SATURS

DARBĀ IZMANTOTIE SAĪSINĀJUMI

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

IEVADS

Noslodzes un trafika balansēšana nav jauna koncepcija un tiek plaši izmantota IT industrijā. Daudzi aparātprogrammatūras risinājumi izpilda dažāda tipa noslodzes un trafika balansēšanu.

Piemēram, tīkla komutatori (switches) var sadalīt trafiku pa vairākām fiziskām sakaru līnijām, bet maršrutētāji (routers) var balansēt trafiku pa vairākiem maršrutiem, tādējādi sadalot noslodzi starp datu pārraides tīkla kanāliem.

No otras puses, serveru noslodzes balansētāji (Server Load Balancer, SLB) sadala trafiku starp vairākiem serveriem, nevis tīkla kanāliem. Mūsdienu serveru noslodzes balansētāji pilda dažādas funkcijas: noslodzes un trafika balansēšanu, trafika intelektuālu pārslēgšanu pēc uzdota algoritma, serveru, aplikāciju un satura pieejamības pārbaudi drošuma paaugstināšanai. Tā kā serveru noslodzes balansētāji tīkla topoloģijā atrodas pirms serveru fermas, tie piedevām aizsargā serverus no dažādiem tīkla uzbrukumiem un paaugstina drošības līmeni.

Savukārt globālie serveru noslodzes balansētāji (Global Server Load Balancer, GSLB) sadala trafiku starp dažādiem datu apstrādes centriem, lai samazinātu atbildes laiku un paaugstinātu drošumu datu apstrādes centra pilnīgas nepieejamības gadījumā.

Slodzes balansētājiem ir daudz nosaukumu: OSI modeļa otrā līdz septītā slāņa intelektuālie komutatori (layer 2 — 7 switches), tīmekļa komutatori (web switches), satura slēdži (content switches) u.c. Neatkarīgi no nosaukuma tie visi būtībā pilda savu galveno funkciju — noslodzes un trafika balansēšanu.

Noslodzes un trafika balansēšanas tēma ir aktuāla ne tikai augsti noslogotiem pakalpojumiem, bet arī visiem projektiem, kas cenšas nodrošināt saviem klientiem labāku servisa kvalitāti, samazinot apkalpošanas atteikumu skaitu un saīsinot aizkaves datu pārraidē pa globālo tīmekli.

Piemēram, šī darba rakstīšanas laikā Google meklētājs uz pieprasījumiem „server load balancing” un „global server load balancing” izdeva attiecīgi 10 milj. un 3 milj. rezultātus, bet Amazon, Cisco, Citrix, Kemp, Radware un citi lielie piegādātāji piedāvā gatavus aparātprogrammatūras risinājumus gan serverus noslodzes balansēšanai, gan globālai serveru noslodzes balansēšanai. Atkarībā no pieteiktās veiktspējas un caurlaidspējas šādu risinājumu izmaksas bieži ir mērāmas desmitos un pat simtos tūkstošu dolāru.

Nevēloties nokļūt atkarībā no viena piegādātāja, kā arī vērā ņemamo izmaksu dēļ, daudzas lielas kompānijas izstrādā pašas savus SLB un GSLB risinājumus uz atvērtās programmatūras bāzes.

Piemēram, Google inženieri (Google Site Reliability Engineers) izstrādāja savām vajadzībām un padarīja visiem pieejamu SLB balansētāju Seesaw uz atvērtā projekta Linux Virtual Server (LVS) bāzes (Sing, 2016). Tādas kompānijas kā Alibaba, Facebook, Mail.ru un Yandex arī izmanto programmētus SLB un GSLB risinājumus.

Uz šī darba uzrakstīšanas brīdi autoram izdevās atrast tikai vienu pastāvošu atvērtās programmatūras projektu globālas serveru noslodzes balansēšanas uzdevuma risināšanai.

Šī darba mērķis ir atvērtās programmatūras izstrāde globālai serveru noslodzes balansēšanai, izmantojot domēnu vārdu sistēmu (Domain Name System based Global Server Load Balancing, DNS GSLB).

1. NOSLODZES UN TRAFIKA BALANSĒŠANAS TEHNOLOĢIJAS

Darba pirmajā daļā tiek apskatītas galvenās noslodzes un trafika balansēšanas tehnoloģijas, kas tiek izmantotas mūsdienu datorsistēmās. Īpaša uzmanība veltīta serveru noslodzes balansēšanas un globālas serveru noslodzes balansēšanas jēdzienu, plusu un mīnusu apskatei.

Noslēgumā sniegts apskatīto noslodzes un trafika balansēšanas tehnoloģiju īss salīdzinājums.

1.1. Noslodzes balansēšana un OSI tīkla modelis

Apskatot un salīdzinot dažādas noslodzes un trafika balansēšanas tehnoloģijas, svarīgs kritērijs ir OSI (Open Systems Interconnection) modeļa slānis, kurā darbojas apskatāmās tehnoloģijas.

OSI tīkla modelis ir konceptuāls modelis, kas raksturo un standartizē tīkla mijiedarbības funkcijas un līdzekļus skaitļošanas sistēmā. OSI nosaka sistēmu mijiedarbības slāņus ar pakešu komutāciju, slāņu standarta nosaukumus un katram slānim pildāmās funkcijas. Turklāt OSI modelis nesatur konkrēta protokolu komplekta realizācijas aprakstu. OSI modelī mijiedarbības līdzekļi ir iedalīti septiņos slāņos: fiziskajā, kanāla, tīkla, transporta, seansa, pasniegšanas un lietojuma (Олифер and Олифер, 2010).

1.1. tabulā uzskaitīti OSI modeļa slāņi, to funkcijas un katram slānim atbilstošo datu pārraides protokolu vai fizisko vižu piemēri.

1.1. tabula

OSI tīkla modelis

Slānis (Layer)	Funkcijas	Piemēri
7. Lietojuma (Application)	Piekļuve tīkla pakalpojumiem	HTTP, FTP, SMTP, SSH, Telnet
6. Pasniegšanas (Presentation)	Datu pasniegšana un šifrēšana	MIME, SSL
5. Seansa (Session)	Sakaru seansa vadība	NetBIOS, SOCKS
4. Transporta (Transport)	Datu pārraide nepieciešamajā drošuma pakāpē	TCP, UDP, SCTP
3. Tīkla (Network)	Vienotas transporta sistēmas izveidošana	IPv4, IPv6, IPsec, IPIP, GRE
2. Kanāla (Data Link)	Savienojuma caurspīdīgums tīkla slānim	ATM, ARP, LLC, MAC, Ethernet, Frame Relay
1. Fiziskais (Physical)	Bitu plūsmas pārraide pa fiziskiem sakaru kanāliem	Vītais pāris, optiskais kabelis, radiosignāls

Noslodzes un trafika balansēšanas tehnoloģijas tiek aktīvi izmantotas praktiski katrā OSI modeļa slānī. Sākumā aplūkosim dažas noslodzes un trafika balansēšanas tehnoloģijas, kas tiek izmantotas tādos mūsdienu datortīklos, kam nav nepieciešami specializēti noslodzes balansētāji.

1.2. Noslodzes balansēšana OSI tīkla modeļa fiziskajā slānī

Fiziskais slānis (physical layer) — OSI modeļa zemākais slānis, kurš nodrošina bitu plūsmu pārraidi pa fiziskiem sakaru kanāliem: koaksiālo kabeli, vīto pāri, optiskās šķiedras kabeli, radiosignālu u.c.

Fiziskā slāņa funkcijas tiek īstenotas visās tīklam pieslēgtajās ierīcēs. No personālā datora puses fiziskā slāņa funkcijas pilda, piemēram, tīkla vai bezvadu adapteris (Олифер and Олифер, 2010).

Galvenais caurlaidspējas un datu pārraides drošuma palielināšanas paņēmiens OSI modeļa fiziskajā slānī ir vairāku fizisko sakaru kanālu vai vairāku frekvenču diapazonu paralēla izmantošana, t.i., noslodzes balansēšana starp vairākiem paralēliem fiziskajiem sakaru kanāliem, tos apvienojot vienā sakaru līnijā.

OSI modeļa fiziskajā slānī kā noslodzes un trafika balansēšanas piemēri minamas elektropārvades līnijas (IEEE 1901), bezvadu tīkli (IEEE 802.11, IEEE 802.16), Ethernet tīkli (IEEE 802.3) un citi tīkla standarti, kas sevī apvieno vairākus frekvenču diapazonus vai vairākus fiziskus signāla nesējus vienā sakaru līnijā.

1.3. Noslodzes balansēšana OSI tīkla modeļa kanāla slānī

Kanāla slānis (data link layer) nodrošina savienojuma caurspīdīgumu nākamajam tīkla slānim. Šai nolūkā tas piedāvā tam noteiktus pakalpojumus:

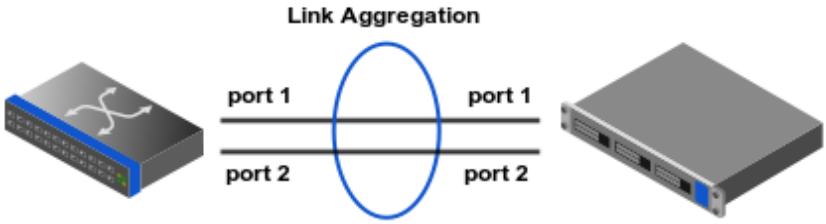
- loģiska savienojuma izveidi starp mezgliem, kas mijiedarbojas;
- informācijas raidītāja un uztvērēja ātrumu saskaņošanu savienojuma ietvaros;
- drošas pārraides nodrošināšanu, kļūdu atklāšanu un korekciju.

Svarīgi atzīmēt, ka adreses, ar ko strādā kanāla slāņa protokols, tiek izmantotas kadru piegādei tikai viena tīkla ietvaros, bet pakešu pārvietošanai starp tīkliem tiek jau izmantotas nākamā, tīkla slāņa adreses (Олифер and Олифер, 2010).

Piemēram, bezvadu tīklā Wi-Fi (IEEE 802.11) un Ethernet tīklā (IEEE 802.3) kanāla slānī katram datoram, vai precīzāk, katram tīkla adapterim ir unikāla aparāta adrese — MAC adrese, kas tiek izmantota izlases raidīšanai, apraidei un multiraidei lokālā tīkla ietvaros.

Noslodzes un trafika balansēšana OSI modeļa kanāla slānī notiek, piemēram, agregējot fiziskos kanālus starp divām komunikācijas ierīcēm vienā loģiskajā kanālā jeb, kā to bieži dēvē, trankā. IEEE 802.3ad specifikācijā aprakstītais agregētu kanālu izveides paņēmiens paredz loģiska porta izveides iespēju, apvienojot vairākus dažādām ierīcēm piederošus fiziskos portus, tādējādi nodrošinot bojājumpiecietību un palielinātu caurlaidspēju. Automātiskai informācijas apmaiņai par kāda fiziskā porta piederību noteiktam loģiskajam portam specifikācijā ir piedāvāts sakaru līniju agregēšanas vadības pakalpojumu protokols (Link Control Aggregation Protocol, LCAP).

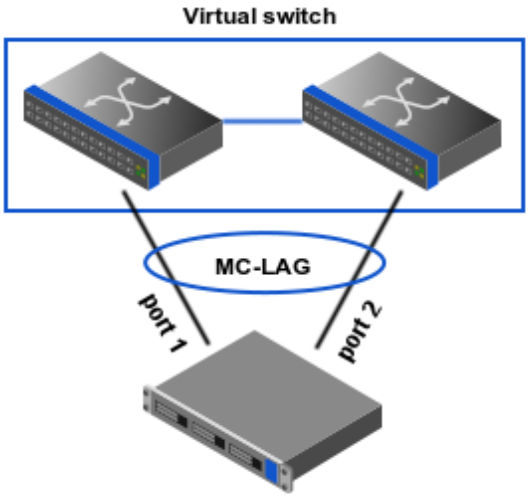
Sakaru līniju agregēšana tiek izmantota gan sakariem starp komutatoru portiem lokālajā tīklā, tā arī sakariem starp komutatoru un serveri, kā norādīts 1.1. attēlā.



1.1. att. Kanālu agregēšana starp komutatoru un serveri

Standarta piemērs kanālu agregēšanai ir tranks starp divām komunikācijas ierīcēm. Angļu valodā sarakstītajos avotos šim kanālu agregēšanas tipam ir nosaukums Link Aggregation Group (LAG). LAG tehnoloģija attīstīta tālāk MC-LAG (Multi-Chassis Link Aggregation Group) — kanālu agregācijas veidā, kas ļauj izmantot vairākas komunikācijas ierīces no katras tranka puses, ar mērķi nodrošināt redundanci gadījumā, ja viena ierīce iziet no ierindas. Un, kaut gan MC-LAG nav rūpniecības standarts, to savos produktos atbalsta visi lielie tīkla aprīkojuma ražotāji.

Izmantojot MC-LAG, sakaru līnijas, kas fiziski vpieslēgtas diviem vai vairākiem atšķirīgiem komutatoriem, veido vienotu agregētu kanālu ar trešo ierīci, kā norādīts 1.2. attēlā. Savukārt komutatori apvienojas vienā loģiskā jeb virtuālā komutatorā ar iekšēja protokola palīdzību.



1.2. att. Kanālu agregēšana starp diviem komutatoriem un serveri

Tīkla veiktspējas paaugstināšana, agregējot vairākas sakaru līnijas, dažos gadījumos ir efektīvāka par vienīgās sakaru līnijas nomainīšanu uz ātrāku.

Piemēram, izmantojot tīklā tikai komutatorus un serverus ar Gigabit Ethernet portiem, ātruma palielināšanai līdz 10 Gbit/s būs nepieciešams nomainīt komutatorus un tīkla adapterus. Tai pat laikā, ja ir brīvi Gigabit Ethernet porti, datu pārraides ātrumu var palielināt, piemēram, līdz 8 Gbit/s, apvienojot agregētā kanālā astoņus Gigabit Ethernet portus tais tīkla sektoros, kur tas patiešām nepieciešams.

1.4. Noslodzes balansēšana OSI tīkla modeļa tīkla slānī

Tīkla slānis (network layer) kalpo vienotas transporta sistēmas izveidei, kas apvieno vairākus tīklus, dēvētus par komplekso tīklu jeb internetu.

Lai tīkla slāņa protokoli varētu nogādāt paketes uz jebkuru kompleksā tīkla mezglu, šiem mezgliem nepieciešamas adreses, kas ir unikālas dotā kompleksā tīkla ietvaros. Tādas adreses sauc par tīkla jeb globālajām adresēm. Katram kompleksā tīkla mezglam, kam jāapmainās ar datiem ar citiem kompleksā tīkla mezgliem, līdztekus adresei, kas tam piešķirta kanāla slānī, jābūt tīkla adresei (Олифер and Олифер, 2010).

Piemēram, populārais TCP/IP steks paredz на IPv4 (32 bitu) vai IPv6 (128 bitu) adrešu izmantošanu tīkla slānī.

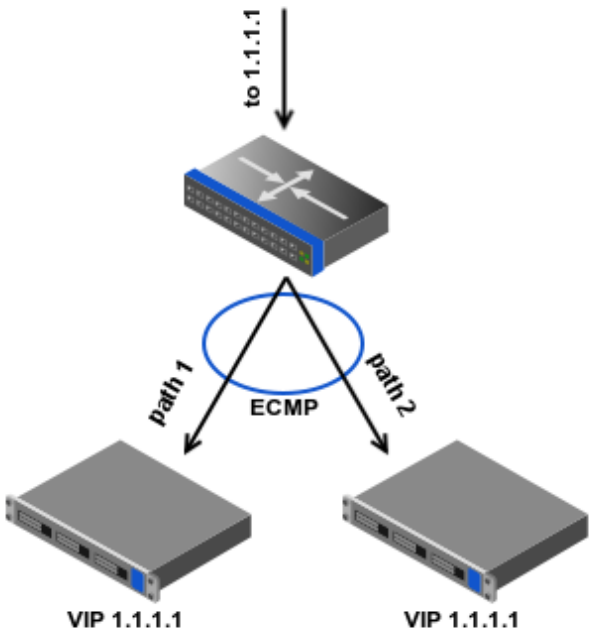
Svarīgs tīkla slāņa uzdevums ir maršruta noteikšana. Maršruts tiek aprakstīts ar tīklu (maršrutētāju) secību, caur kuriem jāiziet paketei, lai tā nokļūtu pie adresāta. Maršrutu

meklēšanu un fiksēšanu nodrošina maršrutēšanas protokoli. Atšķir protokolus, kas izpilda statisko un adaptīvo (dinamisko) maršrutēšanu. Pie statiskās maršrutēšanas visiem ierakstiem tabulā ir nemaināms, statisks statuss, bet ierakstus par maršrutiem sastāda un katra maršrutētāja atmiņā ievada manuāli. Pie dinamiskās maršrutēšanas visas tīkla konfigurācijas izmaiņas automātiski tiek atspoguļotas maršrutēšanas tabulās, pateicoties maršrutēšanas protokoliem.

Paši populārākie atvērtie dinamiskās maršrutēšanas protokoli ir maršruta informācijas protokols RIP (Routing Information Protocol), īsākā ceļa izvēles protokols OSPF (Open Shortest Path First) un robežu vārtejas protokols BGP (Border Gateway Protocol).

Gan statiskā maršrutēšana, gan dažādi dinamiskās maršrutēšanas protokoli ļauj vienlaikus izmantot vairākus vienādas vērtības maršrutus. Bet daži protokoli, piemēram, kompānijas Cisco izstrādātais uzlabotais iekšējās maršrutēšanas protokols EIGRP (Enhanced Interior Gateway Routing Protocol), ļauj vienlaikus izmantot arī vairākus dažādas vērtības maršrutus.

Noslodzes un trafika balansēšanai OSI modeļa tīkla slānī izmanto, piemēram, vienādas vērtības daudzstaru maršrutēšanas algoritmu (Equal-Cost Multi-Path, ECMP), kas aprakstīts RFC 2991. Šis algoritms paredz vairāku alternatīvu tīkla maršrutu vienlaicīgu izmantošanu. Tas dod virkni priekšrocību, tostarp bojājumpiecietību un palielinātu caurlaidspēju. Algoritms ECMP ļauj balansēt noslodzi un trafiku gan starp tīkla maršrutētājiem, gan starp maršrutētāju un serveriem, kā norādīts 1.3. attēlā.



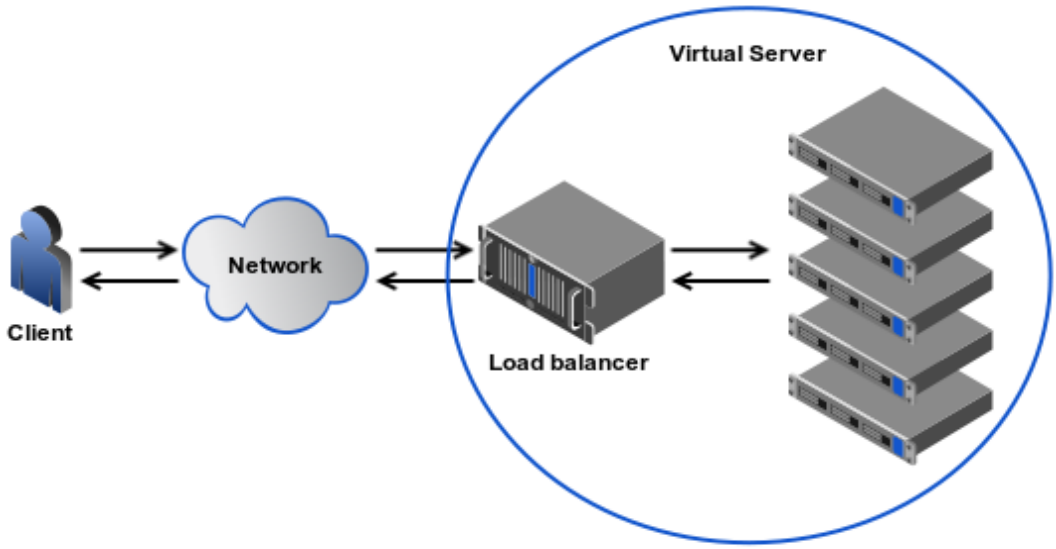
1.3. att. Daudzstaru maršrutēšana starp maršrutētāju un serveriem

1.5. Serveru noslodzes balansēšana

Iepriekš aplūkotās noslodzes un trafika balansēšanas tehnoloģijas var realizēt, izmantojot standarta tīkla komutatorus un maršrutētājus. Specializētie noslodzes balansētāji lēmuma pieņemšanai par trafika balansēšanu izmanto informāciju, sākot ar OSI modeļa transporta slāni, tāpēc izstrādātāji un ražotāji bieži dēvē tos par 4. — 7. slāņa komutatoriem (layer 4 — 7 switches). Tā kā noslodzes balansētāji būtībā ir intelektuālas tīkla ierīces, tie arī realizē OSI modeļa zemāko slāņu funkcijas un var tikt saukti par 2. — 7. slāņa komutatoriem.

Kā darbojas noslodzes balansētājs? No vienas puses, noslodzes balansētājs integrējas tīklā, realizējot kanāla un tīkla slāņu protokolus. No otras puses, noslodzes balansētājs realizē daudzus augstāku slāņu protokolus, tā ka, izmantojot augsta slāņa protokolu, spēj mijiedarboties ar serveriem. Tehniski noslodzes balansētājs ir saistošais posms starp tīklu un serveru fermu, kā norādīts 1.4. attēlā.

Serveru noslodzes balansēšana ir noslodzes un trafika sadalīšana starp daudziem serveriem. Tas ļauj mērogot datorsistēmu tālāk par viena servera iespēju robežām.



1.4. att. Serveru noslodzes balansēšana

Gala lietotājam noslodzes balansētājs un reālo serveru ferma ir pilnīgi caurspīdīgi, un lietotājs mijiedarbojas ar šādu datorsistēmu kā ar vienu augstas veiktspējas virtuālo serveri. Virtuālajam serverim ir jābūt IP adresei, lai tam varētu piekļūt klienti. Šo adresi sauc par virtuālo IP adresi jeb VIP (Virtual IP). VIP adrese tiek iestatīta noslodzes balansētajā un pārstāv visu serveru fermu. Īstenībā vienam noslodzes balansētājam var būt daudz VIP adrešu, t.i., tas var apkalpot daudzus virtuālos serverus vienlaicīgi.

Serveru noslodzes balansēšanas izmantošanai ir virkne priekšrocību:

- Datorsistēmas summārās veiktspējas palielināšanās, jo klientu pieprasījumi tiek sadalīti starp daudziem serveriem.
- Servisa pieejamības uzlabošanās, jo tiek nepārtraukti apsekots serveru stāvoklis un no balansēšanas tiek automātiski izslēgti nepieejamie serveri.
- Klientu apkalpošanas kvalitātes uzlabošanās veiktspējas palielināšanās un pieejamības uzlabošanās rezultātā.
- Datorsistēmas apkalpošanas vienkāršošanās, pateicoties iespējai operatīvi izslēgt no balansēšanas serverus apkopes vai remonta veikšanai.
- Drošības paaugstināšanās no globālā tīkla pieejamo serveru skaita un, attiecīgi, uzbrukumam pakļautās virsmas samazināšanās rezultātā.

Apskatīsim sīkāk trafika plūsmu starp klientu un serveri, izmantojot par piemēru transporta slāņa virtuālo serveri, kas apkalpo TCP savienojumus.

Iestatot TCP savienojumu klients nosūta uz noslodzes balansētāja VIP adresi TCP SYN paketi, kas satur šādu informāciju:

- nosūtītāja IP adrese;
- nosūtītāja TCP ports;
- saņēmēja IP adrese;
- saņēmēja TCP ports.

Uz šīs informācijas pamata noslodzes balansētājs nosaka, kuram virtuālajam serverim domāta saņemtā pakete, ar noslodzes balansēšanas algoritma palīdzību izvēlas no atbilstošās serveru fermas vienu reālu serveri pieprasījuma apstrādei un nogādā TCP paketi līdz šim reālajam serverim.

TCP pakete līdz reālajam serverim jānogādā tā, lai:

- pirmkārt, serveris uz to atbildētu;
- otrkārt, šīi atbilde tiktu nogādāta atpakaļ klientam;
- treškārt, lai klientam tas izskatītos kā viens savienojums.

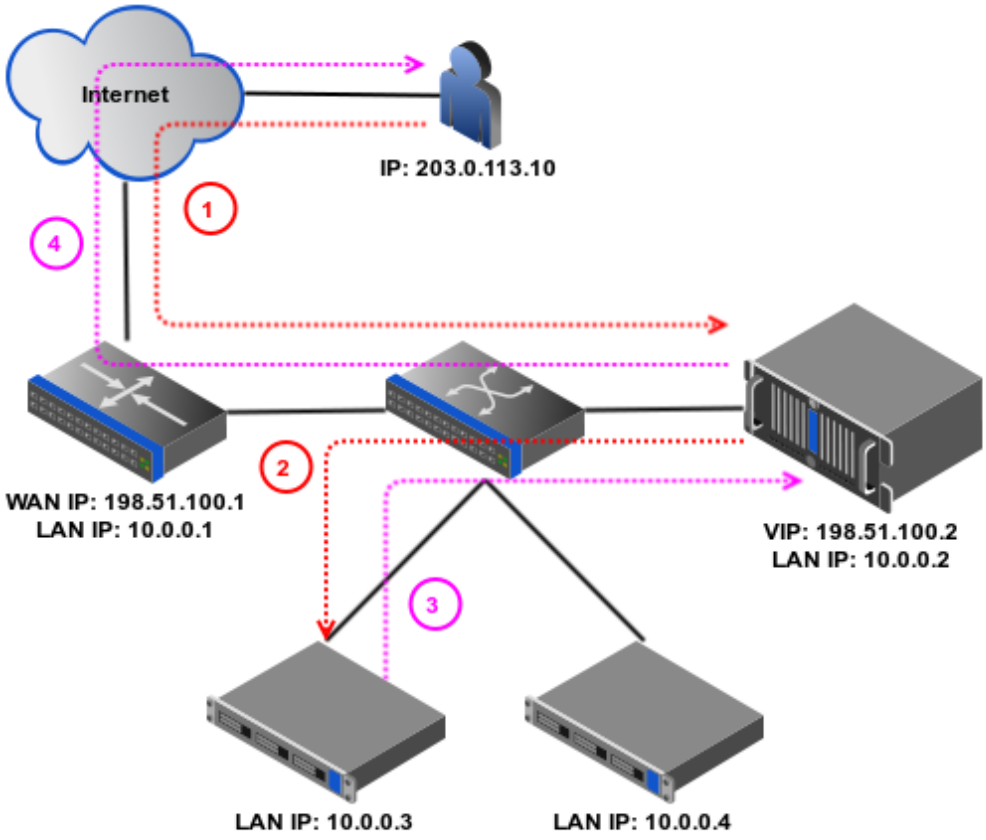
Apskatīsim dažas pakešu piegādes tehnoloģijas, kas tiek izmantotas serveru noslodzes balansēšanā.

1.5.1. Tīkla adrešu translācija

Tīkla adrešu translācija (Network Address Translation, NAT) — ir mūsdienās plaši pielietota tehnoloģija, kas paredz paketes virzīšanu globālajā tīklā, balstoties uz adresēm, kas atšķiras no tām, ko izmanto paketes maršrutēšanai lokālajā tīklā (Олифер and Олифер, 2010).

Izmantojot tīkla adrešu translācijas tehnoloģiju, noslodzes balansētājs saņem paketi no klienta, nomaina paketes galvenē nosūtītāja IP adresi ar savu IP adresi un ar maršrutēšanas starpniecību pārsūta paketi uz reālu serveri. Kad reālais serveris saņem šādu paketi, tam izskatās, it kā klients būtu noslodzes balansētājs. Reālais serveris atbild noslodzes balansētājam, kurš savukārt paketes galvenē nomaina atpakaļ savu IP adresi ar klienta IP adresi un atbild klientam ar maršrutēšanas starpniecību.

1.5. attēlā norādīta pakešu piegādes secīgu darbību shēma, izmantojot tīkla adrešu translācijas tehnoloģiju, bet 1.2. tabulā parādīts pakešu galveņu izmainīšanas rezultāts katrā darbībā.



1.5. att. Pakešu piegādes shēma, veicot tīkla adrešu translāciju

1.2. tabula

Izmaiņas pakešu galvenēs, veicot tīkla adrešu translāciju

Darbība	Nosūtītāja IP adrese	Saņēmēja IP adrese
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

Tīkla adrešu translācijas tehnoloģijas priekšrocība ir tajā, ka tā ļauj uzstādīt noslodzes balansētāju jebkurā tīkla vietā, bez jebkādiem ierobežojumiem tīkla topoloģijā. Vienīgais noteikums ir sakaru esamība tīkla slānī starp noslodzes balansētāju un reālo serveri. Vēl viena priekšrocība ir tā, ka nav nepieciešams izmainīt reālo serveru tīkla konfigurāciju.

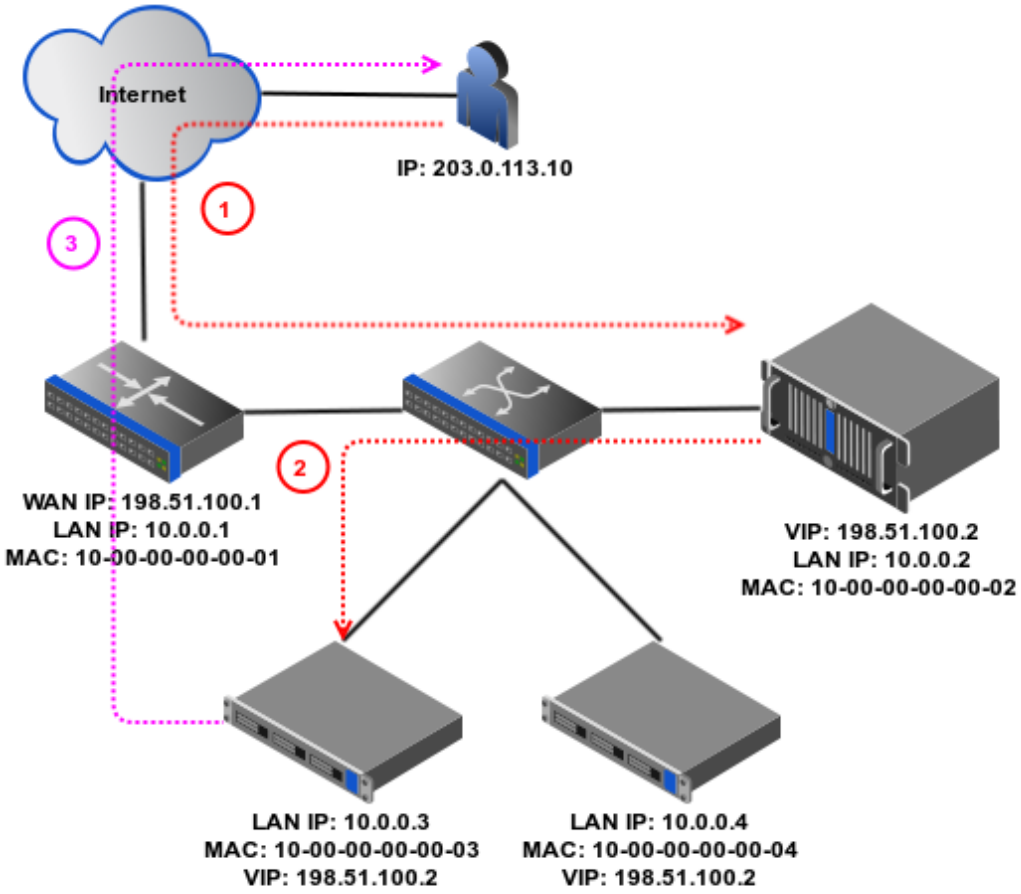
Tīkla adrešu translācijas tehnoloģijas trūkums ir tajā, ka reālie serveri neredz sākotnējā klienta IP adresi, jo noslodzes balansētājs izmaina to uz savu IP adresi.

1.5.2. Tiešā maršrutēšana

Tiešās maršrutēšanas (Direct Routing, DR) tehnoloģija, atšķirībā no tīkla adrešu translācijas, operē nevis ar IP adresēm, bet ar kanāla slāņa adresēm — MAC adresēm.

Izmantojot tiešās maršrutēšanas tehnoloģiju, noslodzes balansētājs saņem paketi no klienta, paketes galvenē nomaina saņēmēja MAC adresi ar reālā servera MAC adresi un ar komutācijas starpniecību pārsūta paketi uz reālu serveri. Kad reālais serveris saņem šādu paketi, tam izskatās, it kā klients būtu pārsūtījis tam paketi tieši, jo tīkla slāņa adreses nemainās. Lai reālais serveris atbildētu uz tādu paketi, tam atgriezeniskās cilpas interfeisā (loopback interface) jābūt norādītai VIP adresei, tāpat kā noslodzes balansētājā. Rezultātā reālais serveris atbild tieši klientam, ar maršrutēšanas starpniecību apejot noslodzes balansētāju. Šādu asimetrisku trafika maršrutu sauc par tiešo servera atgriešanu (Direct Server Return, DSR).

1.6. attēlā norādīta pakešu piegādes secīgu darbību shēma, izmantojot tiešās maršrutēšanas tehnoloģiju, bet 1.3. tabulā parādīts pakešu galveņu izmainīšanas rezultāts katrā darbībā.



1.6. att. Pakešu piegādes shēma, veicot tiešo maršrutēšanu

1.3. tabula

Izmaiņas pakešu galvenēs, veicot tiešo maršrutēšanu

Darbība	Nosūtītāja IP adrese	Saņēmēja IP adrese	Nosūtītāja MAC adrese	Saņēmēja MAC adrese
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

Tiešās maršrutēšanas tehnoloģijas priekšrocība ir tajā, ka tā ļauj panākt augstāku veiktspēju, ja noslodzes balansētājs ir šaurā vieta, jo tādā gadījumā tam jāapstrādā tikai ienākošais trafiks. Tiešā maršrutēšana vislabāk der tādām aplikācijām kā tīmekļa vietne, videostraumējums, failu lejupielāde un citām, kur atbildes pakešu skaits un lielums ievērojami pārsniedz pieprasījuma pakešu skaitu un lielumu.

Tiešās maršrutēšanas tehnoloģijas trūkums ir tajā, ka noslodzes balansētājam un reālajiem serveriem jāatrodas vienā apraides domēnā, jo pakete tiek pārsūtīta uz reālu serveri ar komutācijas starpniecību — tas būtiski ierobežo tīkla topoloģiju. Turklāt nepieciešams izmainīt

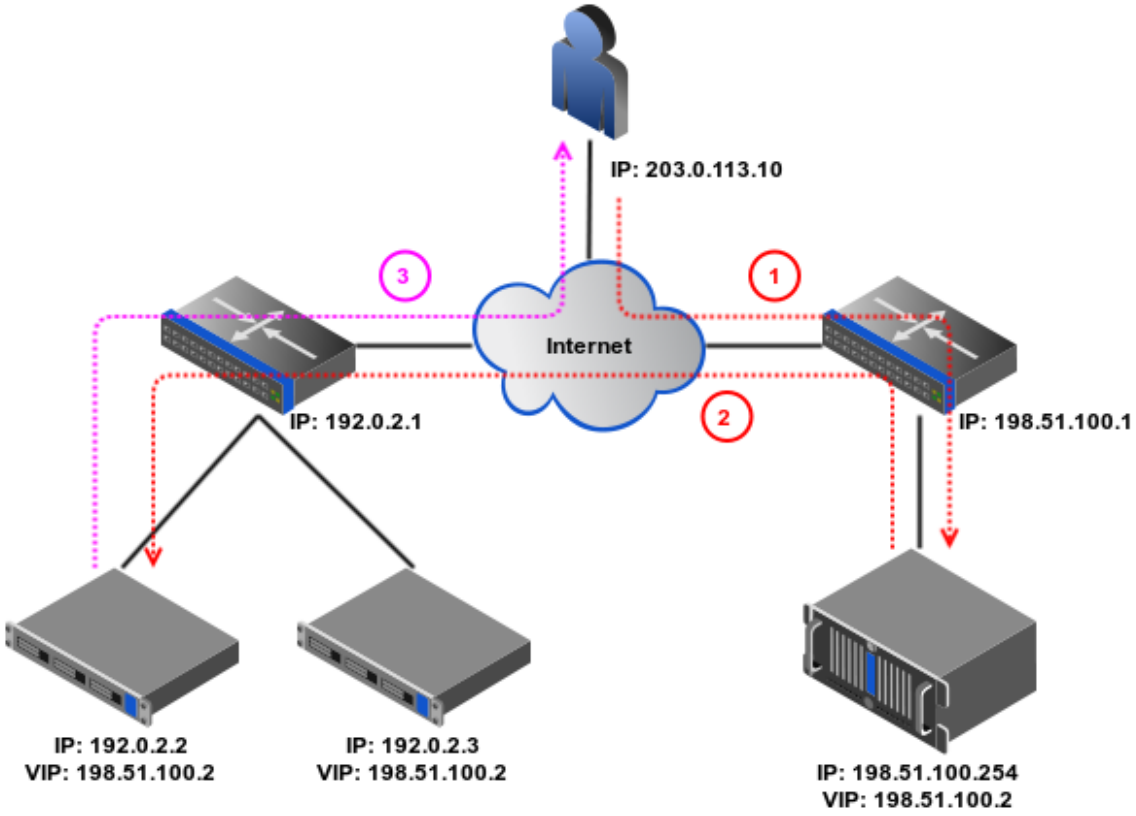
visu reālo serveru tīkla konfigurāciju un norādīt VIP adresi atgriezeniskās cilpas interfeisā, kas ne vienmēr ir iespējams.

1.5.3. Tunelēšana

Tunelēšana (tunneling) ir kopīgs nosaukums tehnoloģijām, ar kurām kāda divu apvienojamu tīklu vai mezglu protokola paketes uz tranzīttīkla robežas tiek iekapsulētas tranzīttīkla protokola paketēs un pārraidītas pa tranzīttīklu (Олифер and Олифер, 2010). Tunelēšanu plaši izmanto, piemēram, dažādu virtuālo privāto tīklu (VPN) uzbūvēšanā.

Izmantojot tunelēšanas tehnoloģiju, noslodzes balansētājs saņem no klienta paketi, pievieno tai papildu IP galveni (iekapsulē paketi) un ar maršrutēšanas starpniecību pārsūta iegūto IPIP paketi uz reālu serveri. Kad reālais serveris saņem šādu paketi, tas atmet papildu IP galveni (atkapsulē paketi), rezultātā iegūstot oriģinālo paketi, un reālajam serverim tas izskatās tā, it kā klients būtu tam pārsūtījis paketi tieši. Lai reālais serveris varētu atkapsulēt šādu paketi un atbildēt uz to, tam tuneļa interfeisā (tunnel interface) jābūt norādītai VIP adresei, tāpat kā noslodzes balansētājā. Rezultātā reālais serveris atbild klientam tieši ar maršrutēšanas starpniecību. Tāpat kā tiešās maršrutēšanas gadījumā, notiek tiešā servera atgriešana (DSR).

1.7. attēlā norādīta pakešu piegādes secīgu darbību shēma, izmantojot tunelēšanas tehnoloģiju, bet 1.4. tabulā parādīts pakešu galveņu izmainīšanas rezultāts katrā darbībā.



1.7. att. Pakešu piegādes shēma, veicot tunelēšanu

1.4. tabula

Izmaiņas pakešu galvenēs, veicot tunelēšanu

Darbība	Nosūtītāja IP adrese	Saņēmēja IP adrese	Nosūtītāja IP adrese	Saņēmēja IP adrese
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

Tunelēšanas tehnoloģijas priekšrocība ir tajā, ka tā, tāpat kā tiešā maršrutēšana, ļauj panākt augstāku veiktspēju, jo noslodzes balansētājam jāapstrādā tikai ienākošais trafiks. Vēl viena priekšrocība ir tā, ka, tāpat kā tīkla adrešu translācijas gadījumā, nav ierobežojumu tīkla topoloģijā.

Tunelēšanas tehnoloģijas tehnoloģijas trūkums ir tajā, ka iekapsulēšana pievieno papildmaksu 20 baitu (IP galvenes lielums) veidā katrai paketei no noslodzes balansētāja uz reālo serveri. Tāpat nepieciešams izmainīt visu reālo serveru tīkla konfigurāciju un norādīt VIP adresi tuneļa tīkla interfeisā, kas ne vienmēr ir iespējams.

1.6. Globāla serveru noslodzes balansēšana

Ar serveru noslodzes balansēšanas tehnoloģijas palīdzību var sadalīt noslodzi un trafiku serveru fermas starpā, taču nevar aizsargāties no datu apstrādes centra, kur izvietota šī ferma, pilnīgas nepieejamības, vai ietekmēt servera atbildes ātrumu, kad klients atrodas ievērojamā attālumā no datu apstrādes centra.

Globālās serveru noslodzes balansēšanas (GSLB) izmantošana ļauj ne tikai sadalīt noslodzi un trafiku starp vairākiem datu apstrādes centriem un automātiski izslēgt datu apstrādes centru no balansēšanas tā nepieejamības gadījumā, bet arī novirzīt klientu uz virtuālu serveri viņam tuvākajā reģionā. Tādējādi galvenie priekšnoteikumi GSLB izmantošanai ir augsta pieejamība un īss atbildes laiks.

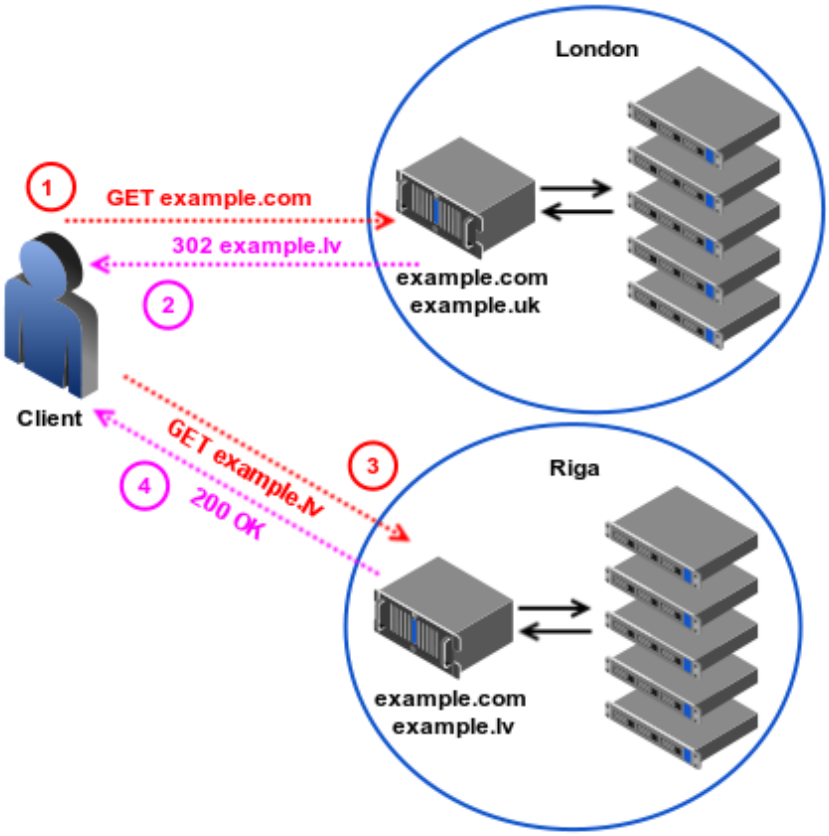
GSLB var realizēt dažādos veidos. Apskatīsim galvenās tehnoloģijas, kas tiek izmantotas globālajā serveru noslodzes balansēšanā.

1.6.1. GSLB, izmantojot HTTP virzienmaiņu

Viena no pašām vienkāršākajām tehnoloģijām globālās serveru noslodzes balansēšanas realizācijai, kuru var izmantot bez būtiskām izmaiņām esošajā tīkla arhitektūrā vai DNS konfigurācijā, ir GSLB, izmantojot HTTP virzienmaiņu. Šī tehnoloģija darbojas OSI modeļa lietojuma slānī.

RFC 2616 aprakstītais HTTP protokols ietver tīmekļa servera atbildes variantu, kurš satur 3xx virzienmaiņas HTTP kodus un jaunu URL. Šāda koda saņemšanas gadījumā tīmekļa pārlūkam automātiski jāpāriet uz jaunu URL.

1.8. attēlā norādīta GSLB darbības shēma, izmantojot HTTP virzienmaiņu. Piemēram, kad klients no Latvijas pieprasa vietni example.com, to apkalpojošais HTTP tīmekļa serveris pārvirza klientu uz tuvāko reģionālo vietni example.lv. Iznākumā klients veic pieprasījumu virtuālam serverim Rīgā un par atbildi saņem tīmekļa lapu. HTTP tīmekļa serveris var pieņemt lēmumu par virzienmaiņu, piemēram, vadoties pēc klienta IP adreses salīdzinājuma ar GeoIP datu bāzi vai IP tīklu adaptīvo sarakstu.



1.8. att. GSLB darbības shēma, izmantojot HTTP virzienmaiņu

GSLB, izmantojot HTTP virzienmaiņu, priekšrocība ir tajā, ka tiek samazināts servera atbildes laiks, jo klients tiek pārvirzīts uz tuvāko reģionālo vietni.

Šī risinājuma trūkums ir tajā, ka tiek atbalstīts tikai HTTP protokols un datu apstrādes centra nepieejamības gadījumā nenotiek rezervēšana, tāpēc HTTP virzienmaiņu izmanto kopā ar citām GSLB tehnoloģijām.

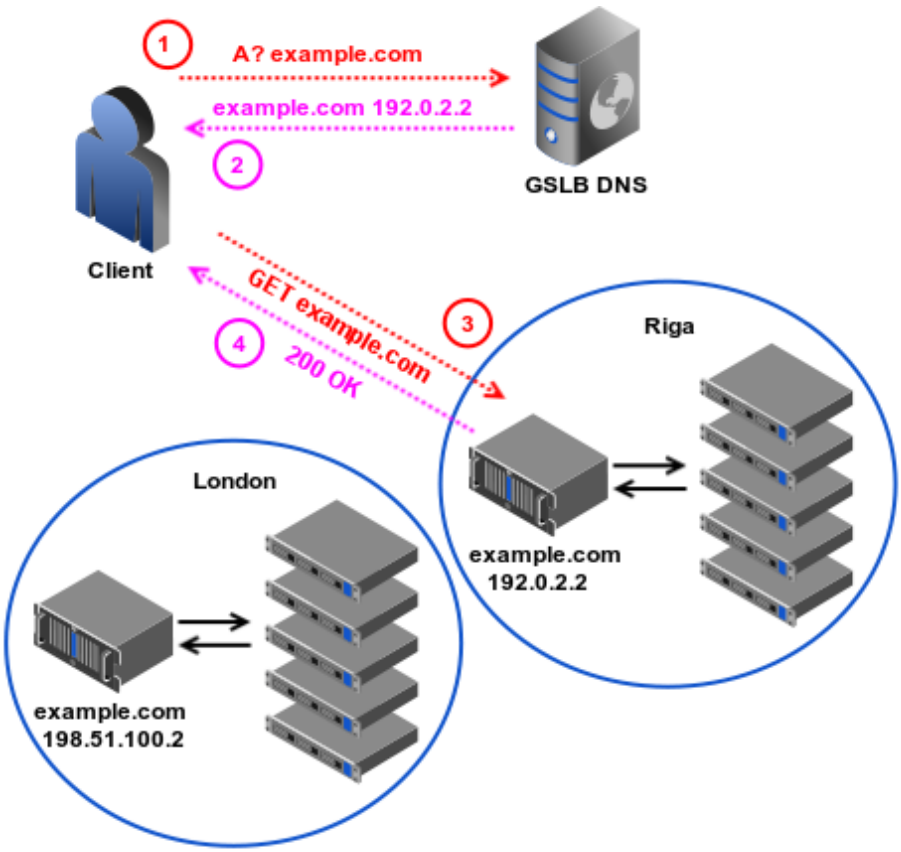
Neraugoties uz ierobežojumiem, HTTP virzienmaiņu izmanto daudzas augsti noslogotas un ģeogrāfiski sadalītas tīmekļa vietnes. Piemēram, kad tiek pieprasīta meklēšanas pakalpojuma Google galvenā lapa https://www.google.com, notiek automātiska HTTP virzienmaiņa uz tuvāko reģionālo vietni.

1.6.2. GSLB, izmantojot DNS

Pati populārākā GSLB tehnoloģija ir globālā serveru noslodzes balansēšana, izmantojot domēnu vārdu sistēmu (DNS GSLB). Tas galvenokārt ir saistīts ar pašas domēnu vārdu sistēmas plašo izplatību: šī sistēma visur tiek izmantota, piemēram, domēnu vārdu pārveidošanā IP adresēs un atpakaļ. RFC 1035 aprakstītais lietojuma slāņa DNS protokols ir viens no vecākajiem interneta protokoliem. Lai nodrošinātu DNS GSLB servisu, serverim jābūt vai nu autoritatīvam DNS serverim, vai DNS Proxy pirms autoritatīva servera.

Vērts atzīmēt, ka arī standarta DNS var izmantot noslodzes balansēšanai starp vairākiem serveriem. Šim nolūkam pietiek norādīt vairākas IP adreses domēna vārdam, un DNS tās izdos pēc apļa sistēmas (round robin) algoritma. Problēma ir tajā, ka DNS nezina, kuras no šīm IP adresēm patiešām strādā. Virtuālais serveris var būt pilnībā nepieejams, bet DNS turpinās izdot tā IP adresi, kas var izsaukt dažādas problēmas klientā. Turklāt standarta DNS neatrisina uzdevumu par klienta pārvirzīšanu uz tuvāko reģionālo vietni.

1.9. attēlā norādīta GSLB darbības shēma, izmantojot DNS. Piemēram, kad klients no Latvijas pieprasa vietnes example.com IP adresi autoritatīvam DNS GSLB serverim, tas izdod example.com tuvākās reģionālās vietne IP adresi. Galu galā klients veic pieprasījumu virtuālajam serverim Rīgā un par atbildi saņem tīmekļa lapu. DNS GSLB serveris var pieņemt lēmumu par pārvirzīšanu, piemēram, vadoties pēc klienta IP adreses salīdzinājuma ar GeoIP datu bāzi vai IP tīklu adaptīvo sarakstu. Rīgā esošās vietnes nepieejamības gadījumā DNS GSLB serveris sāk pēc klienta pieprasījuma izdot nākošo pieejamo vietni, atbilstoši savai balansēšanas politikai.



1.9. att. GSLB darbības shēma, izmantojot DNS

DNS GSLB tehnoloģijas galvenās priekšrocības ir realizācijas vienkāršība un universalitāte, jo to var izmantot kopā ar jebkuriem protokoliem, kas pieļauj komunikāciju ar domēnu vārdu starpniecību, un tādu protokolu ir patiešām ļoti daudz.

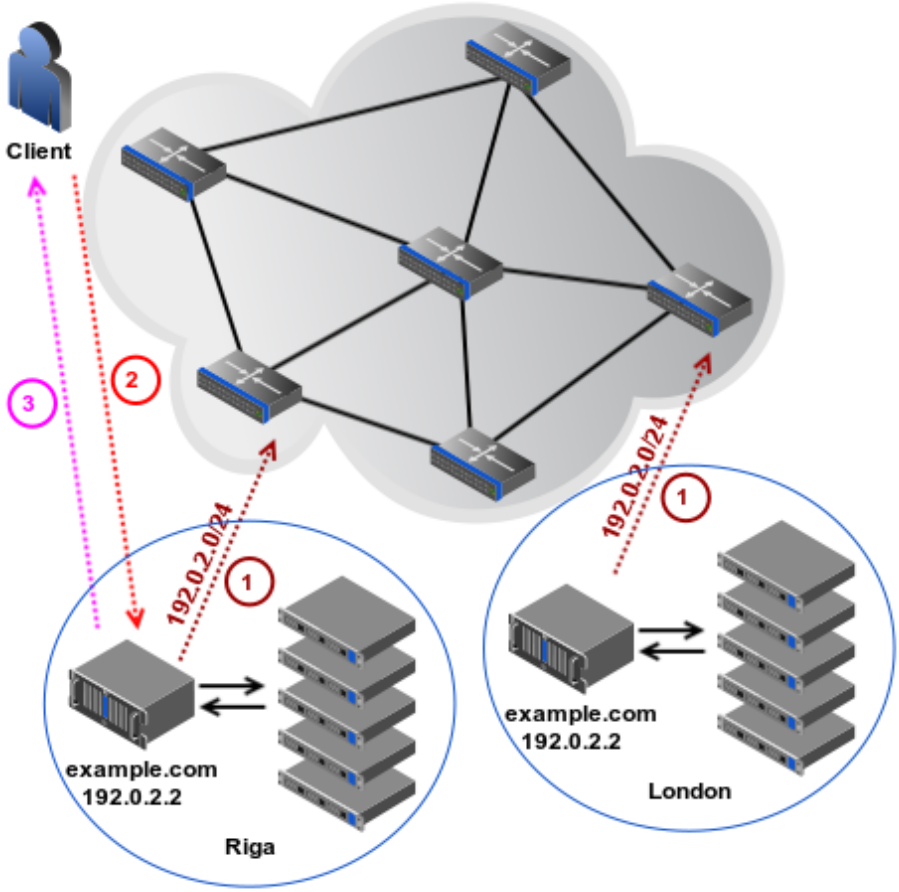
Pie DNS GSLB trūkumiem var pieskaitīt DNS saglabāšanu kešatmiņā (TTL) lokālo DNS serveru pusē un klientā: tās dēļ atteices gadījumā nenotiek momentāna pārslēgšana uz nākamo IP adresi. Cits trūkums ir tas, ka klients nevēršas tieši pie autoritatīva DNS GSLB servera, bet visi pieprasījumi notiek ar lokāla, datus kešatmiņā saglabājoša klienta DNS servera starpniecību. Rezultātā, ja klients izmanto DNS serveri, kas atrodas citā reģionā, tas var tikt nepareizi pārvirzīts nevis uz tuvāko serveri, bet uz citu reģionu.

Neraugoties uz visiem ierobežojumiem, DNS GSLB salīdzinājumā ar standarta DNS sniedz plašu spektru priekšrocību un uzlabojumu.

1.6.3. GSLB, izmantojot maršrutēšanas protokolu

Ne mazāk populāra ir GSLB tehnoloģija, izmantojot maršrutēšanas protokolu vai IP Anycast. Šai tīkla slāņa tehnoloģijā tiek izmantots labākā maršruta meklēšanas algoritms; maršruts tiek izvēlēts ar maršrutēšanas protokola starpniecību. Globālajā tīklā šim nolūkam izmanto, kā likums, BGP protokolu.

1.10. attēlā norādīta GSLB darbības shēma, izmantojot BGP. Piemēram, divi virtuālie serveri, Rīgā un Londonā, nepārtraukti piesaka pa BGP protokolu maršrutu līdz vienam un tam pašam IP tīklam. Klients vēršas šī tīkla IP adresē un, saskaņā ar izvēlēto optimālo maršrutu, trāpa uz tuvāko reģionālo virtuālo serveri Rīgā un saņem atbildi. Gadījumā, ja viens no datu apstrādes centriem izrādās nepieejams, tam kaimiņos esošais maršrutētājs pārstāj saņemt BGP pieteikumus, maršruts noveco un tiek automātiski izslēgts no maršrutēšanas, bet klients sāk vērsties pie nākošā tuvākā virtuālā servera.



1.10. att. GSLB darbības shēma, izmantojot BGP

GSLB tehnoloģijas, izmantojot maršrutēšanas protokolu, galvenā problēma ir pārrāvums jeb taimauts TCP savienojumā starp klientu un serveri pie katras maršruta pārslēgšanas uz alternatīvu ceļu. Šī iemesla pēc šī tehnoloģija vairāk der UDP protokolam, kurš funkcionē bez savienojuma izveides transporta slānī.

Šķiet, vislabākais piemērs šīs tehnoloģijas vispārējai izmantošanai ir publiskie DNS saknes serveri un tādu kompāniju–gigantu DNS serveri kā Amazon, Google, Yandex, kas uzstādīti visā pasaulē un no dažādiem planētas reģioniem pieejami pēc kopīgām IP adresēm.

1.7. Noslodzes un trafika balansēšanas atvērto tehnoloģiju salīdzinājums

Darba pirmās daļas noslēgumā veiksim tajā apskatīto noslodzes un trafika balansēšanas tehnoloģiju īsu salīdzinājumu. Katras tehnoloģijas plusi un mīnusi minēti atbilstošās sadaļās. Salīdzinājuma rezultāts parādīts 1.5. tabulā.

1.5. tabula

Noslodzes un trafika balansēšanas atvērto tehnoloģiju salīdzinājums

Tehnoloģija	OSI slānis	Pielietošanas joma	Mērogojamība	Atvērtās programmatūras piemēri
LAG	2	LAN	zema	Linux Kernel, Open vSwitch
ECMP	3	LAN / WAN	zema	Linux Kernel, Bird, Quagga
SLB NAT	4	LAN / WAN	vidēja	Keepalived, LVS, HAProxy, nginx
SLB DR	4	LAN / WAN	augsta	Keepalived, LVS
SLB IPIP	4	WAN	augsta	Keepalived, LVS
GSLB HTTP	7	LAN / WAN	augsta	Apache httpd, nginx
GSLB DNS	7	LAN / WAN	ļoti augsta	polaris-gslb
GSLB BGP	7	WAN	ļoti augsta	Linux Kernel, Bird, Quagga

2. APLIKĀCIJAS IZSTRĀDE GLOBĀLAI SERVERU NOSLODZES BALANSĒŠANAI

Šai sadaļā tiek secīgi apskatīti procesa posmi aplikācijas izstrādē globālai serveru noslodzes balansēšanai, izmantojot domēnu vārdu sistēmu (Domain Name System based Global Server Load Balancing, DNS GSLB).

Izstrādes procesā tiek noformētas un analizētas produktam izvirzītās prasības, izstrādāts vadības interfeisss, notiek aplikācijas projektēšana, tās realizācija un testēšana.

Projekta noslēguma stadijā tiek noformēta pakete, kas gatava uzstādīšanai serverī un satur aplikāciju, konfigurācijas failus, nepieciešamos moduļus, dokumentāciju aplikācijas noregulēšanai serverī un automātiskas konfigurācijas scenārijus.

Noslēgumā tiek veikts iegūtā programmatūras produkta novērtējums.

2.1. Prasību noformēšana

Šai posmā tiek vāktas un noformētas prasības, kas nosaka topošās programmatūras raksturlielumus un funkcijas. Šīs prasības nosaka pilnu izstrādes uzdevumu (Орлов, 2016).

2.1.1. Funkcionālās prasības

1. Aplikācijai jāatbild klientam uz DNS pieprasījumiem pēc atbilstoša protokola.
2. Aplikācijai jāpārbauda DNS servera izdoto gala serveru ierakstu pieejamība.
3. Aplikācijai jāatbild klientam uz HTTP pieprasījumiem pēc atbilstoša protokola.
4. Aplikācijai jādod administratoram iespēja kontrolēt konfigurāciju.
5. Aplikācijai jādodod operētājsistēmai iespēja palaist aplikāciju kā sistēmas pakalpojumu.

2.1.2. Nefunkcionālās prasības

1. Pielikumam jāfunkcionē operētājsistēmā Linux x86-64.
2. Pielikumam jāatbalsta vairākpavedienošana vai vairākuzdevumu režīms vienlaicīgam darbam ar daudziem klientiem.
3. Pielikumam jānodrošina aizsardzība pret nesankcionētu piekļuvi konfigurācijai.
4. Pielikumam jāatbalsta datu replikācija vairāku serveru klasterī.

2.1.3. Servera sistēmas prasības

Programmai jādarbojas tai piešķirtā serverī vai virtuālajā mašīnā ar šādu konfigurāciju:

1. Vismaz 1 savietojams procesors Intel x86-64.
2. Vismaz 1 GB operatīvās atmiņas.
3. Vismaz 4 GB cietais disks.
4. Tīkla karte.
5. Operētājsistēma Linux x86-64.

2.1.4. Klienta sistēmas prasības

Vadības tīmekļa interfeisam jādarbojas ar šādiem klientiem:

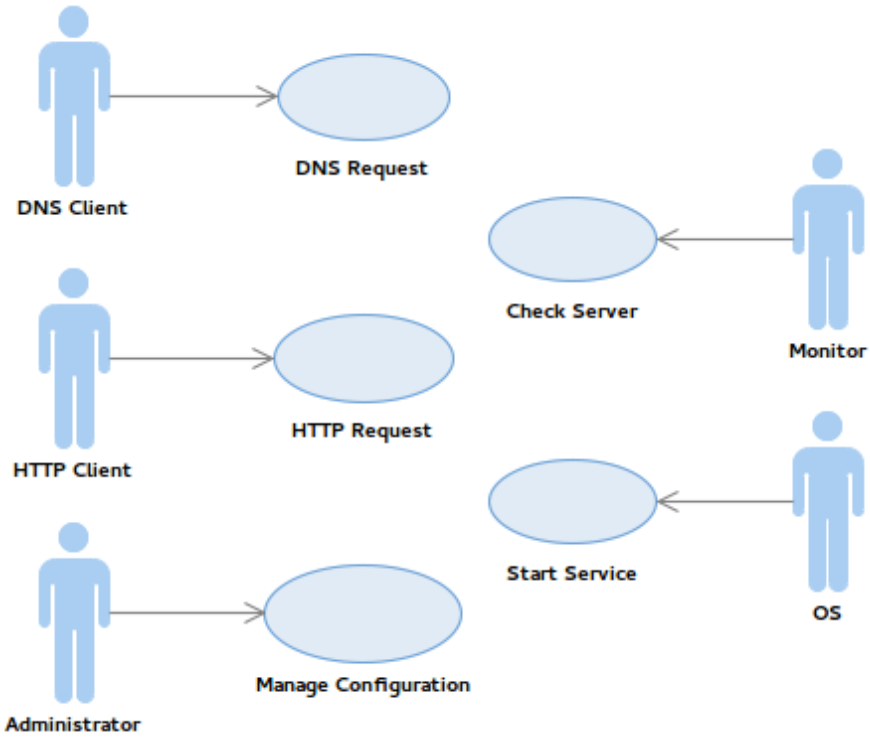
1. Tīmekļa pārlūks Chrome 30, Firefox 27 vai IE 11 ar JavaScript atbalstu.

2.2. Prasību analīze un izstrādāšana

Sistēmas modelēšana sākas ar diagrammas Use Case izveidi. Šo diagrammas tipuu veido aktieri, Use Case elementi un attiecības starp tiem (Орлов, 2016).

Pirmais diagrammas izveides solis sastāv no aktieru noteikšanas, kuros fiksētas ar sistēmu mijiedarbojošos ārējo objektu lomas. Aplūkojamā aplikācijas vidē figurēs pieci aktieri — DNS Client (DNS klients), HTTP Client (HTTP klients), Administrator (sistēmas administrators), Monitor (monitorings) un OS (operētājsistēma). Katrs aktieris izpilda atbilstošu Use Case elementuu.

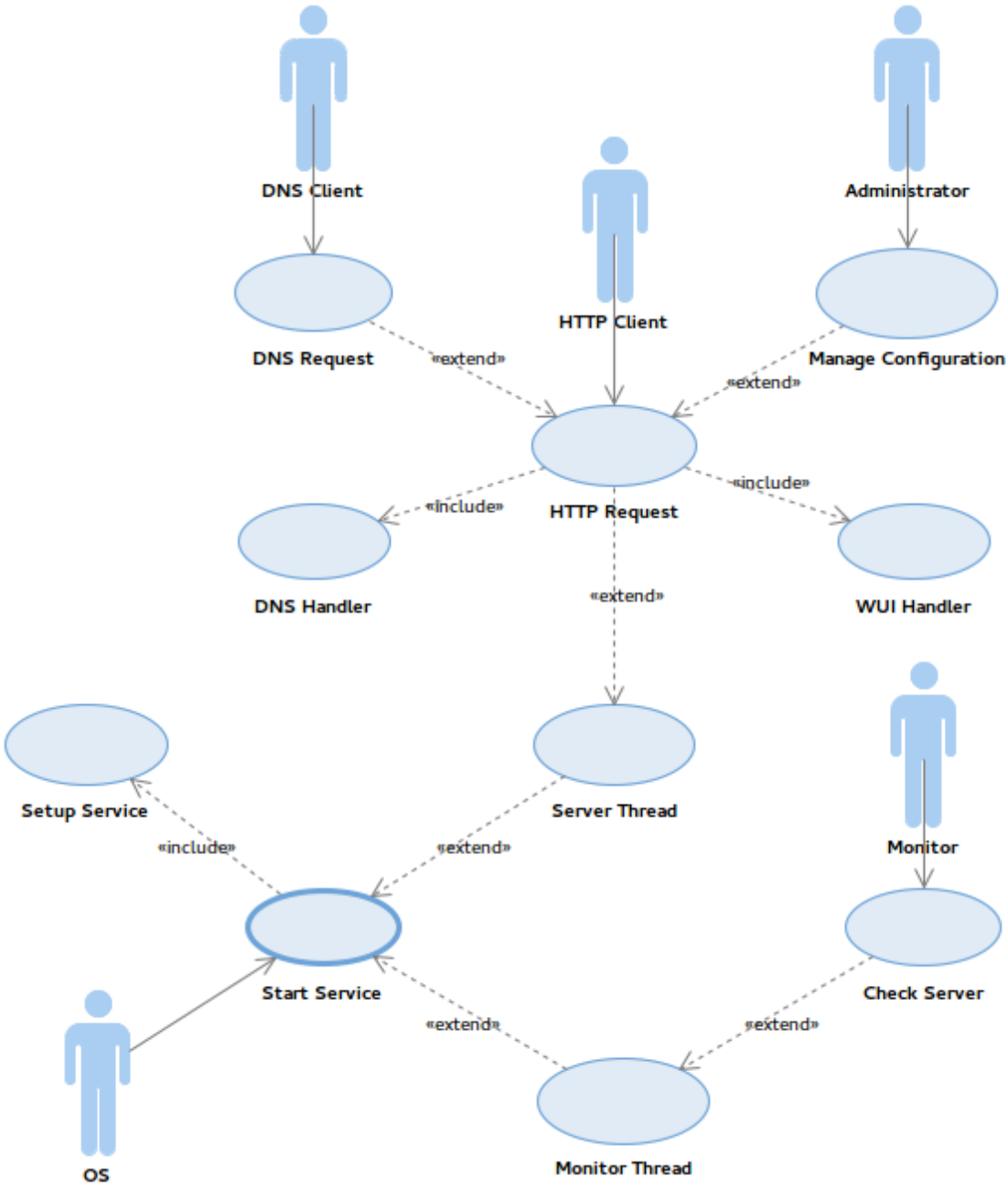
Sākotnējā Use Case diagramma norādīta 2.1. attēlā.



2.1. att. Sākotnējā Use Case diagramma

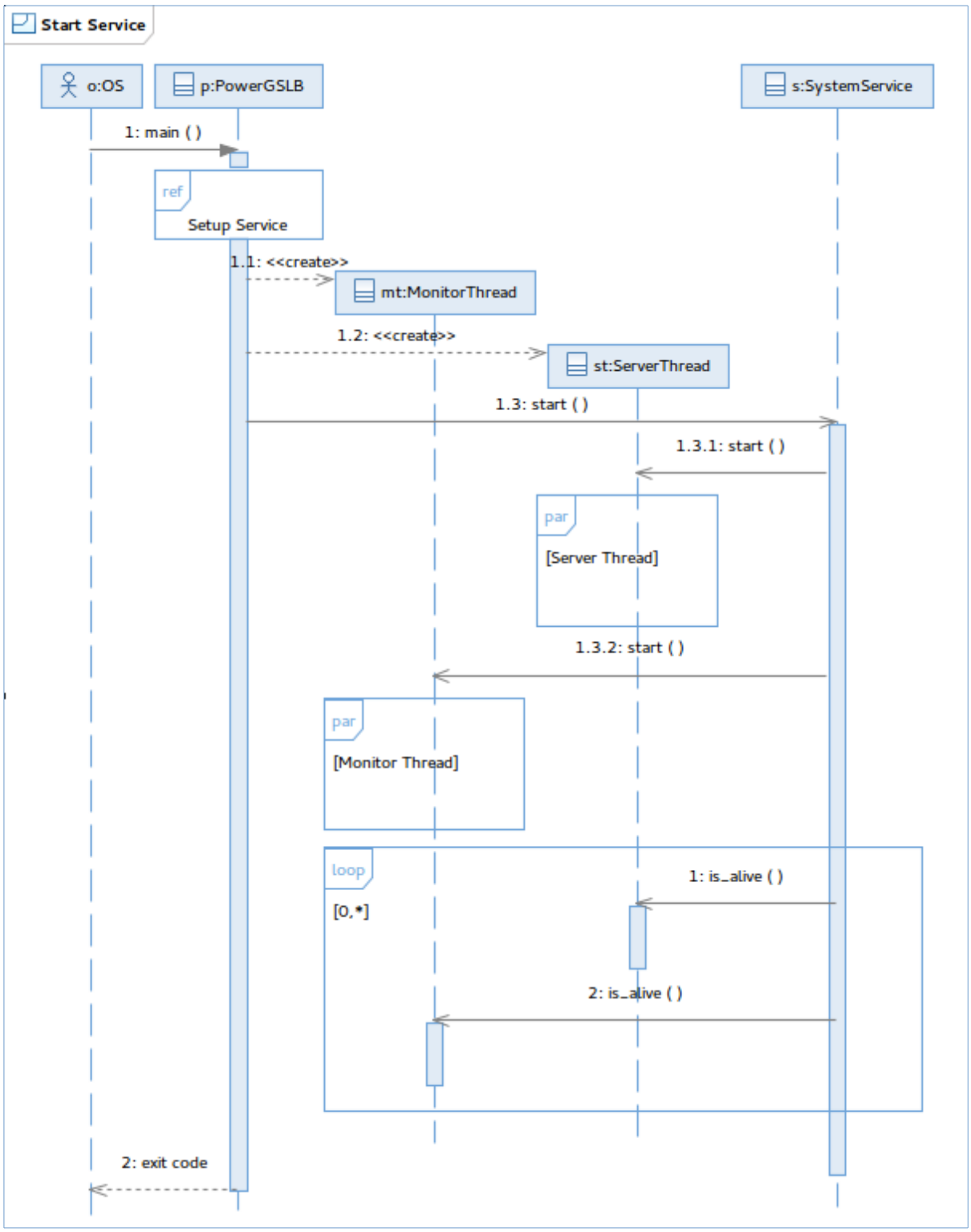
Šai darbā tiek aplūkota viena programma, kurā Use Case elementi nav neatkarīgi, bet paplašina cits citu vai iekļauj cits cita iespējas. Nepieciešams papildināt sākotnējo Use Case diagrammu, lai tajā būtu ņemtas vērā atkarības starp Use Case elementiem.

Programmas izpilde sākas ar tās palaidi, ko veic operētājsistēma. Tādējādi Use Case bāzes elements ir Start Service. 2.2 attēlā norādīta paplašināta Use Case diagramma, kurā ņemti vērā sakari starp Use Case elementiem. Use Case bāzes elements norādīts ar pustreknu līniju.

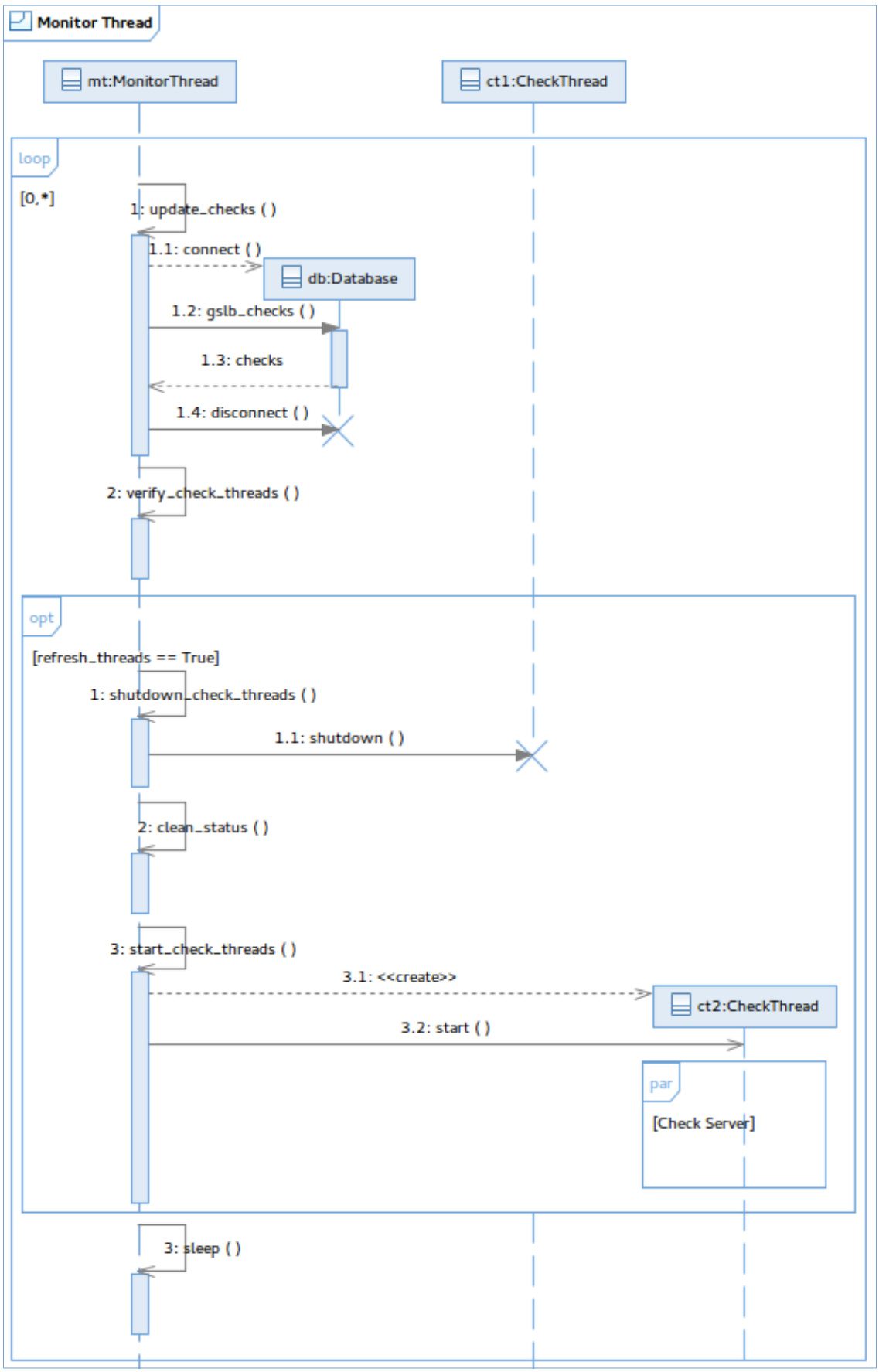


2.2. att. Paplašināta Use Case diagramma

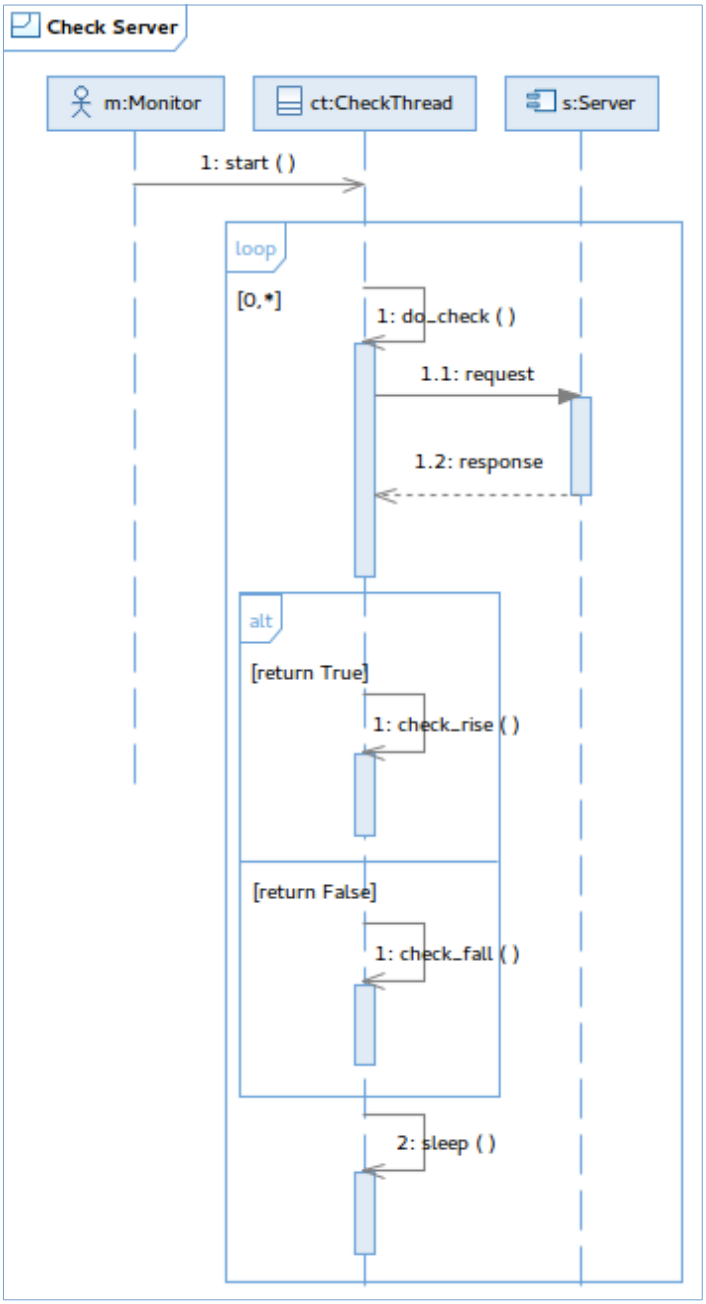
Nākamajā prasību izstrādāšanas solī nepieciešams izanalizēt Use Case diagrammu un izveidot secības diagrammas visiem tās elementiem. Attēlos 2.3. — 2.12. norādītas iegūtās secības diagrammas katram Use Case elementam.



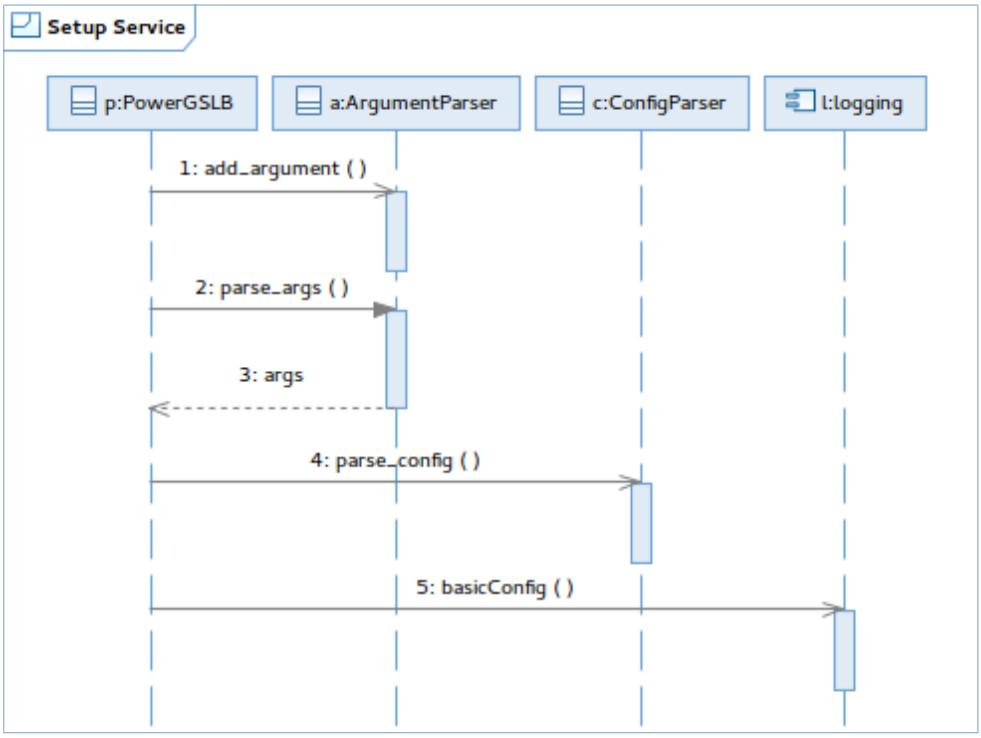
2.3. att. Secības diagramma bāzes elementam Use Case Start Service



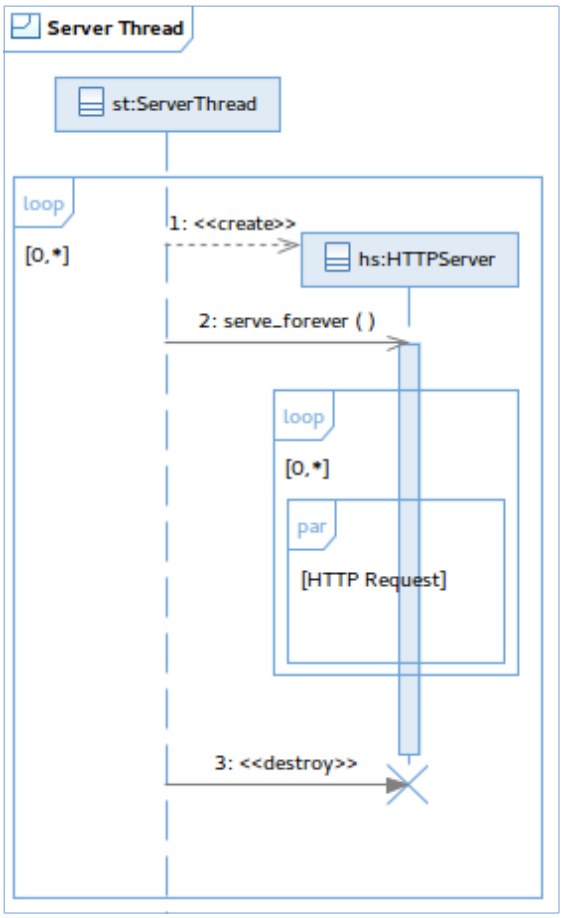
2.4. att. Secības diagramma paplašinājuma elementam Monitor Thread



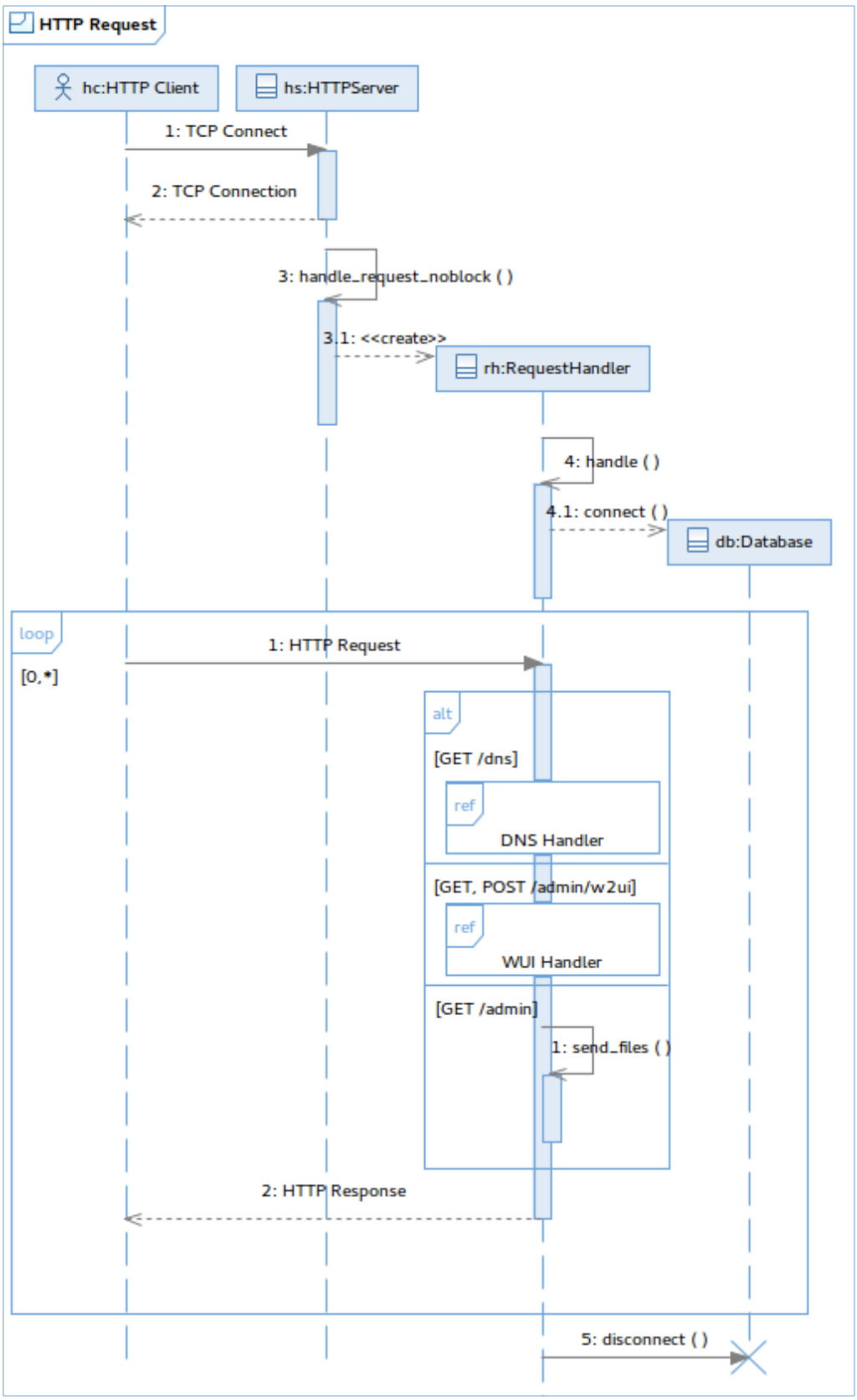
2.5. att. Secības diagramma paplašinājuma elementam Check Server



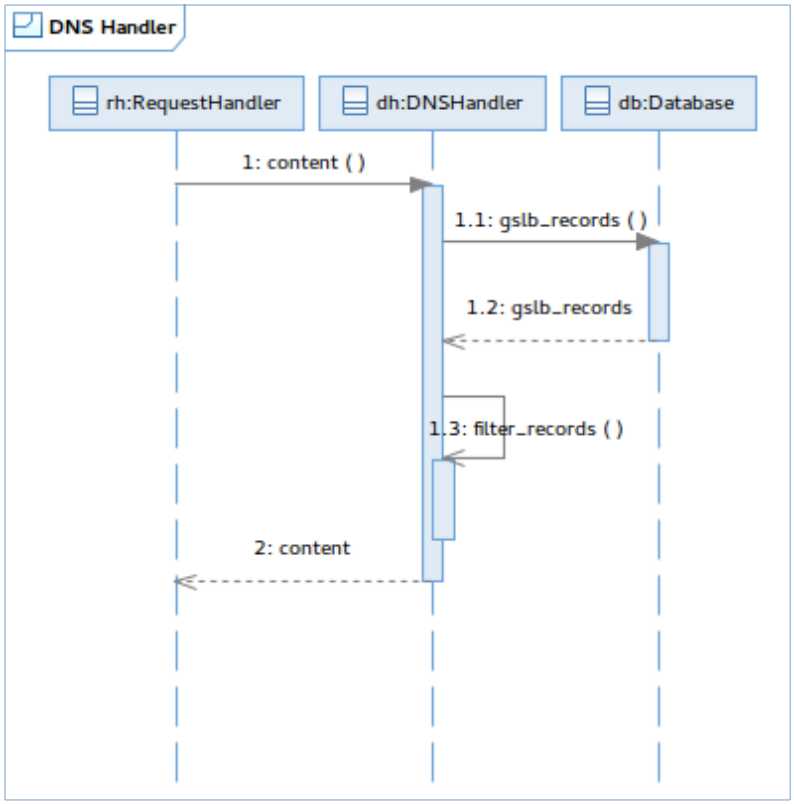
2.6. att. Secības diagramma iekļāvuma elementam Setup Service



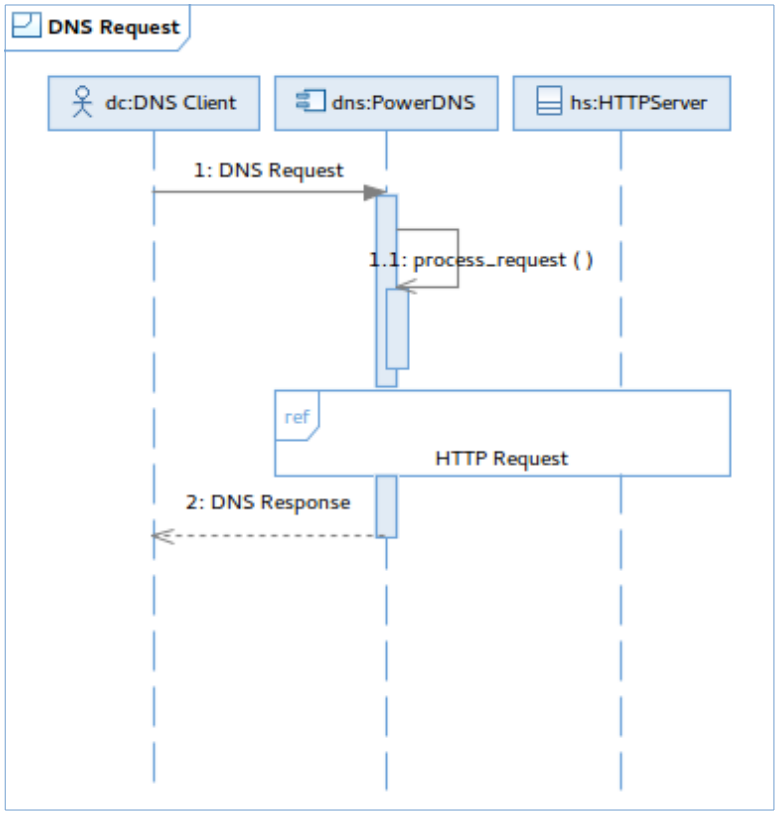
2.7. att. Secības diagramma paplašinājuma elementam Server Thread



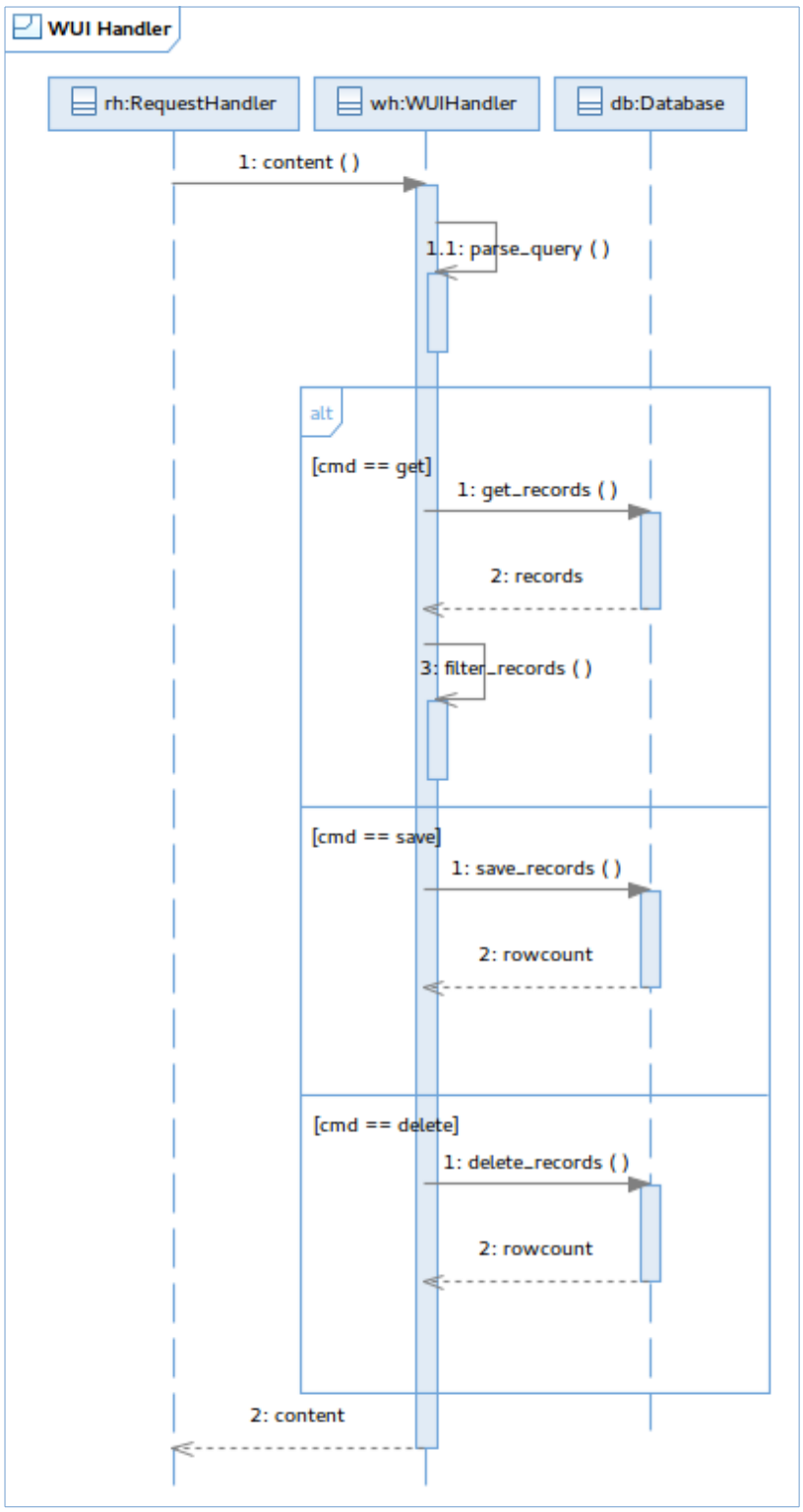
2.8. Secības diagramma paplašinājuma elementam HTTP Request



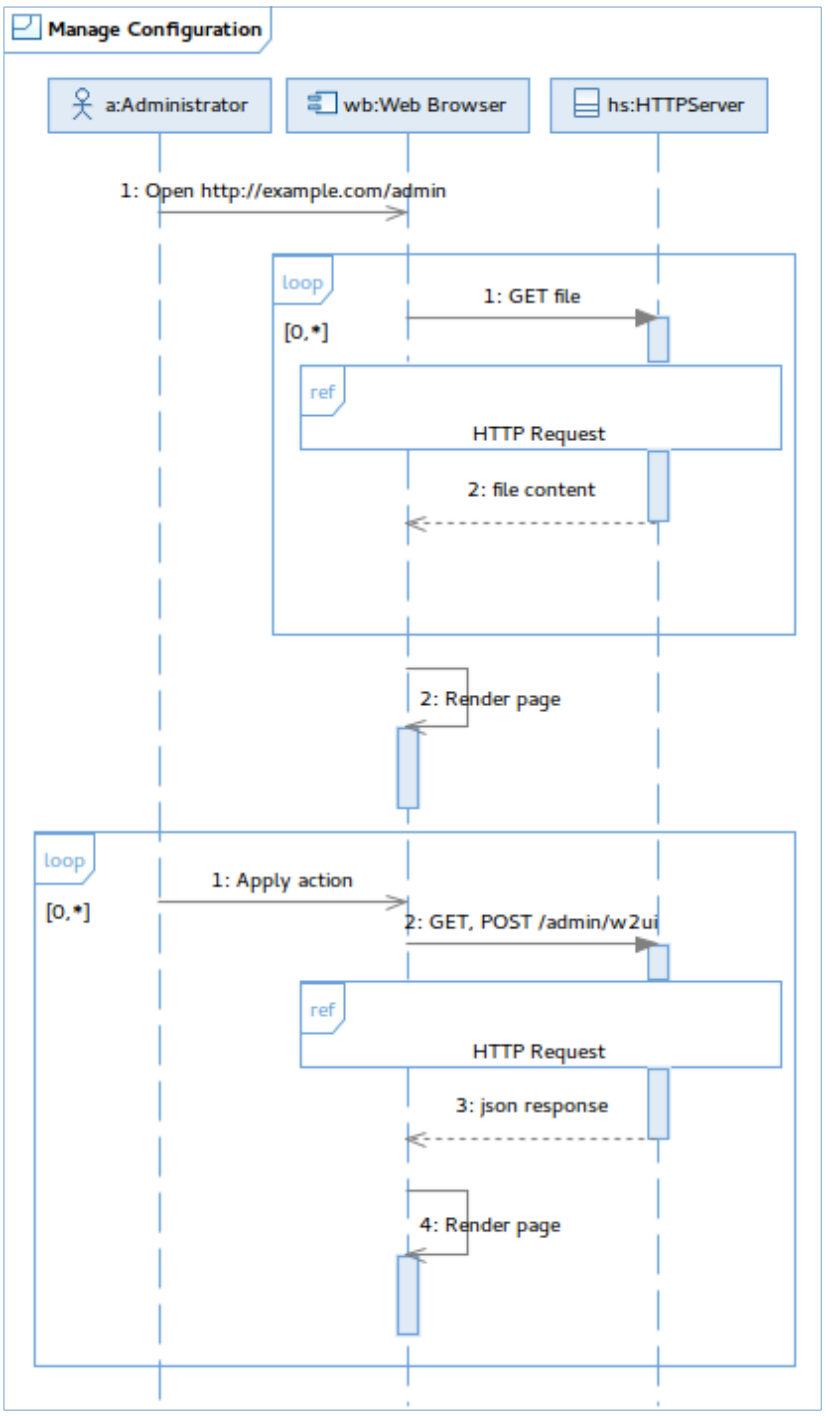
2.9. att. Secības diagramma iekļāvuma elementam DNS Handler



2.10. att. Secības diagramma paplašinājuma elementam DNS Request



2.11. att. Secības diagramma iekļāvuma elementam WUI Handler



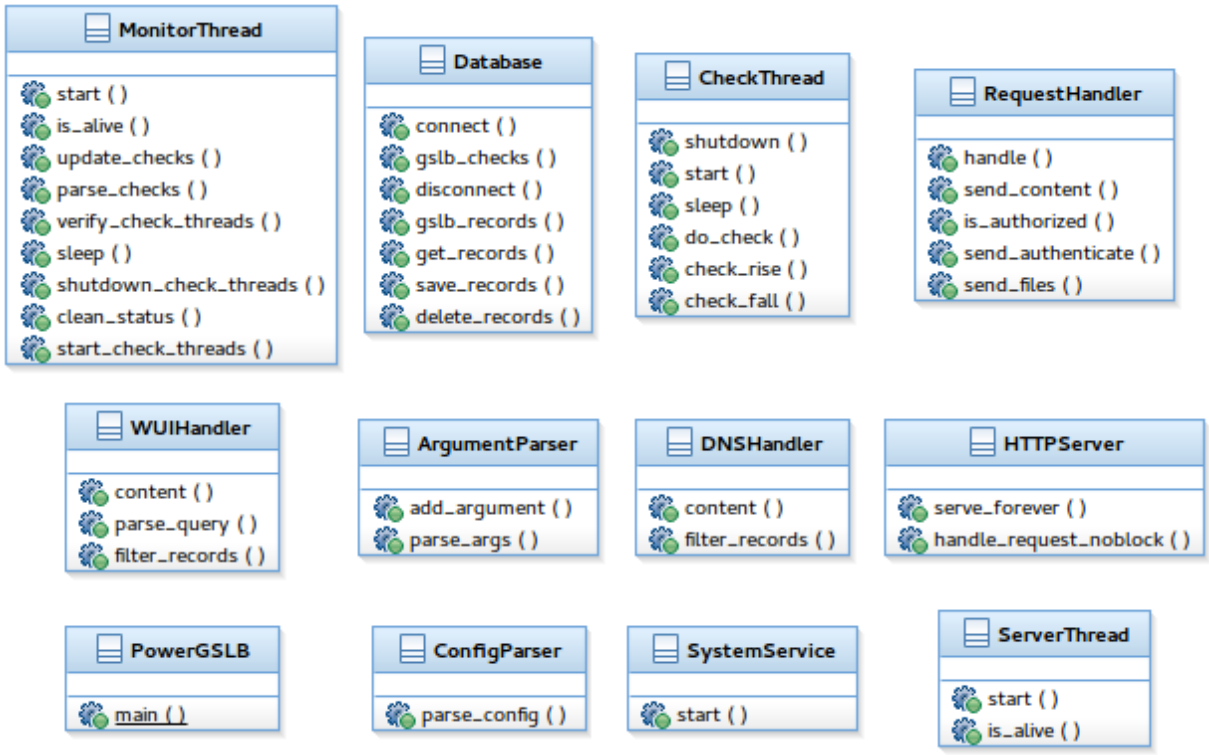
2.12. att. Secības diagramma paplašinājuma elementam Manage Configuration

2.3. Arhitektūras projektēšana

Arhitektūras noformēšana ir pirmais un pamata solis projektēšanas uzdevuma risināšanā, kurš veido fundamentu tādas programmsistēmas priekšstatīšanai, kura spēj izpildīt visu detalizēto prasību spektru (Орлов, 2016).

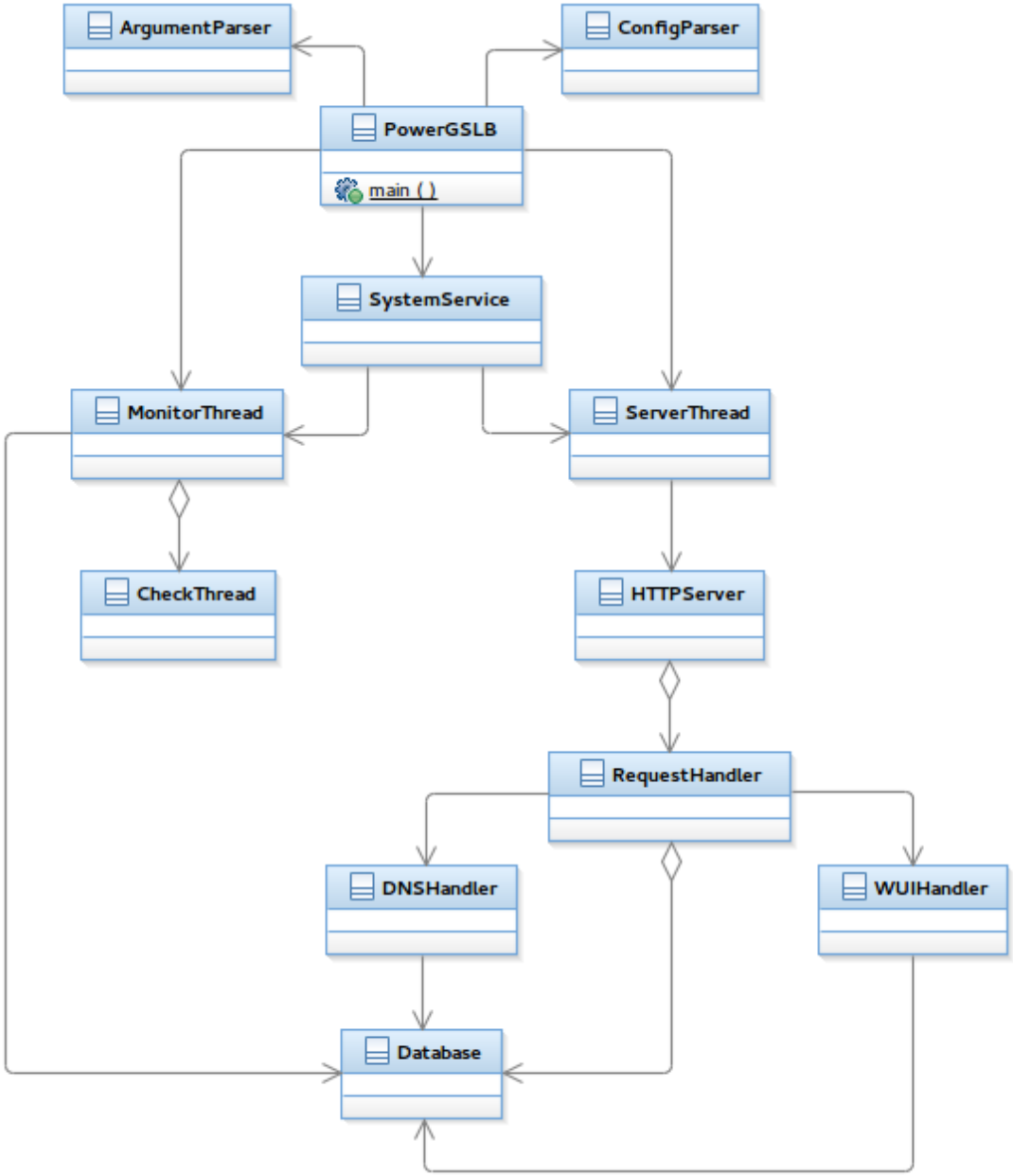
Klašu diagrammas tiek uzskatītas par pamatlīdzekli sistēmu struktūras priekšstatīšanai konstrukcijas bāzes bloku un to savstarpējo attiecību terminoloģijā (Орлов, 2016).

Visu secības diagrammu satura izpētes rezultātā ir noteikti sākuma klašu vārdi un operācijas. 2.13. attēlā norādītas iegūtās klases, neņemot vērā attiecības starp tām.



2.13. att. Sākotnējā klašu diagramma, neņemot vērā to attiecības

2.14. attēlā norādīta sākotnējā klašu diagramma, ņemot vērā attiecības starp tām. Īsuma dēļ diagrammā ir noslēptas klašu operācijas.



2.14. att. Sākotnējā klašu diagramma, ņemot vērā to attiecības

Klašu vizuālā modeļa loģiskās struktūras novērtēšanai tiek izmantots Čidambera-Kemerera metriku komplekts, kurš ietver 6 metrikas:

- WMC — svērtās metodes uz klasi;
- DIT — mantošanas koka augstums;
- NOC — bērnu skaits;
- CBO — saķere starp objektu klasēm;
- RFC — atsauce klasei;
- LCOM4 — saistīto komponentu daudzums klasē.

Vizuālā modeļa loģiskās struktūras kvalitātes sākotnējās novērtēšanas un sistēmas sākotnējās novērtēšanas rezultāti ir atainoti attiecīgi 2.1. un 2.2 tabulā.

2.1. tabula

Vizuālā modeļa loģiskās struktūras kvalitātes sākotnējais novērtējums

Klase	Metrikas					
	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
Vidējā vērtība	**3,42**	**0,00**	**0,00**	**1,33**	**5,83**	**1,00**

2.2. tabula

Sistēmas sākotnējais novērtējums

Metrika	Vērtība
DIT	0
NC	12
NOM	41
LOCΣ	0

No kopainas izdalās negatīvie klases MonitorThread kompleksuma un atsauces un klases PowerGSLB saķeres novērtējumi. Šīs klases ir potenciāls paaugstinātas bīstamības avots kvalitātei. Realizācijas un testēšanas procesā tām jāvelta īpaša uzmanība un jāpadomā par to modificēšanas iespējām.

Balstoties uz sākotnējo novērtējumu, var izdarīt secinājumu, ka sistēmas kvalitātes vidējās vērtības ir apmierinošas.

2.4. Datu bāzes projektēšana

Datu bāze (DB) ir kopīgi izmantojama stabilu datu kopa. Datu bāze ir vienota un liela datu glabātuve (Орлов, 2016).

Prasību analīzes posmā pieņemts lēmums izmantot aplikācijas konfigurācijas glabāšanai datu bāzi. Par datu bāzi var kalpot jebkura datu glabātuve, piemēram, fails, iebūvējamā bibliotēka vai DBMS.

DBMS izmantošanai šai projektā ir virkne priekšrocību:

1. Daudzpavedienu, vairākuzdevumu un vairāklietotāju režīma atbalsts piekļuvei pie datiem.
2. Aizsardzība pret nesankcionētu piekļuvi datu bāzei.
3. Datu replikācija vairāku serveru klasterī.
4. Transakcijas mehānismu atbalsts.
5. Datu integritātes kontrole.
6. Datu bāzes administrēšanas līdzekļu esamība.
7. Standartu atbalsts, piemēram, SQL standarta.

DBMS izmantošanas trūkumi:

1. Sarežģīta uzturēšana.
2. DBMS izmaksas (ieskaitot projektēšanas izmaksas).
3. Sistēmas resursu patēriņš.

DBMS izmantošanas gadījumā izstrādātājam nav nepieciešams saviem spēkiem realizēt datu glabātuvi ar paralēlu piekļuvi tai no dažādiem programmas pavedieniem, nav jārūpējas par datu aizsardzību un datu replikāciju vairāku serveru klasterī. Šos visus uzdevumus pārņem DBMS. Respektīvi, acīmredzams pluss DBMS izmantošanai šajā projektā ir tas, ka tiek nodrošināta vairāku nefunkcionālo prasību izpilde un tiek vienkāršota izstrāde.

Salīdzinot DBMS izmantošanas priekšrocības un trūkumus un ņemot vērā, ka aplikācijas izstrādātājam ir pieredze darbā ar MySQL, pieņemts lēmums izmantot projektā DBMS MySQL. Šai gadījumā darba pieredze ar MySQL neitralizē tādus trūkumus kā sarežģīta uzturēšana un projektēšanas izmaksas, procesa patērētie MySQL resursi nepārsniedz jau noteiktās minimālās prasības serverim, bet DBMS izmaksas nav aktuālas, jo DBMS MySQL tiek izplatīta bez maksas.

Šai aplikācijas vidē datu bāzē nepieciešams glabāt šādus galvenos atribūtus:

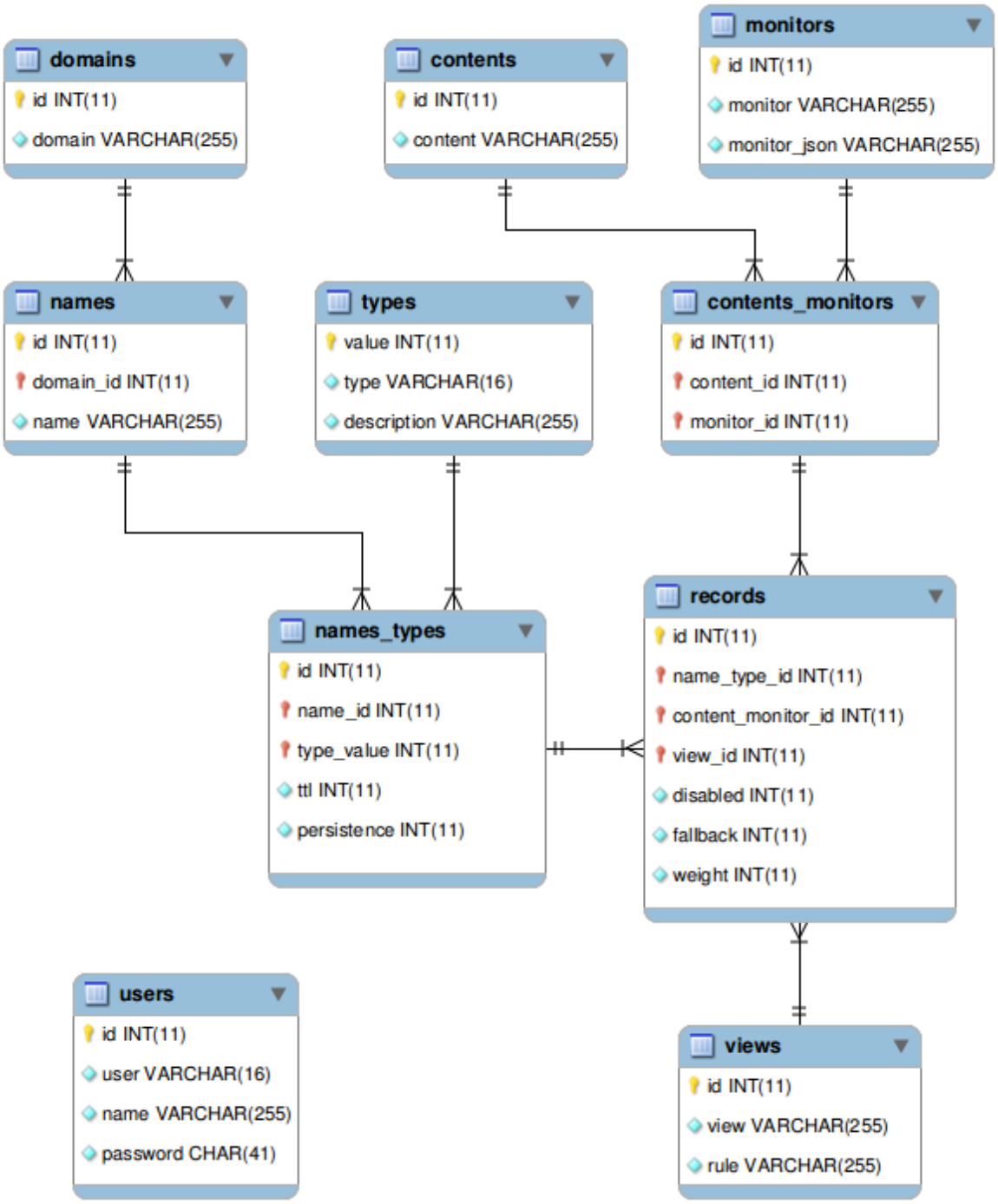
- DNS domēna vārdu — atribūts domain;
- DNS ieraksta vārdu — atribūts name;
- DNS ieraksta tipu — atribūts type;
- DNS ieraksta saturu — atribūts content;
- DNS ieraksta saglabāšanas kešatmiņā laiku — atribūts ttl;
- pazīmi izslēgšanai no balansēšanas — atribūts disabled;
- rezerves servera pazīmi — atribūts fallback;
- pazīmi klienta piesaistei pie servera — atribūts persistence;
- DNS ieraksta svaru — atribūts weight;
- monitora konfigurāciju — atribūts monitor;
- DNS veida konfigurāciju — atribūts view;
- lietotāja vārdu — atribūts username;
- lietotāja kontu — atribūts user;
- lietotāja paroli — atribūts password.

Nepieciešams izveidot normalizētu datu bāzes modeli. Normalizācija nodrošina DB struktūras optimizāciju. Tā ļauj novērst redundanci datu kopās (Орлов, 2016).

Normalizācijas noteikumi ir noformēti normālformu veidā. Pirmā normālforma (1NF) pieprasa, lai visu datu elementu vērtības kolonnās būtu atomāras. Otrā normālforma (2NF) pieprasa, lai katra neatslēgas kolonna būtu pilnīgi atkarīga no primārās atslēgas. Trešā normālforma (3NF) pieprasa, lai visas neatslēgas kolonnas (atribūti) būtu savstarpēji neatkarīgas un pilnīgi atkarīgas no primārās atslēgas (Орлов, 2016).

Datu bāzes modelēšana veikta specializētā modelēšanas vidē MySQL Workbench. Modelēšanas rezultātā iegūta optimāla DB struktūra trešajā normālformā. Iegūtajā DB nav datu redundances.

2.15. attēlā norādīts iegūtais DB ERR (enhanced entity–relationship) modelis Crow's Foot notācija. Īsuma dēļ diagrammā ir noslēpti indeksi.



2.15. att. Datu bāzes modelis Crow's Foot notācija

2.5. Realizācija

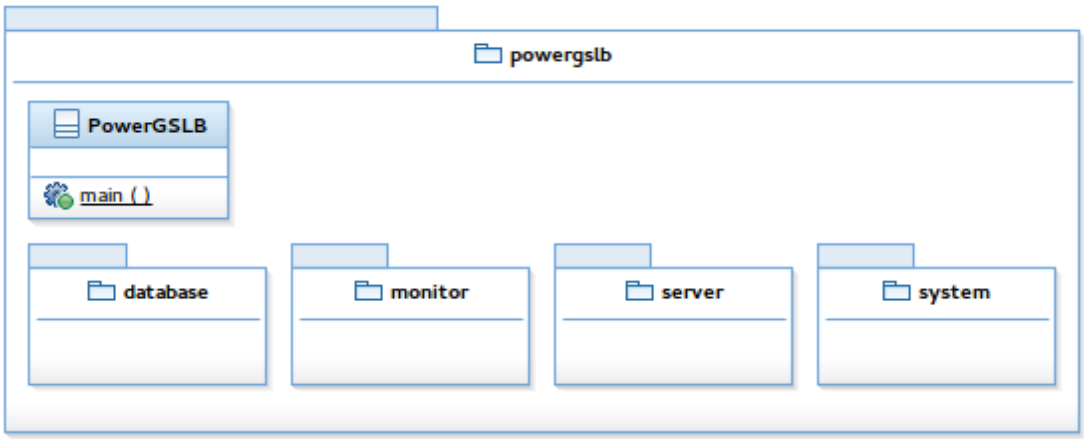
Ņemot vērā aplikācijas realizācijas laiika resursu ierobežojumus un izstrādātāja pieredzi, par realizācijas valodu izvēlēta universālā uz objektiem orientētā programmēšanas valoda Python.

Realizācijas procesā izpētītas valodas Python standarta un ārējo bibliotēku iespējas, noteikti nepieciešamie atribūti un precizētas operāciju klases, noteiktas klašu uzvedības kopīgās iezīmes un izveidoti mantošanas koki, klases iepakotas paketēs, izveidotas pakešu diagrammas un klašu gala diagramma, veikta aplikācijas kodēšana un konfigurēšana.

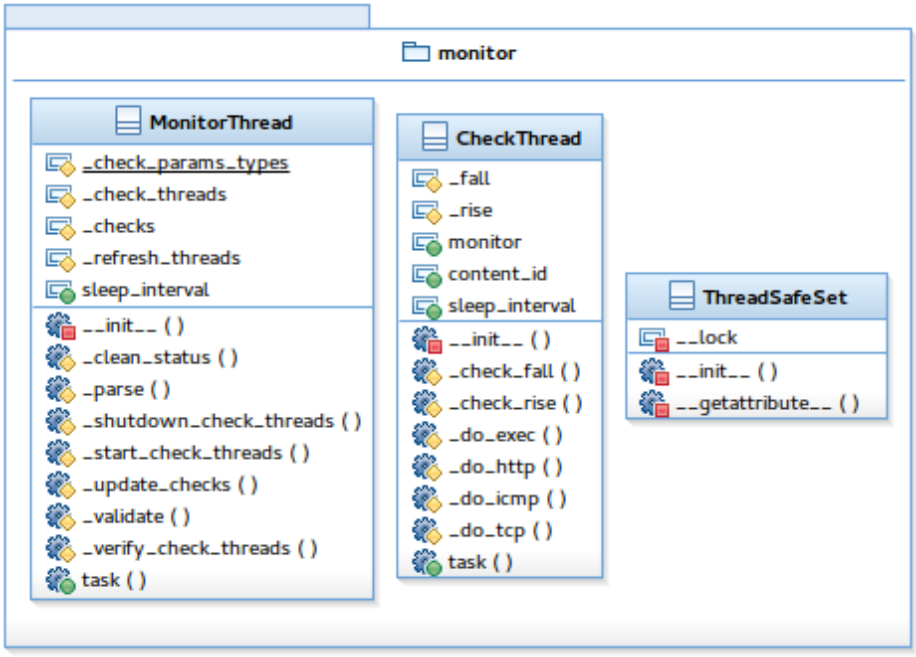
Realizācijas pamatlīdzekļi ar norādītu programmatūras versiju:

- IBM Rational Software Architect 9.5;
- programmēšanas valodas:
  - Python 2.7
  - JavaScript 1.7
- izstrādes vide PyCharm IDE 2016.1;
- modelēšanas vide MySQL Workbench 6.3;
- biroja pakete LibreOffice 4.3;
- attēlu redaktors GIMP 2.8;
- versiju kontroles sistēma Git 1.8;
- operētājsistēma Linux CentOS 7.2.

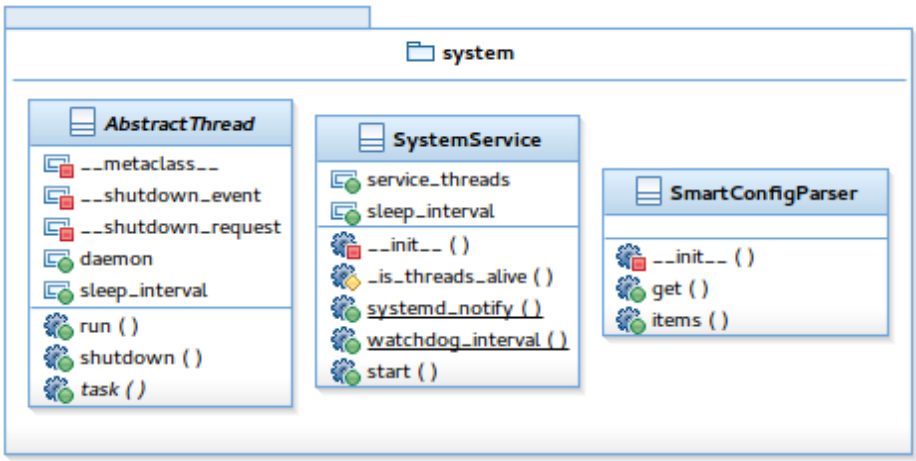
2.16. — 2.20. attēlā norādītas iegūtās diagrammas, neņemot vērā attiecības un ārējās attiecībā pret galveno programmu paketes un klases.



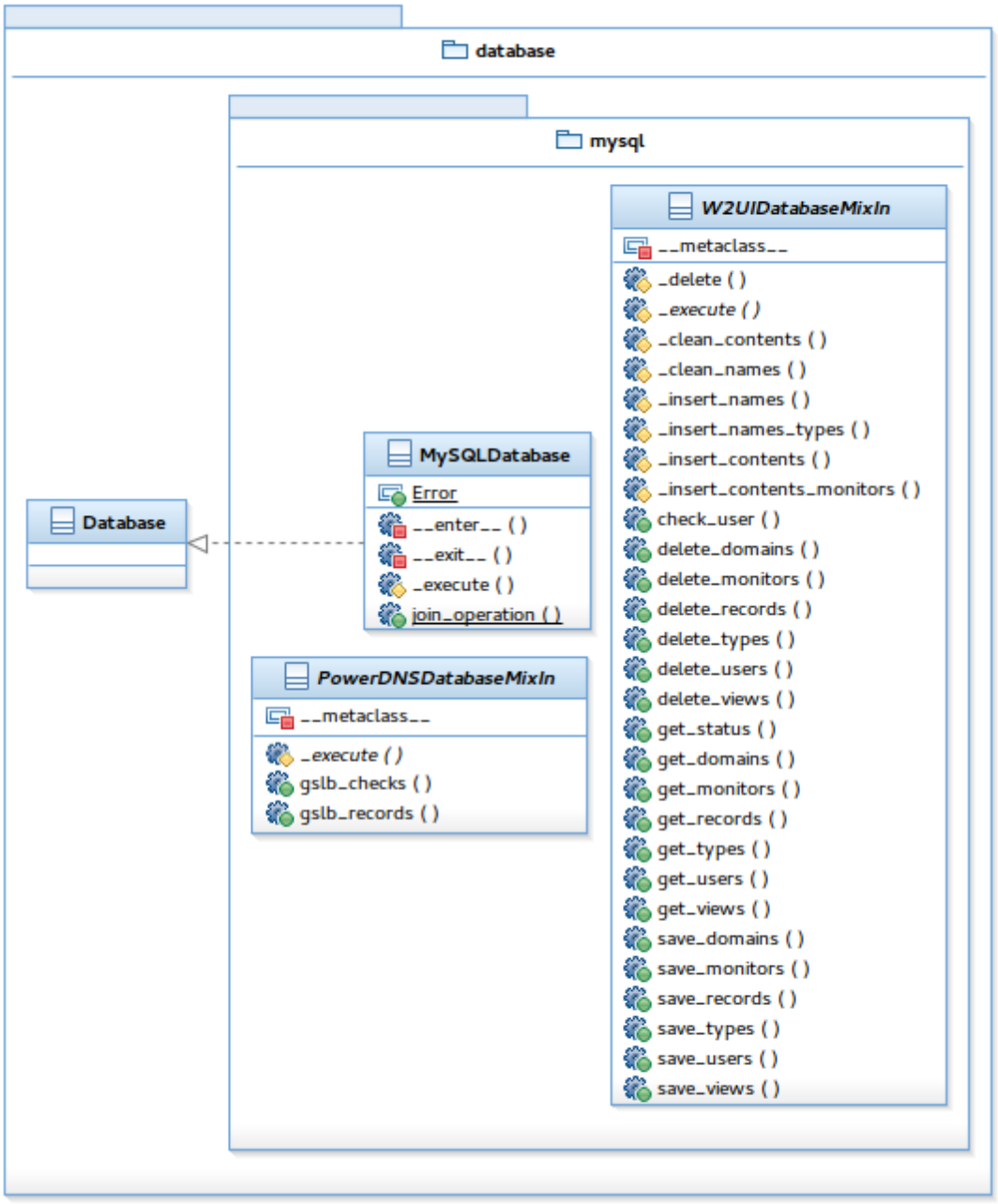
2.16. att. Pakešu diagramma bāzes paketei powergslb



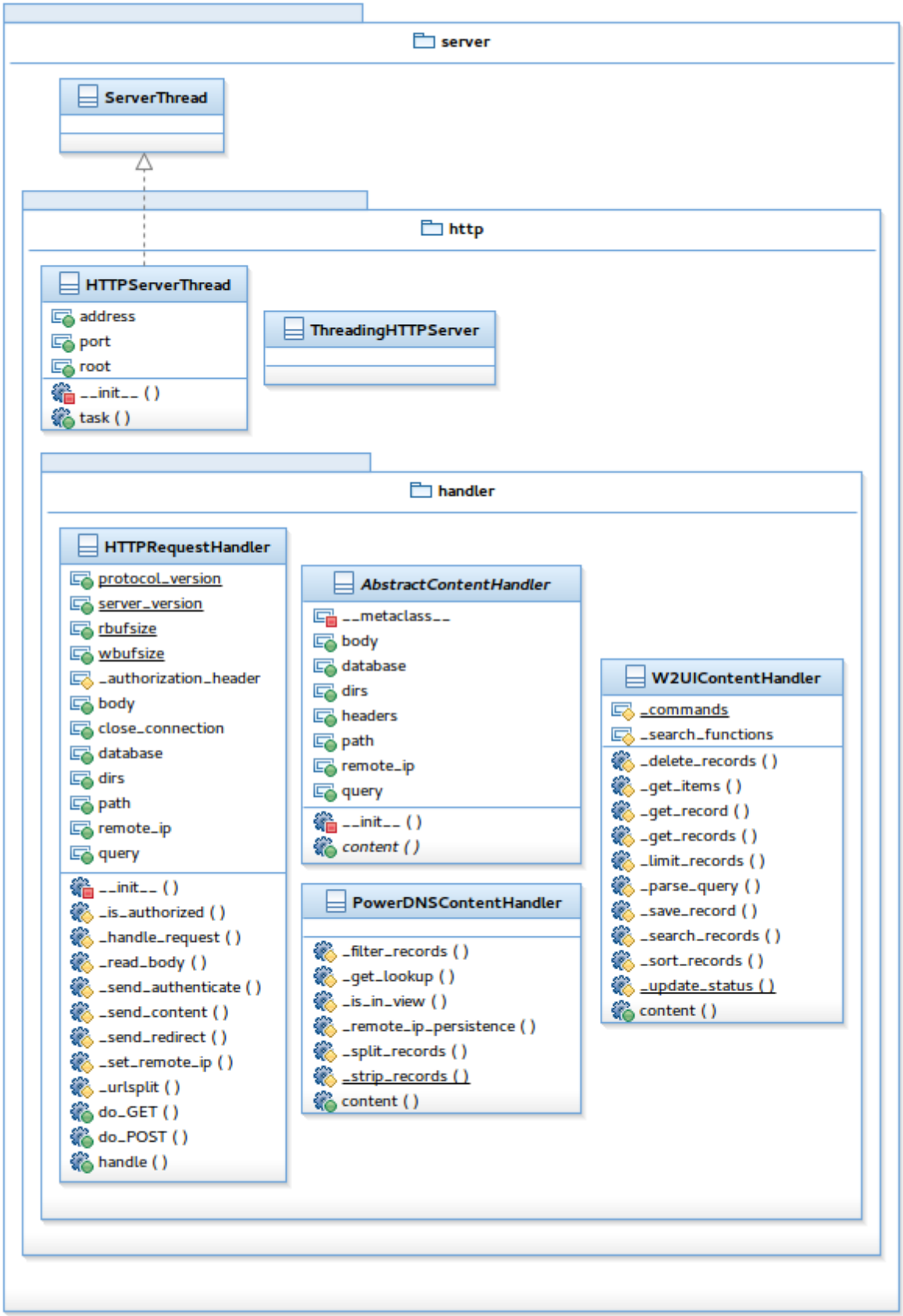
2.17. att. Pakešu diagramma apakšpaketei monitor



2.18. att. Pakešu diagramma apakšpaketei system



2.19. att. Pakešu diagramma apakšpaketei database



2.20. att. Pakešu diagramma apakšpaketei server

Modeļa kvalitātes gala novērtējuma un sistēmas gala novērtējuma rezultāti parādīti attiecīgi 2.3. un 2.4. tabulā.

2.3. tabula

Vizuālā modeļa loģiskās struktūras kvalitātes gala novērtējums

Klases	Metrikas					
	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
Vidējā vērtība	**6,25**	**1,25**	**0,44**	**0,94**	**8,13**	**1,00**

2.4. tabula

Sistēmas gala novērtējums

Metrika	Vērtība
DIT	2
NC	16
NOM	100
LOCΣ	1771

Salīdzinot ar sākotnējo novērtējumu, redzams vērtību WMC, CBO, RFC, NC un NOM pieaugums, kas liecina par programmatūras produkta kompleksuma palielināšanos.

Uzkrītoši atšķiras klases W2UIDatabaseMixIn vērtības WMC и RFC. Un, kaut arī metrika WMC pārsniedz ieteicamo vērtību par 20 metodēm (Орлов, 2016), šai realizācijā tā nav problēma, jo klase satur lielu skaitu SQL pieprasījumu un tikai minimāli – programmas kodu.

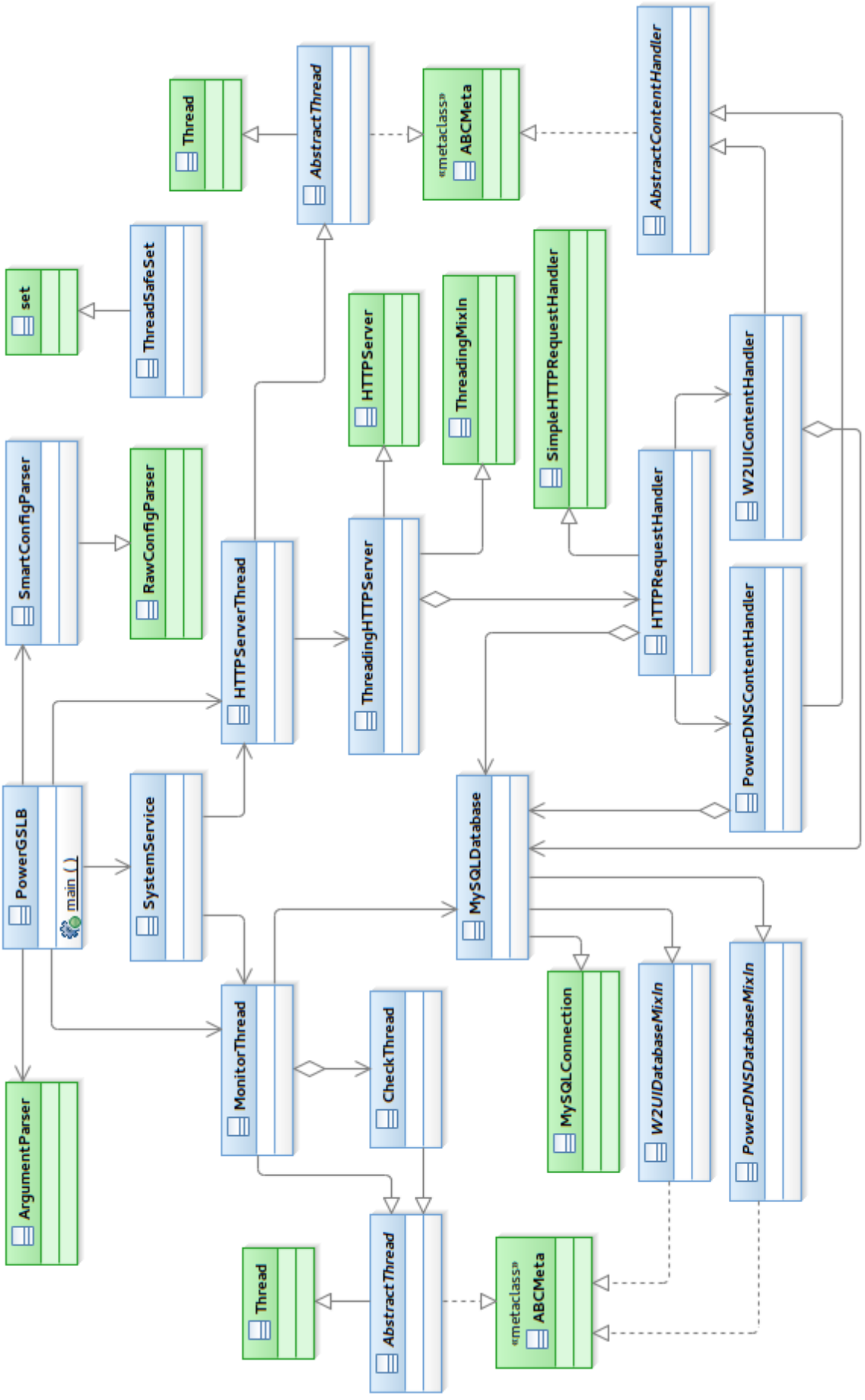
Vērtību DIT un NOC palielināšanās liecina par kvalitātes uzlabošanos, pateicoties koda atkārtotai izmantošanai ar mantošanas starpniecību.

Metrikas LCOM4 vērtības liecina par programmatūras produkta kvalitāti, jo saistītā klasē jābūt tikai vienam saistītam komponentam.

No gala novērtējuma var izdarīt secinājumu, ka modeļa kvalitāte ir apmierinoša.

2.21. attēlā norādīta galīgā klašu diagramma, ņemot vērā ārējās attiecībā pret galveno programmu klases un attiecības starp klasēm. Iekšējās (realizētās) klases ir norādītas zilā krāsā. Ārējās (bibliotēkas) klases ir norādītas zaļā krāsā.

Bāzes paketes powergslb un daļas apakšpakešu kods, atbilstoši stāvoklim uz šī darba uzrakstīšanas brīdi, sniegts pielikumā. Aplikācijas jaunākā versija pieejama projekta Git repozitorijā adresē https://github.com/AlekseyChudov/powergslb. Tur tiek glabāta arī visa programmatūras versiju vēsture.



2.21. att. Galīgā klašu diagramma, ņemot vērā to attiecības

2.6. Testēšana

Testēšana ir programmas izpildes process ar mērķi atklāt kļūdas (Орлов, 2016).

Programmas sistēmas testēšana ir izpildīta manuāli, pēc „melnās kastes" principa, izmantojot sadalīšanas paņēmienu pēc ekvivalentuma.

„Melnās kastes" testēšana (funkcionālā testēšana) ļauj iegūt ieejas datu kombinācijas, kas nodrošina visu programmai izvirzīto funkcionālo prasību pilnu pārbaudi (Орлов, 2016).

Izmantojot sadalīšanas paņēmienu pēc ekvivalentuma, programmas datu ieejas zona tiek sadalīta ekvivalentuma klasēs. Katrai ekvivalentuma klasei tiek izstrādāts viens testa variants (Орлов, 2016).

Apskatīsim operētājsistēmas veiktu aplikācijas palaišanas operāciju.

Priekšnoteikumi

- konfigurācijas fails ir pieejams lasīšanai;
- konfigurācijas failā nav kļūdu;
- konfigurācijas failā norādītā datu bāze ir pieejama;
- konfigurācijas failā norādītā IP adrese pieder serverim;
- konfigurācijas failā norādītais TCP ports serverī ir brīvs.

Pēcnoteikumi

- priekšnoteikumu izpildes gadījumā — korekts aplikācijas starts;
- priekšnoteikumu neizpildes gadījumā — paziņojums par kļūdu sistēmas žurnālā.

Apskatīsim konfigurācijas vadības tīmekļa interfeisa palaišanas operāciju.

Priekšnoteikumi

- administrators ievada pareizu lietotāja vārdu;
- administrators ievada pareizu paroli.

Pēcnoteikumi

- priekšnoteikumu izpildes gadījumā — korekts tīmekļa interfeisa starts;
- priekšnoteikumu neizpildes gadījumā — paziņojums par autorizācijas kļūdu.

Apskatīsim jauna DNS ieraksta pievienošanas operāciju.

Priekšnoteikumi

- administrators ievada eksistējošu domēnu laukā „Domain”;
- administrators ievada eksistējošu tipu laukā „Type”;
- administrators ievada eksistējošu monitoru laukā „Monitor”;
- administrators ievada vērtību, lielāku par 0, laukā „TTL”;
- administrators ievada vērtību no 0 līdz 1 laukā „Disabled”;
- administrators ievada vērtību no 0 līdz 1 laukā „Fallback”.

Pēcnoteikumi

- priekšnoteikumu izpildes gadījumā — ieraksta korekta izveidošana;
- priekšnoteikumu neizpildes gadījumā — paziņojums par ieraksta izveidošanas kļūdu.

Ekvivalentuma klases

1. Nosacījumiem, kas nosaka būla vērtību, noteiksim vienu pieļaujamu un vienu nepieļaujamu ekvivalentuma klasi:
   - V_Class = {true};
   - Inv_Class = {false}.
2. Nosacījumiem, kas nosaka vērtību diapazonu *n..m*, noteiksim vienu pieļaujamu un divas nepieļaujamas ekvivalentuma klases:
   - V_Class = {*n..m*};
   - Inv_Class1 = {$x \mid x: x < n$};
   - Inv_Class2 = {y | $y: y > n$}.
3. Nosacījumiem, kas nosaka vērtību kopu {*a, b, c*}, noteiksim vienu pieļaujamu un vienu nepieļaujamu ekvivalentuma klasi:
   - V_Class = {*a, b, c*};
   - Inv_Class = {$x \mid x: (x \neq a)$ & $(x \neq b)$ & $(x \neq c)$}.
4. Nosacījumiem, kas nosaka konkrēto vērtību *a*, noteiksim vienu pieļaujamu un vienu nepieļaujamu ekvivalentuma klasi:
   - V_Class = {*a*};
   - Inv_Class1 = {$x \mid x: x < a$};
   - Inv_Class2 = {y | $y: y > a$}.

Nepieciešams izstrādāt tādus testa variantus, kas pārbauda vislielāko skaitu ekvivalentuma klašu. Testa variantu piemēri un testu rezultāti ir parādīti 2.5. tabulā.

Analoģiskā veidā veikta visu programmai izvirzīto funkcionālo un nefunkcionālo prasību testēšana. Veiktās testēšanas rezultātā savesta kārtībā programmas darbība, izlabotas atklātās sīkās kļūdas un nav atklātas nopietnas kļūdas, kas pieprasītu izmaiņas projektā.

2.5. tabula

Testa variantu piemēri

Ieejas dati	Gaidāmais rezultāts	Testa rezultāts
Tiek izpildīta aplikācijas palaišanas komanda	1) Korekts aplikācijas starts 2) Sistēmas žurnālā nav paziņojumu par kļūdu	iziets
1) Konf. fails nav pieejams lasīšanai 2) Tiek izpildīta aplikācijas palaišanas komanda	1) Nekorekts aplikācijas starts 2) Sistēmas žurnālā ir paziņojumi par kļūdu	iziets
1) Konf. fails satur kļūdu 2) Tiek izpildīta aplikācijas palaišanas komanda	1) Nekorekts aplikācijas starts 2) Sistēmas žurnālā ir paziņojumi par kļūdu	iziets
1) Datu bāze nav pieejama 2) Tiek izpildīta aplikācijas palaišanas komanda	1) Nekorekts aplikācijas starts 2) Sistēmas žurnālā ir paziņojumi par kļūdu	iziets
1) IP adrese nepieder serverim 2) Tiek izpildīta aplikācijas palaišanas komanda	1) Nekorekts aplikācijas starts 2) Sistēmas žurnālā ir paziņojumi par kļūdu	iziets
1) TCP ports serverī aizņemts 2) Tiek izpildīta aplikācijas palaišanas komanda	1) Nekorekts aplikācijas starts 2) Sistēmas žurnālā ir paziņojumi par kļūdu	iziets
Tiek izpildīta tīmekļa interfeisa palaišana	Korekts tīmekļa interfeisa starts	iziets
1) Tiek izpildīta tīmekļa interfeisa palaišana 2) Tiek ievadīts nepareizs lietotāja vārds	Paziņojumi par autorizācijas kļūdu	iziets
1) Tiek izpildīta tīmekļa interfeisa palaišana 2) Tiek ievadīta nepareiza parole	Paziņojumi par autorizācijas kļūdu	iziets
Tiek pievienots jauns DNS ieraksts	Ieraksta korekta izveidošana	iziets
1) Laukā „Domain” tiek ievadīts neeksistējošs domēns 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	iziets
1) Laukā „Type” tiek ievadīts neeksistējošs tips 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	iziets
1) Laukā „Monitor” tiek ievadīts neeksistējošs monitors 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	iziets
1) Laukā „TTL” tiek ievadīta vērtība, kas mazāka par 0 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	iziets
1) Laukā „Disabled” tiek ievadīta vērtība, kas mazāka par 0 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	iziets
1) Laukā „TTL” tiek ievadīta vērtība, kas lielāka par 1 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	nav kļūdas
1) Laukā „Fallback” tiek ievadīta vērtība, kas mazāka par 0 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	iziets
1) Laukā „Fallback” tiek ievadīta vērtība, kas lielāka par 1 2) Tiek pievienots jauns DNS ieraksts	Paziņojumi par ieraksta izveidošanas kļūdu	nav kļūdas

2.7. Kvalitātes novērtējums

Mūsdienīgas programmu sistēmas neatņemama īpašība ir augsta kvalitāte. Ir daudz dažādu programmatūras kvalitātes definīciju (Орлов, 2016).

Piemēram, autoritatīvajā programminženierijas vārdnīcā IEEE Std 610.12-90 «IEEE Standard Glossary of Software Engineering Terminology» rakstīts:

1. Programmatūras kvalitāte ir sistēmas, komponenta vai procesa atbilstība noteiktām prasībām.
2. Programmatūras kvalitāte ir sistēmas, komponenta vai procesa atbilstība pasūtītāja vai lietotāja vajadzībām vai gaidām (Орлов, 2016).

Darba procesā nodrošināta visu pieteikto, programmatūrai izvirzīto funkcionālo un nefunkcionālo prasību izpilde; tas ir fundamentāls kvalitātes rādītājs (Орлов, 2016).

Daži kvantitatīvi un kvalitatīvi novērtējumi minēti iepriekš 2.1. — 2.4. tabulā. No vizuālā modeļa kvalitātes novērtējuma var izdarīt secinājumu, ka sistēmas kvalitātes vidējās vērtības ir apmierinošas.

Galvenie testu varianti minēti iepriekš 2.5. tabulā. No veiktās testēšanas rezultātiem var izdarīt secinājumu, ka programmatūras funkcijas uzprojektētas un realizētas pareizi, un programmatūras kvalitāte un drošums ir apmierinoši.

Pateicoties moduļu struktūras izmantošanai un programmas klašu sadalīšanai paketēs, tiek vienkāršota funkcionalitātes (functionality) paplašināšana un uzturamība (maintainability) arī tas ir iegūtās programmatūras kvalitātes rādītājs.

NOBEIGUMS

Šī darba mērķis bija atvērtas programmatūras izstrāde globālai serveru balansēšanai, izmantojot domēnu vārdu sistēmu (Domain Name System based Global Server Load Balancing, DNS GSLB).

Galvenie darba uzdevumi bija šādi:

1. Apskatīt un salīdzināt galvenās noslodzes un trafika balansēšanas tehnoloģijas, kas tiek izmantotas dažādos OSI tīkla modeļa slāņos.
2. Apskatīt un aprakstīt aplikācijas izstrādes procesa posmus globālai serveru balansēšanai, izmantojot domēnu vārdu sistēmu.

Darba pirmajā daļā bija aplūkotas tādas noslodzes un trafika balansēšanas tehnoloģijas kā kanālu agregācija (LAG), vienādas vērtības daudzstaru maršrutēšana (ECMP), serveru noslodzes balansēšana (SLB) un globālā serveru noslodzes balansēšana (GSLB). Īpaša uzmanība veltīta SLB un GSLB tehnoloģijām, jo tās visbiežāk tiek realizētas programmatūras līmenī, nevis tīkla aprīkojuma līmenī. Darba pirmās daļas noslēgumā bija veikts apskatīto noslodzes un trafika balansēšanas tehnoloģiju īss salīdzinājums.

Darba otrajā daļā bija apskatīti aplikācijas izstrādes procesa posmi globālajai serveru balansēšanai, izmantojot DNS. Izstrādes procesā bija noformētas un izanalizētas produktam izvirzītās prasības, veikta aplikācijas un datu bāzes projektēšana, realizēta aplikācija, veikta tās testēšana un novērtējums.

Veiktā darba rezultātā iegūtas šādas priekšrocības:

- Izstrādāts lietošanā vienkāršs risinājums globālās serveru noslodzes balansēšanas nodrošināšanai, izmantojot DNS.
- Zemas gala produkta izmaksas, pateicoties atvērto tehnoloģiju izmantošanai.
- Atvērtais izstrādes modelis padara programmatūru vispārpieejamu.

Diemžēl dažas tehniskas detaļas palikušas aiz kadra. Piemēram, tīmekļa aplikācijas izstrādes īpatnības valodā JavaScript un programmatūras izvēršana serveru klasterī. Taču kopumā veiktā darba rezultāti ļauj izdarīt secinājumu, ka darba mērķis ir sasniegts un darba uzdevumi sekmīgi izpildīti.

IZMANTOTĀS LITERATŪRAS UN CITU INFORMĀCIJAS AVOTU SARAKSTS

1. Олифер В. Г. and 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. and 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. *Linux Kernel Documentation* (2016). Available from: <https://www.kernel.org/doc/>. [22.05.2016].
22. *MySQL Documentation* (2016). Available from: <https://dev.mysql.com/doc/>. [22.05.2016].
23. *PEP 8 -- Style Guide for Python Code* (2001). Available from: <https://www.python.org/dev/peps/pep-0008/>. [22.05.2016].
24. PowerGSLB, computer software (2016). Available from: <https://github.com/AlekseyChudov/powergslb>. [22.05.2016].
25. *Python 2.7.11 documentation* (2016). Available from: <https://docs.python.org/2.7/index.html>. [22.05.2016].