В первую очередь, балансировка применяется для распределения нагрузки между нашими серверами. Во-вторых, за счет балансировки мы можем повысить отказоустойчивость нашей системы, т.е., например, если один из наших серверов в кластере выходит из строя, то нагрузку на себя принимает второй, если сможет ее потянуть. Благодаря балансировке также достигается некая защита от некоторых видов атак, например, атаки на все соединения.

К балансировке, как и к любой системе, предъявляются некие требования. Требования такие:

балансировка должна удовлетворять требованиям справедливости, т.е. любой запрос, пришедший в нашу систему, должен быть обслужен, а не просто брошен;

балансировка должна быть эффективной, т.е. мы должны обеспечить такую ее работу, чтобы все наши сервера в кластере работали примерно равномерно, и брали на себя равномерную нагрузку;

благодаря балансировке должно сокращаться время выполнения запроса, т.е. должно обеспечиваться сокращение времени отклика — как только запрос приходит к нам в систему, мы должны как можно быстрее его обслужить, на него ответить;

балансировка должна удовлетворять требованиям предсказуемости, т.е. мы должны четко понимать какой алгоритм балансировки и в каком случае мы должны использовать;

равномерность загрузки системы, в принципе, это требование схоже с эффективностью;

балансировка должна быть масштабируемой, т.е. при резком увеличении нагрузки система балансировки должна обеспечивать стабильную работу нашего сервиса.

Балансировки условно можно разделить на два вида по географическому признаку — балансировка может быть локальной, если у нас сервера расположены внутри одного дата-центра, и балансировка может быть глобальной, если наш ресурс раскидан на сервера по разным дата-центрам.

Подробнее о каждом.

Локально систему балансировки можно применять:

на канальном уровне, как с использованием отдельного балансировщика, так и без него;

на сетевом уровне;

на транспортном уровне.

Это наиболее распространенные способы, которые применяют при локальной балансировке.

Балансировка на канальном уровне выполняется за счет следующего. Мы берем и навешиваем на некий специализированный интерфейс всех наших серверов один и тот же IP-адрес нашего ресурса, на который будут приходить запросы, и с которого будут уходить ответы. Но на ARP-запрос с этого IP-адреса сервера не должны отвечать. И мы навешиваем такой же IP-адрес на наш балансировщик, соответственно, на него будут приходить запросы, и отправляться ответы с него, и он же будет отвечать на ARP запросы. Т.о., получая запрос от клиента, наш балансировщик выбирает по определенному алгоритму тот или иной сервер, который будет обрабатывать этот запрос, подменяет destination MAC и отправляет его на обработку на данный сервер. Сервер его у себя обрабатывает, и т.к. мы не делали подмену заголовков на сетевом уровне, то непосредственно, минуя балансировщик, сразу отвечает клиенту через наш шлюз.

Тут к нам приходит руководство, и говорит: «Ребят, мы тут вконтакте посидели, полистали, всякие мемы почитали и поняли, что мужики от своих жен очень часто борща требуют. Подумали мы и решили, что давайте-ка мы сейчас рекламную компанию свеклы сделаем, и будем свеклу продавать. И все деньги вбухаем туда. А вас попросим как-нибудь сократить расходы на вашу классную систему балансировки».

Мы подумали и решили: а почему бы нам не избавиться от балансировщика? Но при этом прикинули, как мы можем без него реализовать балансировку.

А реализовать мы можем ее очень просто: нам необходимо превратить входящий unicast запрос в broadcast, либо в multicast, кто как хочет.

Делается это следующим образом. Все сервера должны на ARP запрос отвечать одним и тем же MAC-адресом, т.е. это может быть либо несуществующий MAC-адрес, либо какой-то мультикастовый. Либо мы можем навесить этот мультикастовый MAC-адрес на наш шлюз. Соответственно, запрос приходит на наш ресурс, и шлюз его просто размножает ко всем серверам, т.о. запросы поступают на все сервера одновременно, и каждый сервер должен сам понимать, должен ли он отвечать на запрос или нет. И понимать может очень просто, можно поставить, например, поставить деление на нуль srcIP, и дальше дело техники.

Плюсы и минусы алгоритма балансировки на канальном уровне.

В первую очередь, он не зависит от протоколов вышележащих уровней, т.е. можно балансировать нагрузку как по HTTP, так и по FTP, так и по SMTP. Затем, мы сокращаем расходы за счет отказа от выделенного балансировщика, в случае, если мы его не используем. Обратный трафик в сторону клиента не нагружает балансировщик, что тоже хорошо, например, для HTTP, когда у нас входящий запрос, как правило, легкий, а ответ на него весит порой в десятки и сотни раз больше. Затем, в современных реалиях очень полезный для нас плюс — это то, что мы используем только один публичный адрес, т.к. публичные IP нынче дорогие, это несомненно огромный плюс для нашей системы. И такая система способствует быстрому добавлению отключения серверов в кластере.

Из минусов стоит назвать: необходимость размещения сервера в одном сегменте, ограничение по входящей полосе в случае с разделяемыми адресами, т.к. трафик попадает на все сервера одновременно, нагружая нашу систему.

Из решений, использующих данный алгоритм балансировки, можно назвать Linux Virtual Server.

Балансировка на сетевом уровне. В принципе, механизм довольно схожий с балансировкой на канальном уровне, за одним единственным отличием — в данном случае, получая входящий запрос, наш балансировщик подменяет destination IP, соответственно, применяет его на тот сервер, который будет обрабатывать запрос. Сервер получает его, обрабатывает и должен передать его обратно балансировщику, чтобы тот выполнил обратную подмену.

Плюсы такого метода: он также не зависит от протоколов высокого уровня; полная прозрачность работы для сервера, т.е. сервер думает, что т.к. он получает запрос от клиента и работает с ним непосредственно напрямую, минуя всякие балансировщики, он не знает о них ничего; и здесь так же, как и в предыдущем методе, мы используем один публичный адрес, что тоже является большим плюсом.

Из недостатков — это повышенная нагрузка на балансировщик за счет обратного трафика. Каждый сервер должен отправить ответ, в первую очередь, на балансировщик, чтобы тот выполнил обратную подмену. Соответственно, в случае с HTTP нагрузка на балансировщик тоже будет большой.

Балансировка на транспортном уровне. Здесь очень тонкая грань, которая отличает балансировку на сетевом от балансировки на транспортном уровне. Ну, для простоты скажем, что в данном виде балансировки используются при балансировании нагрузки входящие порты источника и адресата.

Очень интересный пример реализации данного вида балансировки — это балансировка при помощи т.н. алгоритма ECMP, т.е. equal-cost multi-path. Выяснилось, что все современные роутеры могут распределять, балансировать нагрузку сами. Для этого нам достаточно на роутер анонсировать одну и ту же подсеть по разным маршрутам. Балансировщик в данном случае имея два одинаковых маршрута по своим общим метрикам будет распределять нагрузку по ним равномерно.

Но существует ряд нюансов, которые тоже надо учитывать. В первую очередь, наш роутер должен распределять нагрузку таким образом, чтобы пакеты в рамках одной TCP сессии попадали на один и тот же сервер. Это раз. Такой режим на роутерах, по-моему, на современных Cisco’ах называется perdestination и perflow и поддерживается по умолчанию.

Следующий нюанс — если мы пропишем на роутере статические маршруты, то мы должны как-то автоматизировать процесс добавления и удаления серверов из нашего кластера. Для того чтобы избежать этой проблемы, мы можем использовать различные протоколы маршрутизации, такие как BGP. Т.е. на каждый наш сервер мы устанавливаем некий софтварный BGP роутер, который будет анонсировать серверную сеть на роутер, который принимает запросы. Из софтварных таких роутеров можно назвать Квага или Bird.

И также необходимо учитывать, что существуют плюсы и минусы этого метода:

Преимущества такого алгоритма балансировки на транспортном уровне: он не зависит от протоколов высокого уровня, как и три предыдущих метода; также используется один публичный адрес; отсутствует необходимость приобретать дополнительное оборудование.

Но существуют свои недостатки, такие как, например, необходимость ставить на сервер дополнительный софт. Как я уже говорил, это программные роутеры BGP, хотя они не сильно нагружают наши сервера. Отсутствие server-affinity, т.е. все соединения будут разрываться в том случае, если мы добавляем или удаляем сервера из кластера. Также существует проблема ограниченности одинаковых маршрутов на разных роутерах. На слайде внизу приведены несколько роутеров марки Cisco — для первых двух одновременно поддерживается восемь одинаковых маршрутов, а для последнего поддерживается до 32-х одинаковых маршрутов. Также мы должны обеспечивать равномерность нагрузки, что предъявляет некие требования к производительности наших серверов, например, если мы добавим более производительный сервер в наш кластер, то нагрузка на него будет распределяться ровно такая же, как и на все остальные. И последний минус — это таймауты BGP протокола, т.е. если сервер перестает слать на роутер, анонсировать свою сеть, т.е. он вышел из строя, роутер в течение некоторого таймаута он может все еще распределять нагрузку по данному маршруту.

Из методов глобальной балансировки можно выделить следующие наиболее распространенные методы:

балансировка на уровне DNS,

балансировка на прикладном уровне — это проксирование и redirect запросов,

балансировка на сетевом уровне — рассмотрим алгоритм Anycast.

В DNS балансировке чаще всего применяют т.н. алгоритмы Round Robin, т.е. это самый простой механизм балансировки, и балансировать таким образом можно любые системы, в которых доступ к сервису происходит по имени. Суть его вот в чем: на DNS сервер просто добавляется несколько А-записей с разными IP-адресами всех наших серверов, и сервер сам будет в цикличном порядке выдавать эти адреса. Т.е. первый запрос получит первый сервер, второй запрос — второй сервер, третий запрос — третий сервер, четвертый запрос — первый сервер и т.д.

Плюсы этого алгоритма: он абсолютно не зависит от протоколов высокого уровня; не зависит от нагрузки сервера, благодаря тому, что на всех клиентах, в основном, есть кэширующие DNS сервера, которые позволяют в случае резкого увеличения нагрузки на наш сервис компенсировать эту проблему; универсальность, т.е. DNS балансировку может выполнять как на локальном уровне в рамках одного дата-центра, так и на глобальном уровне. Последний самый главный плюс такого алгоритма балансировки — это очень низкая стоимость и быстрый старт, т.к. любой сайт фактически имеет доменное имя, имеет DNS сервер, соответственно, добавив несколько А-записей можно добиться балансировки уже на старте.

Из минусов следует назвать следующие. Сложность отключения серверов, в случае, например, их выхода из строя. В данном случае необходимо предусматривать некоторые способы резервирования этих серверов, например, по протоколу CARP или VRRP. Также сложно распределять нагрузку в нужной пропорции, т.к. DNS сервер ничего не знает о том, насколько загружен каждый сервер, а только лишь выдает их IP. Ограниченность IP-адресов — это наиболее весомый минус данного алгоритма балансировки, т.к. каждый сервер должен иметь свой глобальный IP-адрес, а как я уже говорил, IP-адреса нынче дорогие и ограничены. Кроме того, необходимо держать двойной запас серверной мощности, в том случае, когда у нас один из серверов выйдет из строя, а клиент будет ломиться на его IP и не получит нашего сервиса.

Из реализаций можно назвать любой DNS сервер, в качестве примера можно привести сервер Named из пакета BIND, PowerDNS сервер и т.д.

Следующий алгоритм — алгоритм проксирования. Суть его заключается в том, что в качестве балансировщика применяется т.н. умный прокси. Т.е. если балансировщик получает запрос к нашему ресурсу, он анализирует заголовки прикладного уровня, соответственно, он может понимать, запрос к какому ресурсу пришел на наш балансировщик, и направить запрос на тот или иной сервер, на котором этот ресурс содержится. Плюс ко всему, при получении этого запроса балансировщик может добавлять в заголовки HTTP, например, информацию о том, с какого IP пришел клиент, для того, чтобы сервер знал, куда его потом впоследствии отправлять, и с кем он работает. Выполнив запрос, сервер передает его обратно на балансировщик, тот выполняет необходимые манипуляции с новыми заголовками либо третьего уровня, либо седьмого уровня и отдает его клиенту.

Преимущества такого алгоритма балансировки в том, что мы можем обеспечить server-affinity, т.е. мы можем привязать конкретного клиента к определенному серверу, например, используя различные настройки cookie. Затем, мы можем распределять разные типы запросов разным серверам, т.е. мы, например, можем на одном сервере держать статику, которая у нас мало весит, а на другом сервере держать какой-то тяжелый контент, и, соответственно, понимая, анализируя HTTP заголовок, мы можем направлять запрос пользователя на тот или иной сервер. Благодаря этому алгоритму балансировки мы также можем фильтровать запросы по URL, защищаясь тем самым от различных родов атак, и самостоятельно определять работоспособность каждого узла, т.е. не просто доступность каждого сервера на сетевом уровне, но и понимать насколько работоспособен, в данном случае, наш софт на сервере, и не применять при этом никаких дополнительных средств.

Из минусов:

необходимость балансировать нагрузку на самих балансировщиках;

это дополнительная точка отказа нашей системы;

потребление очень большого количества ресурсов, т.к. анализируются запросы прикладного уровня;

необходимость иметь свой прокси для каждого вида протоколов.

Из реализаций можно назвать такие программные веб-сервера как HAProxy либо nginx.

Redirect запросов. Redirect запросов имеет довольно ограниченное применение — применяется, в основном, для глобальной балансировки, и, в частности, для HTTP он хорошо применим. Суть его заключается в том, что мы получаем запрос от клиента на наш балансировщик, балансировщик отвечает ему редиректом на наш сервер, на котором содержатся ресурсы. Например, получая запрос по HTTP, балансировщик отвечает ему в ответ — выдает ошибку 302 move temporary с указанием адреса того сервера, на который дальше будет ходить наш клиент.

Плюсы такого алгоритма балансировки в том, что он распределяет запрос по разным серверам за счет анализа заголовков прикладного уровня.

Минусы — это достаточно малая применимость к протоколам высокого уровня; увеличение времени отклика за счет обращения к редиректору; фактически на каждый запрос клиента мы имеем по два запроса, т.е. первый запрос идет к балансировщику, второй запрос клиент делает непосредственно к серверу. Таким образом, если, например, клиент запрашивает какой-то контент, который порезан кусками, то на каждый кусок клиент будет выполнять по два запроса, что резко увеличивает время обслуживания клиента.

Из решений можно назвать программный веб-сервер nginx.

И последний вид балансировки, о котором я хотел рассказать, — это балансировка на базе Anycast. Этот алгоритм балансировки не требует никакой настройки со стороны клиента, и суть его заключается в следующем: мы из разных географических участков анонсируем один и тот же префикс сети. Таким образом, каждый запрос клиента будет маршрутизироваться на ближайший к нему сервер, который будет его обрабатывать.

Плюсы такого метода в том, что мы обеспечиваем минимальную задержку при обработке запроса, т.к. клиент будет обслуживаться на ближайшем к нему сервере не только с географической точки зрения, но и с топологической. Мы обеспечиваем доставку трафика, минуя магистральные каналы связи, соответственно, получаем некоторое удешевление трафика. Еще плюс в том, что нагрузку распределяет сама сеть, — мы здесь уже фактически никак не участвуем и об этом не заботимся, т.е. этот процесс ложится на плечи Интернет-провайдера. Высокая отказоустойчивость данного алгоритма балансировки, например, если один из серверов выходит из строя, то все запросы к нему будут просто переброшены на ближайший от него сервер. Легко добавлять и выводить из работы любые наши сервера — просто перестаем анонсировать по BGP, например, подсеть, и все запросы пользователей будут маршрутизироваться на другие сервера.

Из минусов следует назвать то, что существует возможность перестроения маршрутов, что для TCP сессии достаточно критично. Например, если в рамках одной сессии пакеты пойдут по другому маршруту на другой сервер, мы получим просто ресет по TCP, и сессия прервется. Отсутствует возможность проконтролировать, с какого узла обслуживается пользователь, т.к. этим рулит сама сеть, мы не можем знать, где сейчас в данный момент обслуживается пользователь, не прибегая к различным дополнительным решениям. Дорогое оборудование, т.е. если, например, мы будем использовать аппаратные роутеры, а это достаточно дорогая штука. Еще такой важный момент, что при использовании балансировки на базе Anycast необходимо учитывать интересы Интернет провайдеров. Т.к. все мы знаем, что Интернет-провайдеры любят пиринговаться друг с другом и, соответственно, в идеале пакеты должны ходить по наиболее короткому маршруту, но по факту получается так, что Интернет-провайдер передает пакеты там, где им дешевле, и это необходимо иметь в виду.

Какие алгоритмы балансировки применяются? О методах уже рассказали, теперь расскажу о том, каким образом выбираем, на какой сервер нам отправлять запросы. Существует несколько наиболее распространенных алгоритмов, о которых я хотел бы рассказать.

Про Round Robin я выше рассказал. Существенным его недостатком является то, что нагрузка в данном случае распределяется без учета особенностей конкретного сервера. Алгоритм Weighted Round Robin позволяет навесить на каждый сервер определенный весовой коэффициент, который будет учитывать мощность и производительность того или иного сервера, т.о. более производительный сервер будет получать запросы чаще.

Следующий алгоритм — Least Connections, т.е. в данном случае будут учитываться не просто нагрузка на сервер, но и количество одновременных соединений с данным сервером в данный момент. Если мы хотим, например, улучшить данный алгоритм, то мы можем использовать алгоритм Least Connections, например, с весами, т.е. навесив дополнительно на каждый сервер некий весовой коэффициент в соответствии с его производительностью и мощностью.

Destination Hash Scheduling и Source Hash Scheduling — это т.н. алгоритмы, которые анализируют IP-адрес либо источника, либо адресата и выбирают из некой статической таблицы тот или иной сервер, на который будет проксироваться дальше запрос.

И алгоритм Sticky Sessions — в данном случае обеспечивается привязка клиента к определенному серверу и, соответственно, все пакеты в рамках одной сессии будут ходить только на этот сервер.

Дальше хочу озвучить несколько решений интеграторов, потому что они часто применяются. Мы в своем сервисе эти решения не используем, поэтому я их просто назову. Здесь перечислены решения от Cisco — это решения Cisco ACE, некий хардварный балансировщик, который имеет теоретически пропускную способность до 16 Гбит в секунду и поддерживает практически все названные мною методы и алгоритмы балансировки. Также от Cisco есть решение под названием Cisco CSS, имеет гигабитные порты, позволяет балансировать трафик на сетевом уровне и выполнять глобальную балансировку на уровне DNS. От компании F5 решение — BigIP, довольно-таки гибкое, имеет свой скриптовый язык и благодаря ему, мы можем настраивать балансировку таким образом, как нам удобнее, и как мы этого хотим. И решение от компании radware, называется оно Alteon NG — выполняет и поддерживает практически все алгоритмы балансировки, позволяет анализировать настройку куки, выполнять привязку клиента к конкретному серверу и ряд других особенностей. Кому интересно, тот может погуглить и найти информацию, она доступна.

И в завершении я бы хотел рассказать о том, как мы применяем различные алгоритмы и методы балансировки, и как их можно всячески соединять.

Т.к. мы являемся провайдером SDN, у нас сеть серверов, которые распределены географически, соответственно, надо не просто балансировать нагрузку между серверами, нам надо еще и перенаправлять все запросы клиентов на ближайший к ним сервер. Как это работает? У нас имеется несколько балансировщиков, балансировщики у нас представляют из себя некий пропатченный DNS сервер, который мы выдаем, просто используя обычный DNS Round Robin. Получив его в свое распоряжение, клиент отправляет на него запрос на получение А-записи и, соответственно, адрес того сервера, который будет обрабатывать его зарос. Каждый балансировщик знает о доступности каждого сервера во всех локациях, которые распределены географически, соответственно, формирует у себя комплексную метрику, которая учитывает загруженность каждого сервера в данный момент, использование ресурсов центрального процессора, утилизацию памяти, доступность сервера с точки зрения сети, потом анализирует расстояние от каждого сервера до клиента и т.д. Т.е. вычисляется некая комплексная метрика, на основе которой принимается решение о том, куда мы должны перенаправить запрос этого клиента, и выбирается, как правило, ближайший не только с точки зрения географии сервер, но и с точки зрения топологии, т.о. сокращается время обслуживания каждого клиента и ускоряется сервис.