для крупных онлайн-магазинов недоступность даже в течении небольшого времени может привести к огромным потерям дохода. Очень важными являются усилия, направленные на увеличение пропускной способности для обработки больших объемов информации, что и является масштабируемостью. Также к масштабируемости относятся наращение емкости запоминающих устройств и увеличение количества транзакций, обрабатываемых в единицу времени. Проектирование системы, которая проста в эксплуатации еще один важный фактор. Для обеспечения управляемости необходимо рассмотреть вопросы диагностики возникающих проблем, легкости проведения обновлений или модификаций [1].

Все вышеперечисленные принципы являются основой для принятия решений в проектировании распределенной веб-архитектуры. Однако они могут находиться в противоречии друг с другом, потому что достижение целей одного происходит за счет пренебрежения другими. При разработке любого веб-приложения важно рассмотреть эти ключевые принципы, чтобы найти наиболее подходящее решение конкретно для каждой задачи.

При рассмотрении архитектуры системы есть несколько вопросов, которые необходимо раскрыть, например, какие компоненты необходимо использовать, как они будут совмещаются друг с другом, и на какие компромиссы можно пойти. Изначальное планирование системы существенно сэкономит время и ресурсы в будущем. Рассмотрим пример хостинга для загрузки изображений. Представим систему, где пользователи могут загрузить свои изображения на сервер, и при этом изображения могут запрашиваться через ссылку на сайт. Количество хранимых изображений может быть безграничным, необходима надежность хранения данных, а также должны быть быстрыми выполнения операций загрузки и запросов изображений. Кроме этого система должна быть простой в обслуживании и экономически эффективной.

При рассмотрении масштабируемой системы полезно разделить функциональность и подумать о каждой части системы как об отдельной службе с четко определенным интерфейсом. У каждой службы существует свой функциональный контекст, и взаимодействие с чем-либо за пределами этого контекста происходит через абстрактный интерфейс. При простом проектировании архитектуры все запросы загрузки и получения изображения обрабатываются одним сервером, но поскольку система должна масштабироваться, целесообразно выделить эти две функции в отдельные сервисы. Другая проблема проектирования состоит в том, что у веб-сервера обычно существует предел количества одновременных соединений, которые он может обслуживать. Поскольку процессы чтения информации могут быть асинхронными, или будет использоваться gzip-сжатие, или передача с делением на порции, то веб-сервер может переключить чтения подачи быстрее и переключиться между клиентами, обслуживая гораздо больше запросов, чем максимальное число соединений. А вот процессы записи обычно поддерживают открытое соединение на протяжении всего времени загрузки, в результате чего веб-сервер не сможет обрабатывать новых клиентов, пока не будут завершены другие процессы записи информации. Предвидение подобной потенциальной проблемы свидетельствует о необходимости разделения чтения и записи изображений в независимые службы. Это позволит не только масштабировать каждую из них по отдельности, но и быть в курсе того, что происходит в каждой службе. Это разграничит проблемы способные возникнуть в будущем, что упростит диагностику и оценку проблемы медленного доступа на чтение.

Чтобы справится с отказом, у веб-архитектуры должна быть избыточность ее служб и данных. Например, если существует только одна копия файла на сервере, то потеря этого сервера будет означать потерю и файла. Обычно эту проблему можно избежать путем создания резервных копий. Этот же принцип применим и к службам. Создание избыточности в системе позволяет избавиться от слабых мест и обеспечить резервную или избыточную функциональность на случай критической ситуации. Например, в случае наличия двух экземпляров одной службы, при выходе из рабочего состояния одного экземпляра системы, отказ может быть обойден за счет переключения на исправный экземпляр. Другая ключевая роль избыточности службы — создание архитектуры, не предусматривающей разделения ресурсов. С этой архитектурой каждый узел в состоянии работать самостоятельно. Она способствует масштабируемости, так как добавление новых узлов не требует специальных условий или знаний. Также в этих системах не найдется критически уязвимой точки отказа, что делает их намного более устойчивыми к отказу [1].

Наборы данных могут быть настолько большими, что их невозможно будет разместить на одном сервере. Может также случиться, что вычислительные операции потребуют слишком больших компьютерных ресурсов. В любом случае есть два варианта: вертикальное или горизонтальное масштабирование. Вертикальное масштабирование предполагает добавление большего количества ресурсов к отдельному серверу. Для очень большого набора данных это означало бы добавление большего количества жестких дисков, и таким образом весь набор данных мог бы разместиться на одном сервере. В случае вычислительных операций это означало бы перемещение вычислений в более крупный сервер с более быстрым ЦП или большим количеством памяти. Горизонтальное масштабирование в отличии от вертикального предполагает добавление большего количества узлов. В случае большого набора данных это означало бы добавление второго сервера для хранения всего объема данных, а для вычислительного ресурса это означало бы разделение работы или загрузки через некоторые дополнительные узлы. В нашем примере единственный файловый сервер, используемый для хранения изображения, можно заменить множеством файловых

серверов, содержащих уникальный набор изображений. Такая архитектура позволит системе заполнять каждый файловый сервер изображениями, добавляя дополнительные серверы, по мере заполнения дискового пространства.

Конечно, есть трудности в распределении данных на множество серверов. Один из ключевых вопросов является местоположение данных. В распределенных системах, чем ближе данные к месту проведения операций, тем лучше производительность системы. Следовательно, распределение данных на множество серверов проблематично, поскольку в любой момент появляется риск того, что данных может не оказаться по месту требования и серверу придется выполнить затратную выборку необходимой информации по сети. Другая потенциальная проблема возникает, когда различные сервисы выполняют считывание и запись на совместно используемом ресурсе. Например, если бы один клиент отправил запрос обновления заголовка изображения в тот момент, когда другой клиент считывал это изображение. В такой ситуации неясно, какой заголовок был бы получен вторым клиентом.

Теперь рассмотрим вопрос масштабирования доступа к данным. Самые простые приложения работают по следующему принципу: пользователь через Интернет обращается к серверу приложений, а тот в свою очередь взаимодействует с сервером базы данных. С ростом приложения возникают две основных сложности: масштабирование доступа к серверу приложений и к базе данных. Большинство систем может быть упрощено к виду, когда пользователи напрямую обращаются к данным. Предположим, что имеется огромный объем данных, и пользователи могут получать доступ к небольшим частям этих данных в произвольном порядке. Получить доступ к конкретным данным особенно трудно, потому что загрузка большого объема данных в память может быть очень накладной и непосредственно влияет на количество дисковых операций ввода-вывода. Кроме того, даже с уникальными идентификаторами, решение проблемы нахождения местонахождения небольшой порции данных может быть очень трудной задачей. К счастью существует много подходов, которые можно применить для упрощения, из них четыре наиболее важных подхода — это использование кэшей, прокси, индексов и балансировщиков нагрузки.

Кэширование дает выгоду за счет базового принципа: недавно запрошенные данные вполне вероятно потребуются еще раз. Кэши используются почти на каждом уровне вычислений: аппаратные средства, операционные системы, веб-браузеры, веб-приложения. Кэш походит на кратковременную память: ограниченный по объему, но более быстрый, чем исходный источник данных, и содержащий элементы, к которым недавно получали доступ. Кэши могут существовать на всех уровнях в архитектуре, но часто находятся на самом близком уровне к фронтэнду, где они реализованы, чтобы возвратить данные быстро без значительной нагрузки бэкэнда.

На базовом уровне прокси-сервер — промежуточная часть аппаратных средств программного обеспечения, которые получают запросы от клиентов и передают их к серверам. Как правило, прокси используются, чтобы фильтровать, преобразовывать и протоколировать запросы. Прокси также очень полезны при координировании запросов, поступающих от большого количества серверов, что дает возможность оптимизировать трафик запроса в масштабе всей системы. Один их способов использования прокси для ускорения доступа к данным заключается в объединении одинаковых или схожих запросов и передачи единого ответа клиентам запроса.

Использование индекса для получения быстрого доступа к данным — известная стратегия для того, чтобы эффективно оптимизировать доступ к данным. Наиболее широкое применение индексирование находит в различных базах данных. Индекс делает взаимные уступки, используя издержки объемов хранения данных и снижая скорости операций записи, позволяя получить выигрыш в виде более быстрых операций чтения.

Балансировщики нагрузки являются важной частью любой архитектуры, поскольку их роль заключается в распределении нагрузки между узлами, ответственными за обслуживание запросов. Их основная цель состоит в том, чтобы обрабатывать много одновременных соединений и направлять эти соединения к одному из запрашиваемых узлов, позволяя системе масштабироваться, просто добавляя узлы, чтобы обслужить большее количество запросов. В распределенной системе балансировщики нагрузки часто находятся на первом этапе обработки информации системой, так что все входящие запросы проходят непосредственно через них. Балансировщики нагрузки также обеспечивают критическую функцию проверки работоспособности узлов. Если по результатам такой проверки узел не отвечает или перегружен, то он может быть удален из пула обработки запросов, и, благодаря избыточности системы, нагрузка будет перераспределена между оставшимися рабочими узлами [2].

Разработка эффективных систем с быстрым доступом к большому количеству данных является очень интересной темой, и существует большое количество различных подходов, которые позво-

ляют правильно сформировать архитектуру систем на ранних стадиях разработки. При проектировании распределенных веб-систем всегда возникает ряд трудностей, при решении которых придется жертвовать одними принципами для наиболее полного использования других. Полезными способом при разработке масштабируемой системы будут: разделение функциональности на сервисы, использование избыточности для решения проблем отказов, использование сегментирования данных. С ростом приложений часто прибегают к подходам упрощения систем, основными из которых являются: использование прокси, индексов, кэшей и балансировщиков нагрузки.

ЛИТЕРАТУРА

1. The Architecture of Open Source Applications [Электронный ресурс]. – Режим доступа: http://www.aosabook.org/en/. – Дата доступа: 29.09.2015.
2. Хабрахабр [Электронный ресурс] / Масштабируемая веб-архитектура и распределенные системы. Режим доступа: http://habrahabr.ru/ post/185636/. – Дата доступа: 29.09.2015.

УДК 621.396.69

ДАТЧИКИ ОХРАНЫ ПОМЕЩЕНИЙ

В.С. ЗЫБАЙЛО
(Представлено: канд. техн. наук, доц. Д.А. ДОВГЯЛО)

Рассмотрены принципы действия датчиков, применяемых в охранных системах, приведена их сравнительная характеристика. Показаны способы повышения их функциональности.

Человек глазами воспринимает форму, размеры и цвет окружающих предметов, ушами слышит звуки, носом чувствует запахи. Для формирования ощущений человеку необходимо внешнее раздражение определенных органов – "датчиков чувств" [1]. Датчики играют ту же роль для аппаратуры, что и органы чувств для человека – они помогают непосредственно общаться аппаратуре с внешним миром. В настоящее время датчики являются неотъемлемой частью технических устройств.

В последнее время в связи с удешевлением электронных систем все чаще применяются датчики со сложной обработкой сигналов, возможностями настройки и регулирования параметров, стандартным интерфейсом системы управления. Имеется определенная тенденция расширительной трактовки и перенесения этого термина на измерительные приборы, появившиеся значительно ранее массового использования датчиков, а также по аналогии - на объекты иной природы, например, биологические[2]. Датчики используются во всех отраслях промышленности и жизнедеятельности человека, так же не стали исключением и охранные системы. Немаловажную роль в развитие систем охраны сыграло и развитие датчиков различных физических величин, которые, включаясь в системы охраны, расширяют их функциональные возможности.

Рассмотрим типичные датчики, входящие в системы охранной сигнализации. Датчики могут строиться на различных принципах работы, но наиболее широкое распространение в охранных системах получили только некоторые типы датчиков. В стандартный комплект датчиков охранных систем входят: датчик разбития стекла, датчик движения и датчик открытия и закрытия дверей. Датчик движения представляет собой пироэлемент, при воздействии на чувствительный элемент которого теплового потока от движущегося объекта, на электродах датчика появляется разность потенциалов. Инфракрасный датчик движения SWAN QUAD имеет широкую диаграмму направленности, что позволяет охватывать обширные зоны размером до 15 м, однако необходимо использовать специальные фильтрующие линзы (“широкий угол”)[2]. За счет хорошо отлаженной системы обработки данных центральным процессором охранной системы практически исключаются ошибки считывания информации с охраняемой зоны, например ошибки связанные с нарушением охраняемой зоны домашними питомцами. Однако иногда ошибки считывания неизбежны, именно поэтому целесообразно подключение еще одного датчика движения для сравнения данных. Весьма перспективными в последнее время стали так называемые комбинированные датчики движения, которые сочетают в себе сразу несколько принципов работы, что позволяет повысить точность измерения. Одним из таких датчиков является комбинированный цифровой датчик PARADOX 525D (рис. 1).

Наличие в системе охраны комбинированных датчиков влечет повышение функциональных возможностей, избежание ложных срабатываний и повышения точности измерений.