или иные услуги, составление отчетов, обработка большого количества данных. Такую задачу предстояло решить автору в ходе работы в ОСП МЦ БГУИР (далее - МЦ), предоставляющего в числе прочего услуги пользования локальной сетью общежитий БГУИР студентам.

Локальная сеть общежитий БГУИР №1 и №2 изначально строилась студентами на добровольных началах. В то время проблемы автоматизации процессов управления не возникало из-за небольшого количества активных пользователей. Однако после введения в эксплуатацию третьего и четвертого общежитий, когда количество активных пользователей резко увеличилось почти в два раза. Тогда необходимость внесения изменений в действующий процесс работы с пользователями стала очевидной.

Вкратце процесс пользования сетью для пользователя можно описать следующим образом: чтобы зарегистрироваться в сети, каждый пользователь должен распечатать договор, заполнить его и лично отнести системному администратору общежития, чтобы тот вручную добавил его данные в систему. Затем нужно оплатить пользование сетью, для чего в свою очередь нужно в один из рабочих дней прийти в один из нескольких пунктов распечатки университета в рабочее время, где заполнить от руки бланк оплаты, произвести оплату и получить чек. Несколько раз в неделю кассир лично относит бланки об оплате в бухгалтерию МЦ, где эта информация также вручную добавляются в систему.

Этот процесс может занять до двух недель в зависимости от суммарного количества оплат и наличия у бухгалтеров других задач (например, отчетность). В виду ручной обработки, информация об оплате иногда теряется, и в таком случае пользователю нужно идти в бухгалтерию и лично решать вопрос о зачислении оплаты. Также не налажен процесс переезда пользователя в другое общежитие, в связи с чем при переезде нужно заново регистрироваться в сети другого общежития, переносить оплаты из одной учетной записи в другую, несмотря на то, что по сути все общежития БГУИР объединены в общую локальную сеть.

Разработанный автором программный комплекс является полностью автоматизированной системой контроля пользования услугами предоставления доступа к сети пользователям всех общежитий БГУИР, которая принимает оплаты в электронном виде и практически полностью исключает участие человека в процессах регистрации пользователей и приема оплат. Архитектура приложения подразумевает наличие головного сервера, который отвечает за сбор и обработку информации о произведенных платежах, регистрацию и авторизацию пользователей и ведение статистики, а также небольшие клиент-серверных приложений-сателлитов, которые предоставляют интерфейс управления пользователями определенного сегмента сети. Данная архитектура позволяет сконцентрировать взаимодействие пользователей и администраторов с базой данных пользователей на работе с одним приложением, вне зависимости от общежития, вести общий учет пользователей по всем общежитиям, но при этом предоставляет возможность относительно легко добавлять новые общежития или другие сегменты сети в систему, в том числе удаленные.

Таким образом разработанное приложение позволяет иметь единую точку авторизации и приема оплат для всех пользователей, что решает проблему смены общежития пользователем. Кроме того благодаря приложениям-сателлитам, разработанным для каждого общежития отдельно, и работающим в режиме реального времени, есть возможность устанавливать достаточно жесткие, но справедливые ограничения на работу пользователей; в частности одна учетная запись может быть активная только в одном сегменте сети, что исключает одновременное использование одной учетной записью людьми из разных общежитий. Также полностью решена проблема личного присутствия пользователя в том или ином месте при регистрации или произведения оплаты: регистрация происходит в режиме онлайн с помощью веб-приложения, а оплаты принимаются через платежную систему ЕРИП. Кроме того разработанное приложение может быть адаптировано для автоматизации других процессов, связанных со студентами, проживающими в общежитиях: обработка заявок пользователей на устранение неисправностей системным администраторам, столяру, сантехнику или электрику; прием оплат за велопарковку; организация выполнения студентами дежурств и общехозяйственных работ, а также организация доступа студентов в общежитие по электронными пропусками при наличии соответствующего оборудования.

Список использованных источников:

1. Дмитрий Котеров, Алексей Костарев. PHP 5 в подлиннике / Алексей Костарев Дмитрий Котеров. — 2-е изд. — BHV (Россия), 2008. — June. — 1104 с.
2. Дюбуа, Поль. MySQL / Поль Дюбуа. — 3-е изд. — «Вильямс», 2006. — 1168 с.

ПРАКТИКА ПОСТРОЕНИЯ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Божко С. С.

Пилецкий И. И. – кандидат физико-математических наук, доцент

При проектировании и разработке высоконагруженных серверных приложений для обработки и анализа большого объема данных основной задачей является обеспечение надежной и быстрой обработки сообщений от клиентских

программ, причем, с возможностью как потоковой обработки в режиме реального времени, так и отложенной обработки для расчета более ресурсоемких задач. Надежность и скорость при обработке информации достигаются применением особых подходов, закладываемых в архитектуру системы. Именно архитектура, как на программном, так и на аппаратном уровнях, является важнейшей частью подобных проектов.

Существует множество технологий и подходов, правильное применение которых способно в значительной степени облегчить разработку высоконагруженных и надежных систем. В дальнейшем речь пойдет об инструментах, позволяющих достичь такого результата.

Сервис-ориентированная архитектура, а в современной интерпретации – это гетерогенная программная композиция слабосвязанных микросервисов. Это значит, что большая система разбивается на части в соответствии с функциональной декомпозицией. Каждая часть – это отдельный сервис. Поскольку сервисы мало зависимы друг от друга, то система в целом становится более устойчива к сбоям ее отдельных элементов: при отказе одной части другие могут продолжать свою работу, за счет частичной потери функциональности или за счет замены другими сервисами. По такому же принципу поострена архитектура Twitter [1].

Как известно, физическое обращение к данным в базе данных является дорогостоящей операцией с точки зрения эффективности и процессорного времени. Поэтому для задач, в которых чтение данных осуществляется довольно часто, а читаемых данных не очень много, имеет смысл введение многоуровневое кэширование данных. Количественная эффективность кэша определяется процентом попаданий в кэш, когда в нем найдено искомое значение. К проблемам данного подхода можно отнести устаревание данных и когерентность кэшей [2].

Возможность масштабирования, как вертикально, так и горизонтально, является неотъемлемой частью архитектур высоконагруженных систем. При этом, у этих двух подходов есть ограничения. Для вертикального масштабирования – это уровень и стоимость необходимого аппаратного обеспечения, но зато результат виден очень быстро. Горизонтальное масштабирование состоит в том, чтобы поставить несколько равноправных, не имеющих ничего общего серверов. Они выступают как единое целое. А отсутствие хранимого состояния, общих узлов и единой точки отказа позволяет с относительной легкостью запускать новые экземпляры при резком росте нагрузки. Для того, чтобы обращаться к этим серверам как к единому целому, вводится новая сущность – балансировщик нагрузки. Наряду с равномерным распределением запросов, не менее важной задачей является обеспечение высокой доступности самого балансировщика, чтобы избежать возникновения единой точки отказа. Поэтому в качестве балансировшика может выступать аппаратное, программное решение, а также их комбинация. Если используется протокол HTTP, то обычно широко применяется механизм DNS в сочетании с алгоритмом балансировки Round Robin [3].

Зачастую обычного горизонтального масштабирования в сочетании с баланировщиками недостаточно. Нужно иметь возможность добавление вычислительной мощности работающему приложению без перезарузки и переконфигурирования. В таком случае широкое применение нашли децентрализованные распределенные системы, построенные на принципе «равный равному» (peer-to-peer, p2p) [4]. Компонентам системы не нужно знать об общей структуре сети. Распространение информации внутри системы происходит через цепную пересылку сообщений через соседей, например, протоколы BitTorrent [5] или Gossip [6]. На этой основе построено множество нереляционных баз данных, потому как добавление серверов без перезагрузки системы крайне важно для масштабирования.

Как правило, в бизнес-приложениях встречаются задачи, которые не требуют немедленного выполнения, которые могут быть выполнены параллельно с текущими или же асинхронно. Примером таких задач служит рассылка почтовых уведомлений, обсчет статистики, обновление ленты новостей. А при работе с большими данными всегда есть долгие, ресурсоемкие задачи (обработка фото, видео). В таких случаях имеет смысл применять очереди сообщений или шины данных. Архитектурное решение «подписка – публикация» позволяет в значительной степени уменьшить связность компонентов системы. А с учетом того, что современные решения позволяют делать очереди персистентными, то это дает колоссальные возможности по обработке большего потока событий без боязни потери транзакции.

Шардинг и партиционирование данных в большей степени отпросятся к архитектуре баз данных. Концептуально это два близких понятия. Концепция шардинга заключается в логическом разделении данных по различным ресурсам исходя из требований к нагрузке. Например, множество пользователей разбивается на N частей, где за каждую часть отвечает отдельный шард – экземпляр хранилища (node). Этот подход позволяет равномерно распределять нагрузку на БД. Термин «партиционирование» подразумевает разделение больших таблиц на логические части по выбранным критериям. Хорошей иллюстрацией здесь является новостной портал, где чтение в большинстве случаев приходится только на самую последнюю часть таблиц (то есть активно читаются только те данные, которые недавно появились).

Инструмент репликации тоже позволяет распределять нагрузку более сбалансированно. Это синхронное либо асинхронное копирование с ведущих серверов на ведомые (master-slave репликация) или другие ведущие (master-master репликация). В таком походе можно разделить чтение данных и их изменение. Чтение всегда осуществлять с ведомых, а изменение – в ведущих.

В зависимости от того, какие данные нужны каждому конкретному компоненту, как часто они будут запрашиваться, иногда возникает необходимость в денормализации данных. Намеренное введение избы-

точности в некоторые случаях позволяет исключить дорогостоящую операцию join на таблицах. Например, анкета пользователя, включающая сотню полей, но при этом часть из них всегда отображается на клиентских приложениях. В такой ситуации имеет смысл ввести избыточность. А если паттерны работы с данными кардинально различны, то есть смысл задуматься о введении для этих случаев своих хранилищ с разными схемами данных, оптимизированными под конкретный вариант использования. Но это все достаточно специфично для ситуации.

Рассмотренные походы для построения высоконагруженных систем с учетом требований к надежности, могут применяться как по отдельности, так и вместе. Все зависит от принятых архитектурных решений и бизнес-логики системы, а также понимания того, что может делать пользователь в системе и правил обработки данных. При принятии архитектурных решений, нужно определять планируемый порядок обработки объема данных, их скорость прироста, соотношение чтения/записи, решить какова допустимая деградация системы, чем можно пренебречь. Понимание всех этих проблем позволяет выполнить проектирование системы с учетом полученной информации, а также с применением вышеперечисленных принципов масштабирования данных.

Список использованных источников:

1. Decomposing Twitter: Adventures in Service-Oriented Architecture [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.infoq.com/presentations/twitter-soa.
2. Cache coherence [Электронный ресурс]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/Cache_coherence.
3. Round-robin DNS [Электронный ресурс]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/Round-robin_DNS.
4. Rüdiger Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002).
5. Bittorrent Protocol Specification v1.0 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://wiki.theory.org/BitTorrentSpecification.
6. Gossip protocol [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Gossip_protocol.

СПОСОБЫ АВТОМАТИЗИРОВАННОГО АНАЛИЗА РАСХОЖДЕНИЙ ЗНАЧЕНИЙ ПАРАМЕТРОВ НА БОЛЬШИХ СОВОКУПНОСТЯХ ДАННЫХ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Мыц С. И.

Волорова Н. А. – канд. техн. наук, доцент

В условиях наличия больших объемов данных возникает необходимость в автоматизированном анализе их параметров. Ни одному человеку или группе людей не будет под силу вручную обработать весь объем информации поступающей от достаточно крупной современной информационной системы. Для решения такой задачи применяются инструменты, созданные с использованием алгоритмов и математических методов.

Взглянем на некоторый частный случай и на его примере рассмотрим возможные подходы к решению проблемы. Допустим, у нас есть некоторый веб-сайт, к которому пользователи задают запросы, а затем как-то взаимодействуют с результатами этих запросов. Чтобы исследовать это взаимодействие в глобальном плане, вводятся параметры, считаемые по некоторому множеству запросов. Эти параметры назовём метриками.

В контексте текущей задачи метрикой будем называть некоторую числовую величину, которая считается на основе данных из множества запросов и является показателем некоторого интересного нам свойства этого множества. В качестве примеров простых метрик можно привести следующие:

- общее количество действий пользователя со всеми результатами запроса
- отношение количества действий пользователя с опрделённым результатом к количеству показов этого результата
- среднее время до первого взаимодействия пользователя с результатами в рамках запроса
- количество пользователей, взаимодействующих с системой

Могут возникать ситуации, когда значения метрик изменяются и нам нужно понять, по каким причинам это произошло и что больше всего повлияло на это изменение. Это можно делать с помощью анализа значений метрик на срезах данных. Срезом данных назовём подмножество исходного множества запросов, взятое по какому-то критерию.

Теперь можно обобщить постановку задачи. Пусть у нас имеется множество объектов (см. множество запросов), имеется числовая функция от произвольного подмножества объектов (см. метрика) и есть