СОДЕРЖАНИЕ

Используемые в работе сокращения....7
Введение....8
1. Технологии балансировки нагрузки и трафика....10
1.1. Балансировка нагрузки и сетевая модель OSI....10
1.2. Балансировка нагрузки на физическом уровне модели OSI....11
1.3. Балансировка нагрузки на канальном уровне модели OSI....12
1.4. Балансировка нагрузки на сетевом уровне модели OSI....14
1.5. Серверная балансировка нагрузки....16
1.5.1. Трансляция сетевых адресов....18
1.5.2. Прямая маршрутизация....20
1.5.3. Туннелирование....22
1.6. Глобальная серверная балансировка нагрузки....24
1.6.1. GSLB посредством HTTP перенаправления....24
1.6.2. GSLB посредством DNS....26
1.6.3. GSLB посредством протокола маршрутизации....28
1.7. Сравнение открытых технологий балансировки нагрузки и трафика....29
2. Разработка приложения для глобальной серверной балансировки нагрузки....30
2.1. Формирование требований....30
2.1.1. Функциональные требования....30
2.1.2. Нефункциональные требования....31
2.1.3. Системные требования сервера....31
2.1.4. Системные требования клиента....31
2.2. Анализ и разработка требований....31
2.3. Проектирование архитектуры....42
2.4. Проектирование базы данных....45
2.5. Реализация....48
2.6. Тестирование....55
2.7. Оценка качества....59
Заключение....60
Список литературы....61
Приложения....63

ИСПОЛЬЗУЕМЫЕ В РАБОТЕ СОКРАЩЕНИЯ

BGP — Border Gateway Protocol

DNS — Domain Name System

DR — Direct Routing

DSR — Direct Server Return

ECMP — Equal-Cost Multi-Path

EIGRP — Enhanced Interior Gateway Routing Protocol

ERR — Enhanced Entity–Relationship model

GSLB — Global Server Load Balancing

IEEE — Institute of Electrical and Electronics Engineers

IP — Internet Protocol

IPIP — IP encapsulation within IP

IETF — Internet Engineering Task Force

LAG — Link Aggregation Group

LAN — Local Area Network

LCAP — Link Control Aggregation Protocol

LVS — Linux Virtual Server project

MC-LAG — Multi-Chassis Link Aggregation Group

NAT — Network Address Translation

OSI — Open Systems Interconnection model

OSPF — Open Shortest Path First

RFC — Request for Comments

RIP — Routing Information Protocol

RPM — RPM Package Manager

SLB — Server Load Balancing

TTL — Time To Live

UML — Unified Modeling Language

URL — Uniform Resource Locator

VIP — Virtual Internet Protocol address

VPN — Virtual Private Network

WAN — Wide Area Network

ВВЕДЕНИЕ

Балансировка нагрузки и трафика не является новой концепцией и повсеместно используется в ИТ-индустрии. Многие программно-аппаратные решения выполняют различные типы балансировки нагрузки и трафика.

Например, сетевые коммутаторы (switches) могут распределять трафик по нескольким физическим линиям связи, а маршрутизаторы (routers) могут балансировать трафик по нескольким маршрутам, распределяя тем самым нагрузку между сетевыми каналами передачи данных.

С другой стороны, серверные балансировщики нагрузки (Server Load Balancer, SLB) распределяют трафик между несколькими серверами, а не сетевыми каналами. Современные серверные балансировщики нагрузки выполняют различные функции: балансировка нагрузки и трафика, интеллектуальное переключение трафика по заданному алгоритму, проверка доступности серверов, приложений и контента для повышения надежности. Поскольку серверные балансировщики нагрузки находятся в сетевой топологии перед фермой серверов, они также защищают серверы от различных сетевых атак и повышают уровень безопасности.

В свою очередь, глобальные серверные балансировщики нагрузки (Global Server Load Balancer, GSLB) распределяют трафик между различными центрами обработки данных для уменьшения времени ответа и повышения надежности в случае полной недоступности центра обработки данных.

Балансировщики нагрузки имеют много названий: интеллектуальные коммутаторы со второго по седьмой уровень модели OSI (layer 2 — 7 switches), веб-коммутаторы (web switches), переключатели контента (content switches) и др. Независимо от названия все они выполняют, по существу, свою основную функцию — балансировку нагрузки и трафика.

Тема балансировки нагрузки и трафика актуальна не только для высоконагруженных сервисов, но и для всех проектов, которые стремятся обеспечить лучшее качество сервиса для своих клиентов за счет уменьшения количества отказов в обслуживании и сокращения задержек при передаче данных через глобальную сеть.

Например, поисковый сервис Google на момент написания работы по запросам „server load balancing” и „global server load balancing” выдавал 10 млн. и 3 млн. результатов соответственно, а крупные вендоры, такие как Amazon, Cisco, Citrix, Kemp, Radware и др. предлагают готовые программно-аппаратные решения как для серверной балансировки нагрузки, так и для глобальной серверной балансировки нагрузки. В зависимости от

заявленной производительности и пропускной способности стоимость таких решений зачастую измеряется десятками и даже сотнями тысяч долларов.

Не желая зависеть от одного вендора и по причине значительной стоимости, многие крупные компании разрабатывают собственные SLB и GSLB решения на базе открытого программного обеспечения.

Например, инженеры Google (Google Site Reliability Engineers) разработали для себя и сделали общедоступным SLB балансировщик Seesaw на базе открытого проекта Linux Virtual Server (LVS) (Sing J. 2016). Такие компании, как Alibaba, Facebook, Mail.ru и Yandex также используют программные SLB и GSLB решения.

На момент написания данной работы автору удалось найти всего один проект существующего открытого программного обеспечения для решения задачи глобальной серверной балансировки нагрузки посредством DNS.

Цель данной работы — разработка открытого ПО для глобальной серверной балансировки посредством системы доменных имен (Domain Name System based Global Server Load Balancing, DNS GSLB).

1. ТЕХНОЛОГИИ БАЛАНСИРОВКИ НАГРУЗКИ И ТРАФИКА

В первой части работы рассматриваются основные технологии балансировки нагрузки и трафика, используемые в современных компьютерных системах. Особое внимание уделяется рассмотрению понятий, плюсов и минусов технологий серверной балансировки нагрузки и глобальной серверной балансировки нагрузки.

В заключении проводится краткое сравнение рассмотренных технологий балансировки нагрузки и трафика.

1.1. Балансировка нагрузки и сетевая модель OSI

При рассмотрении и сравнении различных технологий балансировки нагрузки и трафика одним из важных критериев является уровень модели OSI (Open Systems Interconnection), на котором работают рассматриваемые технологии.

Сетевая модель OSI — это концептуальная модель, характеризующая и стандартизирующая функции и средства сетевого взаимодействия в вычислительной системе. Модель OSI определяет уровни взаимодействия систем с коммутацией пакетов, стандартные названия уровней и функции, которые должен выполнять каждый уровень. При этом модель OSI не содержит описания реализации конкретного набора протоколов. В модели OSI средства взаимодействия делятся на семь уровней: физический, канальный, сетевой, транспортный, сеансовый, представления и прикладной (Олифер В. Г., Олифер Н. А. 2010).

В таблице 1.1 показаны уровни сетевой модели OSI, их функции, примеры протоколов или физических сред передачи данных, которые соответствуют каждому уровню.

Таблица 1.1

Сетевая модель OSI

Уровень (Layer)	Функция	Примеры
7. Прикладной (Application)	Доступ к сетевым службам	HTTP, FTP, SMTP, SSH, Telnet
6. Представления (Presentation)	Представление и шифрование данных	MIME, SSL
5. Сеансовый (Session)	Управление сеансом связи	NetBIOS, SOCKS
4. Транспортный (Transport)	Передача данных с необходимой степенью надежности	TCP, UDP, SCTP
3. Сетевой (Network)	Образование единой транспортной системы	IPv4, IPv6, IPsec, IPIP, GRE
2. Канальный (Data Link)	Прозрачность соединения для сетевого уровня	ATM, ARP, LLC, MAC, Ethernet, Frame Relay
1. Физический (Physical)	Передача потоков битов по физическим каналам связи	витая пара, оптический кабель, радио сигнал

Технологии балансировки нагрузки и трафика активно используются практически на каждом уровне сетевой модели OSI. Для начала рассмотрим некоторые технологии балансировки нагрузки и трафика, используемые в современных компьютерных сетях, которые не требуют специализированных балансировщиков нагрузки.

1.2. Балансировка нагрузки на физическом уровне модели OSI

Физический уровень (physical layer) — нижний уровень модели OSI, который обеспечивает передачу потоков битов по физическим каналам связи, таким как коаксиальный кабель, витая пара, оптоволоконный кабель, радио сигнал и др.

Функции физического уровня реализуются на всех устройствах, подключенных к сети. Со стороны персонального компьютера функции физического уровня выполняются, например, сетевым или беспроводным адаптером (Олифер В. Г., Олифер Н. А. 2010).

Основным способом увеличения пропускной способности и надежности передачи данных на физическом уровне модели OSI является параллельное использование нескольких физических каналов связи или нескольких частотных диапазонов, то есть балансировка нагрузки между несколькими параллельными физическими каналами связи посредством их объединения в одну линию связи.

Примеры балансировки нагрузки и трафика на физическом уровне модели OSI включают в себя, например, линии электропередач (IEEE 1901), беспроводные сети (IEEE 802.11, IEEE 802.16), Ethernet сети (IEEE 802.3) и другие сетевые стандарты, которые

сочетают в себе несколько частотных диапазонов или несколько физических носителей сигнала в одной линии связи.

1.3. Балансировка нагрузки на канальном уровне модели OSI

Канальный уровень (data link layer) обеспечивает прозрачность соединения для следующего сетевого уровня. Для этого он предлагает ему определенные услуги:

- установление логического соединения между взаимодействующими узлами;
- согласование в рамках соединения скоростей передатчика и приемника информации;
- обеспечение надежной передачи, обнаружение и коррекция ошибок.

Важно отметить, что адреса, с которыми работает протокол канального уровня, используются для доставки кадров только в пределах одной сети, а для перемещения пакетов между сетями применяются уже адреса следующего, сетевого, уровня (Олифер В. Г., Олифер Н. А. 2010).

Например, в беспроводной сети Wi-Fi (IEEE 802.11) и сети Ethernet (IEEE 802.3) на канальном уровне каждый компьютер, а точнее каждый сетевой адаптер, имеет уникальный аппаратный адрес — MAC-адрес, который используется для выборочной, широковещательной и групповой рассылки в пределах локальной сети.

Балансировка нагрузки и трафика на канальном уровне модели OSI происходит, например, через агрегирование физических каналов между двумя коммуникационными устройствами в один логический канал или, как его часто называют, транк. Стандартный способ создания агрегированных каналов, описанный в спецификации IEEE 802.3ad, предполагает возможность создания логического порта путем объединения нескольких физических портов, принадлежащих разным устройствам, что обеспечивает отказоустойчивость и увеличение пропускной способности. Для автоматического обмена информацией о принадлежности какого-либо физического порта определенному логическому порту в спецификации предложен служебный протокол управления агрегированием линий связи (Link Control Aggregation Protocol, LCAP).

Агрегирование линий связи применяется как для связей между портами коммутаторов в локальной сети, так и для связей между коммутатором и сервером, как показано на рисунке 1.1.

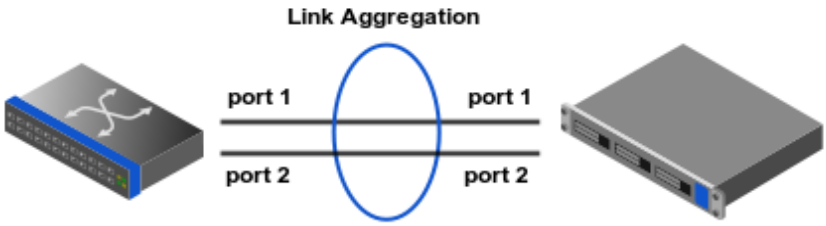

Рис. 1.1. Агрегирование каналов между коммутатором и сервером

Стандартным примером агрегирования каналов является транк между двумя коммуникационными устройствами. В англоязычных источниках этот тип агрегирования каналов имеет название Link Aggregation Group (LAG). Развитием технологии LAG является MC-LAG (Multi-Chassis Link Aggregation Group) — вид агрегации каналов, позволяющий использовать несколько коммуникационных устройств с каждой стороны транка с целью обеспечения избыточности в случае выхода из строя одного устройства. И хотя MC-LAG не является промышленным стандартом, его поддерживают все крупные производители сетевого оборудования в своих продуктах.

При использовании MC-LAG линии связи, физически подключенные в два или более различных коммутаторов, образуют единый агрегированный канал с третьим устройством, как показано на рисунке 1.2. В свою очередь, коммутаторы объединяются в один логический или виртуальный коммутатор посредством внутреннего протокола.

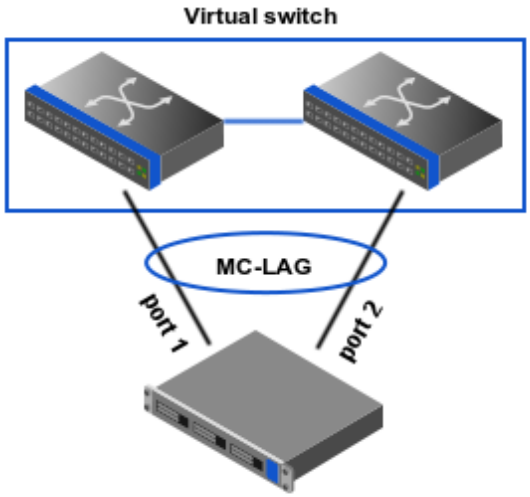

Рис. 1.2. Агрегирование каналов между двумя коммутаторами и сервером

Повышение производительности сети путем агрегирования нескольких линий связи в некоторых случаях является более эффективным, чем замена единственной линии связи на более скоростную.

Например, при использовании в сети коммутаторов и серверов с портами только Gigabit Ethernet повышение скорости до 10 Гбит/с потребует замены коммутаторов и сетевых адаптеров. В то же время при наличии свободных портов Gigabit Ethernet скорость передачи данных можно было бы повысить, например, до 8 Гбит/с, объединив в агрегированный канал восемь портов Gigabit Ethernet на тех участках сети, где это действительно нужно.

1.4. Балансировка нагрузки на сетевом уровне модели OSI

Сетевой уровень (network layer) служит для образования единой транспортной системы, объединяющей несколько сетей, называемой составной сетью, или интернетом.

Для того чтобы протоколы сетевого уровня могли доставлять пакеты любому узлу составной сети, эти узлы должны иметь адреса, уникальные в пределах данной составной сети. Такие адреса называются сетевыми или глобальными. Каждый узел составной сети, который намерен обмениваться данными с другими узлами составной сети, наряду с адресом, назначенным ему на канальном уровне, должен иметь сетевой адрес (Олифер В. Г., Олифер Н. А. 2010).

Например, популярный стек протоколов TCP/IP предполагает использование на сетевом уровне глобальных IPv4 (32 бита) или IPv6 (128 бит) адресов.

Важной задачей сетевого уровня является определение маршрута. Маршрут описывается последовательностью сетей (маршрутизаторов), через которые должен пройти пакет, чтобы попасть к адресату. Поиск и фиксацию маршрутов обеспечивают протоколы маршрутизации. Различают протоколы, выполняющие статическую и адаптивную (динамическую) маршрутизацию. При статической маршрутизации все записи в таблице имеют неизменяемый, статический статус, а записи о маршрутах составляются и вводятся в память каждого маршрутизатора вручную. При динамической маршрутизации все изменения конфигурации сети автоматически отражаются в таблицах маршрутизации благодаря протоколам маршрутизации.

Наиболее популярными открытыми протоколами динамической маршрутизации являются протокол маршрутной информации RIP (Routing Information Protocol), протокол

выбора кратчайшего пути OSPF (Open Shortest Path First) и пограничный шлюзовой протокол BGP (Border Gateway Protocol).

Как статическая маршрутизация, так и различные протоколы динамической маршрутизации позволяют одновременно использовать несколько маршрутов равной стоимости. А некоторые протоколы, например, разработанный компанией Cisco улучшенный протокол внутренней маршрутизации EIGRP (Enhanced Interior Gateway Routing Protocol) позволяет одновременно использовать и несколько маршрутов разной стоимости.

Для балансировки нагрузки и трафика на сетевом уровне модели OSI используется, например, алгоритм многолучевой маршрутизации равной стоимости (Equal-Cost Multi-Path, ECMP), описанный в RFC 2991, который предполагает одновременное использование нескольких альтернативных маршрутов через сеть, что дает ряд преимуществ, таких как отказоустойчивость и увеличение пропускной способности. Алгоритм ECMP позволяет балансировать нагрузку и трафик как между сетевыми маршрутизаторами, так и между маршрутизатором и серверами, как показано на рисунке 1.3.

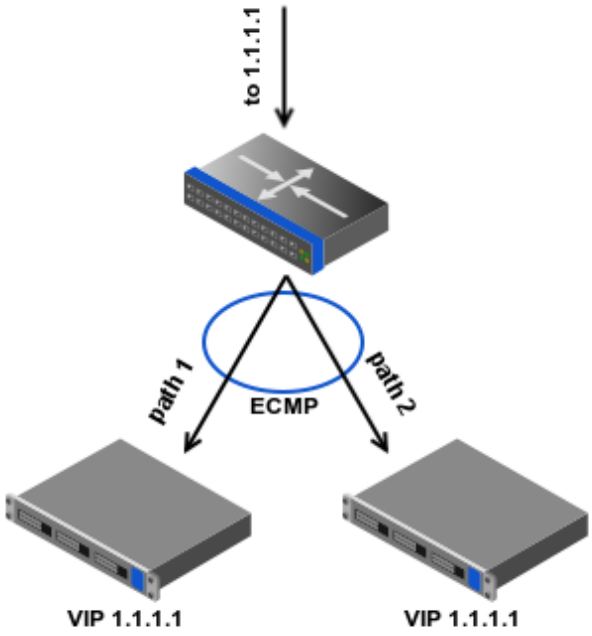

Рис 1.3. Многолучевая маршрутизация между маршрутизатором и серверами

1.5. Серверная балансировка нагрузки

Рассмотренные ранее технологии балансировки нагрузки и трафика могут быть реализованы с использованием стандартных сетевых коммутаторов и маршрутизаторов. Специализированные балансировщики нагрузки используют для принятия решения о балансировке трафика информацию, начиная с транспортного уровня модели OSI, поэтому разработчики и производители часто называют их коммутаторами 4 — 7 уровня (layer 4 — 7 switches). Поскольку балансировщики нагрузки, по сути, являются интеллектуальными сетевыми устройствами, они также реализуют функции нижних уровней модели OSI и могут называться коммутаторами 2 — 7 уровня.

Что же представляет из себя балансировщик нагрузки? С одной стороны, балансировщик нагрузки интегрируется с сетью, реализуя протоколы канального и сетевого уровня. С другой стороны, балансировщик нагрузки реализует многие протоколы более высоких уровней, так что он может взаимодействовать с серверами, используя высокоуровневый протокол. Технически балансировщик нагрузки является связующим звеном между сетью и фермой серверов, как показано на рисунке 1.4.

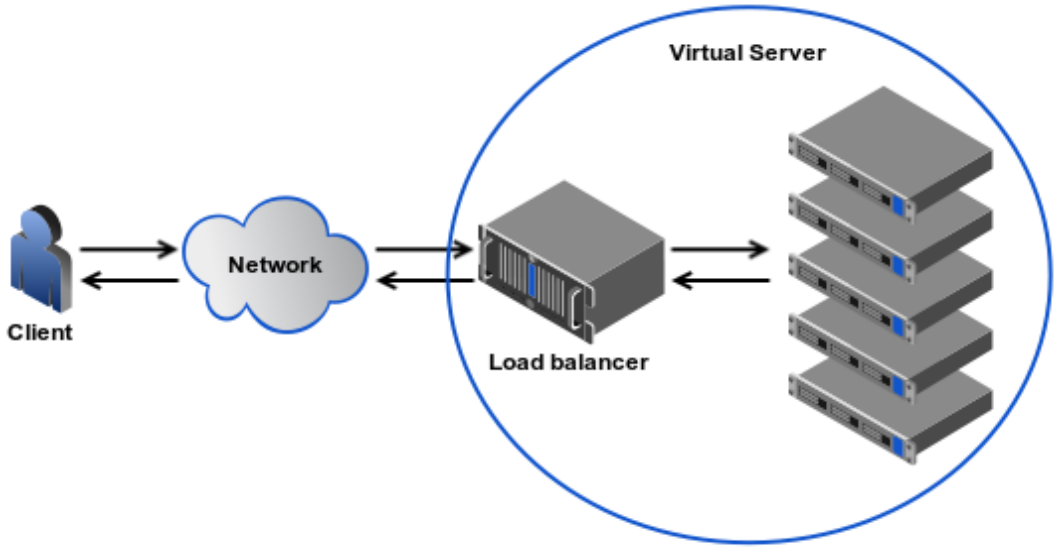

Рис 1.4. Серверная балансировка нагрузки

Серверная балансировка нагрузки — это распределение нагрузки и трафика между многими серверами, что позволяет масштабировать компьютерную систему за пределы возможностей одного сервера.

Для конечного пользователя балансировщик нагрузки и ферма реальных серверов полностью прозрачны, и пользователь взаимодействует с такой компьютерной системой, как с одним высокопроизводительным виртуальным сервером. Виртуальный сервер должен иметь IP-адрес для доступа к нему клиентов. Этот адрес называется виртуальным IP-адресом или VIP (Virtual IP). VIP-адрес настраивается на балансировщике нагрузки и представляет всю ферму серверов. В действительности один балансировщик нагрузки может иметь много VIP-адресов, то есть обслуживать много виртуальных серверов одновременно.

Использование серверной балансировки нагрузки имеет ряд преимуществ:

- увеличение суммарной производительности компьютерной системы за счет распределения запросов клиентов между многими серверами;
- улучшение доступности сервиса через непрерывное отслеживание состояния серверов и автоматическое исключение недоступных серверов из балансировки;
- улучшение качества обслуживания клиентов как следствие увеличения производительности и улучшения доступности;
- упрощение обслуживания компьютерной системы благодаря возможности оперативного выключения серверов из балансировки для проведения работ;
- повышение безопасности за счет сокращения количества доступных из глобальной сети серверов и, соответственно, уменьшения поверхности атаки.

Рассмотрим более детально поток трафика между клиентом и сервером на примере виртуального сервера транспортного уровня, обслуживающего TCP соединения.

При установке TCP соединения клиент посылает на VIP-адрес балансировщика нагрузки TCP SYN пакет, содержащий следующую информацию:

- IP-адрес отправителя;
- TCP порт отправителя;
- IP-адрес получателя;
- TCP порт получателя.

На основании этой информации балансировщик нагрузки определяет, какому виртуальному серверу предназначается полученный пакет, используя алгоритм балансировки нагрузки, выбирает из соответствующей фермы серверов один реальный сервер для обработки запроса и доставляет TCP пакет до этого реального сервера.

Доставить TCP пакет до реального сервера нужно таким образом, чтобы:

- во-первых, сервер на него ответил;

- во-вторых, этот ответ был доставлен обратно клиенту;
- в-третьих, для клиента это выглядело как одно TCP соединение.

Рассмотрим некоторые технологии доставки пакетов, используемые при серверной балансировке нагрузки.

1.5.1. Трансляция сетевых адресов

Трансляция сетевых адресов (Network Address Translation, NAT) — это широко применяемая сегодня технология, предполагающая продвижение пакета в глобальной сети на основании адресов, отличающихся от тех, которые используются для маршрутизации пакета в локальной сети (Олифер В. Г., Олифер Н. А. 2010).

При использовании технологии трансляции сетевых адресов балансировщик нагрузки получает пакет от клиента, изменяет в заголовке пакета IP-адрес отправителя на свой IP-адрес и пересылает пакет на реальный сервер посредством маршрутизации. Когда реальный сервер получает такой пакет, для него это выглядит так, как будто клиентом является балансировщик нагрузки. Реальный сервер отвечает балансировщику нагрузки, который, в свою очередь, обратно изменяет в заголовке пакета свой IP-адрес на IP-адрес клиента и отвечает клиенту посредством маршрутизации.

На рисунке 1.5 показана пошаговая схема доставки пакетов при использовании технологии трансляции сетевых адресов, а в таблице 1.2 показан результат изменения заголовков пакетов на каждом шаге.

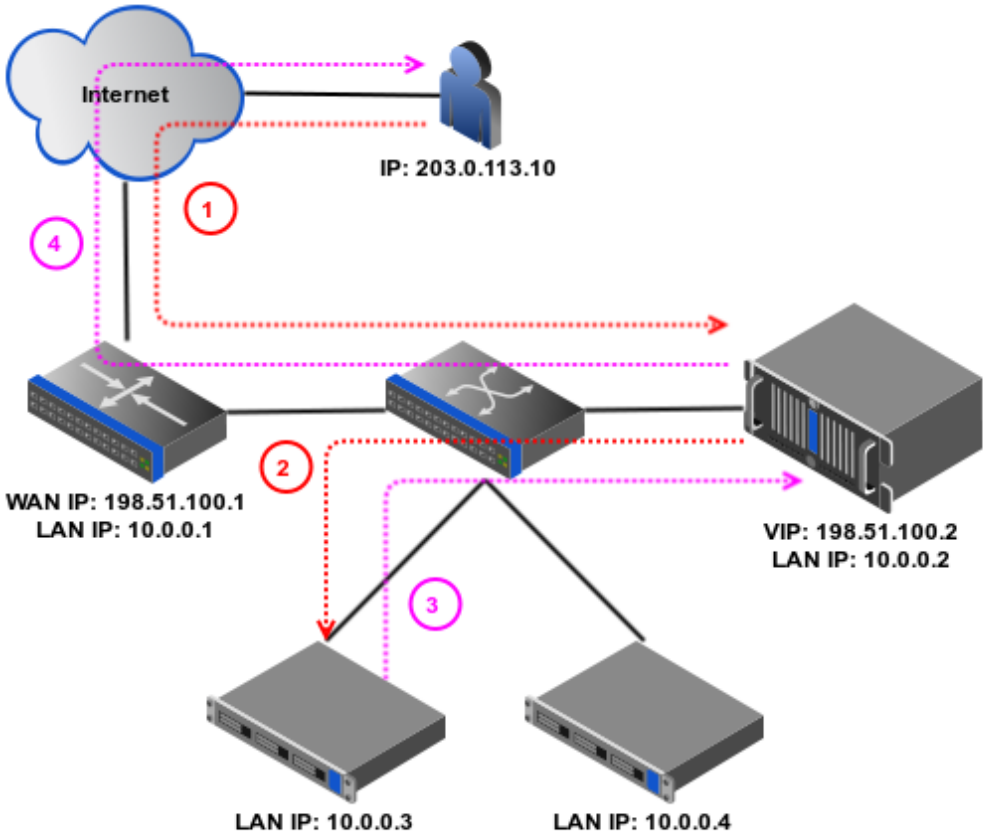

Рис. 1.5. Схема доставки пакетов при трансляции сетевых адресов

Таблица 1.2

Изменение заголовков пакетов при трансляции сетевых адресов

Шаг	IP-адрес отправителя	IP-адрес получателя
1	203.0.113.10	198.51.100.2
2	10.0.0.2	10.0.0.3
3	10.0.0.3	10.0.0.2
4	198.51.100.2	203.0.113.10

Преимуществом технологии трансляции сетевых адресов является то, что она позволяет установить балансировщик нагрузки в любом месте сети без каких-либо ограничений по сетевой топологии. Единственным условием является наличие связи на сетевом уровне между балансировщиком нагрузки и реальным сервером. Еще одним преимуществом является то, что не требуется изменять сетевую конфигурацию реальных серверов.

Недостатком технологии трансляции сетевых адресов является то, что реальные серверы не видят IP-адрес исходного клиента, потому что балансировщик нагрузки изменяет его на свой IP-адрес.

1.5.2. Прямая маршрутизация

Технология прямой маршрутизации (Direct Routing, DR), в отличие от трансляции сетевых адресов, оперирует не IP-адресами, а адресами канального уровня — MAC-адресами.

При использовании технологии прямой маршрутизации балансировщик нагрузки получает пакет от клиента, изменяет в заголовке пакета MAC-адрес получателя на MAC-адрес реального сервера и пересылает пакет на реальный сервер посредством коммутации. Когда реальный сервер получает такой пакет, для него это выглядит так, как будто клиент переслал ему пакет напрямую, так как адреса сетевого уровня не изменяются. Для того чтобы реальный сервер ответил на такой пакет, у него на петлевом сетевом интерфейсе (loopback interface) должен быть назначен VIP-адрес как на балансировщике нагрузки. В результате реальный сервер отвечает напрямую клиенту, минуя балансировщик нагрузки посредством маршрутизации. Такой асимметричный маршрут трафика называется прямым возвратом сервера (Direct Server Return, DSR).

На рисунке 1.6 показана пошаговая схема доставки пакетов при использовании технологии прямой маршрутизации, а в таблице 1.3 показан результат изменения заголовков пакетов на каждом шаге.

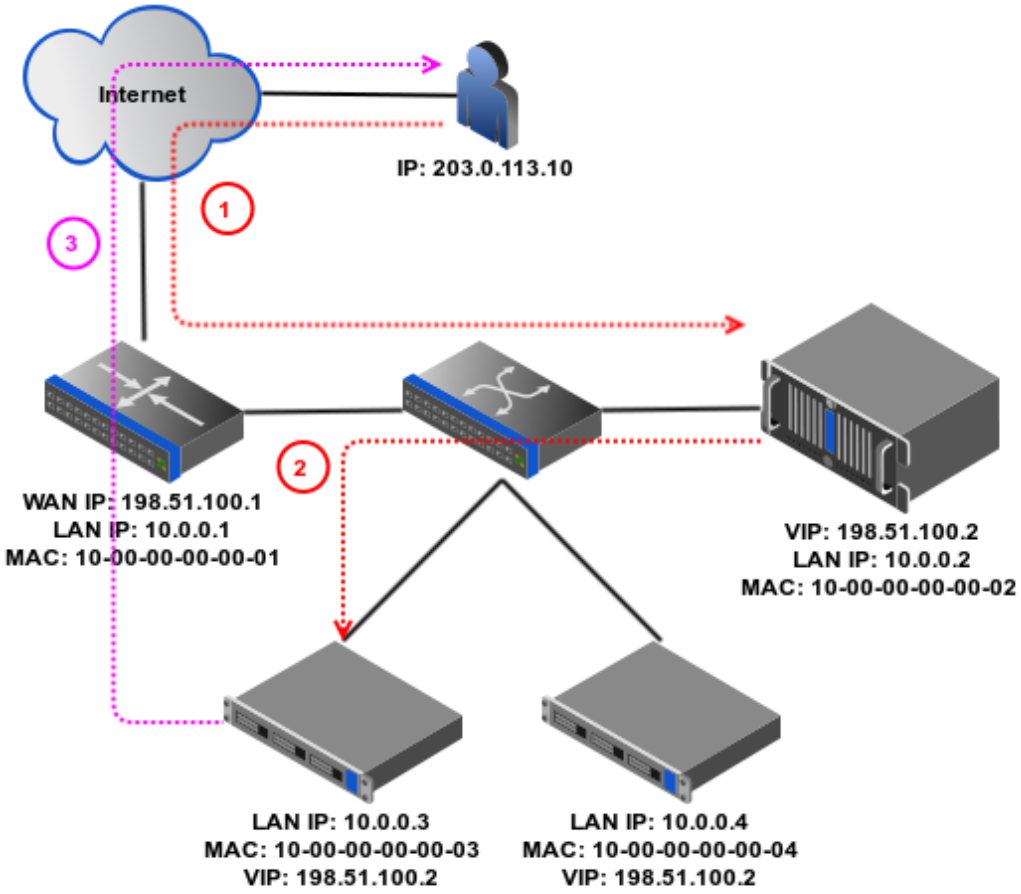

Рис. 1.6. Схема доставки пакетов при прямой маршрутизации

Таблица 1.3

Изменение заголовков пакетов при прямой маршрутизации

Шаг	IP-адрес отправителя	IP-адрес получателя	MAC-адрес отправителя	MAC-адрес получателя
1	203.0.113.10	198.51.100.2	10-00-00-00-00-01	10-00-00-00-00-02
2	203.0.113.10	198.51.100.2	10-00-00-00-00-02	10-00-00-00-00-03
3	198.51.100.2	203.0.113.10	10-00-00-00-00-03	10-00-00-00-00-01

Преимуществом технологии прямой маршрутизации является то, что она позволяет получить более высокую производительность, если балансировщик нагрузки является узким местом, потому что в этом случае он должен обработать только входящий трафик. Прямая маршрутизация лучше всего подходит для таких приложений, как веб-сайт, потоковое видео, скачивание файлов и других, где количество и размер пакетов ответа значительно больше количества и размера пакетов запроса.

Недостатком технологии прямой маршрутизации является то, что балансировщик нагрузки и реальные серверы должны находиться в одном широковещательном домене,

так как пакет пересылается на реальный сервер посредством коммутации — это существенно ограничивает сетевую топологию. Также необходимо изменить сетевую конфигурацию всех реальных серверов и назначить VIP-адрес на петлевом сетевом интерфейсе, что не всегда возможно.

1.5.3. Туннелирование

Туннелирование (tunneling) — это общее название технологий инкапсуляции пакетов некоторого протокола двух объединяемых сетей или узлов в пакеты протокола транзитной сети на ее границе и передача пакетов через транзитную сеть (Олифер В. Г., Олифер Н. А. 2010). Туннелирование широко применяется, например, для построения различных виртуальных частных сетей (VPN).

При использовании технологии туннелирования балансировщик нагрузки получает пакет от клиента, добавляет к нему дополнительный IP-заголовок (инкапсулирует пакет) и пересылает полученный IPIP пакет на реальный сервер посредством маршрутизации. Когда реальный сервер получает такой пакет, он отбрасывает дополнительный IP-заголовок (декапсулирует пакет), получая тем самым оригинальный пакет, и для него это выглядит так, как будто клиент переслал ему пакет напрямую. Для того чтобы реальный сервер смог декапсулировать и ответить на такой пакет, у него на туннельном сетевом интерфейсе (tunnel interface) должен быть назначен VIP-адрес как на балансировщике нагрузки. В результате реальный сервер отвечает напрямую клиенту посредством маршрутизации. Как и в случае прямой маршрутизации, происходит прямой возврат сервера (DSR).

На рисунке 1.7 показана пошаговая схема доставки пакетов при использовании технологии туннелирования, а в таблице 1.4 показан результат изменения заголовков пакетов на каждом шаге.

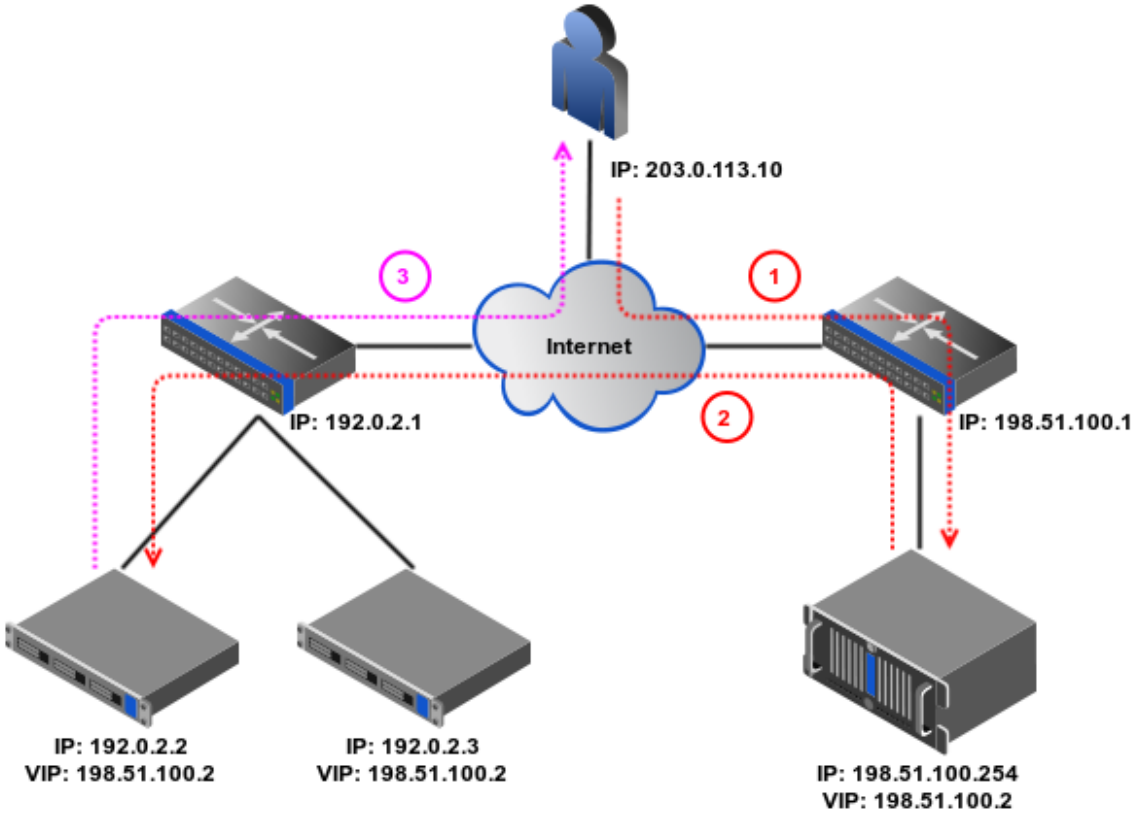

Рис. 1.7. Схема доставки пакетов при туннелировании

Таблица 1.4

Изменение заголовков пакетов при туннелировании

Шаг	IP-адрес отправителя	IP-адрес получателя	IP-адрес отправителя	IP-адрес отправителя
1			203.0.113.10	198.51.100.2
2	198.51.100.254	192.0.2.2	203.0.113.10	198.51.100.2
3			198.51.100.2	203.0.113.10

Преимуществом технологии туннелирования является то, что она, как и прямая маршрутизация, позволяет получить более высокую производительность, потому что балансировщик нагрузки должен обработать только входящий трафик. Еще одним преимуществом является то, что, как и в случае использования трансляции сетевых адресов, нет ограничений по сетевой топологии.

Недостатком технологии туннелирования является то, что инкапсуляция добавляет накладные расходы в виде 20 байт (размер IP-заголовка) к каждому пакету от балансировщика нагрузки к реальному серверу. Также необходимо изменить сетевую конфигурацию всех реальных серверов и назначить VIP-адрес на туннельном сетевом интерфейсе, что не всегда возможно.

1.6. Глобальная серверная балансировка нагрузки

При помощи технологии серверной балансировки нагрузки можно распределить нагрузку и трафик между фермой серверов, но нельзя защититься от полной недоступности центра обработки данных, где эта ферма расположена, или повлиять на скорость ответа сервера, когда клиент находится на значительном удалении от центра обработки данных.

Использование глобальной серверной балансировки нагрузки (GSLB) позволяет не только распределить нагрузку и трафик между несколькими центрами обработки данных и автоматически исключить центр обработки данных из балансировки в случае его недоступности, но и направить клиента на виртуальный сервер в ближайшем к нему регионе. Таким образом, основные предпосылки к использованию GSLB — это высокая доступность и быстрое время отклика.

GSLB может быть реализована различными способами. Рассмотрим основные технологии, использующиеся в глобальной серверной балансировке нагрузки.

1.6.1. GSLB посредством HTTP перенаправления

Одной из самых простых технологий реализации глобальной серверной балансировки нагрузки, которая может быть использована без существенных изменений в существующей сетевой архитектуре или конфигурации DNS, является GSLB посредством HTTP перенаправления. Эта технология работает на прикладном уровне модели OSI.

Протокол HTTP, описанный в RFC 2616, включает вариант ответа веб-сервера, который содержит HTTP коды 3xx перенаправления и новый URL. В случае получения такого кода, веб-браузер должен автоматически перейти на новый URL.

На рисунке 1.8 показана схема работы GSLB посредством HTTP перенаправления. Например, когда клиент из Латвии запрашивает сайт example.com, обслуживающий его HTTP веб-сервер перенаправляет клиента на ближайший региональный сайт example.lv. Наконец, клиент делает запрос на виртуальный сервер в Риге и получает в ответ веб-страницу. HTTP веб-сервер может принимать решение о перенаправлении, например, на основе сравнения IP-адреса клиента с GeoIP базой данных или адаптивным списком IP-сетей.

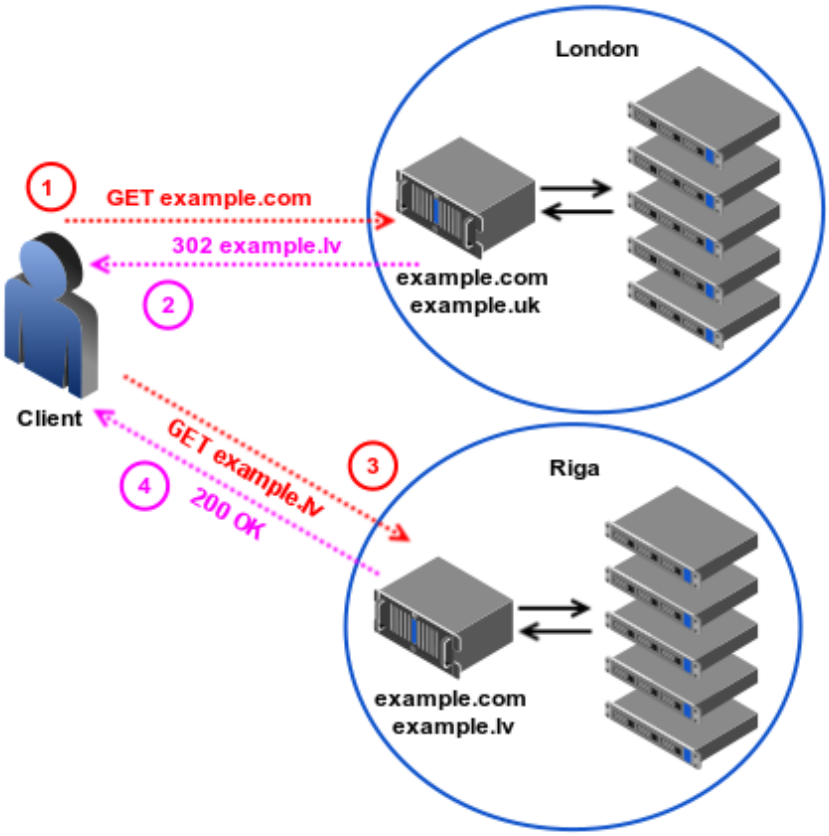

Рис. 1.8. Схема работы GSLB посредством HTTP перенаправления

Преимуществом GSLB посредством HTTP перенаправления является снижение времени ответа сервера, так как клиент перенаправляется на ближайший региональный сайт.

Недостатком данного решения является поддержка только HTTP протокола и отсутствие резервирования в случае недоступности центра обработки данных, поэтому HTTP перенаправление применяют совместно с другими технологиями GSLB.

Несмотря на ограничения, HTTP перенаправление широко применяется многими высоконагруженными и географически распределенными веб-сайтами. Например, при запросе главной страницы поискового сервиса Google https://www.google.com/ происходит автоматическое HTTP перенаправление на ближайший региональный сайт.

1.6.2. GSLB посредством DNS

Наиболее популярная технология GSLB — это глобальная серверная балансировка посредством системы доменных имен (DNS GSLB). Связано это, в первую очередь, с широкой распространенностью самой системы доменных имен, которая повсеместно используется, например, для преобразования доменных имен в IP-адреса и обратно. Описанный в RFC 1035 DNS протокол прикладного уровня является одним из старейших протоколов интернета. Для обеспечения сервиса DNS GSLB сервер должен быть либо авторитетным DNS сервером, либо являться DNS Proxy перед авторитетным сервером.

Стоит отметить, что стандартный DNS также может использоваться для балансировки нагрузки между несколькими серверами. Для этого достаточно назначить несколько IP-адресов для доменного имени, и DNS будет выдавать их по алгоритму кругового обслуживания (round robin). Проблема заключается в том, что DNS не знает, какие из этих IP-адресов на самом деле работают. Виртуальный сервер может быть полностью недоступен, но DNS будет продолжать выдавать его IP-адрес, что может вызвать различные проблемы на стороне клиента. К тому же стандартный DNS не решает задачу перенаправления клиента на ближайший региональный сайт.

На рисунке 1.9 показана схема работы GSLB посредством DNS. Например, когда клиент из Латвии запрашивает IP-адрес сайта example.com у авторитетного DNS GSLB сервера, тот возвращает IP-адрес ближайшего регионального сайта example.com. Наконец, клиент делает запрос на виртуальный сервер в Риге и получает в ответ веб-страницу. DNS GSLB сервер может принимать решение о перенаправлении, например, на основе сравнения IP-адреса клиента с GeoIP базой данных или адаптивным списком IP-сетей. В случае недоступности сайта в Риге, DNS GSLB сервер начнет выдавать по запросу клиента следующий доступный сайт согласно своей политике балансировки.

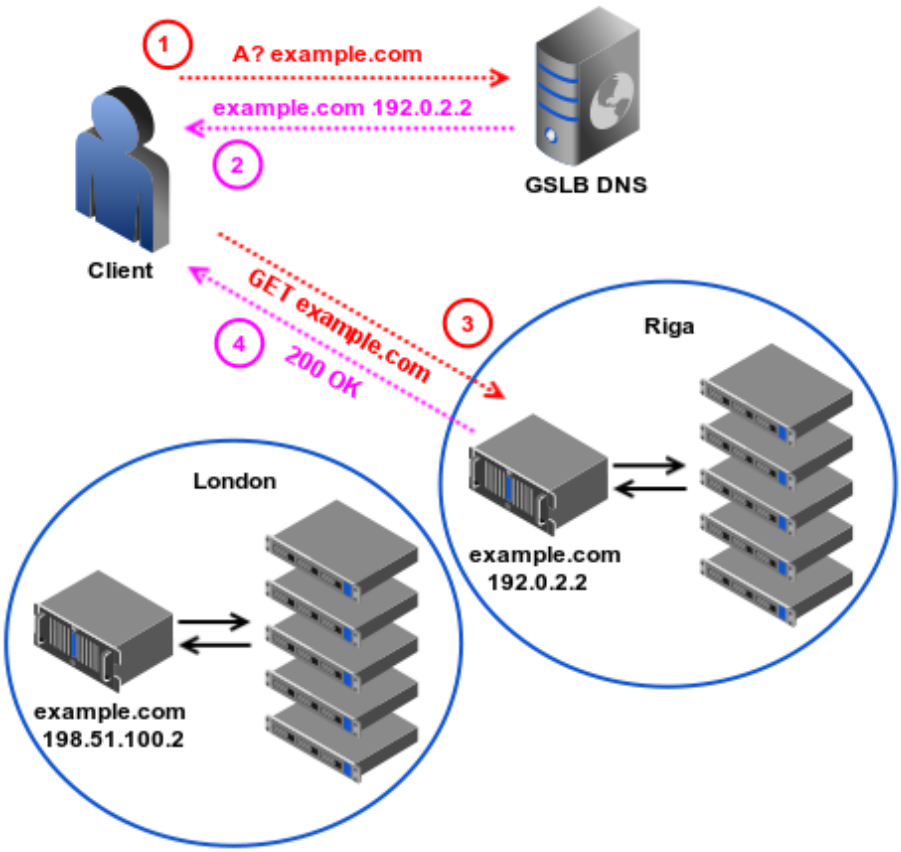

Рис. 1.9. Схема работы GSLB посредством DNS

Основными преимуществами технологии DNS GSLB являются простота реализации и универсальность, так как она может использоваться совместно с любыми протоколами, которые позволяют коммуникацию посредством доменных имен, а это поистине огромный перечень протоколов.

К недостаткам DNS GSLB можно отнести DNS кеширование (TTL) на стороне локальных DNS серверов и на стороне клиента, из-за чего не происходит моментального переключения на следующий IP-адрес в случае сбоя. Еще одним недостатком является то, что клиент не обращается напрямую к авторитетному DNS GSLB серверу, и все запросы происходят через локальный кеширующий DNS сервер клиента. В результате, если клиент использует DNS сервер, находящийся в другом регионе, он может быть неверно направлен не на ближайший сервер, а в другой регион.

Несмотря на все ограничения, DNS GSLB предоставляет широкий спектр преимуществ и улучшений по сравнению со стандартным DNS.

1.6.3. GSLB посредством протокола маршрутизации

Не менее популярна технология GSLB посредством протокола маршрутизации или IP Anycast. В этой технологии сетевого уровня используется алгоритм поиска лучшего маршрута, выбранного посредством протокола маршрутизации. В глобальной сети для этой цели используется, как правило, BGP протокол.

На рисунке 1.10 показана схема работы GSLB посредством BGP. Например, два виртуальных сервера в Риге и в Лондоне непрерывно анонсируют по протоколу BGP маршрут до одной и той же IP-сети. Клиент обращается на IP-адрес из этой сети и, согласно выбранному оптимальному маршруту, попадает на ближайший региональный виртуальный сервер в Риге и получает ответ. В случае если один из центров обработки данных становится недоступен, соседний с ним маршрутизатор перестает получать BGP анонс, маршрут устаревает и автоматически исключается из маршрутизации, а клиент начинает обращаться на следующий ближайший виртуальный сервер.

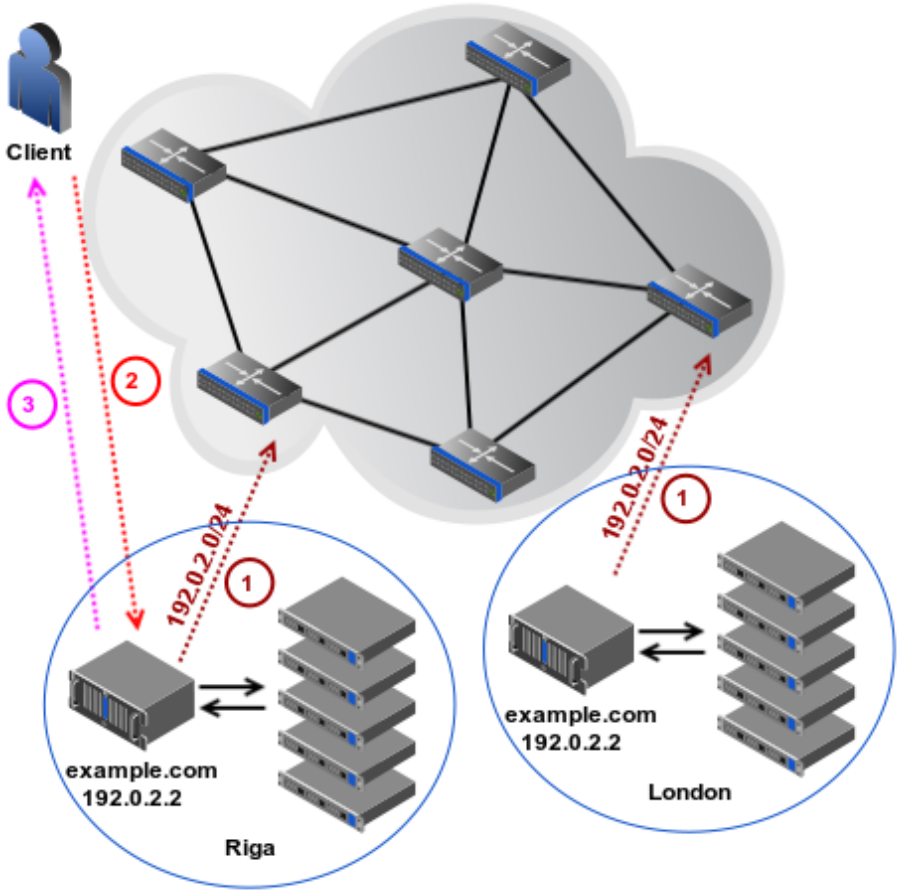

Рис. 1.10. Схема работы GSLB посредством BGP

Главной проблемой технологии GSLB посредством протокола маршрутизации является разрыв или таймаут TCP соединения между клиентом и сервером при каждом переключении маршрута на альтернативный путь. По этой причине эта технология больше подходит для UDP протокола, который функционирует без установления соединения на транспортном уровне.

Пожалуй, лучшим примером повсеместного использования этой технологии являются публичные корневые DNS серверы и DNS серверы таких компаний-гигантов, как Amazon, Google, Yandex и других, которые установлены по всему миру и доступны по общим IP-адресам из разных регионов планеты.

1.7. Сравнение открытых технологий балансировки нагрузки и трафика

В качестве заключения к первой части работы проведем краткое сравнение рассмотренных в ней технологий балансировки нагрузки и трафика. Плюсы и минусы каждой из технологий приведены в соответствующих разделах. Результат сравнения показан в таблице 1.5.

Таблица 1.5

Сравнение открытых технологий балансировки нагрузки и трафика

Технология	Уровень OSI	Область применения	Масштабируемость	Примеры открытого ПО
LAG	2	LAN	низкая	Linux Kernel, Open vSwitch
ECMP	3	LAN / WAN	низкая	Linux Kernel, Bird, Quagga
SLB NAT	4	LAN / WAN	средняя	Keepalived, LVS, HAProxy, nginx
SLB DR	4	LAN / WAN	высокая	Keepalived, LVS
SLB IPIP	4	WAN	высокая	Keepalived, LVS
GSLB HTTP	7	LAN / WAN	высокая	Apache httpd, nginx
GSLB DNS	7	LAN / WAN	очень высокая	polaris-gslb
GSLB BGP	7	WAN	очень высокая	Linux Kernel, Bird, Quagga

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ГЛОБАЛЬНОЙ СЕРВЕРНОЙ БАЛАНСИРОВКИ НАГРУЗКИ

В этом разделе последовательно рассматриваются этапы процесса разработки приложения для глобальной серверной балансировки нагрузки посредством системы доменных имен (Domain Name System based Global Server Load Balancing, DNS GSLB).

В процессе разработки формируются и анализируются требования к продукту, разрабатывается интерфейс управления, происходит проектирование приложения, его реализация и тестирование.

На завершающей стадии проекта формируется готовый к установке на сервер пакет, содержащий приложение, конфигурационные файлы, необходимые модули, документацию по настройке приложения на сервере и сценарии автоматической конфигурации.

В заключении проводится оценка полученного программного продукта.

2.1. Формирование требований

На этом этапе собираются и формируются требования, определяющие характеристики и функции будущего программного обеспечения. Эти требования определяют полное задание на разработку (Орлов С. А. 2016).

2.1.1. Функциональные требования

1. Приложение должно отвечать клиенту на DNS запросы по соответствующему протоколу.
2. Приложение должно проверять доступность выдаваемых DNS сервером записей конечных серверов.
3. Приложение должно отвечать клиенту на HTTP запросы по соответствующему протоколу.
4. Приложение должно предоставлять администратору возможность управления конфигурацией.
5. Приложение должно предоставлять операционной системе возможность запускать приложение, как системный сервис.

2.1.2. Нефункциональные требования

1. Приложение должно функционировать в операционной системе Linux x86-64.
2. Приложение должно поддерживать многопоточность или многозадачность для одновременной работы со многими клиентами.
3. Приложение должно обеспечивать защиту от несанкционированного доступа к конфигурации.
4. Приложение должно поддерживать репликацию данных в кластере из нескольких серверов.

2.1.3. Системные требования сервера

Программа должна работать на выделенном сервере или в виртуальной машине со следующей конфигурацией:

1. Минимум 1 процессор Intel x86-64 совместимый.
2. Минимум 1 GB оперативной памяти.
3. Минимум 4 GB жесткий диск.
4. Сетевая карта.
5. Операционная система Linux x86-64.

2.1.4. Системные требования клиента

Веб-интерфейс управления должен работать со следующими клиентами:

1. Веб-браузер Chrome 30, Firefox 27 или IE 11 с поддержкой JavaScript.

2.2. Анализ и разработка требований

Моделирование системы начинается с построения диаграммы Use Case. Этот тип диаграммы представляется актерами, элементами Use Case и отношениями между ними (Орлов С. А. 2016).

Первый шаг построения диаграммы состоит в определении актеров, фиксирующих роли внешних объектов, взаимодействующих с системой. В рассматриваемой предметной области будут фигурировать пять актеров — DNS Client (DNS-клиент), HTTP Client

(HTTP-клиент), Administrator (администратор системы), Monitor (мониторинг) и OS (операционная система). Каждый актер выполняет соответствующий элемент Use Case.

Начальная диаграмма Use Case изображена на рисунке 2.1.

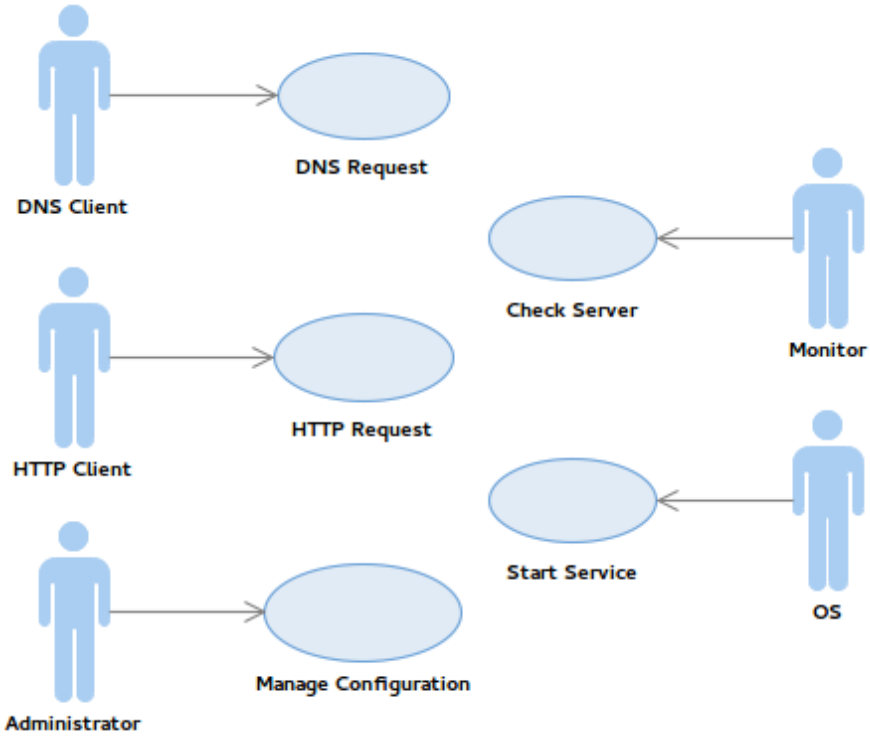

Рис. 2.1. Начальная диаграмма Use Case

В данной работе рассматривается одна программа, в которой элементы Use Case не являются независимыми, а расширяют или включают возможности друг друга. Необходимо дополнить начальную диаграмму Use Case, чтобы в ней были учтены зависимости между элементами Use Case.

Выполнение программы начинается с запуска ее операционной системой. Таким образом, базовым элементом Use Case является Start Service. На рисунке 2.2 показана расширенная диаграмма Use Case, в которой учтены связи между элементами Use Case. Базовый элемент Use Case показан полужирной линией.

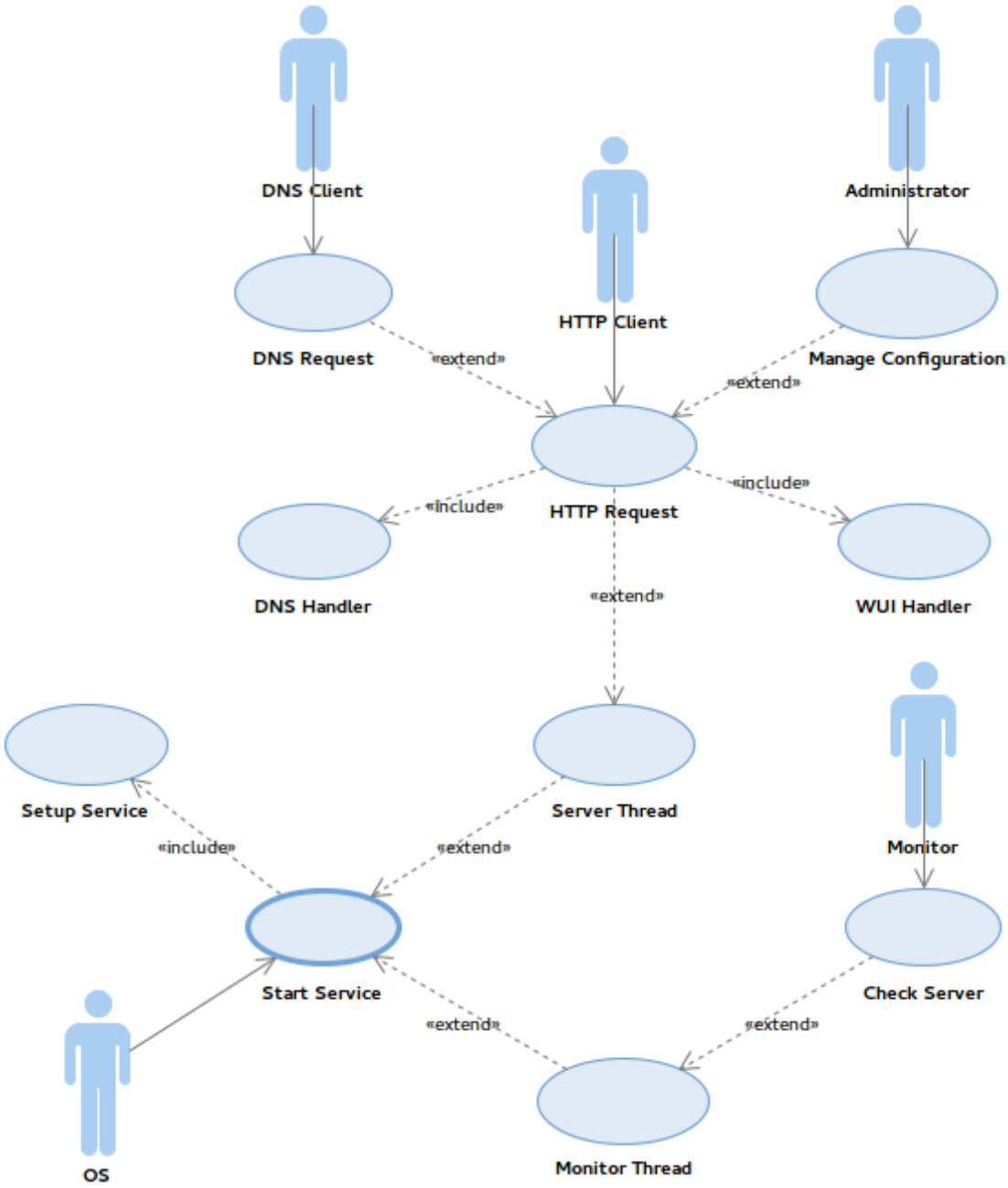

Рис. 2.2. Расширенная диаграмма Use Case

На следующем шаге разработки требований необходимо проанализировать диаграмму Use Case и построить диаграммы последовательности для всех ее элементов. На рисунках 2.3 — 2.12 показаны полученные диаграммы последовательности для каждого элемента Use Case.

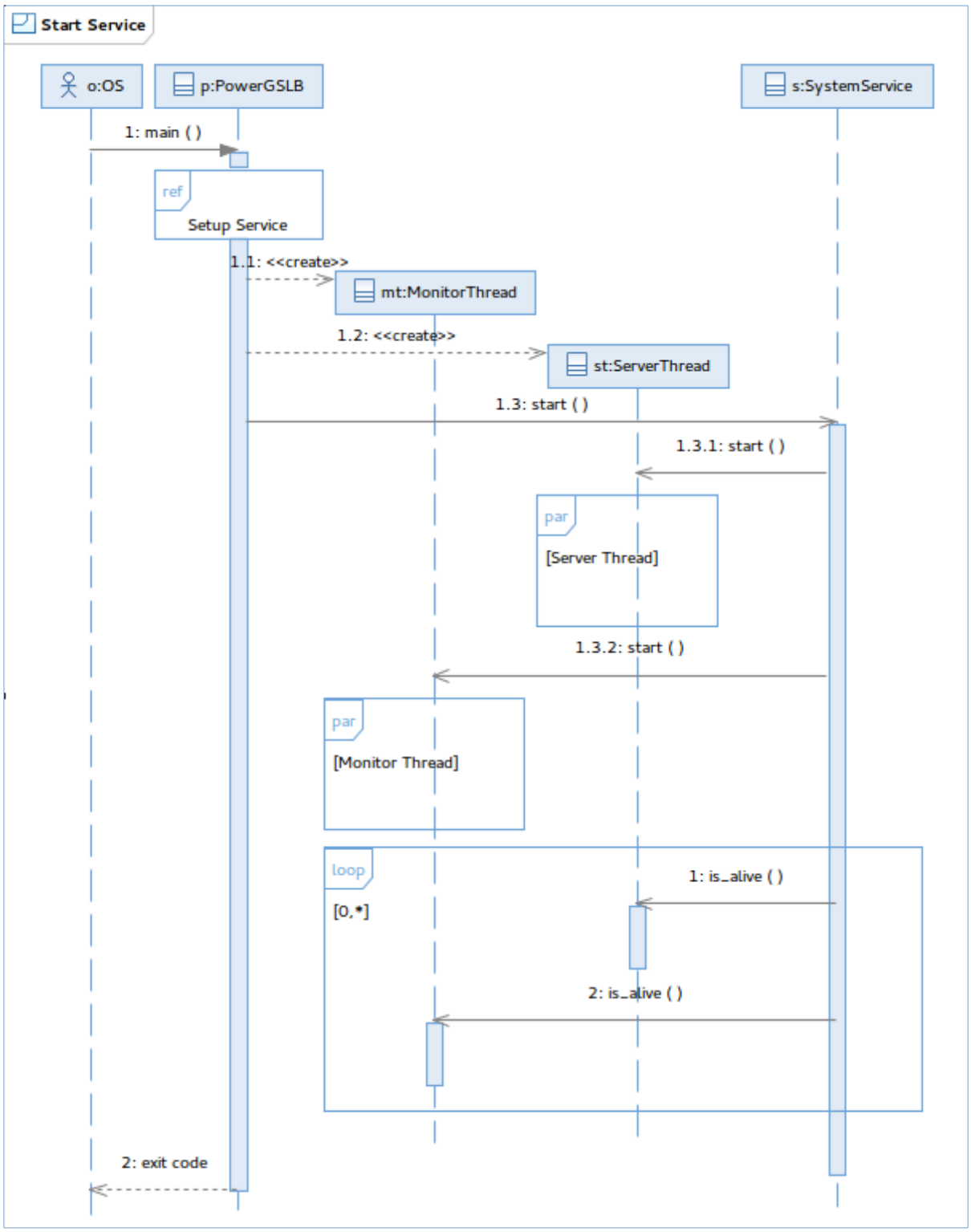

Рис. 2.3. Диаграмма последовательности для базового элемента Use Case Start Service

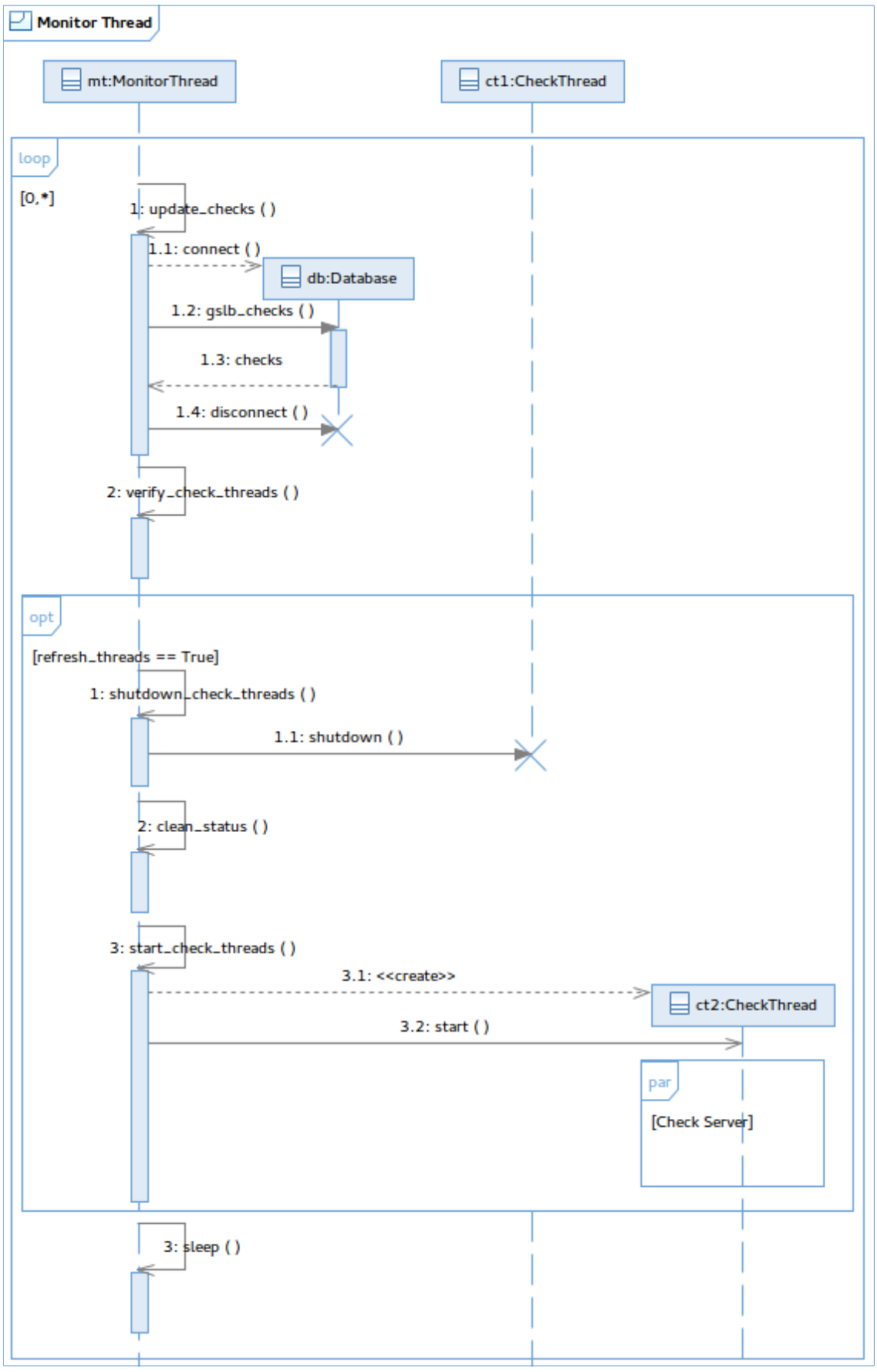

Рис. 2.4. Диаграмма последовательности для элемента расширения Monitor Thread

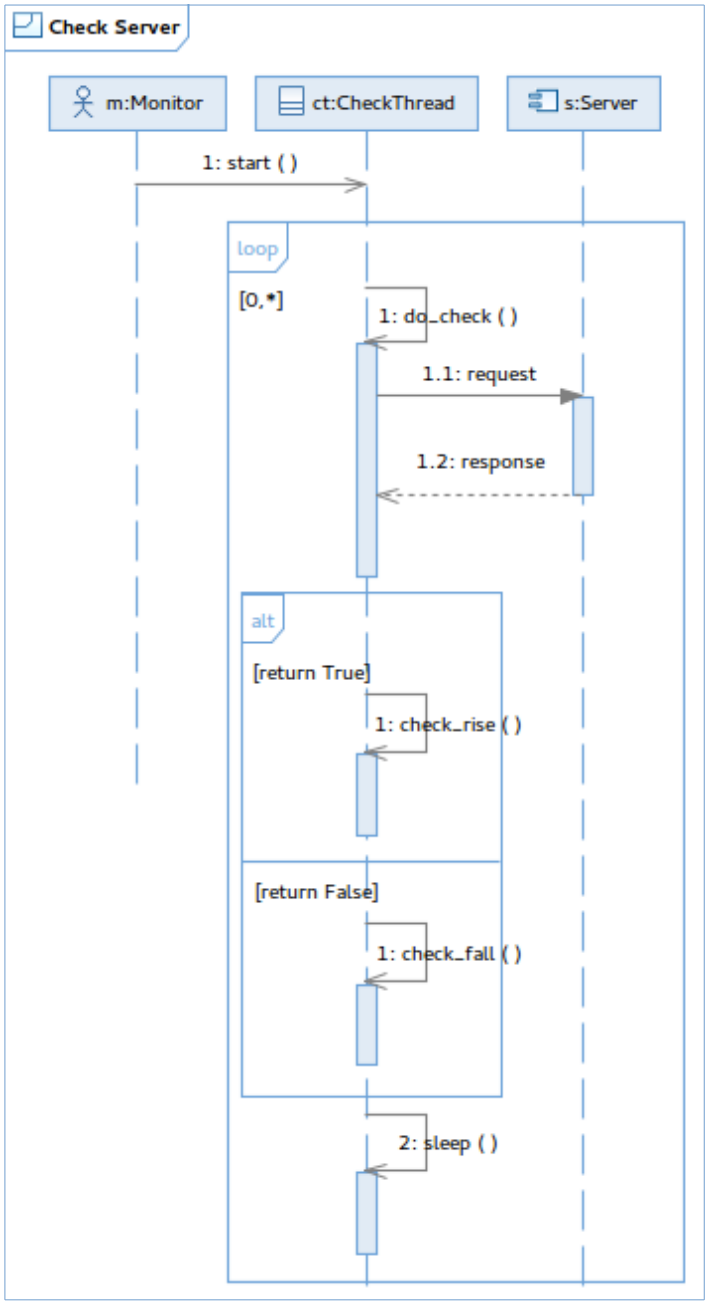

Рис. 2.5. Диаграмма последовательности для элемента расширения Check Server

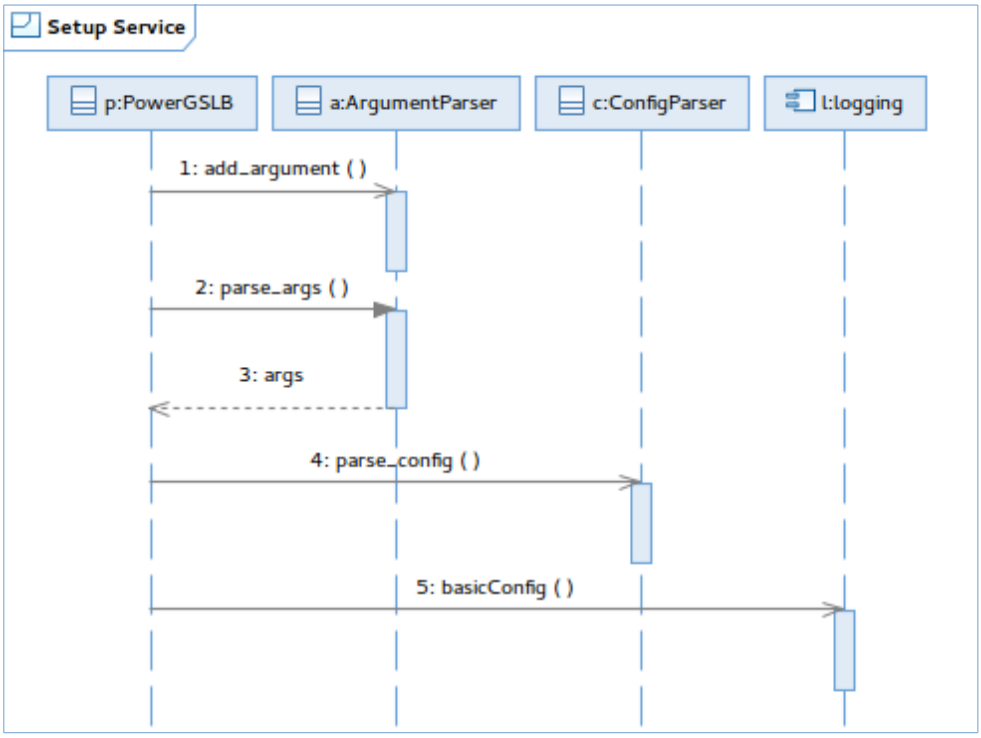

Рис. 2.6. Диаграмма последовательности для включаемого элемента Setup Service

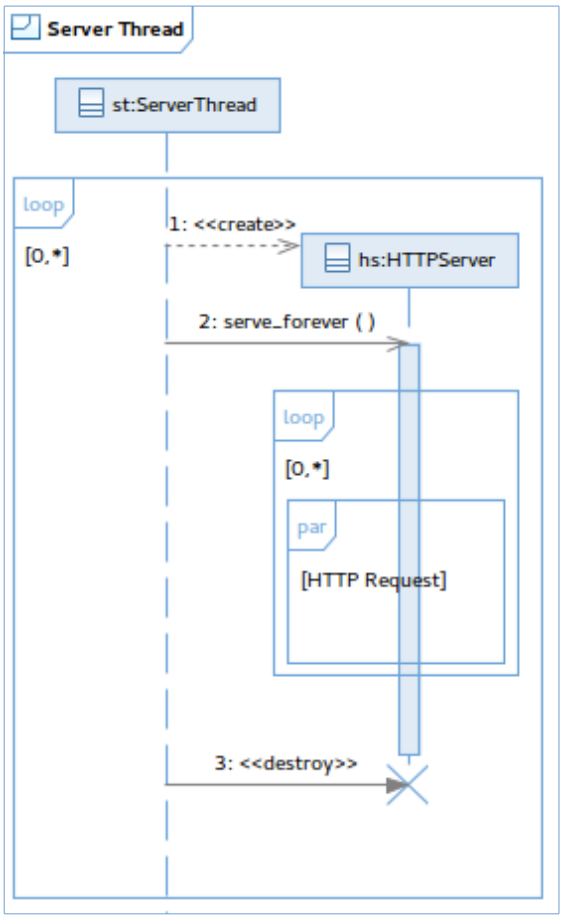

Рис. 2.7. Диаграмма последовательности для элемента расширения Server Thread

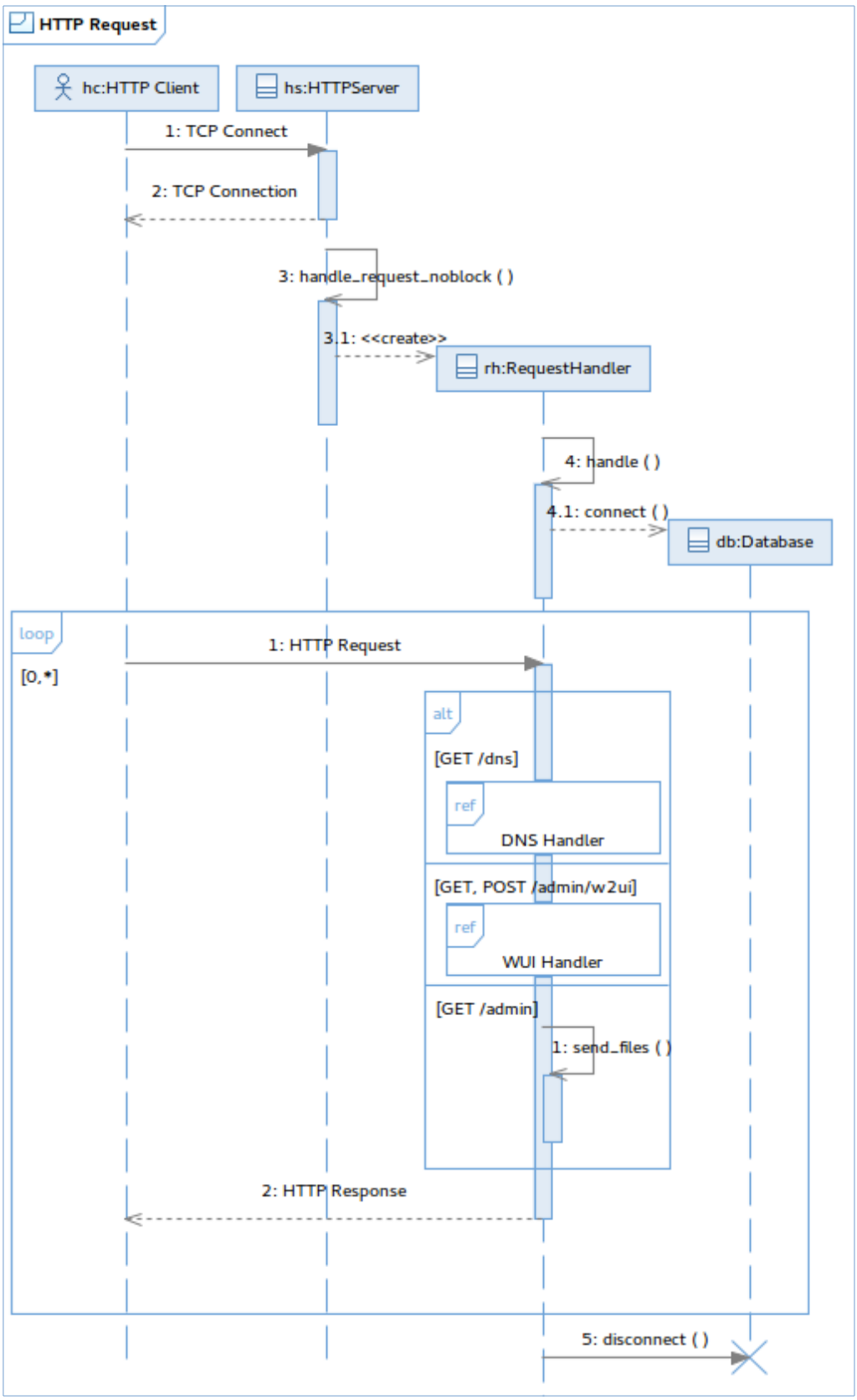

Рис. 2.8. Диаграмма последовательности для элемента расширения HTTP Request

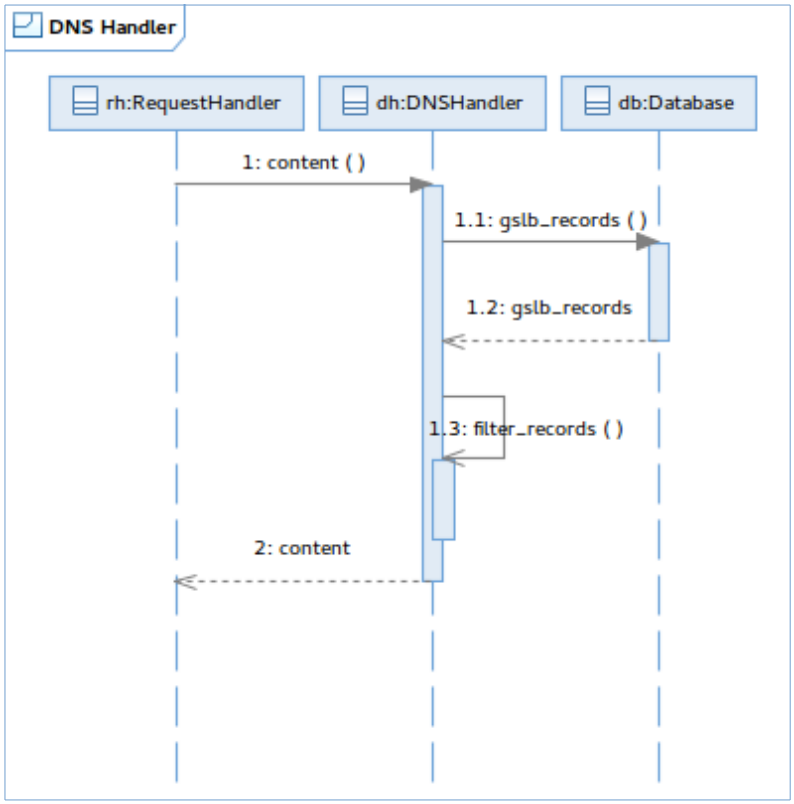

Рис. 2.9. Диаграмма последовательности для включаемого элемента DNS Handler

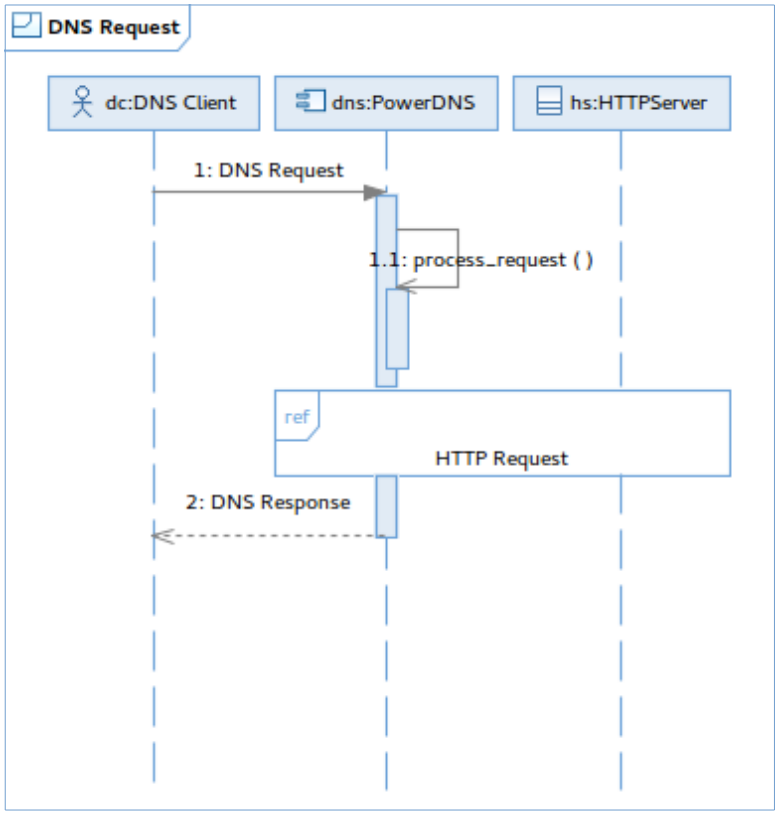

Рис. 2.10. Диаграмма последовательности для элемента расширения DNS Request

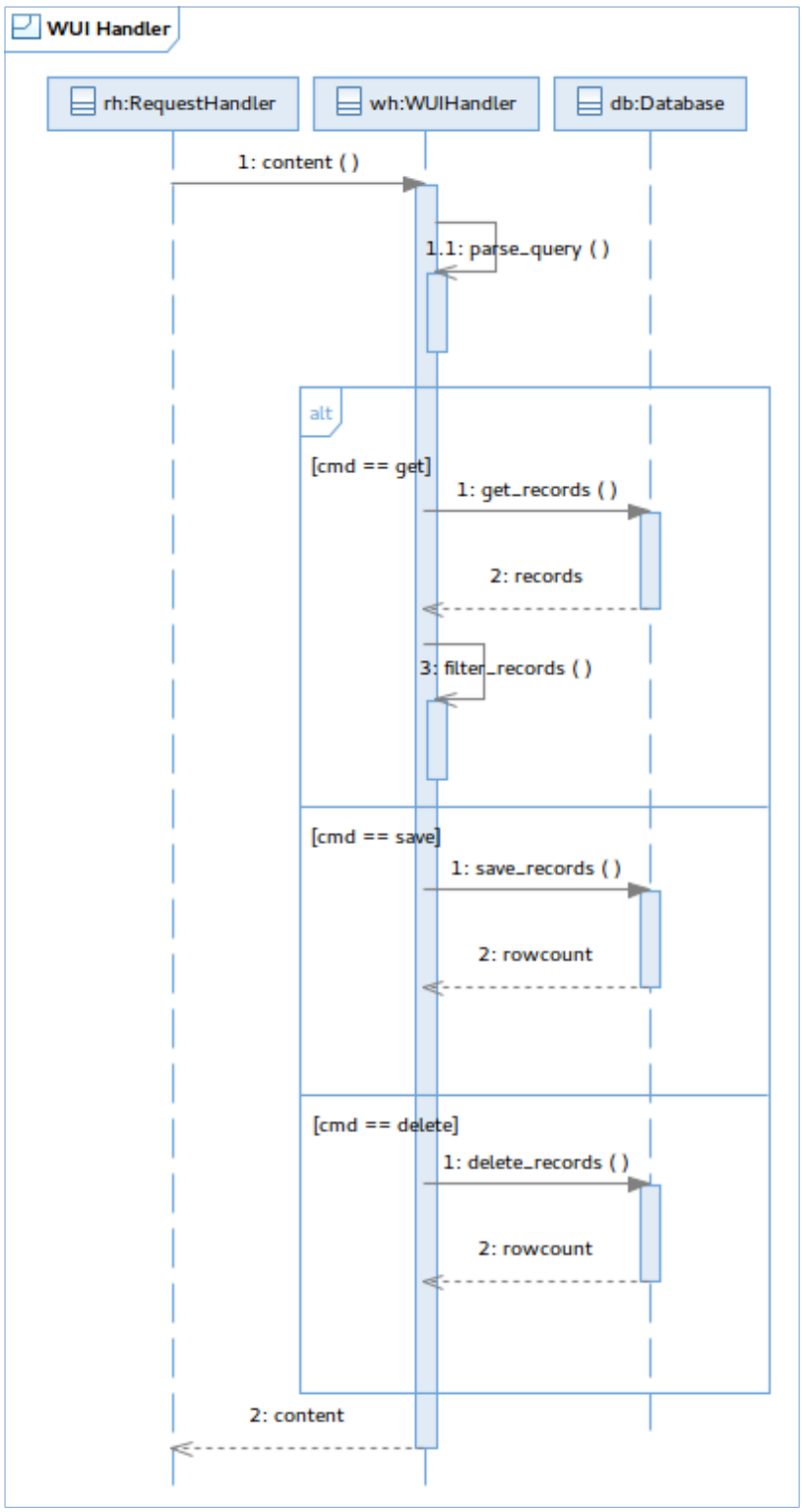

Рис. 2.11. Диаграмма последовательности для включаемого элемента WUI Handler

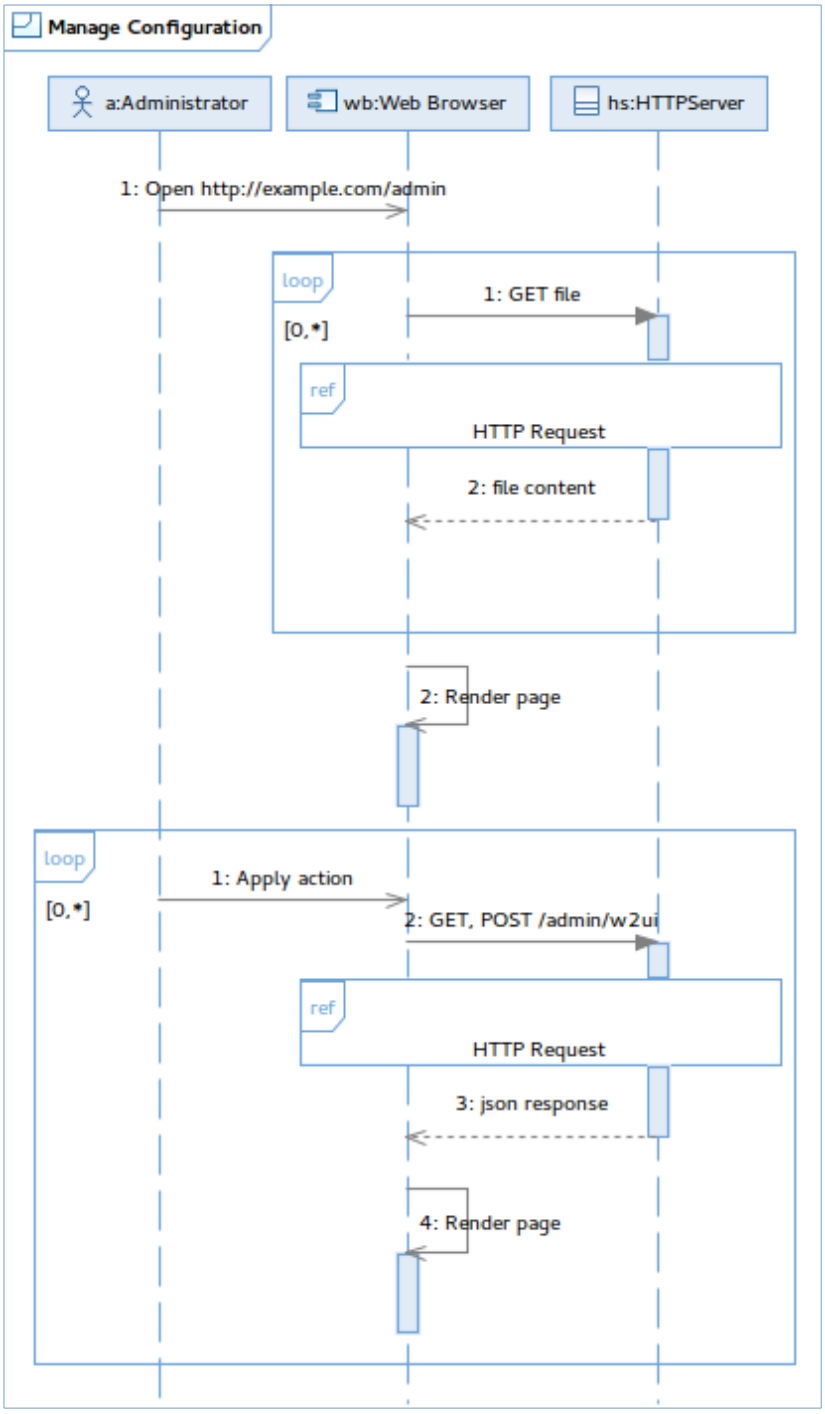

Рис. 2.12. Диаграмма последовательности для элемента расширения Manage Configuration

2.3. Проектирование архитектуры

Формирование архитектуры — первый и основополагающий шаг в решении задачи проектирования, закладывающий фундамент представления программной системы, способной выполнять весь спектр детальных требований (Орлов С. А. 2016).

Диаграммы классов считают основным средством для представления структуры систем в терминах базовых строительных блоков и отношений между ними (Орлов С. А. 2016).

В результате изучения содержания всех диаграмм последовательности определены имена и операции начальных классов. На рисунке 2.13 показаны полученные классы без учета отношений между ними.

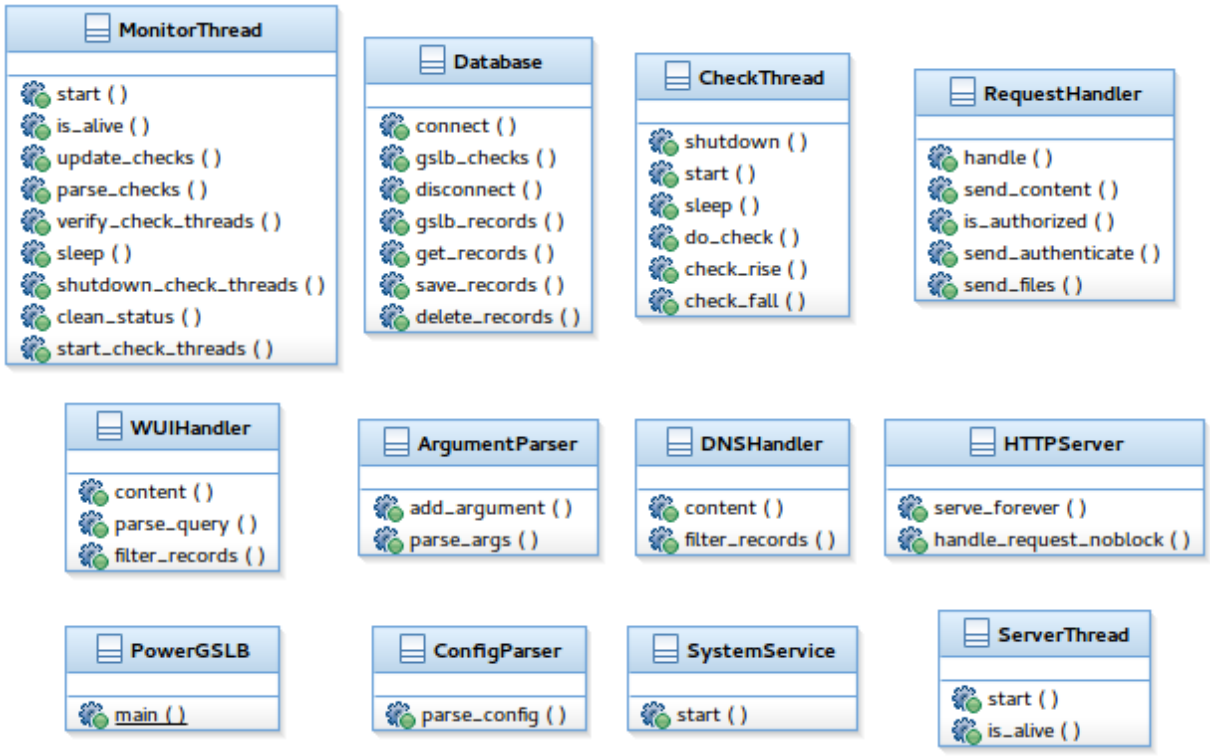

Рис. 2.13. Начальная диаграмма классов без учета отношений

На рисунке 2.14 показана начальная диаграмма классов с учетом отношений между ними. Для сокращения на диаграмме скрыты операции классов.

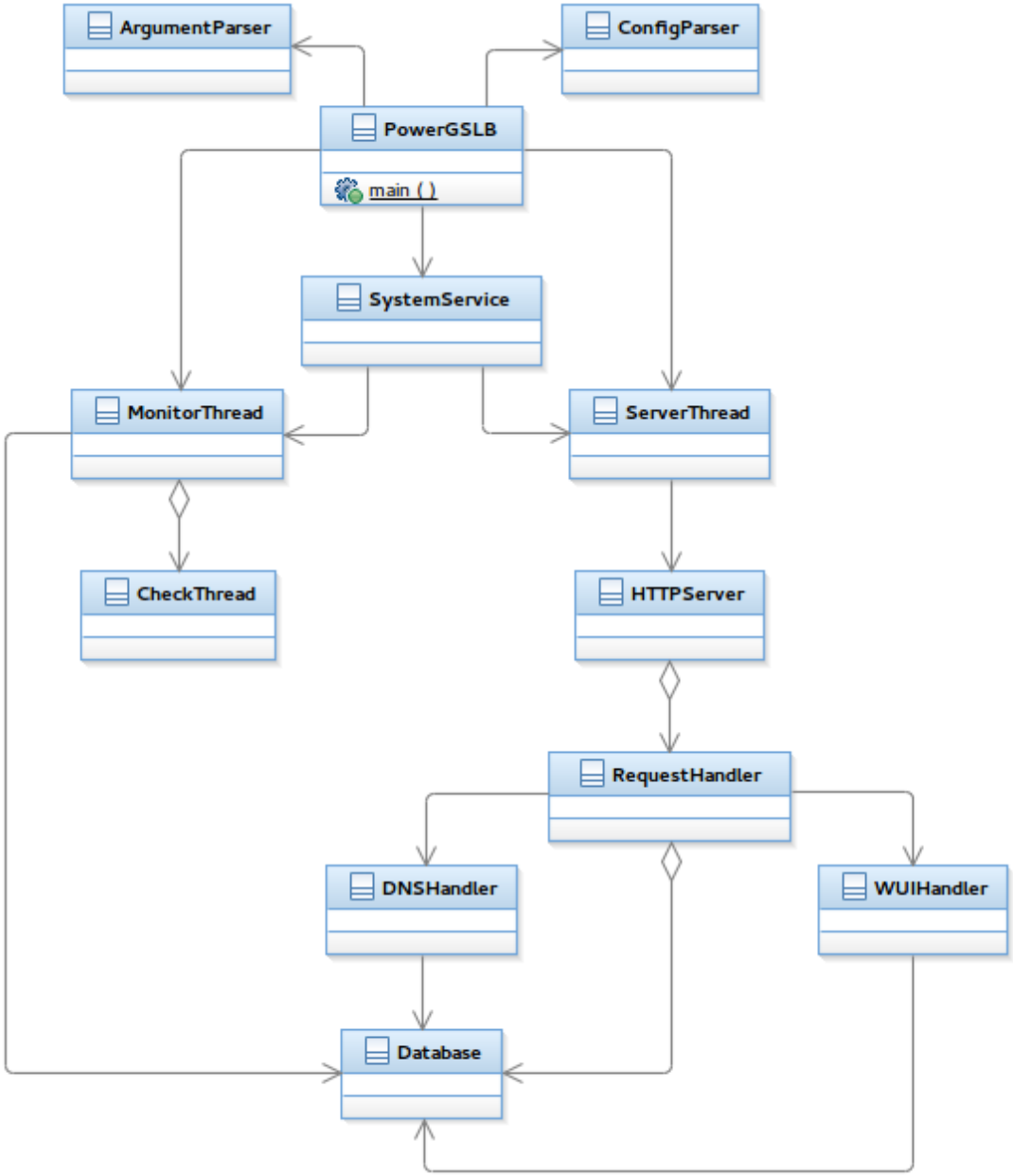

Рис. 2.14. Начальная диаграмма классов с учетом отношений

Для оценки качества логической структуры визуальной модели классов используется набор метрик Чидамбера-Кемерера, включающий 6 метрик:

- WMC — взвешенные методы на класс;
- DIT — высота дерева наследования;
- NOC — количество детей;
- CBO — сцепление между классами объектов;
- RFC — отклик для класса;
- LCOM4 — количество связанных компонентов в классе.

Результаты начальной оценки качества логической структуры визуальной модели и начальной оценки системы показаны в таблицах 2.1 и 2.2 соответственно.

Таблица 2.1

Начальная оценка качества логической структуры визуальной модели

Классы	**Метрики**					
	WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM4**
ArgumentParser	2	0	0	0	2	1
ConfigParser	1	0	0	0	1	1
PowerGSLB	1	0	0	5	8	1
SystemService	1	0	0	2	5	1
MonitorThread	9	0	0	2	15	1
CheckThread	6	0	0	0	6	1
ServerThread	2	0	0	1	5	1
HTTPServer	2	0	0	1	3	1
RequestHandler	5	0	0	3	9	1
DNSHandler	2	0	0	1	3	1
WUIHandler	3	0	0	1	6	1
Database	7	0	0	0	7	1
Среднее значение	**3.42**	**0.00**	**0.00**	**1.33**	**5.83**	**1.00**

Таблица 2.2

Начальная оценка системы

DIT	0
NC	12
NOM	41
LOCΣ	0

Из общего ряда выделяются неблагоприятные оценки сложности и отклика класса MonitorThread и сцепления класса PowerGSLB. Эти классы являются потенциальным источником повышенной опасности для качества. В процессе реализации и тестирования следует уделить им повышенное внимание и подумать об их возможной модификации

На основании начальной оценки можно сделать вывод, что средние значения качества системы являются удовлетворительными.

2.4. Проектирование базы данных

База данных (БД) — это совместно используемый набор устойчивых данных. База данных является единым и большим хранилищем данных (Орлов С. А. 2016).

На этапе анализа требований принято решение использовать для хранения конфигурации приложения базу данных. В качестве базы данных может выступать любое хранилище данных, например, файл, встраиваемая библиотека или СУБД.

Использование в данном проекте СУБД имеет ряд преимуществ:

1. Поддержка многопоточного, многозадачного и многопользовательского режимов доступа к данным.
2. Защита от несанкционированного доступа к базе данных.
3. Репликация данных в кластере из нескольких серверов.
4. Поддержка транзакционных механизмов.
5. Контроль целостности данных.
6. Наличие средств администрирования базы данных.
7. Поддержка стандартов, например, стандарта SQL.

Недостатки использования СУБД:

1. Сложность сопровождения.
2. Стоимость СУБД (включая затраты на проектирование).
3. Потребление системных ресурсов.

В случае использования СУБД разработчику не нужно своими силами реализовывать хранилище данных с параллельным доступом к нему из разных потоков программы, не нужно заботиться о защите данных и репликации данных в кластере из нескольких серверов. Все эти задачи возьмет на себя СУБД. То есть очевидным плюсом использования в данном проекте СУБД является обеспечение выполнения части нефункциональных требований и упрощение разработки.

В результате сравнения преимуществ и недостатков использования СУБД, с учетом наличия у разработчика приложения опыта работы с MySQL, принято решение использовать в проекте СУБД MySQL. В данном случае опыт работы с MySQL нивелирует недостатки сложности сопровождения и затраты на проектирование, потребление ресурсов MySQL процессом вписывается в уже заданные минимальные требования к серверу, а стоимостью можно пренебречь, так как СУБД MySQL распространяется бесплатно.

В данной предметной области в базе данных требуется хранить следующие основные атрибуты:

- имя DNS домена — атрибут domain;
- имя DNS записи — атрибут name;
- тип DNS записи — атрибут type;
- содержимое DNS записи — атрибут content;
- время кеширования DNS записи — атрибут ttl;
- признак исключения из балансировки — атрибут disabled;
- признак резервного сервера — атрибут fallback;
- признак привязки клиента к серверу — атрибут persistence;
- вес DNS записи — атрибут weight;
- конфигурация монитора — атрибут monitor;
- конфигурация DNS вида — атрибут view;
- имя пользователя — атрибут username;
- учетная запись пользователя — атрибут user;
- пароль пользователя — атрибут password.

Необходимо создать нормализованную модель базы данных. Нормализация обеспечивает оптимизацию структуры БД. Она приводит к устранению избыточности в наборах данных (Орлов С. А. 2016).

Правила нормализации оформлены в виде нормальных форм. Первая нормальная форма (1НФ) требует, чтобы значения всех элементов данных в столбцах были атомарными. Вторая нормальная форма (2НФ) требует, чтобы каждый неключевой столбец полностью зависел от первичного ключа. Третья нормальная форма (3НФ) требует, чтобы все неключевые столбцы (атрибуты) были взаимно независимы и полностью зависели от первичного ключа (Орлов С. А. 2016).

Моделирование базы данных выполнено в специализированной среде моделирования MySQL Workbench. В результате моделирования получена оптимальная структура БД в третьей нормальной форме. В полученной БД отсутствует избыточное дублирование данных.

На рисунке 2.15 показана полученная ERR (enhanced entity–relationship) модель БД. Для сокращения на диаграмме скрыты индексы.

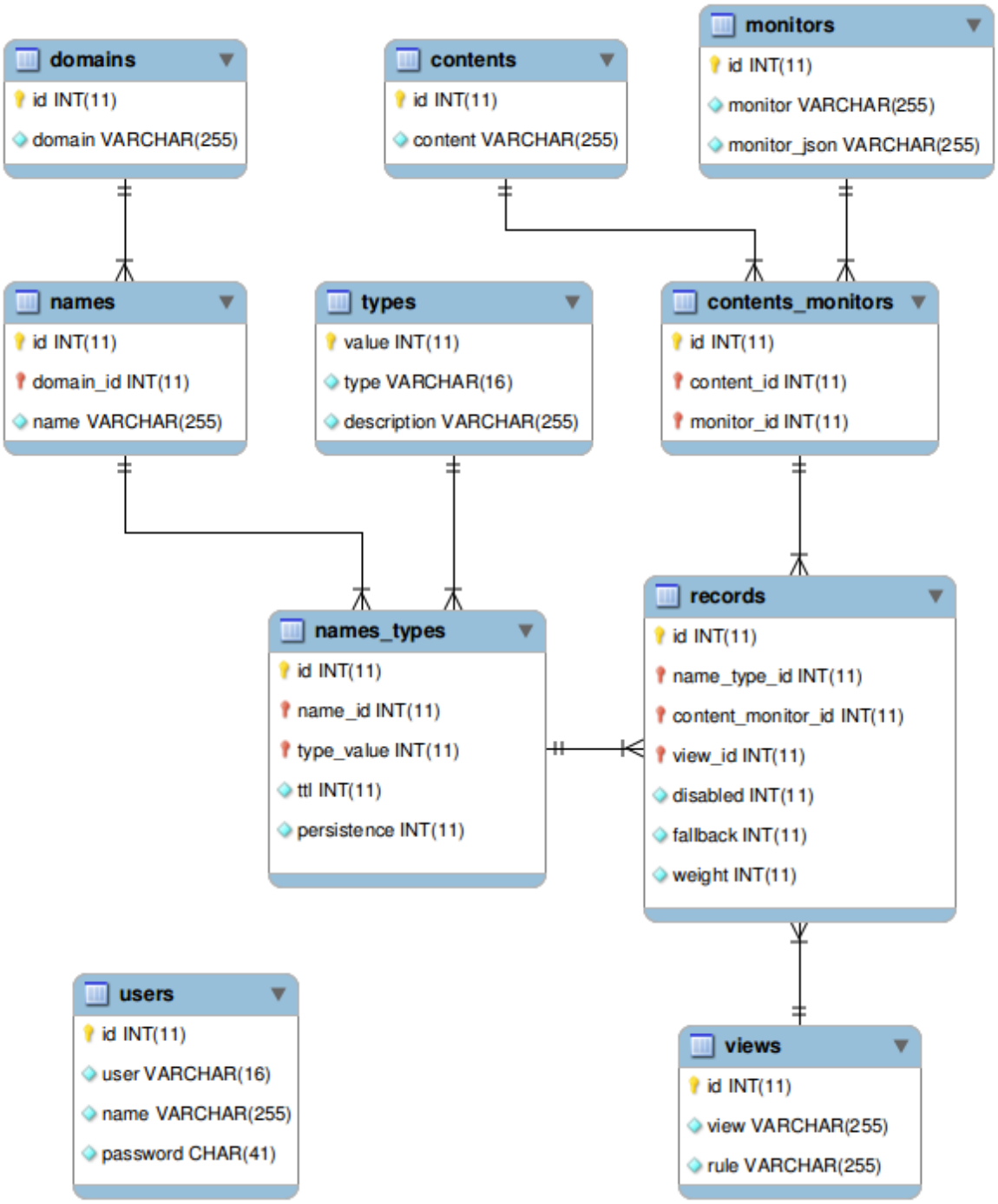

Рис. 2.15. Модель базы данных

2.5. Реализация

Учитывая ресурсные ограничения по времени реализации приложения и опыт разработчика, в качестве языка реализации выбран универсальный объектно-ориентированный язык программирования Python.

В процессе реализации изучены возможности стандартных и внешних библиотек языка Python, определены необходимые атрибуты и уточнены операции классов, определены общие черты поведения классов и построены деревья наследования, классы упакованы в пакеты, построены диаграммы пакетов и конечная диаграмма классов, выполнено кодирование и конфигурация приложения.

Основные средства реализации с указанием версии ПО:

- IBM Rational Software Architect 9.5.
- Языки программирования:
  - Python 2.7
  - JavaScript 1.7
- Среда разработки PyCharm IDE 2016.1.
- Среда моделирования MySQL Workbench 6.3.
- Офисный пакет LibreOffice 4.3.
- Редактор изображений GIMP 2.8.
- Система контроля версий Git 1.8.
- Операционная система Linux CentOS 7.2.

На рисунках 2.16 — 2.20 показаны полученные диаграммы пакетов без учета отношений и внешних, по отношению к основной программе, пакетов и классов.

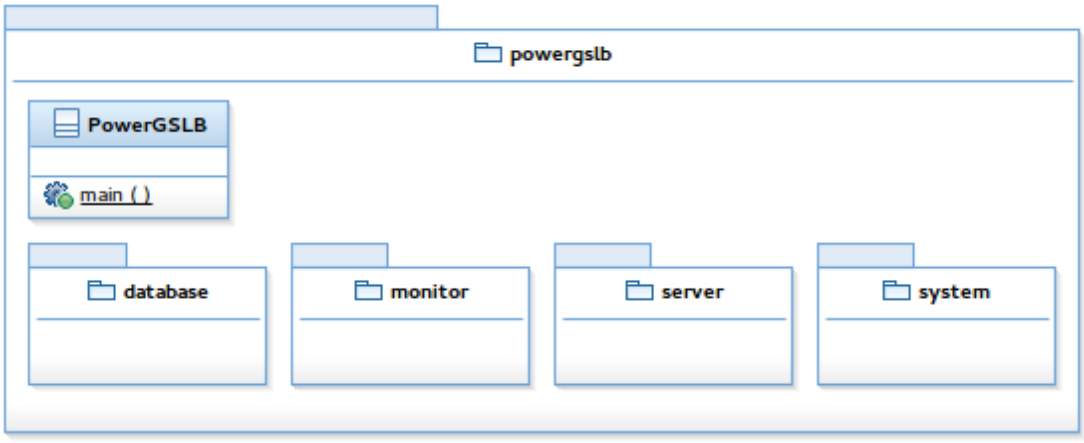

Рис. 2.16. Диаграмма пакетов для базового пакета powergslb

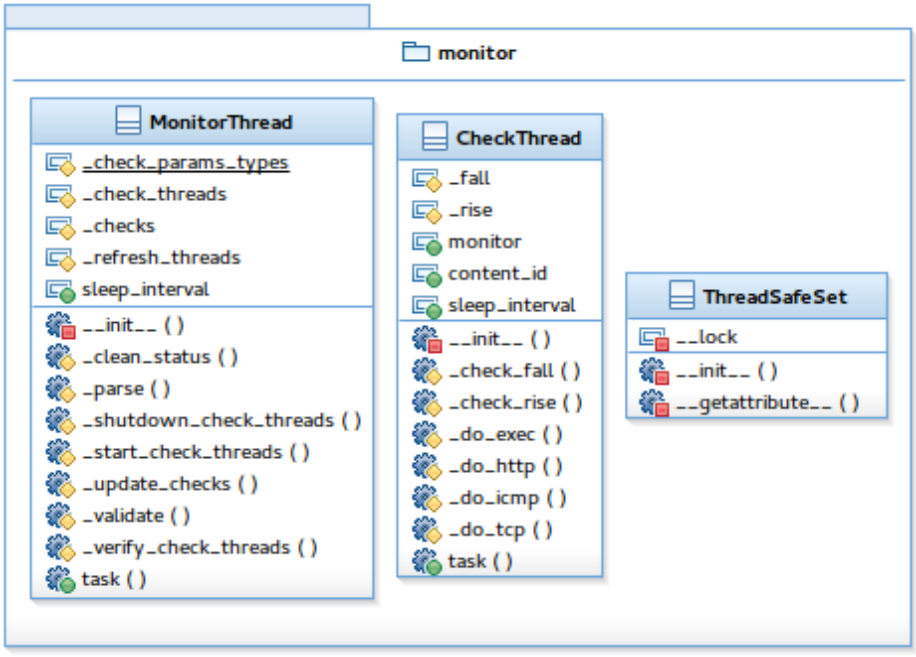

Рис. 2.17. Диаграмма пакетов для вложенного пакета monitor

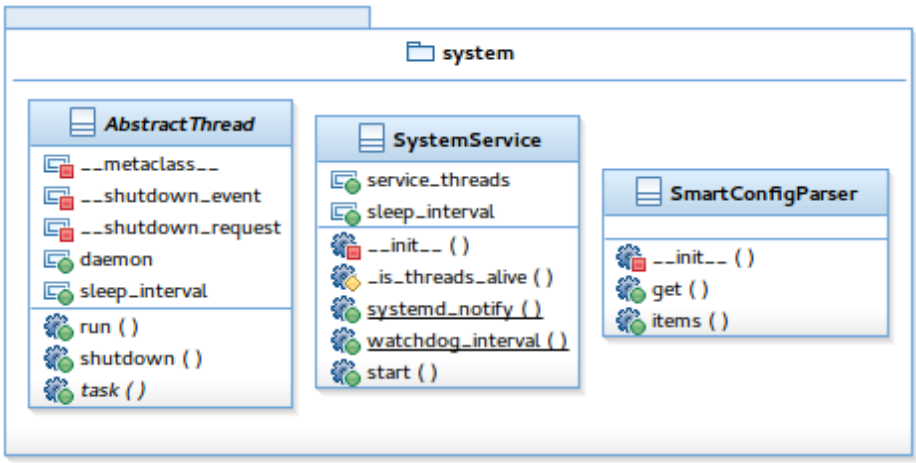

Рис. 2.18. Диаграмма пакетов для вложенного пакета system

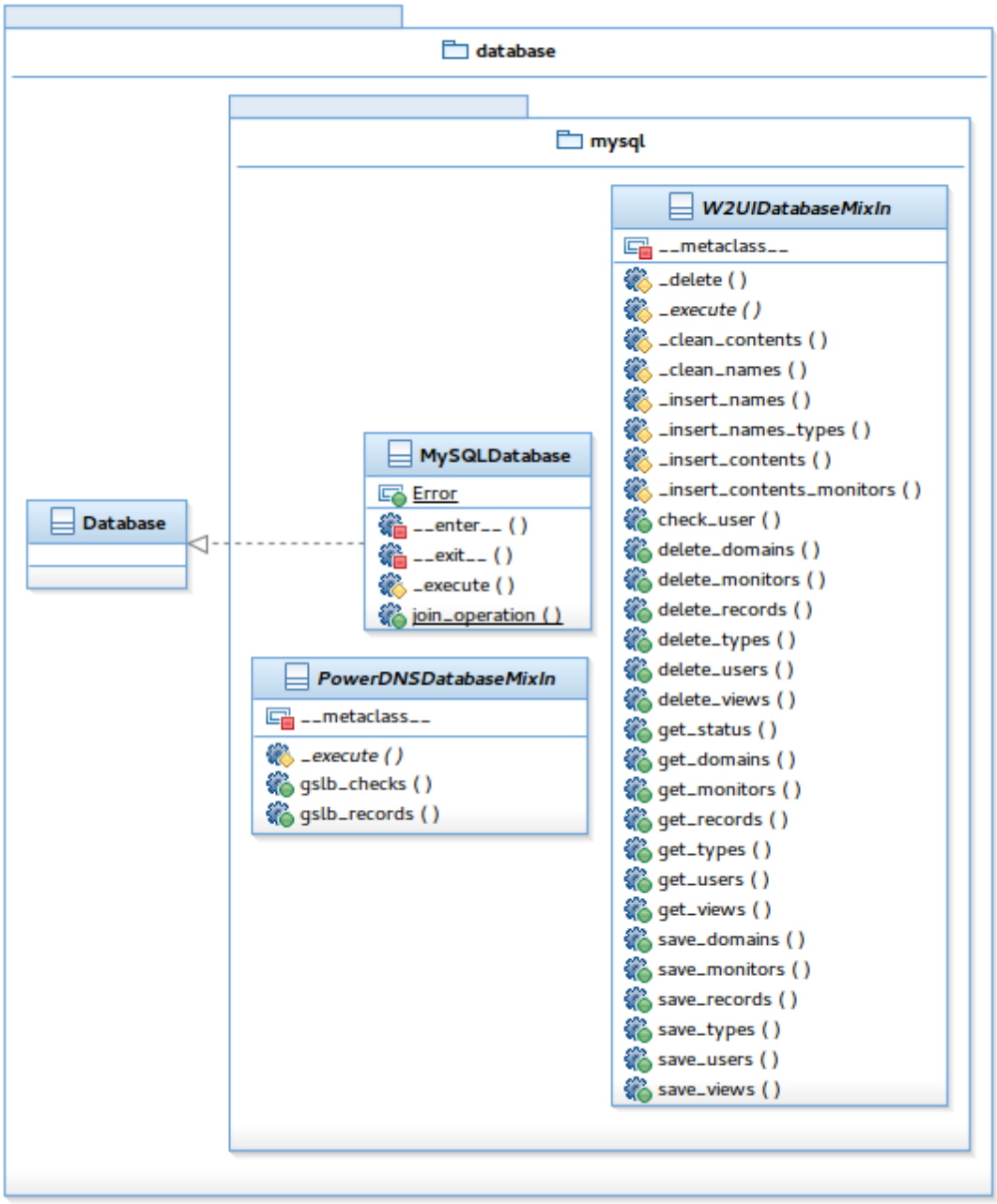

Рис. 2.19. Диаграмма пакетов для вложенного пакета database

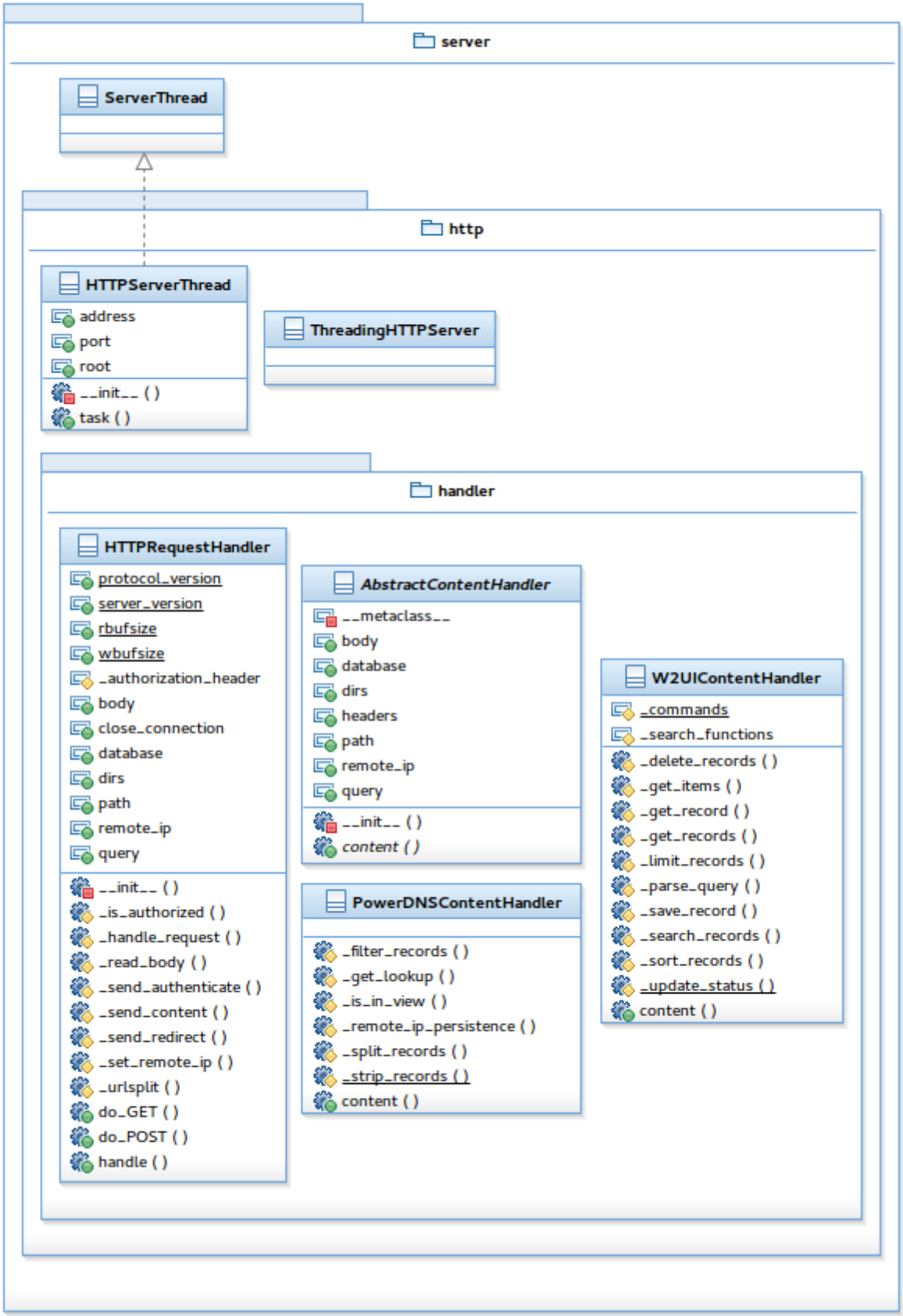

Рис. 2.20. Диаграмма пакетов для вложенного пакета server

Результаты конечной оценки качества модели и конечной оценки системы показаны в таблицах 2.3 и 2.4 соответственно.

Таблица 2.3

Конечная оценка качества логической структуры визуальной модели

Классы	**Метрики**					
	WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM4**
PowerGSLB	1	0	0	5	8	1
SmartConfigParser	3	1	0	0	3	1
SystemService	5	0	0	2	9	1
AbstractThread	3	1	3	0	3	1
MonitorThread	9	2	0	2	15	1
CheckThread	8	2	0	0	8	1
HTTPServerThread	2	2	0	1	5	1
ThreadingHTTPServer	0	1	0	1	1	1
HTTPRequestHandler	12	1	0	2	15	1
AbstractContentHandler	2	1	2	0	2	1
PowerDNSContentHandler	7	2	0	1	9	1
W2UIContentHandler	11	2	0	1	15	1
MySQLDatabase	4	2	0	0	4	1
PowerDNSDatabaseMixIn	3	1	1	0	3	1
W2UIDatabaseMixIn	28	1	1	0	28	1
ThreadSafeSet	2	1	0	0	2	1
Среднее значение	**6.25**	**1.25**	**0.44**	**0.94**	**8.13**	**1.00**

Таблица 2.4

Конечная оценка системы

DIT	2
NC	16
NOM	100
LOCΣ	1771

По сравнению с начальной оценкой наблюдается рост значений WMC, CBO, RFC, NC и NOM, что свидетельствует о возрастании сложности программного продукта.

Резко выделяются значения WMC и RFC класса W2UIDatabaseMixIn. И хотя метрика WMC превышает рекомендуемое значение в 20 методов (Орлов С. А. 2016), в данной реализации это не является проблемой, так как класс содержит большое количество SQL запросов и минимум программного кода.

Увеличение значений DIT и NOC свидетельствует об улучшении качества за счет повторного использования кода посредством наследования.

Значения метрики LCOM4 свидетельствует о качестве программного продукта, так как в связанном классе должен быть только один связанный компонент.

На основании конечной оценки можно сделать вывод, что качество модели является удовлетворительным.

На рисунке 2.21 показана конечная диаграмма классов с учетом внешних, по отношению к основной программе, классов и отношений между классами. Для сокращения на диаграмме скрыты пакеты, атрибуты и операции классов. Внутренние (реализованные) классы отображены синим цветом. Внешние (библиотечные) классы отображены зеленым цветом.

Код базового пакета powergslb и части вложенных пакетов, по состоянию на момент написания данной работы, приведен в приложении. Последняя версия приложения доступна в Git репозитории проекта по адресу https://github.com/AlekseyChudov/powergslb. Там же хранится вся история версий ПО.

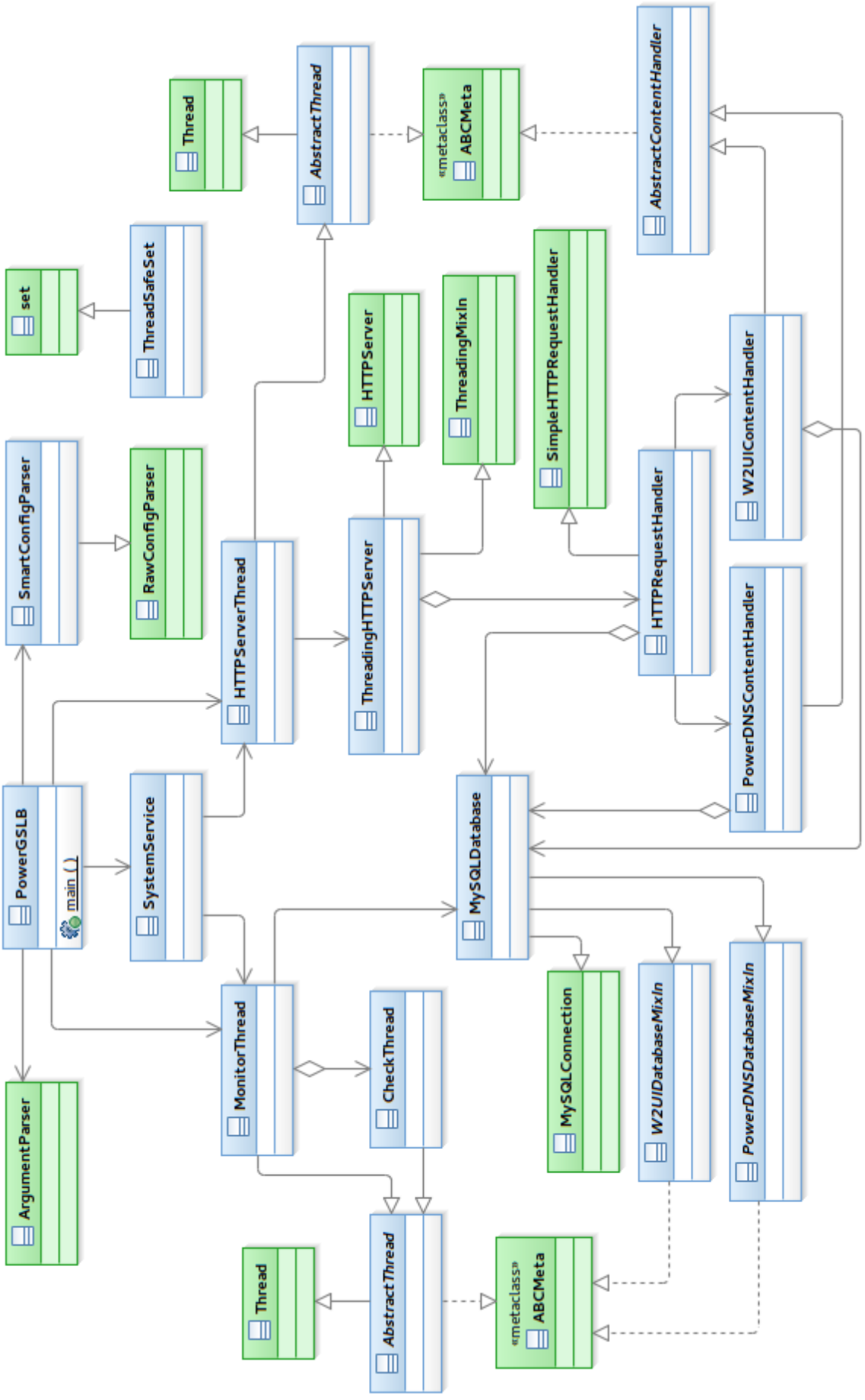

Рис. 2.21. Конечная диаграмма классов с учетом отношений

2.6. Тестирование

Тестирования — это процесс выполнения программы с целью обнаружения ошибок (Орлов С. А. 2016).

Тестирование программной системы выполнено вручную по принципу «черного ящика» способом разбиения по эквивалентности.

Тестирование «черного ящика» (функциональное тестирование) позволяет получить комбинации входных данных, обеспечивающих полную проверку всех функциональных требований к программе (Орлов С. А. 2016).

При использовании способа разбиения по эквивалентности входная область данных программы делится на классы эквивалентности. Для каждого класса эквивалентности разрабатывается один тестовый вариант (Орлов С. А. 2016).

Рассмотрим операцию запуска приложения операционной системой.

Предусловия

- конфигурационный файл доступен для чтения;
- конфигурационный файл не содержит ошибок;
- указанная в конфигурационном файле база данных доступна;
- указанный в конфигурационном файле IP адрес принадлежит серверу;
- указанный в конфигурационном файле TCP порт свободен на сервере.

Постусловия

- в случае выполнения предусловий — корректный старт приложения;
- в случае невыполнения предусловий — сообщения об ошибке в системном логе.

Рассмотрим операцию запуска веб-интерфейса управления конфигурацией.

Предусловия

- администратор вводит правильное имя пользователя.
- администратор вводит правильный пароль.

Постусловия

- в случае выполнения предусловий — корректный старт веб-интерфейса;
- в случае невыполнения предусловий — сообщения об ошибке авторизации.

Рассмотрим операцию добавление новой DNS записи.

Предусловия

- администратор вводит существующий домен в поле «Domain»;
- администратор вводит существующий тип в поле «Type»;
- администратор вводит существующий монитор в поле «Monitor»;
- администратор вводит значение больше 0 в поле «TTL»;
- администратор вводит значение от 0 до 1 в поле «Disabled»;
- администратор вводит значение от 0 до 1 в поле «Fallback».

Постусловия

- в случае выполнения предусловий — корректное создание записи;
- в случае невыполнения предусловий — сообщения об ошибке создания записи.

Классы эквивалентности

1. Для тех условий, которые задают булевое значение, определим один допустимый и один недопустимый класс эквивалентности:
   - V_Class = {true};
   - Inv_Class = {false}.
2. Для тех условий, которые задают диапазон значений *n..m*, определим один допустимый и два недопустимых класса эквивалентности:
   - V_Class = $\{n..m\}$;
   - Inv_Class1 = $\{x \mid x: x < n\}$;
   - Inv_Class2 = $\{y \mid y: y > n\}$.
3. Для тех условий, которые задают множество значений $\{a, b, c\}$, определим один допустимый и один недопустимый класс эквивалентности:
   - V_Class = $\{a, b, c\}$;
   - Inv_Class = $\{x \mid x: (x \neq a) \,\&\, (x \neq b) \,\&\, (x \neq c)\}$.
4. Для тех условий, которые задают конкретное значение *a*, определим один допустимый и два недопустимых класса эквивалентности:
   - V_Class = $\{a\}$;
   - Inv_Class1 = $\{x \mid x: x < a\}$;
   - Inv_Class2 = $\{y \mid y: y > a\}$.

Нужно разработать такие тестовые варианты, которые проверяют наибольшее количество классов эквивалентности. Примеры тестовых вариантов и результаты тестов показаны в таблице 2.5.

Аналогичным образом выполнено тестирование всех функциональных и нефункциональных требований к программе. В результате проведенного тестирования отлажена работа приложения, исправлены обнаруженные мелкие ошибки и не обнаружено серьезных ошибок, требующих проектных изменений.

Таблица 2.5

Примеры тестовых вариантов

Входные данные	Ожидаемый результат	Результат теста
Выполняется команда запуска приложения	1) Корректный старт приложения 2) Нет сообщений об ошибке в системном логе	пройден
1) Конф. файл недоступен для чтения 2) Выполняется команда запуска приложения	1) Некорректный старт приложения 2) Есть сообщения об ошибке в системном логе	пройден
1) Конф. файл содержит ошибку 2) Выполняется команда запуска приложения	1) Некорректный старт приложения 2) Есть сообщения об ошибке в системном логе	пройден
1) База данных недоступна 2) Выполняется команда запуска приложения	1) Некорректный старт приложения 2) Есть сообщения об ошибке в системном логе	пройден
1) IP адрес не принадлежит серверу 2) Выполняется команда запуска приложения	1) Некорректный старт приложения 2) Есть сообщения об ошибке в системном логе	пройден
1) TCP порт занят на сервере 2) Выполняется команда запуска приложения	1) Некорректный старт приложения 2) Есть сообщения об ошибке в системном логе	пройден
Выполняется запуск веб-интерфейса	Корректный старт веб-интерфейса	пройден
1) Выполняется запуск веб-интерфейса 2) Вводится неправильное имя пользователя	Сообщения об ошибке авторизации	пройден
1) Выполняется запуск веб-интерфейса 2) Вводится неправильный пароль	Сообщения об ошибке авторизации	пройден
Добавляется новая DNS запись	Корректное создание записи	пройден
1) Вводится несуществующий домен в поле «Domain» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	пройден
1) Вводится несуществующий тип в поле «Type» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	пройден
1) Вводится несуществующий монитор в поле «Monitor» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	пройден
1) Вводится значение меньше 0 в поле «TTL» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	пройден
1) Вводится значение меньше 0 в поле «Disabled» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	пройден
1) Вводится значение больше 1 в поле «Disabled» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	Нет ошибки
1) Вводится значение меньше 0 в поле «Fallback» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	пройден
1) Вводится значение больше 1 в поле «Fallback» 2) Добавляется новая DNS запись	Сообщения об ошибке создания записи	Нет ошибки

2.7. Оценка качества

Неотъемлемым свойством современной программной системы является высокое качество. Существует множество определений качества программного обеспечения (Орлов С. А. 2016).

Например, в авторитетном словаре программной инженерии IEEE Std 610.12-90 «IEEE Standard Glossary of Software Engineering Terminology» записано:

1. Качество ПО — это степень соответствия системы, компонента или процесса определенным требованиям.
2. Качество ПО — это степень соответствия системы, компонента или процесса нуждам или ожиданиям заказчика, пользователя (Орлов С. А. 2016).

В процессе работы обеспечено выполнение всех заявленных функциональных и нефункциональных требований к ПО, что является фундаментальным показателем качества (Орлов С. А. 2016).

Некоторые количественные и качественные оценки приведены ранее в таблицах 2.1 — 2.4. На основании оценки качества визуальной модели можно сделать вывод, что средние значения качества системы являются удовлетворительными.

Основные тестовые варианты приведены ранее в таблице 2.5. На основании результатов проведенного тестирования можно сделать вывод, что функции ПО спроектированы и реализованы правильно, а качество и надежность ПО являются удовлетворительными.

Благодаря использованию модульной структуры и разбиению классов программы на пакеты, упрощается расширение функциональности (functionality) и сопровождаемость (maintainability), что также является показателем качества полученного ПО.

ЗАКЛЮЧЕНИЕ

Целью данной работы была разработка открытого ПО для глобальной серверной балансировки посредством системы доменных имен (Domain Name System based Global Server Load Balancing, DNS GSLB).

Основные задания по работе были следующие:

1. Рассмотреть и сравнить основные технологии балансировки нагрузки и трафика, используемые на различных уровнях сетевой модели OSI.
2. Рассмотреть и описать этапы процесса разработки приложения для глобальной серверной балансировки посредством системы доменных имен.

В первой части работы были рассмотрены такие технологии балансировки нагрузки и трафика, как агрегация каналов (LAG), многолучевая маршрутизация равной стоимости (ECMP), серверная балансировка нагрузки (SLB) и глобальная серверная балансировка нагрузки (GSLB). Особое внимание уделено технологиям SLB и GSLB как наиболее часто реализуемым на уровне программного обеспечения, а не на уровне сетевого оборудования. В заключении первой части работы было проведено краткое сравнение рассмотренных технологий балансировки нагрузки и трафика.

Во второй части работы были рассмотрены этапы процесса разработки приложения для глобальной серверной балансировки посредством DNS. В процессе разработки были сформированы и проанализированы требования к продукту, проведено проектирование приложения и базы данных, реализовано приложение, проведено его тестирование и оценка.

В результате проведенной работы получены следующие преимущества:

- Разработано простое в использовании решение для обеспечения глобальной серверной балансировки нагрузки посредством DNS.
- Низкая стоимость конечного продукта за счет использования открытых технологий.
- Открытая модель разработки делает ПО общедоступным.

К сожалению, некоторые технические детали остались за кадром. Например, особенности разработки веб-приложения на языке JavaScript и развертывание ПО на кластере серверов. Но в целом по результатам проделанной работы можно сделать вывод, что цель работы достигнута, и задания по работе успешно выполнены.

СПИСОК ЛИТЕРАТУРЫ

1. Олифер В. Г., Олифер Н. А. 2010, *Компьютерные сети. Принципы, технологии, протоколы*. Учебник для вузов. 4-е издание. Питер, СПб.
2. Орлов С. А. 2016, *Программная инженерия. Технологии разработки программного обеспечения*. Учебник для вузов. 5-е издание. Питер, СПб.
3. Орлов С. А., Комашилова О. Я. 2006, *Технологии разработки программного обеспечения*. Методические указания по выполнению курсового проекта. TSI, Rīga.
4. Bourke T. 2001, *Server Load Balancing*. O'Reilly Media.
5. Kopparapu C. 2002, *Load Balancing Servers, Firewalls, and Caches*. Wiley Computer Publishing.
6. Sing J. 2016, *Seesaw: scalable and robust load balancing*. Available from: <http://google-opensource.blogspot.com/2016/01/seesaw-scalable-and-robust-load.html>. [22.05.2016].
7. Zhang W. 2011, *Virtual Server via NAT*. Available from: <http://www.linuxvirtualserver.org/VS-NAT.html>. [22.05.2016].
8. Zhang W. 2011, *Virtual Server via IP Tunneling*. Available from: <http://www.linuxvirtualserver.org/VS-IPTunneling.html>. [22.05.2016].
9. Zhang W. 2011, *Virtual Server via Direct Routing*. Available from: <http://www.linuxvirtualserver.org/VS-DRouting.html>. [22.05.2016].
10. IEEE Std. 1901 2010, *IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications*. Available from: <https://standards.ieee.org/findstds/standard/1901-2010.html>. [22.05.2016].
11. IEEE Std. 802.3, *ETHERNET*. Available from: <https://standards.ieee.org/about/get/802/802.3.html>. [22.05.2016].
12. IEEE Std. 802.11, *Wireless LANs*. Available from: <https://standards.ieee.org/about/get/802/802.11.html>. [22.05.2016].
13. IEEE Std. 802.16, *BROADBAND WIRELESS METROPOLITAN AREA NETWORKS (MANs)*. Available from: <https://standards.ieee.org/about/get/802/802.16.html>. [22.05.2016].
14. IEEE Std. 802.1AX 2008, *IEEE Standard for Local and metropolitan area networks - Link Aggregation*. Available from: <https://standards.ieee.org/findstds/standard/802.1AX-2008.html>. [22.05.2016].
15. IETF RFC 1035 1987, *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*. Available from: <https://tools.ietf.org/html/rfc1035>. [22.05.2016].

16. IETF RFC 2178 1998, *OSPF Version 2*. Available from: <https://tools.ietf.org/html/rfc2178>. [22.05.2016].
17. IETF RFC 2453 1998, *RIP Version 2*. Available from: <https://tools.ietf.org/html/rfc2453>. [22.05.2016].
18. IETF RFC 2616 1999, *Hypertext Transfer Protocol — HTTP/1.1*. Available from: <https://www.ietf.org/rfc/rfc2616.txt>. [22.05.2016].
19. IETF RFC 2991 2000, *Multipath Issues in Unicast and Multicast Next-Hop Selection*. Available from: <https://tools.ietf.org/html/rfc2991>. [22.05.2016].
20. IETF RFC 4271 2006, *A Border Gateway Protocol 4 (BGP-4)*. Available from: <https://tools.ietf.org/html/rfc4271>. [22.05.2016].
21. PowerGSLB, computer software 2016. Available from: <https://github.com/AlekseyChudov/powergslb>. [22.05.2016].