

Load Balancer

Softwarový projekt, MFF UK Praha

Programátorská dokumentace

Obsah

1	Úvod	5
2	Init	7
2.1	Startování Load Balanceru	7
2.2	Spuštění a kontrola konfiguračního souboru	7
2.3	Vytvoření spojení	8
2.4	Spuštění jádra a statistik	8
2.5	Kontrola, restartování a ukončení Load Balanceru	9
3	Jádro	10
3.1	Architektura jádra	10
3.2	Spuštění a komunikace s procesem init	12
3.3	Inicializace	13
3.4	TCP spojení	14
3.4.1	Management TCP vláken	14
3.4.2	Přijetí spojení a vybrání cílového serveru	15
3.4.3	Navázání spojení	16
3.4.4	Provádění spojení	17
3.5	UDP vlákna	17
3.5.1	Management socketů	18
3.5.2	Přijetí spojení, vybrání cílového serveru a přenos dat	18
3.6	Ping vlákno	19
4	Podmoduly	20
4.1	Funkce na první vrstvě	22
4.2	Funkce na druhé vrstvě	24
4.3	Funkce na třetí vrstvě	27
4.4	Funkce na datové vrstvě	30
4.5	Vytvoření vlastního podmodulu	32
4.5.1	Příklad vytvoření podmodulů http protokolu	33
4.5.2	Příklad vytvoření podmodulů ftp protokolu	33

4.5.3	Vytvoření vlastního dynamického algoritmu	34
5	Statistický modul	38
5.1	Úvod	38
5.2	Databáze	39
5.2.1	Práce s databází pomocí ODBC rozhraní	39
5.2.2	Filosofie ukládání dat	39
5.2.3	Datový a databázový model	41
5.3	Databázový buffer	43
5.4	Buffer v paměti	46
5.5	Inicializační vlákno	46
5.5.1	Start statistického modulu	46
5.5.2	Inicializace globálních proměnných a struktur	47
5.5.3	Připojení k databázi a příprava databázových operací	48
5.5.4	Start pracovních vláken	49
5.6	Vlákno pro ukládání dat	49
5.7	Vlákno pro zpracování dotazů a odeslání odpovědí	50
5.7.1	Zpracování dotazů od podmodulů jádra	50
5.7.2	Zpracování dotazů od prezentačního modulu	52
5.7.3	Zpracování dotazů na stav serverů	53
6	Konfigurace	54
6.1	Úvod	54
6.2	Návrh konfigurační knihovny	54
6.3	Reprezentace konfiguračního souboru	56
6.4	Parser	57
6.4.1	Flex	57
6.4.2	Bison	58
6.5	Zpřístupnění proměnných modulům	60
6.5.1	Zpřístupnění proměnných jednoduchých datových typů	61
6.5.2	Zpřístupnění tabulek	62
6.6	Podpora pro webové rozhraní	64
6.7	Rozšíření konfigurační knihovny	64
7	Web	66
7.1	Úvod	66
7.2	PHP extension balancer	66
7.2.1	Seznam funkcí	66
7.3	Konfigurace webového rozhraní	68
7.4	Ovládání balanceru z webu	69
7.5	Monitoring serverů	69

7.6	Zobrazování dat ze statistického modulu	69
7.7	Editace konfiguračního souboru	70
7.7.1	Main	70
7.7.2	Aliases	70
7.7.3	Sections	70
7.8	Zabezpečení webového rozhraní	70
8	Knihovny	71
8.1	knihovna cache.c	71
8.2	Komunikační knihovna commun.c	73
8.3	knihovna groups.c	75
8.4	knihovna ping.c	76
8.5	knihovna port.c	76
8.6	knihovna startup.c	78
A	Komunikační protokol procesu init	79
A.1	Tabulka zpráv	79
A.2	Sekvence příkazů	81
B	Přehled některých existujících balancerů	83
B.1	WebMux	83
B.2	Coyote Point's Equalizer	83
B.3	Prestwood Load Balancer	84
B.4	Zeus Load Balancer	84
B.5	Distributor Load Balancer	84
B.6	Pure Load Balancer	84
C	Chronologický průběh prací	85

Kapitola 1

Úvod

Load Balancer je rozdělen do několika modulů. Tím je umožněno dosáhnout různé úrovně funkčnosti a výkonu. Aby se co nejvíce omezila možnost, kdy se Load Balancer stane "úzkým hrdlem", jsou jednotlivé moduly samostatné procesy, které mohou běžet na různých počítačích a komunikují spolu pomocí protokolu TCP/IP a UDP/IP.

Proces Init slouží ke spuštění, řízení a ukončení Load Balanceru. Jádro je výkonnou částí, která provádí samotné přepojování spojení. Statistiky ukládají informace o provedených spojeních, odpovídají na dotazy jádra o stavu serverů a poskytují informace pro webové statistiky a informace. Webové rozhraní umožňuje vzdálenou správu Load Balanceru a prezentaci statistik.

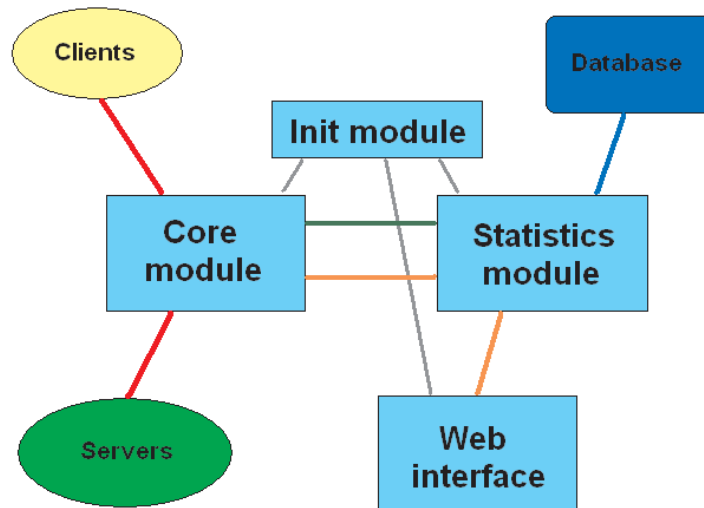
Na obrázku je znázorněna architektura Load Balanceru a tok dat mezi jednotlivými částmi. Load balancer tvoří části označené světle modrou barvou. Tmavě modrou barvou je naznačena externí databáze, která je k Load Balanceru připojena přes ODBC rozhraní. Žlutou barvu mají klienti, kteří se připojují z Internetu na Load balancer a ten jejich požadavky přesměrovává na servery (zelené), které jsou na lokální síti.

Barevné spojnice na představují tok dat. Červeně jsou naznačena uživatelská data, která se přenesou k jádru a to je po výběru serveru přeneseno na vybraný server. Tato zátěž je vysoká a spoje by proto měly být dostatečně dimenzovány.

Šedou barvou je naznačena servisní komunikace mezi modulem init a ostatními částmi Load Balanceru. Používá se pro přenos konfiguračních dat a restartování Load Balanceru.

Zelenou barvou jsou naznačena data odesílaná pomocí protokolu UDP z jádra do statistik o prováděných spojeních. Objem těchto dat se dá regulovat pomocí konfiguračního souboru.

Oranžová barva představuje dotazy jádra a webového rozhraní do statistik.



Obrázek 1.1: Architektura Load Balanceru

Jádro dotazy používá pro balancování v dynamických algoritmech. Webové rozhraní pro zobrazování statistik o spojeních.

Modrá spojnice představuje práci statistického modulu s externí databází.

Celý Load Balancer by z důvodu výkonosti měl běžet na lokální síti. Jednotlivé části ale lze spustit i na WAN. Servisní komunikace (šedá barva) probíhá nezabezpečeně a proto se toto nastavení nedoporučuje.

Kapitola 2

Init

V této kapitole bude podrobněji popsána funkce procesu init. Hlavní funkce initu tvoří spuštění celého Load Balanceru, načtení konfiguračního souboru, jeho kontrola a rozeslání ostatním částem Load Balanceru a následná kontrola jejich stavu. Dále je možné pomocí signálu SIGHUP celý Load Balancer restartovat a pomocí signálu SIGTERM ukončit. Proces init je implementován jako stavový automat.

2.1 Startování Load Balanceru

Celý Load Balancer se spouští v následujícím pořadí. Nejdříve se spustí procesy jádro a statistiky, které po načtení přepínačů z příkazového řádku začnou čekat na určených portech na startovací sekvenci od initu. Pak se spustí proces init, který si z konfigu načte, na kterých IP adresách a portech se nachází jádro a statistiky a jestli se mají statistiky spustit. Pokud ano, následuje pokus o jejich spuštění. Poté následuje spouštění jádra i s informací o spuštění statistik.

2.2 Spuštění a kontrola konfiguračního souboru

Init se je možné spustit s s následujícími přepínači:

- d nespouštět jako službu daemon. Init se (stejně jako jádro a statistiky) implicitně spouští na pozadí jako daemon.
- f **FILE** jako konfigurační soubor použít FILE. Implicitně se použije soubor /etc/lb.cfg

- o provést pouze kontrolu konfiguračního souboru a nespouštět Load Balancer.
- a při nastavení tohoto příznaku se hledají všechny chyby v konfiguračním souboru. Implicitně se kontrola souboru zastaví po nalezení první chyby
- e vypisovat nalezené chyby na chybový výstup stderr. Implicitně se chyby zapisují pouze do syslogu.

Po spuštění se nejdříve init přepne do režimu daemon (pokud není zapnutý přepínač d). Pak se načte buď standardní konfigurační soubor nebo soubor FILE při použití přepínače f pomocí funkce `cfg_get_file` konfigurační knihovny. Provede se kontrola konfiguračního souboru pomocí funkce `cfg_check`. Pokud je nastaven přepínač o je po provedení kontroly init ukončen. Program pokračuje nastavením dočasného blokování signálů SIGHUP a SIGTERM během inicializace. Pak si init z konfiguračního souboru načte proměnné, které potřebuje pro svůj vlastní běh pomocí funkce `read_config` a nastaví úroveň logování do syslogu.

2.3 Vytvoření spojení

Init pokusí navázat spojení s jádrem a statistikami pomocí funkcí `connect_core_socket` a `connect_stats_socket`. Pokud je nastaveno spuštění statistik, pokusí se init připojit na TCP port `INIT_STATS_PORT`, implicitně na port 44784. Pokud se připojení nepodaří, zkouší se init opakovaně připojit každou vteřinu do uplynutí času `INIT_STATS_STARTUP_TIMEOUT`.

Pak je stejným způsobem navázáno spojení s jádrem. Init se pokusí připojit na TCP port `INIT_CORE_PORT`, implicitně 44783. Pokud se připojení nepodaří, zkouší se init opakovaně připojit každou vteřinu do uplynutí času `INIT_CORE_STARTUP_TIMEOUT`.

2.4 Spuštění jádra a statistik

Spuštění se provádí pomocí funkce `init_main_loop`, která je hlavní funkcí stavového automatu. Pokud je nastaveno spuštění procesu statistiky, je s ním zahájena startovací sekvence. Zprávy jsou posílány v textové podobě pro větší čitelnost v případě chyb. Na začátku každé zprávy je třímístné číslo určující zprávu i její typ. Seznam zpráv a jejich různé sekvence jsou v příloze Komunikační protokol procesu init.

2.5 Kontrola, restartování a ukončení Load Balanceru

Po ukončení startovací sekvence `init` pravidelně kontroluje jádro i statistiky, pokud to bylo nastaveno v konfiguračním souboru. Střídavě zasílá zprávy jádru a statistikám a čeká na jejich odpověď. Při zjištění nekomunikace procesu se ho `init` pokusí restartovat. Zasláním signálu `SIGHUP` je možné celý Load Balancer restartovat včetně znovunačtení konfiguračního souboru. Zasláním signálu `SIGTERM` se celý Load Balancer ukončí.

Kapitola 3

Jádro

V této kapitole bude podrobněji popsána funkce jádra, které je hlavním výkonným procesem Load Balanceru. Při spuštění se rozdělí na dvě části. První komunikuje s procesem `init`, stará se o restart jádra, jeho ukončení a případně o automatickou kontrolu, jestli jádro v pořádku pracuje. Druhá část je samotné výkonné jádro, které pracuje v několika vláknech. Po počáteční inicializaci hlavní vlákno čeká na příchozí spojení. Po přijetí požadavku ho předá buď příslušnému UDP vláknu nebo vybere jedno z volných TCP vláken, které nový požadavek obslouží. Volitelně se vytvoří vlákno, které umožňuje kontrolu dostupnosti serverů.

3.1 Architektura jádra

U jádra je kladem důraz především na modularitu, škálovatelnost a stabilitu. Je postaveno tak, aby mohlo vykonávat alespoň základní činnosti bez spuštění některých dalších součástí, případně po jejich pádu.

Při spouštění a komunikaci s procesem `init` jsou vytvořeny dva procesy. Synovský je samotné výkonné jádro a rodičovský zajišťuje komunikaci s procesem `init`, pomocí něhož může dojít k ukončení jádra, jeho restartování (např. pro znovunačtení konfiguračního souboru) nebo pro automatickou kontrolu funkčnosti výkonné části (nastavením proměnné `INIT_CORE_FREEZE_CONTROL`). Pokud rodičovský proces nedostane odpověď na svůj dotaz, výkonný synovský proces automaticky restartuje. To zvyšuje celkovou stabilitu jádra a možnost automatického obnovení při pádu.

Při samotném přepojování dat od klientů k serverům je kladen důraz především na modularitu a škálovatelnost. Pro výběr správného serveru se používají dll knihovny na třech vrstvách, které se mohou nahrávat i přidávat

za běhu Load Balanceru (nová knihovna vyžaduje úpravu konfiguračního souboru a proto následné restartování Load Balanceru).

Podmodul a jeho funkce na první vrstvě se určuje v konfigurační části jádra a volá se pro každé příchozí spojení. Samotná funkce se dá plně konfigurovat, ale v případě speciálních požadavků lze napsat a vložit svoji vlastní. Funkce na první vrstvě je povinná. Rozhoduje se na základě portu, na kterém Load Balancer poslouchá, protokolu, adresy klienta a dalších parametrů. Výstupem této funkce je skupina serverů, které mohou daný požadavek obsloužit, knihovna a funkce druhé nebo třetí vrstvy, případně datového podmodulu a data načtená pro vykonání rozhodnutí.

Podmodul a jeho funkce na druhé vrstvě je volána na základě výsledku funkce na první vrstvě. Typicky je pro každý port/protokol napsán zvláštní podmodul. Druhá vrstva nadále upřesňuje skupinu serverů, které jsou schopny požadavek řešit a které dostala od podmodulu na první vrstvě. Výstupem je knihovna a funkce pro třetí vrstvu, případně datový podmodul a data načtená pro vykonání rozhodnutí. Typicky se podmoduly na druhé vrstvě rozhodují podle obsahu dat (Layer 7 switching). Funkce na druhé vrstvě se mohou volat opakovaně a tak zjemňovat dělení. Naimplementovaný je podmodul pro rozhodování na základě obsahu dat v protokolu HTTP.

Po průchodu prvních dvou vrstev by skupina vybraných serverů měla být ekvivalentní z hlediska schopnosti vyřízení požadavku daného typu spojení. Ve funkcích v podmodulech na třetí vrstvě se určuje pořadí vybrané skupiny serverů buď na základě statických hodnot (statická váha serveru, zdrojová adresa, ...) nebo pomocí dotazování se statistik. Jádro pak postupně zkouší vytvořit spojení na těchto serverech podle jejich priority. Funkce na třetí vrstvě by neměla být volána pouze v případě kdy daný požadavek může vyřídit pouze jeden server nebo pokud je priorita serverů nastavená z předchozích vrstev.

Dalším podmodulem, který lze využít, je datový podmodul. Slouží k řízení nových spojení a k úpravě přenášených dat. Datový modul se nahraje a používá pokud je nastaven při průchodu první až třetí vrstvou. Volá se při vytvoření hlavního spojení, při ukončení hlavního spojení a vždy při přenosu dat oběma směry v hlavním spojení. V tom případě se nejdříve data načtou, pak se zavolá funkce datového podmodulu, a následně se data zapíší. Datový podmodul má možnost měnit data před jejich zapsáním a vytvářet, otevírat a zavírat pomocná datová spojení potřebná pro příslušný protokol směrem od i k serveru. Po ukončení hlavního spojení jsou všechna datová spojení také ukončena. Naimplementovaný je datový podmodul pro protokol FTP, který umožňuje přepojování aktivního i pasivního přenosu.

Všechny podmoduly se nahrávají jako dll knihovny a pro další rozšíření o nové protokoly tak stačí upravit nebo připsat nové dll knihovny. Load Ba-

lancer pak stačí restartovat pro uplatnění změn. Tím je dosaženo vysoké modularity a možnosti dalšího rozšíření i o zcela speciální nestandardní protokoly.

Výběrem různých funkcí na třetí vrstvě je pak dosaženo vysoké škálovatelnosti. Pro některý typ serverů stačí rychlejší rozhodování pomocí statických vah, pro jiné je lepší se rozhodovat na základě aktuálního zatížení dotazem do statistik.

Přidávání, změna parametrů nebo ubírání serverů je možné úpravou konfiguračního souboru a následným restartem Load Balanceru.

Výkonnost jádra lze pak nastavovat automatickým řízením počtu obsluhujících vláken na základě konfiguračního souboru. Je zde tak možnost kompromisu mezi výkonem a množstvím systémových prostředků (především paměti) na základě aktuálního vytížení Load Balanceru.

3.2 Spuštění a komunikace s procesem init

Jádro se je možné spustit s následujícími prepínači:

- d** nespouštět jako službu daemon. Jádro se (stejně jako Init a statistiky) implicitně spouští na pozadí jako daemon.
- p** **PORT** čekat na startovací sekvenci na portu PORT. Implicitně jádro poslouchá na portu 44783.
- l** **IP_ADDR** čekat na startovací sekvenci na rozhraní IP_ADDR. Implicitně jádro poslouchá na rozhraní INADDR_ANY.

Po spuštění se nejdříve jádro přepne do režimu daemon (pokud není zapnutý prepínač d). Poté se zavolá funkce `startup_init_core` z knihovny `startup`, ve které jádro poslouchá na portu 44783 (pokud není uvedený jiný port prepínačem p) na startovací sekvenci procesu `init`. Po navázání spojení s jádrem dostane od procesu `init` konfigurační soubor a informaci, jestli byl úspěšně spuštěn proces `statistiky`. Pak se načtou proměnné potřebné pro další inicializaci. Po nastavení úrovně logování se proces jádra rozdělí. Synovský proces je výkonný proces jádra, rodičovský proces čeká na případnou komunikaci s procesem `init`.

Synovský proces pak provede svoji inicializaci a po jejím ukončení zavolá funkci `startup_init_core_ready` a tím odešle procesu `init` zprávu o dokončení své inicializace.

Rodičovský proces může obdržet zprávu od procesu `init`, aby ukončil činnost jádra. V tom případě nejdřív ukončí synovský proces signálem `SIGKILL` a poté sám skončí.

Další možností je zpráva, aby byl proveden restart jádra. To se opět provede ukončením synovského procesu pomocí signálu SIGKILL a novým rozdělením rodičovského procesu.

Poslední funkcí rodičovského procesu může být automatická kontrola "živosti" synovského procesu. Pokud je v konfiguračním souboru nastavena hodnota `INIT_CORE_FREEZE_CONTROL` na `YES`, pak rodičovský proces posílá synovskému procesu signál `SIGUSR1` a čeká `INIT_CORE_FREEZE_TIMEOUT` vteřin na odpověď. Pokud do té doby neobdrží od synovského procesu signál `SIGUSR1`, považuje ho za nefunkčního a provede automatický restart synovského procesu stejně jako při restartu zvolaném procesem `init`.

3.3 Inicializace

Pokud je úspěšně spuštěn proces statistik, je po vytvoření výkonného synovského procesu zavolána funkce `commun_init_core`, která vytvoří udp socket pro odesílání informací do statistik.

Pak se funkcí `fill_const` načtou z konfiguračního souboru proměnné potřebné pro běh jádra.

Nakonec se zavolá funkce `core_init`, která postupně provede inicializaci jádra v následujících krocích:

proměnné alokování potřebných struktur (pole záznamů o vláknech, ...) a nastavení proměnných (mutexy, podmínkové proměnné, ...)

ping vlákno pokud je v konfiguračním souboru nastaveno použití "pingání" služeb, je zde vytvořeno vlákno příslušné vlákno

TCP vlákna Je vytvořeno `CORE_THREADS_INIT` vláken a čeká se, až ukončí vlastní inicializaci. Během ní vlákno inicializuje své lokální proměnné (kopie seznamu serverů, ...), pokud jsou spuštěny statistiky, je zavolána funkce `commun_core_socket`, která vytvoří spojení se statistikami.

sockety a UDP vlákna prochází se seznam portů, které se mají otevřít. Pokud se jedná o TCP port, vytvoří se příslušný socket (socket), pojmenuje se (bind) a začne se na něm poslouchat (listen). Pro každý UDP port se vytvoří nové UDP vlákno, které daný port obsluhuje.

knihovna první vrstvy nalezení a nahrání knihovny a funkce na první vrstvě podle proměnných `CORE_LAYER1_LIBRARY` a `CORE_LAYER1_FUNCTION`

Pokud se nějaký z těchto kroků nepodaří, je většinou funkce jádra ukončena.

3.4 TCP spojení

Přepojování TCP spojení je bráno jako hlavní funkce Load Balanceru. Poslouchání na TCP portech je prováděno v hlavní smyčce funkce main. Pomocí funkce select se čeká na událost na socketu přiřazeném příslušnému TCP portu. Po přijetí spojení se vybere volné vlákno určené pro TCP spojení a přidělí se mu přijaté spojení. Po jeho převzetí se hlavní smyčka vrátí na select a čeká na další spojení. Pokud počet aktivních vláken přeteče (podteče) jsou automaticky vytvořena nová (zrušena existující). Obsluhující vlákno nejdříve postupným voláním podmodulů na první až třetí vrstvě rozhodne, na který server se má připojit. Pak zajišťuje přepojování dat, případné vytváření a rušení pomocných spojení, a odesílání příslušných zpráv do statistik.

3.4.1 Management TCP vláken

Počet TCP vláken se řídí několika proměnnými s prefixem CORE_THREADS. Při inicializaci se nejdříve vytvoří pole záznamů o CORE_THREADS_MAX prvcích, které obsahuje informace o jednotlivých vláknech (ukazatel na vlákno, jestli existuje, jestli je právě v činnosti, seznam informací o cachování). Pak se vytvoří CORE_THREADS_INIT vláken, u kterých se nastaví příznak exist na 1 a busy na 0. Vlákna po své inicializaci čekají na podmínkové proměnné na přidělení nového spojení.

Při přijetí nového spojení se u vybraného vlákna změní atribut busy na 1 a provede se obslužení daného spojení. Po jeho ukončení se nastaví atribut busy zpět na 0 a vlákno se opět uspí na podmínkové proměnné, kde čeká na další spojení.

V hlavní smyčce se po přidělení nového spojení kontroluje poměr počtu existujících vláken ku počtu aktuálně pracujících vláken. Pokud tento poměr přesáhne hodnotu CORE_THREADS_FULL, je zavolána funkce thread_increase, která se pokusí počet vláken zvětšit. Pokud poměr klesne pod hodnotu CORE_THREADS_EMPTY, je zavolána funkce thread_decrease, která se pokusí počet vláken zmenšit. Obě proměnné by tedy měly být v intervalu $(0, 1)$.

Funkce thread_increase vynásobí aktuální počet existujících vláken koeficientem CORE_THREADS_RATE_UP, který by měl být větší než jedna. Pokud nový počet přesáhne hodnotu CORE_THREADS_MAX, je na tuto hodnotu snížen. Pak jsou postupně inicializována nová vlákna a přidána jako existující do pole informací o vláknech.

Funkce `thread_decrease` vydělí aktuální počet existujících vláken koeficientem `CORE_THREADS_RATE_DOWN`, který by měl být větší než jedna. Zároveň by měla platit nerovnost $1/\text{CORE_THREADS_RATE_DOWN} \geq \text{CORE_THREADS_EMPTY}$. Pokud je nový počet menší než hodnota `CORE_THREADS_MIN`, je na tuto hodnotu zvýšen. Pak je postupně procházeno pole záznamů o vláknech a jsou rušena vlákna, která právě nejsou v činnosti. Protože při spouštění balanceru je po prvním požadavku počet aktivních vláken jedna a proto by pravděpodobně došlo postupně k jejich snížení až na minimum, je umožněno snížení počtu vláken až po jeho prvním zvyšování.

3.4.2 Přijetí spojení a vybrání cílového serveru

Po probuzení TCP vlákna hlavním vláknem se provede minimum operací, aby hlavní vlákno mohlo co nejrychleji pokračovat v činnosti. Změní se atribut `busy` v tabulce TCP vláken a ověří se jejich aktuální počet, uloží se číslo socketu, portu a protokolu do vnitřních proměnných vlákna a přijme se spojení (`accept`). Poté se pomocí podmínkové proměnné umožní pokračovat hlavnímu vlákně v činnosti.

Po inicializaci proměnných potřebných pro volání funkcí jednotlivých podmodulů a vytvoření kopie seznamu serverů se volá funkce podmodulu na první vrstvě. Pokud je po návratu nastavena funkce podmodulu na druhé vrstvě, projde se tabulka funkcí na druhé vrstvě. Po nalezení se vybraná funkce zavolá, v opačném případě se ji vlákno pokusí nahrát do paměti a uložit do tabulky. Pokud je po návratu z funkce opět nastavena na druhé vrstvě, je zavolána tato funkce. Tím je umožněno jemnější oddělování jednotlivých skupin. Pokud funkce na druhé vrstvě není nastavena nebo se jí nepodaří nahrát do paměti, je volána funkce na třetí vrstvě.

Pokud se funkce podmodulu na třetí vrstvě nenajde v tabulce pro funkce na třetí vrstvě, vlákno se ji opět pokusí nahrát do paměti. Po zavolání a návratu z funkce začne vlákno vybírat cílový server. Na třetí vrstvě může být volána pouze jedna funkce. Pokud funkce není nastavena nebo se jí nepodaří nahrát do paměti, je do logu zaneseno upozornění a přejde se na vybírání serveru.

Pokud je po projití všemi vrstvami nastavená funkce datového podmodulu, pokusí se ho jádro nahrát do paměti a uložit do tabulky pro další použití.

Po projití všemi funkcemi jednotlivých vrstev začne vlákno vybírat cílový server podle kopie seznamu serverů, které mohou jednotlivé funkce modifikovat. Seznam obsahuje položku `priority`, která rozhoduje o pořadí serverů. Seznam se prochází od začátku a vybere se první server, který má nejvyšší priority a je dostupný. Na vybraný server se vlákno pokusí připojit. Pokud

se to nepodaří, pokračuje se v procházení seznamu a vezme se další server s nejvyšší prioritou. Pokud už žádný takový není, prochází se seznam znovu od začátku a vybírá se server s druhou nejvyšší prioritou atd. dokud je jejich priorita větší než nula. U všech serverů, ke kterým se nepodařilo připojit se zvýší počet neúspěšných pokusů a při přesažení hodnoty PING_TTD je server označen za nefunkční. Pokud není vybrán nebo není dostupný žádný server, je spojení uzavřeno a vlákno se připraví do stavu na přijetí dalšího požadavku.

3.4.3 Navázání spojení

Navazování spojení se provádí voláním funkce `connect()`. Ta má v systému nastavena příliš dlouhý timeout, který by při vysokém zatížení mohl zablokovat celý Load Balancer. Proto je k dispozici několik nastavení, která mohou implicitní timeout zkrátit. K nastavení se používají tyto proměnné:

CORE_CONNECT_TIMEOUT Určuje délku timeoutu v milisekundách. Pokud je nastaveno na nulu, použije se autokonfigurace.

CORE_CONNECT_SEND_TO_STATS Pokud je nastaveno, odesílají se informace o úspěchu či neúspěchu do statistik.

CORE_CONNECT_STARTING_TIME Určuje počáteční délku timeoutu při autokonfiguraci.

CORE_CONNECT_NUMBER Určuje počet spojení, které reprezentuje průměrná hodnota.

CORE_CONNECT_RATE Určuje, jakou hodnotou se má vynásobit průměrná hodnota pro nastavení délky timeoutu.

Connect timeout se nastavuje proměnnou `CORE_CONNECT_TIMEOUT`, která určuje délku timeoutu v milisekundách. Pokud je nastavena na nulovou hodnotu, použije pro stanovení délky timeoutu autokonfigurace.

Při autokonfiguraci si jádro počítá průměrnou hodnotu trvání connectu (spojení se serverem). Průměr se počítá z `CORE_CONNECT_NUMBER` hodnot, kde při výpočtu nové hodnoty průměru se použije aktuální hodnota trvání a `CORE_CONNECT_NUMBER-1` hodnot předchozího průměru. Při startu se jako počáteční průměrná hodnota použije `CORE_CONNECT_STARTING_TIME` zadaná v milisekundách. Pro nastavení timeoutu pro spojení se spočtený průměr vynásobí hodnotou proměnné `CORE_CONNECT_RATE`. Pro shrnutí se nová

průměrná hodnota spočítá takto: $new_average_connect_time = \frac{average_connect_time * (CORE_CONNECT_NUMBER - 1) + new_connect_time}{CORE_CONNECT_NUMBER}$ kde `average_connect_time` je průměrný čas předchozích spojení a `new_connect_time` je connet čas aktuálního spojení.

3.4.4 Provádění spojení

V tomto okamžiku je již navázáno spojení s vybraným serverem. Pak se na server odešlou data, která si načetly podmoduly na první až třetí vrstvě. Pokud je vyžadováno použití datového podmodulu, zavolá se před odesláním těchto dat, aby je mohl případně modifikovat. Pak se vlákno zastaví na začátku smyčky, ve které čeká pomocí `selectu` na událost na socketech pro čtení z klienta, ze serveru, případně na vytvoření nebo čtení z pomocných datových spojení vytvořených datovým podmodulem. Pokud je pomocí podmodulů na první až třetí vrstvě nastaveno odesílání informací do statistik, jsou odeslány při těchto událostech:

- při čtení dat z klienta a odesílání na server (pokud se jedná o první data, je součástí informace doba odezvy).
- při čtení dat ze serveru a odesílání na klienta (pokud se jedná o první data, je součástí informace doba odezvy).
- při uzavření spojení ze strany serveru.
- při uzavření spojení ze strany klienta.
- při předávání dat v datovém spojení vytvořeném datovým podmodulem.
- při otevření nového spojení se serverem.

Při ukončení spojení nebo při výskytu chyby je provedeno případné zalogování, uzavření všech socketů, nastavení vlákna do stavu připraveného k použití a přesunu do čekání na signál od hlavního vlákna.

3.5 UDP vlákna

UDP vláken je stejný počet, kolik je otevřených UDP portů a každé obsluhuje svůj vlastní port. Každé vlákno nejdříve provede inicializaci lokálních proměnných (vytvoření vlastní kopie seznamu serverů, vytvoření a inicializace pole s otevřenými sockety, vytvoření a nastavení socketu pro poslouchání na UDP portu, ...).

3.5.1 Management socketů

Protože UDP komunikace není spojovaná, ale probíhá po jednotlivých packetech, dá se u ní rozlišit začátek, ale nedá se rozpoznat konec komunikace (uzavření spojení u TCP spojení). Proto je po vytvoření socketu pro komunikaci určitého klienta s určitým serverem nutné zachovat socket otevřený pro případnou další komunikaci. Tím vzniká problém postupného růstu otevřených socketů. Nelze totiž rozlišit, jestli bude po výměně jednoho packetu spojení dále nevyužíváno nebo po jednom packetu od klienta bude server jednou za den posílat nějaká data.

Tato situace je řešena pomocí pole `udps` o délce `CORE_UDP_CONNECTIONS`, které obsahuje záznam o otevřených spojeních (číslo socketu, adresa serveru a čas posledního packetu). Nejdříve se postupně zaplňuje toto pole a po jeho naplnění se vezme nejstarší spojení (s nejstarším datem posledního packetu), uzavře se a otevře se pro nově příchozí spojení.

3.5.2 Přijetí spojení, vybrání cílového serveru a přenos dat

UDP vlákno čeká pomocí `selectu` na změnu na portu, který je vláknu přiřazen, a na všech otevřených socketech z pole `udps`.

Při události na portu je packet přijat a vlákno zavolá funkci podmodulu na první vrstvě. Pokud je po návratu nastavena funkce podmodulu na druhé vrstvě, projde se tabulka funkcí na druhé vrstvě. Po nalezení se vybraná funkce zavolá, v opačném případě se ji vlákno pokusí nahrát do paměti a uložit do tabulky. Pokud je po návratu z funkce opět nastavena na druhé vrstvě, je zavolána tato funkce. Tím je umožněno jemnější oddělování jednotlivých skupin. Pokud funkce na druhé vrstvě není nastavena nebo se ji nepodaří nahrát do paměti, je volána funkce na třetí vrstvě.

Pokud se funkce podmodulu na třetí vrstvě nenajde v tabulce pro funkce na třetí vrstvě, vlákno se ji opět pokusí nahrát do paměti. Po zavolání a návratu z funkce začne vlákno vybírat cílový server. Na třetí vrstvě může být volána pouze jedna funkce. Pokud funkce není nastavena nebo se ji nepodaří nahrát do paměti, je do logu zaneseno upozornění a přejde se na vybírání serveru.

Po projití všemi funkcemi jednotlivých vrstev začne vlákno vybírat cílový server podle kopie seznamu serverů, které mohou jednotlivé funkce modifikovat. Seznam obsahuje položku priority, která rozhoduje o pořadí serverů. Seznam se prochází od začátku a vybere se první server, který má nejvyšší priority a je dostupný. Na vybraný server vlákno přepoše data. Protože u UDP

přenosu nejde rozeznat, jestli byla data v pořádku doručena, nepokračuje se v případném dalším odesílání dat.

Při události na některém z otevřených spojení se data přečtou a přeošlou se na příslušný server podle adresy uvedené v poli `udps`.

Pokud je pomocí podmodulů na první až třetí vrstvě nastaveno odesílání informací do statistik, jsou odeslány po odeslání dat na server nebo klienta.

3.6 Ping vlákno

Pokud je v konfiguračním souboru nastavena proměnná `PING_ENABLE`, je při inicializaci vytvořeno a spuštěno ping vlákno. Vlákno postupně prochází seznam serverů a na každý z nich volá funkci `ping_server` z knihovny `ping.c`, která podle podle portu daného serveru provede test jeho dostupnosti. Pokud dojde ke změně jeho stavu, je odeslána zpráva statistikám (pokud jsou spuštěny) a příslušně změněna hodnota `alive` v seznamu serverů.

Ping vlákno testování provádí každých `PING_DELAY` milisekund.

Kapitola 4

Podmoduly

V této kapitole bude podrobněji popsána funkce a rozhraní podmodulů na první až třetí vrstvě a datového podmodulu. Podmoduly na první až třetí vrstvě rozhodují o vybrání serveru pro příchozí požadavek. Datový podmodul umožňuje modifikaci dat během spojení a vytváření a zavírání pomocných datových spojení. Podmodul je dll knihovna s příponou .so, která je uložena v adresáři /usr/lib. Každá knihovna obsahuje jednu nebo více funkcí. V jedné knihovně mohou být funkce z více vrstev.

Podmodulům se předávají veškeré informace, které by pro svoje rozhodování mohly použít. Dále pak několik vnitřních struktur jádra, které využívají případné funkce z knihoven, které podmoduly mohou volat (cachování, rozvinutí skupin, ...). Tyto struktury by podmoduly neměly nijak měnit, slouží pouze pro předání potřebných dat pomocným knihovnám.

Návratová hodnota všech funkcí v podmodulech je typu int a měla by nabývat hodnoty -1 při chybě a hodnoty 0 opačně.

V této kapitole je popsáno rozhraní, volání a další informace důležité pro vlastní implementaci podmodulů. Podmoduly, které jsou součástí Load Balanceru jsou popsány v uživatelské dokumentaci.

Tabulka 4.1: Popis struktur a typů používaných v podmodulech.

ENUM_PROTOCOL	výčtový typ definovaný v core.h { TCP, UDP }
ENUM_DM_TYPE	výčtový typ definovaný v core.h { DM_START, DM_CTOS, DM_STOC, DM_CLOSE_CLIENT, DM_CLOSE_SERVER } používá při volání datového modulu, aby se dozvěděl při jaké příležitosti je volán.
pokračování na další straně	

	DM_START - při navázání nového spojení DM_CTOS - při přenosu dat od klienta k serveru DM_STOC - při přenosu dat od serveru ke klientu DM_CLOSE_CLIENT, DM_CLOSE_SERVER - při uzavření spojení ze strany klientu, serveru
struct sockaddr_in	systémová struktura obsahující informace o IP adrese a portu.
struct timeval	systémová struktura obsahující informaci o času tv_sec - počet sekund, tv_usec - počet mikrosekund
t_core_server	struktura definovaná v core.h obsahuje informace o jednotlivých serverech - int id - identifikace serveru - char *nick - textové pojmenování serveru - int priority - priorita serveru, server s největší prioritou bude vybrán jako první - int static_weight - statická váha, určuje statickou výkonnost jednotlivých serverů - int alive - 1 pokud je server dostupný, 0 pokud ne - int ttd (time to death) - určuje kolik pokusů v řadě bylo neúspěšných při pokusu o připojení na daný server - ENUM_PROTOCOL protocol - protokol daného serveru - struct sockaddr_in address - obsahuje adresu serveru
t_core_module	struktura definovaná v core.h obsahuje identifikaci podmodulu - char library[256] - jméno knihovny včetně přípony .so - char function[256] - jméno funkce v knihovně
t_cache_info	struktura definovaná v core.h struktura je určena pro vnitřní použití jádra a knihovny podporující cachování. V podmodulech by se neměla nijak pozměňovat, aby funkce cachování probíhala správně
t_core_groups	struktura definovaná v core.h struktura je určena pro vnitřní použití jádra a knihovny podporující výčet skupin. V podmodulech by se neměla nijak pozměňovat, aby funkce výčtu skupin probíhala správně
t_core_dm_ports	struktura definovaná v core.h struktura je určena pro vnitřní použití jádra a knihovny podporující vytváření pomocných datových spojení. V podmodulech by se neměla nijak pozměňovat, aby funkce
pokračování na další straně	

pro vytváření pomocných datových spojení probíhala správně
--

4.1 Funkce na první vrstvě

Funkce na první vrstvě je vždy pouze jedna a je volána jako první při otevření každého nového spojení. Typicky se rozhoduje podle tabulky pravidel na základě parametrů poskytnutých při volání jádrem (zdrojová adresa/port, cílový port, ...). Hlavička funkce první vrstvy vypadá následovně:

```
int
layer1_function (int thread_id,
                 ENUM_PROTOCOL protocol,
                 struct sockaddr_in * source_address,
                 int *destination_port,
                 struct timeval * timestamp,
                 char *text, t_core_server * servers,
                 const int servers_num,
                 t_core_module_id * module2,
                 t_core_module_id * module3,
                 t_core_module_id * data_module,
                 char *cfgfile, int *query_socket,
                 t_cache_info * cache_info,
                 t_core_groups groups, int *send_to_stats);
```

Tabulka 4.2: Význam jednotlivých parametrů funkce podmodulu na první vrstvě.

typ	název	popis
int	thread_id	identifikace vlákna, které se používá při volání některých knihoven.
ENUM_PROTOCOL	protocol	informace o typu protokolu, proměnná nabývá hodnot TCP, UDP.
struct sockaddr_in *	source_address	informace o IP adrese klienta.
int *	destination_port	číslo portu, na kterém jádro poslouchá a na který bylo provedeno spojení.
struct timeval *	timestamp	časové razítko při vytvoření spojení.
char *	text	do této proměnné se postupně ukládá informace o podmodulech, kterými
pokračování na další straně		

		spojení prošlo. Její hodnota se pak uloží do statistik a lze se pomocí ní dotazovat na určitý typ spojení.
t_core_server *	servers	lokální kopie seznamu (pole) serverů, ve kterém jednotlivé podmoduly mění u jednotlivých serverů jejich prioritu.
const int	servers_num	počet serverů v poli serverů.
t_core_module_id *	module2	struktura, do které může funkce uložit jméno podmodulu na druhé vrstvě.
t_core_module_id *	module3	struktura, do které může funkce uložit jméno podmodulu na třetí vrstvě.
t_core_module_id *	data_module	struktura, do které může funkce uložit jméno datového podmodulu.
char *	cfgfile	ukazatel na řetězec obsahující kopii konfiguračního souboru.
int *	query_socket	číslo socketu, který se používá při dotazování na statistiky.
t_cache_info *	cache_info	struktura jádra, která se předává funkcím na podporu cachování.
t_core_groups	groups	struktura jádra, která se předává funkci pro vracení skupin serverů.
int *	send_to_stats	v tomto parametru může funkce nastavit, jestli je potřeba odesílat informace o spojení do statistik.

Při volání funkce podmodulu na první vrstvě se její parametry naplní podle dostupných informací. Parametry `thread_id`, `protocol`, `source_address`, `destination_port` a `timestamp` se naplní podle aktuálního stavu.

Parametr `text` je při volání prázdný. Funkce by do něho měla zapsat informaci o typu spojení, případně pro jaký výběr serverů se rozhodla. Tuto proměnnou mohou upravit všechny další funkce podmodulů na dalších vrstvách. V jádře je tento text alokovan jako pole, takže pro něj není potřeba alokovat novou paměť. Typicky se do textu zapíše jméno pravidla, které bylo pro dané spojení vybráno (`http`, `ftp`, `ssh`, ...).

Při volání jsou všechny položky struktury `servers` nastaveny podle aktuálního stavu, položka `priority` je nastavena na nulovou hodnotu. Funkce by měla změnit hodnotu `priority` na větší než nula u všech serverů, které

jsou schopné obsloužit daný typ požadavku. Nastavovat lze buď přímo nebo pomocí funkce `get_groups_servers` z knihovny `groups.c`, která rozvine název skupiny na příslušné servery. Počet serverů je nastaven v parametru `servers_num`.

Parametry `module2`, `module3` a `data_module` jsou před voláním nastaveny na prázdné hodnoty. Parametry `module2` a `module3` se nastavují na prázdné hodnoty při každém volání libovolné funkce na všech vrstvách. `Data_module` se nastaví na prázdnou hodnotu pouze před funkcí na první vrstvě a pak se propaguje nižším vrstvám. Podle těchto parametrů se pak volají funkce na dalších vrstvách.

V parametru `cfgfile` je uložena textová podoba konfiguračního souboru, který si daná funkce může načíst s využitím konfigurační knihovny a jejích funkcí.

Parametr `query_socket` obsahuje socket vytvořený pro spojení se statistikami. Předává se odkazem, protože během volání funkcí z knihovny `commun.c` určené pro komunikaci může dojít ke změně socketu. Sama funkce by tento socket měnit neměla.

V parametrech `cache_info` a `groups` jsou uloženy vnitřní data jádra a jsou použity pro předání dat příslušným knihovnám. Funkce by je neměla měnit.

V parametru `send_to_stats` může funkce nastavit, jestli se mají o daném spojení odesílat informace do statistik, hodnota 1 znamená informace odesílat, 0 neodesílat. Implicitně je tento parametr nastaven na 1. Odesílání statistiky zvyšuje vytížení spojení mezi jádrem a statistikami, ale také umožňuje vyhodnocování zátěže jednotlivých serverů, jak pro dotazování funkcemi podmodulů nebo zobrazování statistik a historie na webu.

Součástí Load Balanceru je funkce `common` v knihovně `lib_layer1_main.so`, která je podrobněji popsána v uživatelské sekci dokumentace.

4.2 Funkce na druhé vrstvě

Funkcí na druhé vrstvě může být více a volají se podle nastavení parametru `module2` ve funkci na první vrstvě. Funkce na druhé vrstvě mohou být volány opakovaně, což může sloužit ke zpřesnění určení typu příchozího spojení, nebo nemusí být zavolány vůbec.

Funkce podmodulů na druhé vrstvě dokončují výběr serverů, které jsou schopné obsloužit daný požadavek a z hlediska funkčního obslužení spojení jsou ekvivalentní. Funkce podmodulu na druhé vrstvě typicky pro rozhodování využívá načtení části dat ze spojení (Layer 7 switching) a na jejich základě ponechá nebo omezí skupinu serverů již vybranou funkcí podmodulu na první vrstvě. Hlavička funkce druhé vrstvy vypadá následovně:

```

int
layer2_function (int thread_id,
                 ENUM_PROTOCOL protocol,
                 struct sockaddr_in * source_address,
                 int *destination_port,
                 struct timeval * timestamp,
                 char *text, t_core_server * servers,
                 const int servers_num,
                 t_core_module_id * module2,
                 t_core_module_id * module3,
                 t_core_module_id * data_module,
                 const int socket, char **data,
                 int *data_len, char *cfgfile,
                 int *query_socket,
                 t_cache_info * cache_info,
                 t_core_groups groups, int *send_to_stats);

```

Tabulka 4.3: Význam jednotlivých parametrů funkce podmodulu na druhé vrstvě.

typ	název	popis
int	thread_id	identifikace vlákna, které se používá při volání některých knihoven.
ENUM_PROTOCOL	protocol	informace o typu protokolu, proměnná nabývá hodnot TCP, UDP.
struct sockaddr_in *	source_address	informace o IP adrese klienta.
int *	destination_port	číslo portu, na kterém jádro poslouchá a na který bylo provedeno spojení.
struct timeval *	timestamp	časové razítko při vytvoření spojení.
char *	text	do této proměnné se postupně ukládá informace o podmodulech, kterými spojení prošlo. Její hodnota se pak uloží do statistik a lze se pomocí ní dotazovat na určitý typ spojení.
t_core_server *	servers	lokální kopie seznamu (pole) serverů, ve kterém jednotlivé podmoduly mění u jednotlivých serverů jejich prioritu.
const int	servers_num	počet serverů v poli serverů.

pokračování na další straně

t_core_module_id *	module2	struktura, do které může funkce uložit jméno podmodulu na druhé vrstvě.
t_core_module_id *	module3	struktura, do které může funkce uložit jméno podmodulu na třetí vrstvě.
t_core_module_id *	data_module	struktura, do které může funkce uložit jméno datového podmodulu.
const int	socket	socket spojení, ze kterého si funkce může číst data.
char **	data	parametr, do kterého se uloží všechna načtená data.
int *	data_len	délka načtených dat.
char *	cfgfile	ukazatel na řetězec obsahující kopii konfiguračního souboru.
int *	query_socket	číslo socketu, který se používá při dotazování na statistiky.
t_cache_info *	cache_info	struktura jádra, která se předává funkcím na podporu cachování.
t_core_groups	groups	struktura jádra, která se předává funkci pro vrácení skupin serverů.
int *	send_to_stats	v tomto parametru může funkce nastavit, jestli je potřeba odesílat o spojení informace do statistik.

Při volání funkce podmodulu na druhé vrstvě se její parametry naplní podle dostupných informací. Parametry `thread_id`, `protocol`, `source_address`, `destination_port` a `timestamp` se naplní podle aktuálního stavu.

Parametr `text` při volání obsahuje řetězec zapsaný funkcí podmodulu na první vrstvě. Funkce by do něho měla zapsat upřesnění informace o typu spojení, případně pro jaký výběr serverů se rozhodla. Tuto proměnnou mohou upravit všechny další funkce podmodulů na dalších vrstvách. V jádře je tento text alokovan jako pole, takže není potřeba pro něj alokovat novou paměť. Typicky se do textu přes dvojtečku připojí jméno pravidla, které upřesňuje dané spojení (`http:html`, `http:php`, `http:bin.data`, ...).

Při volání jsou všechny položky struktury `servers` nastaveny podle aktuálního stavu, položka `priority` je nastavena podle hodnot vrácených funkcí podmodulu na první vrstvě. Funkce by měla změnit hodnotu `priority` na větší než nula u všech serverů, které jsou schopné obsloužit daný typ požadavku. Nastavovat lze buď přímo nebo pomocí funkce `get_groups_servers`

z knihovny `groups.c`, která rozvine název skupiny na příslušné servery. Typicky funkce ponechá nebo zúží (opětovným nastavením nuly v `priority`) výběr serverů provedených na první vrstvě. Počet serverů je nastaven v parametru `servers_num`.

Parametry `module2`, `module3` jsou před voláním nastaveny na prázdné hodnoty. `Data_module` je nastavena podle funkce podmodulu na první vrstvě.

Parametry `socket`, `data` a `data_len` slouží pro načtení a uložení dat z přijímaného spojení. `Socket` je před prvním voláním funkce na druhé vrstvě ve stavu po přijetí spojení vláknem a je připraven na čtení. Parametr `data` je při zahájení spojení nastaven na hodnotu '0'

a `data_len` na nula. Pokud již byla dříve volána funkce na druhé vrstvě, měla by být uložena v parametru `data`. Funkce by měla veškerá načtená data připojit k parametru `data` a příslušně změnit parametr `data_len`. Pro připojení dat je nutné naalokovat novou paměť. Po průchodu všech funkcí na všech vrstvách jádro zapíše data na klienta.

V parametru `cfgfile` je uložena textová podoba konfiguračního souboru, který si daná funkce může načíst s využitím konfigurační knihovny a jejích funkcí.

Parametr `query_socket` obsahuje socket vytvořený pro spojení se statistikami. Předává se odkazem, protože během volání funkcí z knihovny `commun.c` určené pro komunikaci může dojít ke změně socketu. Sama funkce by tento socket měnit neměla.

V parametrech `cache_info` a `groups` jsou uloženy vnitřní data jádra a jsou použity pro předání dat příslušným knihovnám. Funkce by je neměla měnit.

V parametru `send_to_stats` může funkce nastavit, jestli se mají o daném spojení odesílat informace do statistik, hodnota 1 znamená informace odesílat, 0 neodesílat. Implicitně je tento parametr nastaven na 1. Odesílání statistických informací zvyšuje vytížení spojení mezi jádrem a statistikami, ale také umožňuje vyhodnocování zátěže jednotlivých serverů, jak pro dotazování funkcemi podmodulů nebo zobrazování statistik a historie na webu.

Součástí `Load Balanceru` je funkce `http_headers` v knihovně `http_layer2.so`, která je podrobněji popsána v uživatelské sekci dokumentace.

4.3 Funkce na třetí vrstvě

Funkcí na třetí vrstvě je více a volají se podle nastavení parametru `module3` ve funkci na první nebo druhé vrstvě. Funkce na třetí vrstvě je zavolána nejvýše jednou.

Funkce podmodulů na třetí vrstvě určují pořadí serverů, které byly

vybrány na předchozích vrstvách. Funkce podmodulu na třetí vrstvě se typicky rozhoduje na základě výkonnosti serverů (statická váha, aktuální zatížení, ...), podle informací o spojení (timestamp, zdrojová adresa, ...) nebo použije nějaký algoritmus (round robin, random access, hash IP address, ...). Hlavička funkce třetí vrstvy vypadá následovně:

```
int
layer3_function (int thread_id,
                 ENUM_PROTOCOL protocol,
                 struct sockaddr_in * source_address,
                 int *destination_port,
                 struct timeval * timestamp,
                 char *text, t_core_server * servers,
                 const int servers_num,
                 char *cfgfile,
                 int *query_socket,
                 t_cache_info * cache_info,
                 t_core_groups groups,
                 int *send_to_stats);
```

Tabulka 4.4: Význam jednotlivých parametrů funkce podmodulu na třetí vrstvě.

typ	název	popis
int	thread_id	identifikace vlákna, které se používá při volání některých knihoven.
ENUM_PROTOCOL	protocol	informace o typu protokolu, proměnná nabývá hodnot TCP, UDP.
struct sockaddr_in *	source_address	informace o IP adrese klienta.
int *	destination_port	číslo portu, na kterém jádro poslouchá a na který bylo provedeno spojení.
struct timeval *	timestamp	časové razítko při vytvoření spojení.
char *	text	do této proměnné se postupně ukládá informace o podmodulech, kterými spojení prošlo. Její hodnota se pak uloží do statistik a lze se pomocí ní dotazovat na určitý typ spojení.
t_core_server *	servers	lokální kopie seznamu (pole) serverů, ve kterém jednotlivé podmoduly mění
pokračování na další straně		

		u jednotlivých serverů jejich prioritou.
const int	servers_num	počet serverů v poli serverů.
char *	cfgfile	ukazatel na řetězec obsahující kopii konfiguračního souboru.
int *	query_socket	číslo socketu, který se používá při dotazování na statistiky.
t_cache_info *	cache_info	struktura jádra, která se předává funkcím na podporu cachování.
t_core_groups	groups	struktura jádra, která se předává funkci pro vracení skupin serverů.
int *	send_to_stats	v tomto parametru může funkce nastavit, jestli je potřeba o spojení odesílat informace do statistik.

Při volání funkce podmodulu na třetí vrstvě se její parametry naplní podle dostupných informací. Parametry `thread_id`, `protocol`, `source_address`, `destination_port` a `timestamp` se naplní podle aktuálního stavu.

Parametr `text` při volání obsahuje řetězec zapsaný funkcemi podmodulů na první a druhé vrstvě. Funkce by do něho měla zapsat upřesnění informace o typu spojení, případně pro jaký výběr serverů se rozhodla. V jádru je tento text alokovan jako pole, takže není potřeba pro něj alokovat novou paměť. Typicky se do textu na třetí vrstvě nepřipisuje nic, ale může se využít v rozhodovacím algoritmu při dotazu na statistiky.

Při volání jsou všechny položky struktury `servers` nastaveny podle aktuálního stavu. Položka `priority` je nastavena podle hodnot vrácených funkcí podmodulů na první a druhé vrstvě. Funkce typicky již nemění seznam vybraných serverů, ale mění položku `priority` u již vybraných. Jádro pak přešle požadavek serveru s nejvyšší prioritou. Nastavovat lze buď přímo nebo pomocí funkce `get_groups_servers` z knihovny `groups.c`, která rozvine název skupiny na příslušné servery. Počet serverů je nastaven v parametru `servers_num`.

V parametru `cfgfile` je uložena textová podoba konfiguračního souboru, který si daná funkce může načíst s využitím konfigurační knihovny a jejích funkcí.

Parametr `query_socket` obsahuje socket vytvořený pro spojení se statistikami. Předává se odkazem, protože během volání funkcí z knihovny `commun.c` určené pro komunikaci může dojít ke změně socketu. Sama funkce by tento socket měnit neměla.

V parametrech `cache_info` a `groups` jsou uloženy vnitřní data jádra a jsou

použity pro předání dat příslušným knihovnám. Funkce by je neměla měnit.

V parametru `send_to_stats` může funkce nastavit, jestli se mají o daném spojení odesílat informace do statistik, hodnota 1 znamená informace odesílat, 0 neodesílat. Implicitně je tento parametr nastaven na 1. Odesílání statistikckých informací zvyšuje vytížení spojení mezi jádrem a statistikami, ale také umožňuje vyhodnocování zátěže jednotlivých serverů, jak pro dotazování funkcemi podmodulů nebo zobrazování statistik a historie na webu.

Součástí Load Balanceru jsou funkce `basic_round_robin`, `random_select`, `hash_IP_select`, `self_weighted_round_robin` a `static_select_first` v knihovně `static_layer3.so` a funkce `minimal_server_response`, `minimal_average_server_response`, `least_open_connections`, `weighted_least_open_connections`, `weighted_minimal_sessions_count`, `weighted_minimal_flow_from_server`, `predictive_minimal_server_response`, `predictive_minimal_flow_from_server` a `predictive_minimal_sessions_count_per_second` v knihovně `dynamic_layer3.so`, které jsou podrobněji popsány v uživatelské sekci dokumentace.

4.4 Funkce na datové vrstvě

Funkcí na datové vrstvě může být více a volají se podle nastavení parametru `data_module` ve funkci na první nebo druhé vrstvě. Pokud je funkce na datové vrstvě nastavena, je volána při každém přeposílání dat od klienta k serveru nebo obráceně.

Funkce podmodulů na datové vrstvě řídí samotný přenos dat. Při přijetí dat od klienta (serveru) je zavolána funkce datového podmodulu a jsou jí předána načtená data. Funkce je může upravit nebo na jejich základě otevřít nebo zavřít pomocná datová spojení a to ve směru od klienta k serveru i obráceně. Po ukončení funkce jsou jí změněná data přeposílána serveru (klientu).

```
int
data_layer_function (ENUM_DM_TYPE type,
                    int thread_id, int client_socket,
                    int server_socket, int service_port,
                    struct sockaddr_in client_addr,
                    struct sockaddr_in server_addr,
                    struct sockaddr_in balancer_inner_addr,
                    struct sockaddr_in balancer_outer_addr,
                    char *data, int *data_len,
                    t_core_dm_ports *core_ports);
```

Tabulka 4.5: Význam jednotlivých parametrů funkce podmodulu na datové vrstvě.

typ	název	popis
ENUM_DM_TYPE	type	informace o stavu, ve kterém se volá funkce datového podmodulu.
int	thread_id	identifikace vlákna, které se používá při volání některých knihoven.
int	client_socket	socket hlavního spojení ke klientovi.
int	server_socket	socket hlavního spojení k serveru.
int	service_port	číslo portu, na kterém jádro poslouchá danou službu.
struct sockaddr_in *	client_address	informace o IP adrese klienta.
struct sockaddr_in *	server_address	informace o IP adrese serveru.
struct sockaddr_in *	balancer_inner_address	informace o IP adrese Load Balanceru, která patří do vnitřní sítě.
struct sockaddr_in *	balancer_outer_address	informace o IP adrese Load Balanceru, která patří do vnější sítě.
char *	data	načtená data, která funkce může pozměnit.
int *	data_len	délka načtených dat.
t_core_dm_ports *	core_ports	struktura jádra, která se předává funkci pro otevírání a zavírání pomocných datových spojení.

Při volání funkce podmodulu na datové vrstvě se její parametry naplní podle dostupných informací. Parametr type se vyplní podle místa, ze kterého je funkce volána. Parametry thread_id, client_socket, server_socket a service_port se nastaví podle aktuálního spojení.

Adresa client_address je adresa, ze které přichází spojení od klienta. Adresa server_address je adresa serveru, na který se přeposílá spojení. Adresy balancer_inner_address a balancer_outer_address jsou adresy načtené z konfiguračního souboru, které určují rozhraní do vnitřní sítě (farma serverů a ostatní součásti Load Balanceru) a do vnější sítě (Internet a sítě s klienty).

V parametru data jsou uložena načtená data z klienta nebo serveru, která může funkce datového podmodulu měnit předtím, než budou přeposlána. Při změně velikosti dat se musí příslušně změnit hodnota data_len, která udává velikost dat. Pokud je typ spojení DM_START (při otevření nového spojení), DM_CLOSE_CLIENT (uzavření spojení ze strany klienta) nebo DM_CLOSE_SERVER (uzavření spojení ze strany serveru) jsou obě hodnoty data i data_len rovny hodnotě NULL.

Parametr `core_ports` obsahuje vnitřní data jádra, které jsou použity pro předání dat knihovně `port.c`. Funkce by je neměla měnit.

Součástí Load Balanceru je funkce `ftp` v knihovně `data_layer.so`, která umožňuje používání pomocných datových spojení v protokolu `ftp` v aktivním i pasivním režimu. Podrobněji je popsána v uživatelské sekci dokumentace.

4.5 Vytvoření vlastního podmodulu

Load Balancer umožňuje používání základních již vytvořených podmodulů na všech vrstvách. Pro méně časté a nestandardní protokoly a požadavky je možné implementovat vlastní podmoduly a funkce na všech vrstvách. Nový podmodul je `dll` knihovna, která se přeloží a umístí do adresáře `/usr/lib`. Každá knihovna může obsahovat více funkcí, které musí dodržovat hlavičky popsané výše. Pro uplatnění nových funkcí se musí příslušně upravit konfigurační soubor.

Na různých vrstvách lze používat různé funkce z knihoven, které spolupracují s jádrem nebo statistikami. Zde je uveden pouze stručný přehled, podrobnější informace o funkcích je uveden v kapitole knihovny. Knihovny se nacházejí v adresáři `src/lib/`.

cache.h knihovna pro použití funkce `cache`. Každý podmodul si může vytvořit více `cache`.

create_cache - vytvoření nové `cache`

query_cache - položení dotazu na `cache`

update_cache - žádost o aktualizování položky v jádru

commun.h knihovna pro dotazování statistik.

commun_query_3rdlayer_to_stat - funkce která umožní podmodulům položit dotaz statistikám a vrátí odpověď, pokud dříve nevyprší `timeout`

groups.h knihovna pro převod jmen na seznam serverů

get_group_servers - funkce, která vrací podle jména skupiny nebo serveru seznam serverů

port.h knihovna pro otevírání a zavírání pomocných datových spojení

open_port - otevře nové datové spojení v určeném směru

close_port - zavře určené datové spojení

close_all_ports - zavře všechna pomocná datová spojení

4.5.1 Příklad vytvoření podmodulů http protokolu

V následujícím příkladu je popsáno vytvoření potřebných funkcí pro balancování http protokolu. Tento protokol je součástí základních knihoven Load Balanceru a zde jsou popsány části kódu ukazující využití různých knihoven Load Balanceru.

Nejdříve je třeba upravit první vrstvu balanceru. Základní funkce `common` je postačující a proto je třeba pouze rozšířit její pravidla v konfiguračním souboru. Přidá se pravidlo, které při příchozím požadavku na port 80 zavolá funkci `http_headers` v knihovně `http_layer2.c`. Tato funkce je na druhé vrstvě a bude se starat o balancování na aplikační vrstvě podle obsahu příchozích dat.

Nejdříve se vytvoří sekce `<HTTP_SUBMODULE>` v konfiguračním souboru, kde se uloží např. parametry cache, velikosti používaných polí, pravidla pro samotné balancování podle obsahu. Po načtení se data předzpracují. V každém pravidle je položka `servers`, která obsahuje skupinu nebo server, který se má vrátit v případě, že dané pravidlo bude úspěšné. Během předzpracování se tento název převede na pole typu `int`, kde 1 znamená, že skupina server obsahuje, 0 naopak. Pro převod se použije funkce `get_group_servers`. Pokud je nastaveno použití cache, zavolá se funkce `create_cache` s příslušnými parametry, která cache vytvoří. Hodnoty se načtou a předzpracují na začátku funkce a pouze při prvním volání, aby se tím později nezdržoval běh programu.

Pak se ze spojení načtou data o potřebné velikosti (obsahující celou hlavičku). Načtená data se uloží do proměnné data, přes kterou se předají jádru. Hlavička se zpracuje a rozloží na jednotlivé části. Pak se postupně procházejí pravidla z konfigu a kontroluje se, jestli data danému pravidlu odpovídají. V této fázi seznam serverů odpovídá skupině serverů odpovídající vybranému pravidlu. Pokud je nastaveno použití cache, nastaví se podle konfigu potřebné parametry a zavolá se funkce `query_cache`. Pokud se v cache najde příslušný záznam, vrátí funkce číslo serveru s posledním přístupem. Funkcí `update_cache` se zažádá jádro, aby po ukončení výběru serverů aktualizovalo příslušnou položku v cache podle nové hodnoty.

Součástí pravidla je jméno knihovny a funkce na třetí vrstvě, která se má použít pro další balancování. Další funkce pro http protokol už není potřeba.

4.5.2 Příklad vytvoření podmodulů ftp protokolu

V následujícím příkladu je popsáno vytvoření potřebných funkcí pro balancování ftp protokolu. Tento protokol je součástí základních knihoven Load Balanceru a zde jsou popsány části kódu ukazující využití různých knihoven

Load Balanceru.

Nejdříve je třeba upravit první vrstvu balanceru. Základní funkce `common` je postačující a proto je třeba pouze rozšířit její pravidla v konfiguračním souboru. Přidá se pravidlo, které při příchozím požadavku na port 80 zavolá příslušnou funkci na třetí vrstvě balanceru. Funkce na druhé vrstvě není potřeba. Je ale nutné přidat funkci na datové vrstvě - v knihovně `data_layer.c` funkce `ftp`. Tato funkce se stará o vytváření pomocných datových spojení.

Funkce `ftp` se poprvé zavolá při vytvoření spojení. Podle standardu protokolu `ftp` se otevře port pro datové spojení na portu číslo služby - 1. Při dalších volání, ve kterých se přenášejí data v obou směrech, se kontroluje jejich obsah. Pokud ve směru od klienta k serveru funkce najde příkaz `PORT`, otevře nové datové spojení pomocí funkce `open_port` na portu určeném příkazem `PORT`. Pro server se musí změnit oba parametry - adresa i port. Funkce je příslušně upraví a nová data vrátí jádru, které je přepošle vybranému serveru. V opačném směru funkce čeká na zprávu číslo 227, která je odpovědí na žádost klienta o přenos v pasivním režimu. Funkce otevře nové datové spojení pomocí funkce `open_port` a změní příslušné parametry v přeposílaných datech. Upravená data vrátí jádru, které je odešle klientovi.

Zavírání je v tomto případě provedeno automaticky jádrem, které při uzavření jednoho konce pomocného datového spojení uzavře i druhý konec. Při uzavření hlavního spojení jádro uzavře všechna pomocná datová spojení.

4.5.3 Vytvoření vlastního dynamického algoritmu

Pro vytvoření vlastního dynamického algoritmu je potřeba napsat novou funkci se všemi parametry, které mají podmoduly třetí vrstvy (viz. část Funkce na třetí vrstvě). Všechny funkce implementující dynamické algoritmy jsou umístěny v souboru `dynamic_layer3.c`. Servery, které vyhovují předchozím pravidlům Load Balanceru, mají kladnou hodnotu atributu *priority* ve struktuře `t_core_server`. Seznam struktur pro všechny servery je jedním z parametrů podmodulu třetí vrstvy. Pro všechny takové servery je potřeba sestavit dotaz(y) na nějaké statistické hodnoty, podle kterých se bude rozhodovat, na který server bude nejvýhodnější daný požadavek poslat. Rozsah dotazovaných dat je dvojího typu, buď posledních X požadavků nebo posledních X sekund. Seznam všech možných hodnot je definován výčtovým typem `t_commun_query_range`. Typy hodnot dotazovaných dat jsou dány kombinacemi parametrů definovanými výčtovými typy `t_commun_query_value_type` a `t_commun_query_info_type`, přičemž možné kombinace jsou následující:

- *MAX RESPONSE* – Maximální odezva serveru

- *MAX RESPONSE_TIME* – Před kolika sekundami došlo k maximální odezvě serveru
- *MAX FLOW* – Maximální tok dat
- *MAX FLOW_TIME* – Před kolika sekundami došlo k maximálnímu toku dat
- *MAX FLOW_FROM_SERVER* – Maximální tok dat od serveru ke klientům
- *MAX FLOW_FROM_SERVER_TIME* – Před kolika sekundami došlo k maximálnímu toku dat od serveru ke klientům
- *MAX FLOW_FROM_CLIENT* – Maximální tok dat od klientů k serveru
- *MAX FLOW_FROM_CLIENT_TIME* – Před kolika sekundami došlo k maximálnímu toku dat od klientů k serveru
- *MAX CONNECTIONS_PER_SECOND* – Maximální počet nových spojení navázaných se serverem během jedné sekundy
- *MAX CONNECTIONS_TIME* – Před kolika sekundami došlo k navázání maximálního počtu nových spojení se serverem během jedné sekundy
- *MAX REQUESTS_PER_SECOND* – Maximální počet požadavků zpracovaných během jedné sekundy
- *MAX REQUESTS_TIME* – Před kolika sekundami došlo ke zpracování maximálního počtu požadavků během jedné sekundy
- *MAX CONNECT_TIME* – Maximální doba trvání navázání spojení se serverem
- *MIN RESPONSE* – Minimální odezva serveru
- *MIN RESPONSE_TIME* – Před kolika sekundami došlo k minimální odezvě serveru
- *MIN FLOW* – Minimální tok dat
- *MIN FLOW_TIME* – Před kolika sekundami došlo k minimálnímu toku dat

- *MIN FLOW_FROM_SERVER* – Minimální tok dat od serveru ke klientům
- *MIN FLOW_FROM_SERVER_TIME* – Před kolika sekundami došlo k minimálnímu toku dat od serveru ke klientům
- *MIN FLOW_FROM_CLIENT* – Minimální tok dat od klientů k serveru
- *MIN FLOW_FROM_CLIENT_TIME* – Před kolika sekundami došlo k minimálnímu toku dat od klientů k serveru
- *MIN CONNECTIONS_PER_SECOND* – Minimální počet nových spojení navázaných se serverem během jedné sekundy
- *MIN CONNECTIONS_TIME* – Před kolika sekundami došlo k navázání minimálního počtu nových spojení se serverem během jedné sekundy
- *MIN REQUESTS_PER_SECOND* – Minimální počet požadavků zpracovaných během jedné sekundy
- *MIN REQUESTS_TIME* – Před kolika sekundami došlo ke zpracování minimálního počtu požadavků během jedné sekundy
- *MIN CONNECT_TIME* – Minimální doba trvání navázání spojení se serverem
- *AVG RESPONSE* – Průměrná odezva serveru
- *AVG FLOW* – Průměrný tok dat
- *AVG FLOW_FROM_SERVER* – Průměrný tok dat od serveru ke klientům
- *AVG FLOW_FROM_CLIENT* – Průměrný tok dat od klientů k serveru
- *AVG CONNECTIONS_PER_SECOND* – Průměrný počet nových spojení navázaných se serverem během jedné sekundy
- *AVG REQUESTS_PER_SECOND* – Průměrný počet požadavků zpracovaných během jedné sekundy
- *AVG CONNECT_TIME* – Průměrná doba trvání navázání spojení se serverem
- *VAL CONNECTIONS* – Počet nových spojení navázaných se serverem

- *VAL REQUESTS* – Počet zpracovaných požadavků
- *VAL OPEN_CONNECTIONS* – Počet aktuálně otevřených spojení se serverem

Zpracování dotazů není časově zanedbatelné, proto je tu možnost volby neprovádět dotazy při každém zpracovávaném požadavku. Pro tyto účely je možné použít funkci *init_rates_from_config*, kterou se při prvním použití daného algoritmu načtou z konfiguračního souboru parametry *request_rate* a *milisec_rate*, ve kterých si uživatel definuje po kolika zpracovaných požadavcích nebo po kolika milisekundách se znovu dotázat a získat tak nová data. Po odeslání dotazu a přijetí odpovědi pomocí funkce *commun_query_3rdlayer_to_stat* z komunikační knihovny je třeba podle hodnot odpovědí setřídit priority seznamu serverů. Server, který je podle přijatých hodnot nejméně zatížený, by měl dostat přiřazenu největší prioritu.

Kapitola 5

Statistický modul

5.1 Úvod

Statistický modul je ta část Load Balanceru, která se stará o uchovávání rozmanitých statistických dat a atributů popisujících práci Load Balanceru. Dále také zajišťuje poskytování informací na základě těchto statistických dat ostatním částem Load Balanceru.

Data jsou ukládána do několika různých úložišť, z nichž největším je libovolný databázový server podporující normu SQL99, kam se ukládají převážně data pro informativní, historické a archivní účely. Než se data uloží do databáze, jsou uchovávána a agregována do záznamů o jednotlivých sekundách běhu v tzv. databázovém bufferu, odkud se po uživatelem stanovené době přesouvají do databáze. Třetí možností uložení dat je buffer v paměti, kam se ukládají statistická data jednotlivě. Tento buffer se používá převážně pro rychlé získání informací o posledním průběhu balancování a tyto informace jsou pak využity při následném balancování.

Veškerá funkčnost statistického modulu je obstarána třemi vlákny. První z nich zajišťuje samotné nastartování modulu, inicializaci všech globálních proměnných a struktur, připojení k databázi a spuštění dalších dvou vláken. Druhé vlákno se stará o příjem jednotlivých údajů o běhu od jádra a o jejich ukládání. Třetí vlákno má na starosti příjem dotazů na statistické informace od různých dalších modulů, následné zpracování těchto dotazů a odeslání získaných odpovědí.

5.2 Databáze

5.2.1 Práce s databází pomocí ODBC rozhraní

Veškerá komunikace s databází byla implementována pomocí rozhraní ODBC, proto lze k ukládání dat ze statistického modulu použít libovolný databázový server podporující normu SQL99. ODBC je standardní aplikační rozhraní, které poskytuje klientovi jednotný přístup k datům nezávisle na tom, jakým systémem řízení báze dat jsou tato data spravována. Toto rozhraní se skládá ze čtyř vrstev:

- aplikační – Pokud aplikace potřebuje data z databáze, pak provede volání SQL příkazu pomocí ODBC funkcí.
- spravující ODBC ovladače – Úkolem této vrstvy je zajistit propojení mezi aplikací a příslušným ODBC ovladačem (ODBC ovladače tvoří třetí vrstvu, podrobněji viz dále). Jakmile aplikace potřebuje data, správce ovladačů vyhledá a nahraje příslušný ovladač ve formě dynamické knihovny. Dále zjistí jaké konkrétní funkce jsou podporovány jednotlivými ovladači a uschová si jejich adresy v paměti do tabulky. V případě, že aplikace volá konkrétní funkci, pak tato vrstva zjistí, ke kterému ovladači funkce patří a zavolá ji. Tímto způsobem může být prováděn souběžný přístup k více ovladačům, což se hodí v případě programování aplikací přistupujících souběžně k několika zdrojům dat.
- ODBC ovladače – Provedou zpracování volané ODBC funkce, přeložení požadavku do SQL pro příslušný systém řízení báze dat a jeho následné poslání.
- systém řízení báze dat – Provede zpracování operace požadované ODBC ovladačem a výsledky této operace mu vrátí.

5.2.2 Filosofie ukládání dat

Statistická data jsou ukládána do databáze jako agregované údaje vztahující se k danému časovému intervalu, který je dlouhý minimálně jednu sekundu. Takový způsob ukládání byl navržen z několika důvodů:

- Pokud by se ukládala data o všech událostech jednotlivě, vznikl by obrovský počet záznamů, který by bylo velice obtížné uložit a následně v něm vyhledávat. Pokud by například balancer zpracovával průměrně tisíc požadavků za vteřinu, každý den by se uložilo 86400000 záznamů o zpracovaných požadavcích o celkové velikosti přibližně 8GB.

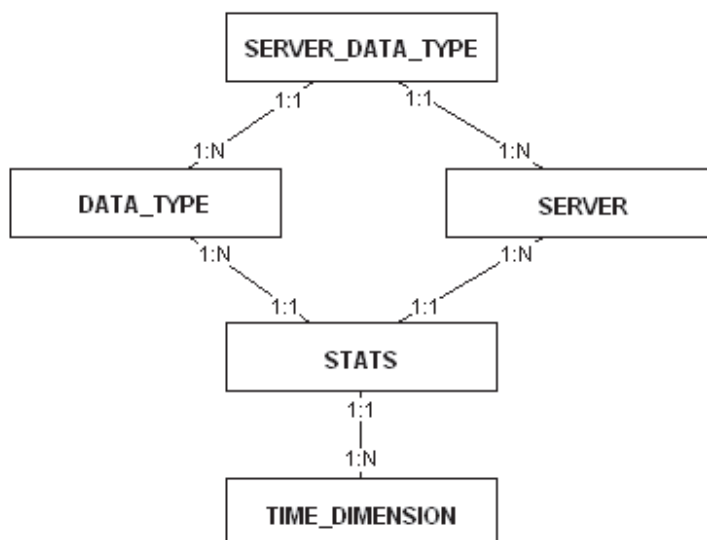
- Pro balancování jsou potřebná hlavně přesná a podrobná data o aktuální zátěži serverů, proto je zbytečné uchovávání těchto dat po dlouhou dobu a pro taková data stačí jako úložiště použít buffer v paměti.
- Pro informování uživatelů o aktuální zátěži dostačují změny po vteřinových intervalech, v přesnějších údajích by se uživatel nemusel tak snadno orientovat.

I tak by docházelo k ukládání velkého množství záznamů, proto byl předchozí myšlenkový postup ještě jednou zopakován a data jsou dále po uplynutí stanovené doby agregována a přesouvána do záznamů vztahujícím se k větším časovým intervalům. Seznam všech časových intervalů, do kterých je možné data agregovat, je definován ve struktuře *t_commun_time_aggregation_level* a je následující: *sekunda*, *minuta*, *hodina*, *den*, *měsíc* a *rok*. Pro každý z těchto časových intervalů je v konfiguračním souboru použit jeden parametr, který určuje minimální dobu trvání dat agregovaných za daný časový interval než může dojít k další agregaci a přesunu těchto dat do většího intervalu. Tyto přesuny pokračují dokud data nejsou agregována do největšího časového intervalu. Seznam těchto parametrů je následující:

- *STATS_MIN_SECOND_DATA_DURATION* – minimální doba uváděná v minutách, po kterou budou uložena data agregovaná v sekundových intervalech
- *STATS_MIN_MINUTE_DATA_DURATION* – minimální doba uváděná v hodinách, po kterou budou uložena data agregovaná v minutových intervalech
- *STATS_MIN_HOUR_DATA_DURATION* – minimální doba uváděná ve dnech, po kterou budou uložena data agregovaná v hodinových intervalech
- *STATS_MIN_DAY_DATA_DURATION* – minimální doba uváděná v měsících, po kterou budou uložena data agregovaná v denních intervalech
- *STATS_MIN_MONTH_DATA_DURATION* – minimální doba uváděná v letech, po kterou budou uložena data agregovaná v měsíčních intervalech

Pokud uživatel nechce využít databázi, může v konfiguračním souboru nastavit hodnotu proměnné *STATS_USE_DATABASE* na "NO". Při této volbě se data ukládají pouze do bufferu databáze a pokud je nastaven parametr *STATS_QUERY_MODE* na hodnotu "memory", pak i do bufferu v paměti. Při nastavení proměnné *STATS_USE_DATABASE* na hodnotu "YES" jsou data z bufferu databáze přesouvána do databáze. Pro určení databáze, uživatelského schématu a hesla pro přihlášení jsou použity parametry *STATS_DB_NAME*, *STATS_DB_USER* a *STATS_DB_PASSWORD* z konfiguračního souboru.

5.2.3 Datový a databázový model



Obrázek 5.1: Datový model

time_dimension

<i>id_time</i>	Primární klíč
<i>timestamp</i>	Časové razítko určující začátek časového intervalu
<i>dimension_level</i>	Identifikátor délky časového intervalu z <i>t_commun_time_aggregation_level</i>

stats

<i>id_stats</i>	Primární klíč
<i>id_time</i>	Cizí klíč do tabulky <i>time_dimension</i> určující časový interval, za který jsou data agregována
<i>id_server</i>	Cizí klíč do tabulky <i>server</i> určující server, ke kterému jsou data vztažena
<i>id_data_type</i>	Cizí klíč do tabulky <i>data_type</i> určující datový typ, ke kterému jsou data vztažena
<i>requests_count</i>	Počet zpracovaných požadavků
<i>max_requests_per_second</i>	Maximální počet požadavků zpracovaných za sekundu
<i>min_requests_per_second</i>	Minimální počet požadavků zpracovaných za sekundu
<i>session_count</i>	Počet nově otevřených spojení
<i>max_sessions_per_second</i>	Maximální počet nově otevřených spojení za sekundu
<i>min_sessions_per_second</i>	Minimální počet nově otevřených spojení za sekundu
<i>recieved_data</i>	Množství dat přijatých od klientů (v Bytech)
<i>send_data</i>	Množství dat odeslaných od serveru klientům (v Bytech)
<i>max_flow</i>	Maximální množství dat, které prošlo za jednu sekundu (v Bytech)
<i>min_flow</i>	Minimální množství dat, které prošlo za jednu sekundu (v Bytech)
<i>max_flow_from_client</i>	Maximální množství dat, které přišlo od klientů za jednu sekundu (v Bytech)
<i>min_flow_from_client</i>	Minimální množství dat, které přišlo od klientů za jednu sekundu (v Bytech)
<i>max_flow_from_server</i>	Maximální množství dat, které přišlo od serveru za jednu sekundu (v Bytech)
<i>min_flow_from_server</i>	Minimální množství dat, které přišlo od serveru za jednu sekundu (v Bytech)
<i>avg_response</i>	Průměrná odezva serveru (v mikrosekundách)
<i>max_response</i>	Maximální odezva serveru (v mikrosekundách)
<i>min_response</i>	Minimální odezva serveru (v mikrosekundách)
<i>avg_connect_time</i>	Průměrná doba pro vytvoření spojení se serverem (v mikrosekundách)
<i>max_connect_time</i>	Maximální doba pro vytvoření spojení se serverem (v mikrosekundách)
<i>min_connect_time</i>	Minimální doba pro vytvoření spojení se serverem (v mikrosekundách)

server

<i>id_server</i>	Primární klíč
<i>ip</i>	IP adresa serveru
<i>port</i>	Port serveru

data_type

<i>id_data_type</i>	Primární klíč
<i>name</i>	Název datového typu, který je specifikován řetězcem typů od nejobecnějšího k nejkonkrétnějšímu a tyto typy jsou odděleny dvojtečkou. Např. <i>http:php</i>

server_data_type

<i>id_server</i>	Cizí klíč do tabulky <i>server</i>
<i>id_data_type</i>	Cizí klíč do tabulky <i>data_type</i>

5.3 Databázový buffer

Databázový buffer je použit pro agregování informativních dat přicházejících od modulu jádra do sekundových intervalů, které jsou poté z tohoto bufferu přesouvány do databáze. Každý server má svůj vlastní databázový buffer, který je implementován jako obousměrný kruhový spojový seznam a je stříděn podle času od nejnovějších po nejstarší intervaly. Prvky tohoto seznamu tvoří jednotlivé sekundové intervaly s agregovanými daty a pokud dojde k zaplnění celého bufferu, začnou se další data ukládat na místo nejstarších intervalů, které jsou předtím zapsány do databáze. Pokud má dojít k přesunu nějakého intervalu z databázového bufferu do databáze, jsou zkontrolovány databázové buffery všech serverů a všechny intervaly s daty staršími nebo rovnými stáří přesouvaného intervalu jsou také přesunuty do databáze. Tím je zaručeno, že pokud přestanou chodit požadavky o nějakém serveru, tak se poslední data o daném serveru stejně dostanou do databáze a nezůstanou v bufferu. To by totiž působilo problémy při restartu statistického modulu (došlo by ke ztrátě těchto dat z bufferu).

Každý prvek spojového seznamu tvořící databázový buffer určuje sekundový interval, je v něm vždy specifikována sekunda, ke které se daný interval vztahuje, a dále pro každý datový typ jsou ukládány následující údaje:

<i>id_data_type</i>	Identifikace datového typu
<i>requests_count</i>	Počet zpracovaných požadavků
<i>session_count</i>	Počet nově otevřených spojení
<i>recieved_data</i>	Množství dat přijatých od klientů (v Bytech)
<i>send_data</i>	Množství dat odeslaných od serveru klientům (v Bytech)
<i>response_sum</i>	Součet všech odezev serveru
<i>max_response</i>	Maximální odezva serveru
<i>min_response</i>	Minimální odezva serveru
<i>connect_time_sum</i>	Součet všech dob pro vytvoření spojení k serveru
<i>max_connect_time</i>	Maximální doba pro vytvoření spojení k serveru
<i>min_connect_time</i>	Minimální doba pro vytvoření spojení k serveru

Dále jsou všechna data z intervalu před jeho přesunutím do databáze vložena do pomocných intervalů vyšších časových dimenzí s údaji o posledním průběhu. Tyto intervaly jsou používány pro optimalizaci dotazů od prezentačního modulu (bude upřesněno dále). Každý interval ukládá pro každý datový typ následující data vzhledem k dané časové dimenzi a začátku intervalu:

<i>id_data_type</i>	Identifikace datového typu
<i>requests_count</i>	Počet zpracovaných požadavků
<i>max_requests_per_sec</i>	Maximální počet požadavků zpracovaných za sekundu
<i>min_requests_per_sec</i>	Minimální počet požadavků zpracovaných za sekundu
<i>session_count</i>	Počet nově otevřených spojení
<i>max_sessions_per_sec</i>	Maximální počet spojení navázaných za sekundu
<i>min_sessions_per_sec</i>	Minimální počet spojení navázaných za sekundu
<i>recieved_data</i>	Množství dat přijatých od klientů (v Bytech)
<i>send_data</i>	Množství dat odeslaných od serveru klientům (v Bytech)
<i>max_flow</i>	Maximální tok dat (v Bytech za sekundu)
<i>min_flow</i>	Minimální tok dat (v Bytech za sekundu)
<i>max_flow_from_client</i>	Maximální tok dat od klientů (v Bytech za sekundu)
<i>min_flow_from_client</i>	Minimální tok dat od klientů (v Bytech za sekundu)
<i>max_flow_from_server</i>	Maximální tok dat od serveru (v Bytech za sekundu)
<i>min_flow_from_server</i>	Minimální tok dat od serveru (v Bytech za sekundu)
<i>avg_response</i>	Průměrná odezva serveru
<i>max_response</i>	Maximální odezva serveru
<i>min_response</i>	Minimální odezva serveru
<i>avg_connect_time</i>	Průměrná doba pro vytvoření spojení k serveru
<i>max_connect_time</i>	Maximální doba pro vytvoření spojení k serveru
<i>min_connect_time</i>	Minimální doba pro vytvoření spojení k serveru

Databázový buffer je používán vždy, tedy i v případě, že není využito ukládání statistik do databáze. V tom případě slouží databázový buffer pro poskytování dat prezentačnímu modulu. Velikost databázového bufferu (a tím tedy počet prvků spojového seznamu) každého serveru je určena parametrem *STATS_DB_BUFFER_INTERVAL_COUNT* v konfiguračním souboru, který tak určuje dobu v sekundách, po kterou v něm budou data uložena.

5.4 Buffer v paměti

Buffer v paměti je použit pro ukládání informací o jednotlivých aktuálních událostech a informacích přijímaných od modulu jádra. Využíván je pouze pro rychlý zisk aktuálních informací o zatížení serverů potřebných pro balancování dalších požadavků, prezentační modul využívá pouze data z databázového bufferu a databáze. Každý server má svůj vlastní buffer, který je implementován jako obousměrný kruhový spojový seznam, který je seříděný podle časových razítek událostí od nejnovějších k nejstarším. V každém prvku spojového seznamu jsou uloženy následující údaje:

<i>id_data_type</i>	Identifikace datového typu
<i>timestamp</i>	Časové razítko, kdy byl požadavek přijat
<i>msg_type</i>	Identifikátor typu požadavku
<i>data_size</i>	Množství dat v daném požadavku (v Bytech)
<i>response</i>	Velikost odezvy serveru (v mikrosekundách)

Parametrem *STATS_BUFFER_SIZE* z konfiguračního souboru je definována velikost paměti v Bytech, která je vyhrazena pro buffery všech serverů, proto velikost jednoho bufferu je poměrnou částí této hodnoty vzhledem k počtu serverů definovaných v konfiguračním souboru. Po zaplnění celého bufferu dojde k přepisu dat s nejstarším časovým razítkem. Buffer v paměti je použit, pokud má parametr *STATS_QUERY_MODE* z konfiguračního souboru hodnotu "memory".

5.5 Inicializační vlákno

5.5.1 Start statistického modulu

Statistický modul je možné spustit s následujícími prepínači:

- d** nespouštět jako službu daemon. Statistiky se (stejně jako Init a jádro) implicitně spouští na pozadí jako daemon.
- p PORT** čekat na startovací sekvenci na portu PORT. Implicitně statistiky poslouchají na portu 44784.
- l IP_ADDR** čekat na startovací sekvenci na rozhraní IP_ADDR. Implicitně statistiky poslouchají na rozhraní INADDR_ANY.

Po spuštění se nejdříve statistický modul přepne do režimu daemon (pokud není zapnutý prepínač d). Poté se zavolá funkce `startup_init_stats` z knihovny `startup`, ve které statistiky poslouchají na portu 44784 (pokud není uvedený

jiný port přepínačem p) na startovací sekvenci procesu init. Po navázání spojení s procesem init od něj dostane konfigurační soubor. Na základě obsahu konfiguračního souboru je pak inicializována komunikace s modulem jádra. Pomocí komunikační knihovny je nastaven port, na kterém bude statistický modul přijímat jednotlivé údaje o běhu od modulu jádra (tento port je definován parametrem *COMMUN_STATS_PORT*). Dále je spuštěno zvláštní vlákno, které (na portu určeném parametrem *COMMUN_CORE3RD_PORT*) přijímá spojení od ostatních modulů a všechny přijaté dotazy uloží do fronty zpráv, které jsou pak zpracovávány vláknem pro zpracování dotazů.

5.5.2 Inicializace globálních proměnných a struktur

Po inicializaci spojení s ostatními moduly se nainicializují všechny globální proměnné, které se vztahují ke statistickému modulu. Poté dojde k vytvoření spojení s databází a přípravě databázových operací, což je podrobněji popsáno v následující kapitole.

Dále je v paměti vytvořena struktura *data_type_list*, která se používá pro uchovávání informací o všech datových typech požadavků, které byly zpracovány Load Balancerem. Informace uchovávané o každém datovém typu jsou následující:

- *data_type* – název datového typu, který je vytvořen z názvů všech balancovacích pravidel, kterými požadavek při balancování prošel, oddělených dvojtečkami
- *ID_data_type* – ID datového typu, shodné s tím, pod kterým je tento typ uložen v databázi
- *inferior_IDs* – seznam ID všech datových typů, které jsou potomky daného datového typu (např. pro datový typ *http* to budou ID typů *http:html*, *http:php* atd.)
- *inferior_IDs_count* – počet ID uložených v seznamu *inferior_IDs*

Vytvoření této struktury proběhne na základě dat uložených v databázi v tabulce *data_type* a pokud poté přijde požadavek s jiným datovým typem, tento typ je zaznamenán jak do struktury *data_type_list*, tak do databáze. Dále je v paměti vytvořena struktura *server_list*, která se používá pro uchovávání informací o všech serverech, které byly použity ve farmě serverů, na které je Load Balancer nasazen. Informace uchovávané o každém serveru jsou následující:

- *ID_server* – ID serveru, shodné s tím, pod kterým je tento server uložen v databázi

- *address* – IP adresa a port serveru
- *status* – stav, ve kterém se server aktuálně nachází (*OK*–server běží, *ERROR*–při běhu došlo k chybě, *DISABLE*–server není použit)
- *open_session_count* – počet aktuálně otevřených spojení se serverem
- *data_type_count* – počet různých datových typů požadavků, které byly poslány na daný server
- *data_type_names* – seznam různých datových typů požadavků, které byly poslány na daný server
- *data_type_IDs* – seznam identifikátorů různých datových typů požadavků, které byly poslány na daný server
- *db_buffer* – odkaz na databázový buffer daného serveru
- *buffer* – odkaz na buffer daného serveru

Vytvoření této struktury proběhne na základě dat uložených v databázi v tabulkách *server* a *server_data_type* a na základě tabulky *SERVERS* z konfiguračního souboru.

5.5.3 Připojení k databázi a příprava databázových operací

Před připojením k databázi dojde k vytvoření handlerů pro prostředí a samotné připojení a poté je možné se k databázi připojit. Uživatel se připojuje k databázi a databázovému schématu uvedenými v konfiguračním souboru. Po úspěšném připojení proběhne příprava všech databázových operací, které jsou opakovaně prováděny v průběhu práce statistického modulu, aby při každém volání daného příkazu nedocházelo znovu k operacím parse a prepare a příkaz se mohl pouze zavolat s aktuálními parametry, čímž je snížena doba potřebná na provedení dotazu.

Příprava každého příkazu se sestává z alokování handleru pro daný příkaz, dále rozparsování textu příkazu a svázání vstupních a výstupních proměnných daného příkazu s konkrétními proměnnými. K volání příkazu je pak vždy pouze potřeba naplnit jeho vstupní proměnné konkrétními hodnotami a příkaz provést. Pokud je příkazem dotaz, pak se po jeho provedení postupně vyzvedávají jednotlivé záznamy z databáze a hodnoty jednotlivých parametrů se načítají do odpovídajících výstupních proměnných.

5.5.4 Start pracovních vláken

Po provedení veškeré inicializace jsou vytvořena dvě vlákna, která obstarávají veškerou další práci statistického modulu. Toto inicializační vlákno už pouze pomocí startup modulu pošle init modulu informaci, že statistický modul byl úspěšně spuštěn a případně ještě přijme informace o stavu spuštění ostatních modulů. Dále už jenom čeká na ukončení pracovních vláken a ukončí běh statistického modulu.

5.6 Vlákno pro ukládání dat

Vlákno pro ukládání dat v cyklu přijímá a ukládá informace od modulu jádra o jednotlivých údajích o běhu Load Balanceru. Příjem těchto informací je zajišťován komunikační knihovnou pomocí komunikace přes protokol UDP. Struktura přijímaných údajů je následující:

- *server_addr* – IP adresa a port serveru, na který byl daný požadavek odeslán nebo ze kterého byl daný požadavek přijat
- *timestamp* – Časové razítko okamžiku, kdy byl daný požadavek přijat jádrem
- *msg_type* – Typ požadavku, seznam těchto typů je definován ve výčtovém typu *t_commun_msg_type*
- *data_size* – Velikost dat daného požadavku v Bytech
- *response* – Odezva serveru, používá se pouze u první odpovědi na požadavek daného spojení
- *connect_time* – Doba pro vytvoření spojení k serveru
- *text* – Textová identifikace typu dat daného požadavku

Po přijetí dat od jádra se ve struktuře *server_list* nalezne server odpovídající parametru *server_addr* z příchozích dat. Na základě parametru *msg_type* se pak buď změní stav odpovídajícího serveru (při hodnotách *accessible_service* nebo *unaccessible_service*) nebo se data uloží do bufferů odpovídajícího serveru. Před jejich uložením se ještě ve struktuře *data_type_list* nalezne ID datového typu, který odpovídá parametru *text* z příchozích dat, a do bufferů se uloží pouze dané ID.

5.7 Vlákno pro zpracování dotazů a odeslání odpovědí

Po spuštění tohoto vlákna se nainicializují všechny struktury potřebné pro zpracování všech typů dotazů a pak se v cyklu vždy přijme a následně zpracuje dotaz a odešle nalezená odpověď. Pro příjem dotazů od jádra je použito, jak už bylo zmíněno u inicializačního vlákna, samostatné vlákno, které všechny dotazy ukládá do fronty zpráv. Vlákno pro zpracování dotazů tedy vždy vyzvedne dotaz z fronty zpráv a podle jeho typu ho dále příslušně zpracuje.

5.7.1 Zpracování dotazů od podmodulů jádra

Přijatý dotaz je nejprve převeden do interní struktury, která obsahuje data o jednotlivých dotazovaných serverech, v rámci každého serveru data o jednotlivých dotazovaných datových typech a v rámci každého datového typu všechny atributy o jednotlivých dotazovaných časových intervalech. Takto navržená struktura umožňuje rychlejší zpracování dotazů v rámci struktur, ve kterých jsou požadovaná data ve statistickém modulu uložena. Dále je ještě vytvořena struktura obsahující seznam všech různých dotazovaných časových intervalů, která je použita pro zpracování dotazů do databáze.

Následně je pro každý server a každý datový typ proveden příslušný dotaz buď do bufferu v paměti nebo do databázového bufferu (podle nastavení parametru *STATS_QUERY_MODE*). Pokud je některá z odpovědí získaných z databázového bufferu neúplná, provede se ještě dotaz do databáze. Na závěr je interní struktura dotazu s hodnotami odpovědí převedena na strukturu odpovědi a je odeslána podmodulu jádra.

Struktura dotazu

Jeden dotaz od podmodulu obsahuje následující údaje:

- *range* – Identifikátor rozsahu dat, na která je dotazováno. Jsou dva základní typy rozsahů hodnot, časový interval posledních X sekund a posledních X požadavků zpracovaných Load Balancerem. Seznam všech identifikátorů je definován ve výčtovém typu *t_communic_range_type*.
- *value_type* – Identifikátor typu hodnoty, na kterou je dotazováno (např. maximum, minimum atd.). Seznam všech identifikátorů je definován ve výčtovém typu *t_communic_query_value_type*.

- *info_type* – Identifikátor typu informace, na kterou je dotazováno (např. odezva serveru, tok dat atd.). Seznam těchto typů je definován ve výčtovém typu *t_commun_query_info_type*.
- *data_type* – Textový identifikátor typu dat, na který je dotazováno.
- *server_num* – Počet serverů, o kterých chceme získat požadovaná data.
- *servers* – Seznam serverů, o kterých chceme získat požadovaná data.

Přijatý dotaz se může skládat z více jednotlivých dotazů, jejichž struktura byla popsána výše. Jednak se tím ušetří síťový provoz a jednak se zrychlí zpracování dotazů uvnitř statistického modulu.

Dotazy do bufferu v paměti

Buffer je uzamknut a je procházen od nejnovějšího záznamu po nejstarší, dokud nejsou nalezeny odpovědi na všechny dotazy. U každého záznamu z bufferu jsou projity všechny datové typy z dotazu a pokud některým z nich odpovídá typ záznamu, jsou data tohoto záznamu použita a zahrnuta do odpovědi. Pro počítání hodnot odpovědí je použita interní struktura *t_stats_buffer_select_process*, která obsahuje následující údaje:

- *value* – Aktuální hodnota odpovědi.
- *temp_sum_value* – Slouží k nasčítávání hodnot odpovědí, jejichž hodnoty jsou uváděny za sekundový interval.
- *amount* – Počet hodnot, které se musí ještě do odpovědi zahrnout, než bude hotová.
- *time_amount* – Počet vteřin, které se musí ještě do odpovědi zahrnout, než bude hotová.
- *extreme_time* – Identifikuje vteřinu, ve které se objevila extrémní hodnota dané odpovědi.

Dotazy do databázového bufferu

Postup je stejný jako u bufferu v paměti, jenom s tím rozdílem, že v databázovém bufferu jsou uloženy hodnoty agregované za sekundový interval, takže se jednodušeji získávají hodnoty odpovědí na dotazy ohledně dat za sekundu. Stejně jako u bufferu v paměti je pro počítání odpovědí použita interní struktura *t_stats_buffer_select_process* a pokud nedojde k nalezení všech odpovědí v databázovém bufferu, databáze pak pokračuje při svém zpracování dotazů v plnění této struktury nalezenými hodnotami.

Dotazy do databáze

Je procházena interní struktura obsahující seznam všech různých dotazovaných časových intervalů a každý interval z ní je zpracován samostatně. Z tabulky *time_dimension* jsou vždy vybrána ID všech agregovaných sekundových záznamů, které spadají do dotazovaného časového intervalu a pro každé takto získané ID jsou vždy sestaveny a provedeny všechny dotazy, které se ptají na daný interval. Hodnoty získané těmito dotazy z databáze jsou ukládány a zpracovávány ve struktuře *t_stats.buffer_select_process*, odkud jsou po dokončení dotazů a získání všech odpovědí přesunuty do struktury *t_stats.query_data*.

5.7.2 Zpracování dotazů od prezentačního modulu

Struktura dotazu

- *range_from* – Časové razítko, které tvoří dolní hranici intervalu, na který se dotazuje.
- *range_to* – Časové razítko, které tvoří horní hranici intervalu, na který se dotazuje.
- *aggregation_level* – Identifikátor úrovně agregace požadovaných dat. Určuje délku časového intervalu, ze kterého se data agregují do jedné hodnoty v odpovědi. Seznam všech identifikátorů je definován ve výčtovém typu *t_commun_time_aggregation_level*.
- *value_type* – Identifikátor typu hodnoty, na kterou je dotazováno (např. maximum, minimum atd.). Seznam všech identifikátorů je definován ve výčtovém typu *t_commun_query_value_type*.
- *info_type* – Identifikátor typu informace, na kterou je dotazováno (např. odezva serveru, tok dat atd.). Seznam těchto typů je definován ve výčtovém typu *t_commun_query_info_type*.
- *data_type* – Textový identifikátor typu dat, na který je dotazováno.
- *server_num* – Počet serverů, o kterých chceme získat požadovaná data.
- *servers* – Seznam serverů, o kterých chceme získat požadovaná data.

Počet hodnot vrácených v odpovědi na jeden dotaz je určen délkou časového intervalu mezi *range_from* a *range_to* vydělený délkou intervalu určenou parametrem *aggregation_level*.

Dotazy do databázového bufferu

Pro každý server z dotazu je uzamknut a procházen jeho databázový buffer. Pokud aktuální interval z bufferu spadá do časového intervalu dotazu, pak jsou do odpovědi zahrnuty hodnoty všech datových typů, které jsou shodné nebo jsou potomky datového typu specifikovaného v dotazu. Pokud se zpracuje počet intervalů, který je roven agregaci vyjádřené v sekundách, zapíše se výsledná hodnota na aktuální pozici do pole odpovědi ve struktuře *t_communications_present_answer*.

Dotazy do databáze

Pro danou úroveň agregace jsou z tabulky *time_dimension* vybrána ID všech časových intervalů, které spadají do intervalu dotazu. Pro každé takové ID je sestaven a proveden dotaz na data z tabulky *stats* a vrácená hodnota je uložena na příslušné místo do seznamu hodnot ve struktuře odpovědi. Pokud je horní hranice intervalu větší než největší časové razítko intervalu vybraného z databáze, přidají se do odpovědi příslušné hodnoty z intervalu příslušné časové dimenze, který je uložen v paměti jako součást databázového bufferu.

5.7.3 Zpracování dotazů na stav serverů

Tento dotaz je přijímán také od prezentačního modulu. V dotazu nejsou potřebné žádné jiné informace mimo specifikace, že se jedná právě o tento typ dotazu. Pokud je jako úložiště dat použita databáze, pak se z ní načtou všechny servery, které byly do farmy obsluhované Load Balancerem zapojeny a jejich stav je nastaven na *UNKNOWN*. Následně je všem serverům, které jsou uvedeny v konfiguračním souboru v tabulce *SERVERS*, jejich stav přenastaven na jejich aktuální hodnotu, která je u serveru uvedena v parametru *status* ve struktuře *server_list*. Pokud není přístupná databáze, pak se do struktury s odpovědí načtou pouze data o všech serverech ze struktury *server_list*.

Kapitola 6

Konfigurace

6.1 Úvod

Jednotlivé moduly Load Balanceru mohou běžet na různých počítačích, konfigurační soubor je ale pouze na jednom počítači a to na počítači, na kterém běží modul Init. O přesun konfiguračního souboru z počítače na počítač se stará komunikační knihovna. Struktura konfiguračního souboru (sekce MAIN, sekce ALIASES a uživatelské sekce) je detailně popsána v uživatelské dokumentaci. Konfigurační knihovna má dvě části, *cfg* a *cfgweb*. Knihovna *cfg* má za úkol základní věci týkající se konfiguračního souboru, jako je rozparsování konfiguračního souboru, ověření syntaktické správnosti konfiguračního souboru, zápis obsahu paměti do konfiguračního souboru a zpřístupnění proměnných z konfiguračního souboru modulům. Knihovna *cfgweb* rozšiřuje tyto služby o další, jež se používají v prezentačním modulu (webové rozhraní Load Balanceru). Jedná se o služby jako zpřístupnění sekce MAIN, zpřístupnění sekce ALIASES, zpřístupnění jednotlivých uživatelských sekcí a změna jejich obsahů.

6.2 Návrh konfigurační knihovny

Obsah konfiguračního souboru se do modulu Load Balanceru může dostat dvojí cestou. Buď je načten funkcí *cfg_get_file()* nebo jej příslušný modul přijme pomocí komunikační knihovny.

Obsah konfiguračního souboru se pak rozparsuje funkcí *cfg_parse_config_file()*, která vrací ukazatel na interní reprezentaci konfiguračního souboru a její použití je následující:

```
t_cfg_config *r;  
r=cfg_parse_config_file(...);
```

Tato funkce využívá parsovacích nástrojů *flex* a *bison* a zároveň kontroluje syntaktickou správnost konfiguračního souboru.

Všechny funkce z konfigurační knihovny pak následně pracují s interní reprezentací konfiguračního souboru, můžou ho měnit a posléze i zapsat do souboru na externí paměti.

Proces Init, který se stará o rozparsování konfiguračního souboru a poslání jeho obsahu ostatním modulům Load Balanceru, vypadá následovně:

```
char *filename;
char *cfgfile;
t_cfg_config *r;

/* filename obsahuje umístění standartního konfiguračního souboru,
   nebo soubor zadány z příkazové řádky
   */
if ((cfgfile = cfg_get_file (filename)) == NULL)
{
    /* ERROR */
}

/* druhý argument nenulový znamená, že chyby se budou kromě do logu
   vypisovat i na stderr
   */
if ((r = cfg_parse_config_file (cfgfile,1)) == NULL)
{
    /* ERROR */
    cfg_get_file_done (cfgfile);
    return 1;
}

/* druhý argument nenulový znamená, že chyby se budou kromě do logu
   vypisovat i na stderr;

   třetí argument nenulový znamená, že se budou vypisovat všechny nalezené
   chyby, nejenom první nalezená
   */
if (!cfg_check(r,1,1))
{
    /* ERROR */
    cfg_get_file_done (cfgfile);
    cfg_parse_config_file_done (&r);
    return 1;
}
```

```

/* konfiguracni soubor je ok,
   Init ho muze rozeslat ostatnim modulom Load Balanceru
*/

/* po ukonceni prace
*/
cfg_get_file_done (cfgfile);
cfg_parse_config_file_done (&r);

```

6.3 Reprezentace konfiguračního souboru

Reprezentace konfiguračního souboru byla navržena s cílem snadné rozšiřitelnosti pro nově přidávané moduly (nové balancovací algoritmy).

Konfigurační soubor je zhruba řečeno reprezentován jako spojový seznam sekcí plus navíc některá další propojení. To platí doslova pro uživatelské sekce (které jsou reprezentovány jako spojový seznam svých řádků a každý řádek uživatelské sekce je reprezentován jako spojový seznam slov na dané řádce), ale ne pro sekci MAIN a sekci ALIASES. Sekce MAIN (více viz. uživatelská dokumentace) slouží k přiřazení uživatelských sekcí danému modulu. Sekce ALIASES má za úkol to samé, jenom přiřazení uživatelských sekcí (obecně více) modulům pojmenovává. Z programátorského hlediska tedy mezi sekcí MAIN a sekcí ALIASES není velký rozdíl. Každá řádka sekce MAIN je vlastně jakýsi "anonymní" alias.

Program sám si tedy uchovává seznam aliasů, kde si u každého aliasu pamatuje:

- jméno aliasu - určené prvním slovem na řádce v sekci ALIASES, v případě tzv. anonymního aliasu (viz. předchozí odstavec - jedná se vlastně o přiřazení uživatelských sekcí modulu v sekci MAIN) je jméno prázdné
- jméno modulu - kterému se v rámci aliasu přiřazují uživatelské sekce
- seznam uživatelských sekcí - které se přiřazují modulu
- aktivita - určuje, zda-li je alias aktivní. Anonymní aliasy jsou vždy aktivní, neboť pocházejí ze sekce MAIN. Pojmenované aliasy (což jsou právě ty, jejichž definice se našla v sekci ALIASES) jsou aktivní právě tehdy, pokud se vyskytly v sekci MAIN

Z poznámky o aktivitě aliasu mimochodem vyplývá, že aliasy uvedené v sekci MAIN, ale nedefinované v sekci ALIASES se ignorují.

Je třeba upozornit, že pokud alias přiřazuje obecně různé množiny uživatelských sekcí různým modulům, vyskytne se tento alias v seznamu aliasů vícekrát. Pokaždé s jiným modulem. Tedy alias je vlastně úplně charakterizován nejen svým jménem, ale dvojicí jméno aliasu a jméno modulu, kterému tento alias přiřazuje uživatelské sekce.

Všechny aliasy (určené svým jménem a jménem modulu) téhož jména ale mají stejnou aktivitu (tzn. buď jsou aktivní nebo pasivní).

6.4 Parser

Úkolem parseru je vytvořit výše popsanou datovou strukturu, tedy spojový seznam obecných aliasů (kde obecným aliasem rozumíme jak alias z konfiguračního souboru, tak i anonymní alias) a spojový seznam uživatelských sekcí.

Parser při své tvorbě interní reprezentace konfiguračního souboru odstraňuje následující duplicity:

- v sekci MAIN na jedné řádce uvedena některá uživatelská sekce vícekrát, obdobně v definici aliasu v sekci ALIASES
- alias (jenž je definován v sekci ALIASES) je uveden v sekci MAIN vícekrát
- vícenásobná definice aliasu a modulu (tj. přiřazení uživatelských sekcí modulu) v sekci ALIASES - vezme se sjednocení uživatelských sekcí
- uživatelské sekce stejného jména - obsah se spojí - toto může způsobit nechtěné chování programu, uživatel by měl být opatrný a volit jména uživatelských sekcí a také identifikátory proměnných (i mezi uživatelskými sekcemi) jednoznačně

6.4.1 Flex

Flex je nástroj, který posloužil jako tzv. tokenizátor, tedy program, který čte vstup znak po znaku a kdykoli přečte posloupnost znaků, které odpovídají danému vzoru, vrátí definovaný token. Vstupní soubor pro *flex* je *cfg_parse.l* a definuje následující tokeny:

- MAIN_BEGIN - začátek sekce MAIN
- MAIN_END - konec sekce MAIN

- ALIASES_BEGIN - začátek sekce ALIASES
- ALIASES_END - konec sekce ALIASES
- SECTION_BEGIN - začátek uživatelské sekce, jejíž jméno je posloupnost znaků, kde první znak může být libovolné písmeno anglické abecedy nebo podtržítka a další znaky to samé plus navíc libovolná cifra
- SECTION_END - konec uživatelské sekce
- WORD - slovo (posloupnost znaků, pro níž platí stejné omezení jako pro identifikátor uživatelské sekce)
- QUOTED_VALUE - text v uvozovkách, ve kterém se mohou vyskytovat uvozovky uvozené zpětným lomítkem
- SEMICOLON - středník

Tokenizátor navíc odstraňuje bílé znaky (mezery, tabulátory a konce řádků) a komentáře.

6.4.2 Bison

Pro parsování konfiguračního souboru byl použit nástroj *bison*, který na základě dané gramatiky a zdrojového kódu (akcí) k jednotlivým pravidlům gramatiky vygeneruje parser, který přijímá pouze slova (kde znakovým slova se myslí token) patřící do jazyka dané gramatiky. Dále je uvedena gramatika popisující syntakticky správný konfigurační soubor. Z důvodu obecnosti a možnosti snadného přidávání nových datových typů není určen pevný počet argumentů v závislosti na klíčovém slově (struktura konfiguračního souboru viz. uživatelská dokumentace). Správný počet argumentů kontroluje funkce *cfg_check()* která je volána modulem Load Balanceru při každém načtení konfiguračního souboru.

Terminální symboly (tokeny) jsou psány velkými písmeny, neterminální symboly gramatiky jsou psány malými písmeny.

```
input :=
    main_section alias_section user_sections

main_section :=
    MAIN_BEGIN MAIN_END
    | MAIN_BEGIN main_lines MAIN_END
```

```

main_lines :=
    main_line
    | main_lines main_line

main_line :=
    WORD SEMI_COLON
    | rest_of_main_line WORD SEMI_COLON

rest_of_main_line :=
    WORD
    | rest_of_main_line WORD

alias_section :=
    ALIASES_BEGIN ALIASES_END
    | ALIASES_BEGIN aliases_lines ALIASES_END

aliases_lines :=
    aliases_line
    | aliases_lines aliases_line

aliases_line :=
    WORD WORD WORD SEMI_COLON
    | WORD WORD rest_of_aliases_line WORD SEMI_COLON

rest_of_aliases_line :=
    WORD
    | rest_of_aliases_line WORD

user_sections :=
    /* empty */
    | user_sections user_section

user_section :=
    SECTION_BEGIN SECTION_END
    | SECTION_BEGIN section_lines SECTION_END

section_lines :=
    section_line
    | section_lines section_line

```

```

section_line :=
    WORD SEMI_COLON
  | WORD QUOTED_VALUE SEMI_COLON
  | WORD rest_of_section_line QUOTED_VALUE SEMI_COLON

rest_of_section_line :=
    QUOTED_VALUE
  | rest_of_section_line QUOTED_VALUE

```

Zdrojový kód v jazyce C pro jednotlivá pravidla je v souboru *cfg_parse.y*. Zdrojový kód je zdokumentován. Jednotlivá pravidla operují se slovy na aktuálně čteném řádku a řadí ho do průběžně budované datové struktury.

Na konci se ještě struktura projde a vytvoří se ukazatele pro urychlení některých vyhledávání. Více viz. funkce *cfg_parse_config_file()*.

6.5 Zpřístupnění proměnných modulům

Knihovna *cfg* obsahuje sadu funkcí pro zpřístupnění proměnných datových typů jednotlivým modulům Load Balanceru také balancovacím podmodulům. Tyto podmoduly jsou rozdělené do tří vrstev (více viz. kapitola o Podmodulech programátorské dokumentace). Uživatel si může dopsat nové podmoduly (nové balancovací algoritmy) a proto je třeba popsat API, jak si nový podmodul zpřístupní svá data z konfiguračního souboru.

Funkce pro všechny jednoduché datové typy jsou stejné, využívají funkce, které prochází strukturu a pro daný modul (jenž je součástí aktivního aliasu) prochází všechny sekce jemu přiřazené a hledá proměnnou daného jména a typu, to určuje samotná funkce. Ve všech následujících ukázkách předpokládáme následující deklarace:

```

t_cfg_config *r;
char *name;

```

První proměnná je ukazatel na strukturu reprezentující konfigurační soubor (podmodul Load Balanceru ji dostane jako argument). Druhá proměnná je řetězec s názvem daného modulu či podmodulu (který každý modul resp. podmodul ví; tj. zná své jméno, díky němuž jsou mu v konfiguračním souboru přiřazena data).

Dále jsou uvedeny ukázkové zdrojové kódy pro zpřístupnění proměnných z konfiguračního souboru. Definice příslušných knihovních funkcí, spolu s významem argumentů a možných návratových hodnot lze nalézt v technické dokumentaci.

6.5.1 Zpřístupnění proměnných jednoduchých datových typů

Následující zdrojový kód ukazuje, jak se dá přistoupit k proměnným jednoduchých datových typů. Pro syntaxi konfiguračního souboru a způsob vyhodnocování níže používaných funkcí viz. kapitola o Konfiguraci v uživatelské dokumentaci.

Všechny funkce mají jako první argument ukazatel na strukturu reprezentující konfigurační soubor, dále jméno (pod)modulu, jméno proměnné a defaultní hodnotu. Datový typ proměnné je určen danou funkcí.

```
int MAX_THREAD;
long limit;
double d;
int b;
char c;
char *s;

/* zakladni datove typy
 */
MAX_THREAD=cfg_get_varint(r, name, "MAX_THREAD", 10);
limit=cfg_get_varlint(r, name, "limit", 1000L);
c=cfg_get_varchar(r, name, "CHAR1", 's');
d=cfg_get_vardouble(r, name, "AVG_OVERLOAD", 0.0);
b=cfg_get_varbool(r, name, "BOOL1", 0);

/* datovy typ string (retezec)
   (programator musi dealokovat)
 */
s=cfg_get_varstring(r, name, "STR", "daemon");
do_something_with_string(s);
free((void*) s);

/* datovy typ list (seznam): oddelovac je carka
   (programator musi dealokovat)
 */
char *str;
char **list;
int i,n;

str=cfg_get_varlist(r, name, "LIST", "");
/* parsovani stringu do pole jednotlivych polozek seznamu
 */
```

```

list=cfg_list2str(str,&n);

if (list == NULL) {
    if (n < 0)
        /* ERROR: not enough memory */
    else
        /* list is empty */
} else {
    for (i=0;i<n;i++) {
        do_something_with_item_of_list(list[i]);
    }
}
free((void*) str);
cfg_free_list(&list, n);

```

6.5.2 Zpřístupnění tabulek

Práce s tabulkami je o něco komplikovanější, programátor si musí dát pozor na správné přetypování ukazatelů. Programátor ví, co si do konfiguračního souboru uložil a proto si vrácený ukazatel typu *void* správně přetypuje. Ve všech následujících ukázkách předpokládáme následující deklarace:

```

void ***table; /* tabulka */
int row;      /* pocet radek */
int i;

```

Pro první příklad předpokládáme v konfiguračním souboru definici následující tabulky (pro syntax zápisu tabulek v konfiguračním souboru viz kapitola o Konfiguraci v uživatelské dokumentaci):

```

tabledef "cisla" "3" "varint" "varint" "varint";

```

Následující kód vypíše obsah tabulky "cisla" na standartní výstup (tabulka obsahuje 3 celočíselné sloupečky):

```

table=cfg_get_vartable(r, name, "cisla", &row);
if (table == NULL)
    /* ERROR */

for (i=0;i<row;i++) {
    printf("%2d: '%d' '%d' '%d'\n", i,
        *((int*) table[i][0]),
        *((int*) table[i][1]),
        *((int*) table[i][2]));
}

```

```
cfg_free_vartable(&table);
```

Pro druhý příklad předpokládáme v konfiguračním souboru definici následující tabulky:

```
tabledef "mix" "3" "varbool" "varstring" "varint";
```

Následující kód vypíše obsah tabulky "mix" na standartní výstup.

```
table=cfg_get_vartable(r, name, "mix", &row);
if (table == NULL)
    /* ERROR */
for (i=0;i<row;i++) {
    printf("%2d: '%d' '%s' '%d'\n", i,
           *((int*) table[i][0]),
           *((char **) table[i][1]),
           *((int*) table[i][2]));
}
cfg_free_vartable(&table);
```

Pro třetí příklad předpokládáme v konfiguračním souboru definici následující tabulky:

```
tabledef "stattable" "3" "varint" "varlist" "varstring";
```

Následující kód vypíše obsah tabulky "stattable" na standartní výstup.

```
char **list;
int j,n;

table=cfg_get_vartable(r, name, "stattable", &row);
if (table == NULL)
    /* ERROR */
for (i=0;i<row;i++) {
    printf("%2d: ", i);
    /* int */
    printf("'%'d' ", *((int*) table[i][0]));

    /* vypis list jako retezec */
    printf("'%'s' ", *((char **) table[i][1]));

    /* zkonvertuj list do pole */
    list=cfg_list2str(*((char **) table[i][1]),&n);
    if (list == NULL) {
        if (n < 0)
```

```

        /* ERROR: not enough memory */
    else
        /* list is empty */
} else {
    printf("(items of list: ");
    for (j=0;j<n;j++)
        printf("%d:'%s' ", j+1, list[j]);
    printf(") ");
}
cfg_free_list(&list, n);

/* string */
printf("'%s'\n", *((char **) table[i][2]));
}

cfg_free_vartable(&table);

```

6.6 Podpora pro webové rozhraní

Knihovna *cfgweb* rozšiřuje sadu funkcí, jež nabízí knihovna *cfg*. Obsahuje funkce pro zpřístupnění všeho, co se v konfiguračním souboru nachází (funkce s prefixem *cfg_get*), funkce pro editaci sekce MAIN (funkce s prefixem *cfg_main*), funkce pro editaci sekce ALIASES (funkce s prefixem *cfg_aliases*) a konečně funkce pro editaci uživatelských sekcí (funkce s prefixem *cfg_section*). Rozhraní pro uživatelské sekce je "nízkoúrovňové", sekci interpretuje pouze jako seznam řádek, takže například práce s tabulkami je plně v režii webového rozhraní (na rozdíl od pohodlnější práce s tabulkami pro moduly Load Balanceru via *cfg_get_vartable()*). To je z důvodu, aby webové rozhraní mohlo zkušeným administrátorům nabídnout stejné možnosti jako přímá editace konfiguračního souboru.

Pokud uživatel přidá do konfiguračního souboru nové proměnné, neměl by nechávat prázdný popis pro webové rozhraní. Podobně pro tabulky. Více viz. uživatelská dokumentace.

6.7 Rozšíření konfigurační knihovny

Pokud si chce uživatel připsat nový podmodul (další balancovací algoritmus), stačí mu uložit si do konfiguračního souboru svá data a zpřístupnit je prostřednictvím funkcí z knihovny *cfg*. Podrobný popis těchto funkcí, přípustné argumenty, návratové hodnoty, uvolňování paměti a další podrob-

nosti jsou popsány v technické dokumentaci.

Pokud uživatel potřebuje ve svém novém modulu i nové datové typy (resp. jejich zpracování), stačí mu přiřadit si novou funkci typu *cfg_get_mydatatype* po vzoru ostatních, již připravených funkcí pro standardní datové typy. Dále je třeba přiřadit do *cfg_check* kontrolu pro daný datový typ (minimálně správný počet argumentů).

To je taky důvod, proč byl konfigurační modul navržen tak, jak byl navržen. Oddělit parser, jehož kód je generován, od zbytku funkčnosti. Parser nemusí být přepsán (kromě přidání případných nových klíčových slov) ani při komplikovanějším požadavku na změnu obsahu konfiguračního souboru.

Kapitola 7

Web

7.1 Úvod

V této kapitole je popsána implementace webového rozhraní k Load Balanceru. Webové rozhraní umožňuje editovat konfigurační soubor, zobrazovat statistiky běhu v podobě tabulek a grafů, zobrazovat stav serverů, ovládat běh Load Balanceru. Vše je implementováno pomocí php skriptů, které využívají php extension balancer.

Web je vytvořen pomocí rámců. Vlevo je navigační panel s odkazy:

General odkaz na stránky projektu

Monitoring zobrazení stavu serverů

Reporting zobrazení statistik běhu pomocí tabulek a grafů

Configuration ovládání běhu balanceru, editace konfiguračního souboru

7.2 PHP extension balancer

Co je PHP extension balancer? Je to rozšíření skriptovacího jazyka php o vlastní funkce. Tento způsob byl zvolen ze dvou důvodů. Použití konfigurační knihovny *webcfg* k editaci konfiguračního souboru. Komunikace skriptů se statistickým modulem. Více o PHP extension na <http://php.paradoxical.co.uk/manual/cs/zend.php>. Použití je možné dvěma způsoby, buď jako externí nebo jako vestavěný modul do PHP.

7.2.1 Seznam funkcí

open_cfg_file otevření konfiguračního souboru

close_cfg_file zavření konfiguračního souboru

get_main_modules funkce vrací pole všech modulů v sekci MAIN

get_all_sections funkce vrací pole všech sekcí v konfiguračním souboru

get_main_aliases funkce vrací pole všech aliasů použitých v sekci MAIN

get_aliases funkce vrací pole všech aliasů v sekci ALIASES

get_modules_for_alias funkce vrací pro zadaný alias pole všech modulů, které daný alias obsahuje

get_sections_for_alias_and_module funkce vrací pro zadaný alias a modul pole všech sekcí, které obsahuje daný modul v daném aliasu

get_section_for_main_module funkce vrací pro zadaný modul v sekci MAIN pole všech sekcí, které daný modul obsahuje

alias_del smaže daný alias v sekci ALIASES

alias_add_module přidá daný modul do definice daného aliasu

alias_del_module smaže daný modul v definici daného aliasu

alias_add_section přidá sekci do definice daného modulu v daném aliasu

alias_del_section smaže sekci v definici daného modulu v daném aliasu

alias_get_active přidá alias do sekce MAIN

alias_set_active smaže alias v sekci MAIN

main_add_module přidá modul do sekce MAIN

main_del_module smaže modul v sekci MAIN

main_add_section k definici danému modulu přidá danou sekci

main_del_section v definici danému modulu smaže danou sekci

section_add přidá do konfiguračního souboru danou sekci

section_del smaže v konfiguračním souboru danou sekci

section_get vrací obsah celé sekce

section_get_line vrací obsah daného řádku sekce

section_del_line smaže daný řádek sekce

section_add_line přidá daný řádek do sekce

section_change_line změní daný řádek v sekci

main_can_del_section vrací true pokud lze smazat danou sekci z definice modulu v sekci MAIN, tedy zda není poslední, jinak vrací false

section_can_del vrací true, pokud lze smazat danou sekci, tedy pokud není použita v definici nějakého modulu

commun_monitor_query funkce pro zaslání monitorovacího dotazu statistickému modulu, funkce využívá knihovnu `commun.c`

commun_send_query funkce pro zaslání dotazu statistickému modulu, slouží k získání dat o běhu balanceru

get_enum vrací pole všech hodnot daného typu enum

get_varservers vrací pole všech hodnot typu `varservers`

7.3 Konfigurace webového rozhraní

Konfigurace webového rozhraní je uložena v kořenovém adresáři v souboru `config.php`, ve kterém jsou tato nastavení:

statsIP IP adresa stroje, na kterém běží statistický modul

statsPORT port, na kterém poslouchá statistický modul

init_pid_file soubor obsahující pid modulu `init`

running_config umístění konfiguračního souboru, který si `init` modul načítá při startu

bin_dir adresář, kde jsou uloženy programy `copy` a `restart` na ovládní balanceru z webu

7.4 Ovládání balanceru z webu

Ovládání webu umožňuje restartování celého Load Balanceru zasláním signálu SIGHUP, jeho zastavení zasláním signálu SIGTERM, operace s konfiguračním souborem.

Existují tři konfigurační soubory:

running aktuální konfigurace Load Balanceru, při startu ho načítá modul `init`

default slouží jako výchozí nastavení Load Balancer

editing uchovává změny provedené webovým rozhraním

Ovládání webového rozhraní obsahuje dva programy:

copy zajišťuje kopírování konfiguračních souborů `running`, `default`, `editing`

restart zajišťuje posílání signálů SIGHUP a SIGTERM modulu `init`

K funkčnosti ovládání balanceru z webu jsou nutná určitá práva programů `copy` a `restart`. Program `copy` implementuje funkce `load config` (zkopíruj `running` do `editing`), `load default config` (zkopíruj `default` do `editing`), `save config` (zkopíruj `editing` do `running`). Program musí mít práva na zápis do `running` konfiguračního souboru. Program `restart` implementuje zasílání signálů SIGHUP a SIGTERM. Též musí mít práva k zaslání signálů procesu modulu `init`. Oba programy musí být spustitelné pod uživatelem, pod kterým běží webový server. Jako vhodné řešení práv se jeví použití programu `sudo` (<http://www.courtesan.com/sudo/>).

7.5 Monitoring serverů

Monitoring serverů zobrazuje tabulku serverů a jejich aktuální stav. Funkčnost zajišťuje skript `servers.php`, který získává data voláním funkce `commun_monitor_query`, která je součástí PHP extension `balancer`. Stránka je pomocí `html refresh` tagu automaticky jednou za 10 sekund obnovována.

7.6 Zobrazování dat ze statistického modulu

K získávání dat ze statického modulu slouží funkce `commun_send_query`, která je součástí PHP extension `balancer`. Pomocí `html` formuláře, který generuje skript `query.php`, uživatel vytvoří dotaz, který se předá již

zmiňované funkci. Ta se připojí ke statistickému modulu a vrátí pole s hodnotami, které jsou pak zobrazeny v tabulce a pomocí knihovny `jpgraph` (<http://www.aditus.nu/jpgraph/>) v grafu. Zobrazení hodnot mají na starosti skripty `show_table.php` a `show_graph.php`. Výsledná html stránka má pak nastaven `html refresh` tag a každou minutu se obnovuje pro pohodlné sledování aktuálního stavu.

7.7 Editace konfiguračního souboru

Přes web se konfiguruje soubor `config.cfg`, který je považován za editing konfigurační soubor. Jeho obsah je možné shlédnout pod odkazem `view config`.

7.7.1 Main

Tuto část generuje skript `main.php`, který zobrazuje seznam modulů v sekci `MAIN` a ke každému modulu seznam sekcí, které obsahuje. Dále je zde seznam aliasů v sekci `MAIN` a jejich možné smazání ze sekce `main`.

7.7.2 Aliases

Tuto část generuje skript `aliases.php`, který zobrazuje definice aliasu v sekci `ALIASES` a umožňuje jejich editaci. Pomocí grafických ikon, je možné alias přidat či odebrat ze sekce `main`.

7.7.3 Sections

Tuto část generuje skript `sections.php`, který zobrazuje seznam sekcí, umožňuje přidání nové sekce a pokud je to možné, smazání existující sekce. Po kliknutí na název sekce se skript zobrazuje obsah sekce, umožňuje editaci jednotlivých záznamů a jejich přidávání.

7.8 Zabezpečení webového rozhraní

Webové rozhraní nemá žádnou autorizaci přístupu. Doporučuje se omezit přístup k webu na úrovni web serveru. Takový nástroj nám například nabízí web server Apache v podobě modulů `mod_auth`, `mod_access` (<http://httpd.apache.org/docs-2.0/howto/auth.html>). Pomocí těchto modulů můžeme vytvořit autorizaci přístupu na základě loginu a hesla a navíc omezit přístup jen z některých domén.

Kapitola 8

Knihovny

V této kapitole jsou popsány knihovny používané různými částmi Load Balanceru a jsou uloženy v adresáři lib/. U každé knihovny jsou stručně popsány jednotlivé volané funkce. Funkce, které se mohou využívat z uživatelských podmodulů, jsou popsány podrobněji. Seznam funkcí, které lze volat z podmodulů, je v kapitole podmoduly v části 3.5.

8.1 knihovna cache.c

Knihovna cache obsahuje funkce zajišťující použití cache z podmodulů a k jejich aktualizaci z jádra. Počet cache není omezen a každý podmodul si jich může vytvořit libovolné množství.

Cache je seznam řádek, kde každý odpovídá jednomu hashovacímu číslu. Každý řádek obsahuje položky, které mají stejné hashovací číslo (kolizní doména). V každé položce je uloženo, zda obsahuje hodnotu nebo je prázdná, hashovaný řetězec, id serveru a timestamp. V cache je uloženo číslo serveru, kam bylo naposledy přeposláno spojení a zároveň aktualizována příslušná položka z jádra.

create_cache

```
int create_cache (t_cache_info * cache,  
                 int size,  
                 int domain_size,  
                 int string_size)
```

Tato funkce umožňuje podmodulům vytvořit novou cache. Parametr cache obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a

neměla by je nijak měnit. Parametr `size` udává počet řádků v cache. Parametr `domain_size` udává velikost kolizní domény (počet položek na řádku). Parametr `string_size` udává maximální velikost hashovaného řetězce. Velikost alokované paměti je `size * domain_size * sizeof (položka)`. Velikost položky je `string_size + sizeof(int + int + time_t)`. Při úspěšném vytvoření vrací číslo nově vytvořené cache, při chybě -1.

query_cache

```
int query_cache (t_cache_info * cache,
                 int cache_id,
                 int hash_num,
                 char *hash_string,
                 int *server_id,
                 time_t * time)
```

Pomocí této funkce se mohou funkce podmodulů dotazovat na položky cache. Postupně se projde určená kolizní doména a pokud se v ní nalezne příslušný záznam, vrátí funkce číslo serveru a timestamp. Parametr `cache` obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a neměla by je nijak měnit. Parametr `cache_id` je číslo použité cache. Parametr `hash_num` je hodnota řetězce po zahashování a určuje index řádku v cache. Parametr `hash_string` je hledaný řetězec. `Server_id` a `time` jsou výstupní parametry, pokud řetězec není nalezen, pak obsahují hodnoty -1 a 0. Při chybě funkce vrací -1, jinak 0.

update_cache

```
int update_cache (t_cache_info * cache,
                  int thread_id,
                  int cache_id,
                  int hash_num,
                  char *hash_string)
```

Touto funkcí se požaduje, aby jádro po ukončení výběru serveru zapsala číslo serveru, na které bylo dané spojení přepojeno, do příslušné cache. Při úspěšné aktualizaci cache se položce přiřadí timestamp v okamžiku vybrání příslušného serveru. Parametr `cache` obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a neměla by je nijak měnit. Parametry `cache_id` a `thread_id` obsahují identifikaci vlákna předané z jádra a číslo použité cache. Parametry `hash_num` a `hash_string` obsahují hodnotu řetězce po zahashování a samotný řetězec. Při chybě vrací funkce -1, jinak 0.

update_cache_from_core

Tato funkce je volána z jádra a zajišťuje aktualizace všech potřebných cache, které byly příslušně nastaveny voláním funkce `update_cache` z jednotlivých podmodulů.

8.2 Komunikační knihovna `commun.c`

Komunikační knihovna `commun` poskytuje funkce zajišťující veškerou komunikaci mezi moduly při běhu Load Balanceru. Zajišťuje odesílání a přijímání informací o jednotlivých požadavcích procházejících přes Load Balancer, odesílání dotazů podmoduly jádra a přijímání odpovědí na ně, přijímání dotazů od různých podmodulů statistickým modulem a odesílání odpovědí zpět. Seznam dostupných funkcí komunikační knihovny je následující:

commun_init_core

Tato funkce inicializuje všechny proměnné jádra z konfiguračního souboru, které se vztahují ke komunikaci s jinými moduly. Tuto funkci musí modul jádra zavolat dřív, než použije nějakou jinou funkci z komunikační knihovny.

commun_init_stats

Tato funkce inicializuje všechny proměnné statistického modulu z konfiguračního souboru, které se vztahují ke komunikaci s jinými moduly. Tuto funkci musí statistický modul zavolat dřív, než použije nějakou jinou funkci z komunikační knihovny. Dále tato funkce vytvoří socket a na portu `STATS_PORT` začne poslouchat a čekat na UDP packety s informacemi o jednotlivých požadavcích procházejících Load Balancerem, které mu posílá modul jádra. Dále spustí v novém vlákně funkci `commun_thread_3rd_function()`, jejíž funkčnost je popsána dále.

commun_thread_3rd_function

Tato funkce zajišťuje příjem dotazů ve statistickém modulu od ostatních modulů. Je spuštěna jako samostatné vlákno, které v nekonečném cyklu čeká na množině deskriptorů na příchod dotazu. Po spuštění obsahuje pouze deskriptor určený pro příjem dotazů od prezentačního modulu, při každém spojení si dané vlákno jádra vytvoří vlastní socket a jeho deskriptor je následně do množiny přidán. Po příchodu paketu s dotazem je porovnána velikost přijatých dat s kontrolní délkou dat uvedenou na konci přijatých

dat. Pokud si tato čísla odpovídají, je do fronty zpráv přidána nová zpráva obsahující typ dotazu, identifikaci socketu, ze kterého byl dotaz přijat, časové razítko, kdy byl dotaz přijat a data dotazu.

commun_send_core_to_stat

Tato funkce zajišťuje posílání informací o jednotlivých požadavcích procházejících přes Load Balancer z modulu jádra do statistického modulu. Obsah struktury *t_commun_core_to_stat* naplní do bufferu, který pošle pomocí UDP protokolu spolu s informacemi o typu dat, identifikačním čísle paketu a délce paketu statistickému modulu.

commun_receive_stat

Tato funkce zajišťuje příjem informací o jednotlivých požadavcích procházejících přes Load Balancer od modulu jádra. V nekonečném cyklu čeká na deskriptoru vytvořeném ve funkci *commun_init_stats()* na příjem UDP paketů, jejichž obsah skládá zpět do struktury *t_commun_core_to_stat*, pokud mají data z UDP paketu správnou velikost.

commun_query_3rdlayer_to_stat

```
int commun_query_3rdlayer_to_stat
    (const t_commun_3rdlayer_to_stat send_data,
     t_commun_stat_to_3rdlayer * recv_data,
     int *sock,
     const int wait)
```

Tato funkce zajišťuje v podmodulech jádra posílání dotazů statistickému modulu a přijímání odpovědí od něj a používá se v dynamických vyvažovacích algoritmech. Parametr *send_data* obsahuje strukturu s dotazem, parametr *recv_data* odkazuje na strukturu, do které se uloží odpověď na daný dotaz. Pro oba uvedené parametry je nutné před použitím této funkce naalokovat paměť. Pokud již existuje otevřený socket pro dotaz, předává se jeho deskriptor v parametru *sock*, jinak je parametr roven -1 a nový socket je otevřen v rámci této funkce. Parametr *wait* určuje v milisekundách dobu, po kterou bude funkce čekat na vrácení odpovědi. Pokud odpověď v požadované době nedostane, vrací funkce hodnotu -1. Funkce pracuje následovně: pokud dané vlákno ještě nemá vytvořeno příslušný socket pro dotazy, pak ho vytvoří. Poté naplní obsah struktury *t_commun_3rdlayer_to_stat* s dotazem do bufferu a ten pak pošle pomocí TCP protokolu přes socket statistickému modulu. Následně čeká na daném socketu na odpověď, kterou po jejím příjmu naskládá do

struktury *t_commun_stat_to_3rdlayer*. Pokud odpověď nepřijde do specifikovaného timeoutu, skončí funkce s návratovou hodnotou -1.

commun_receive_query_stat

Tato funkce zajišťuje ve statistickém modulu vyzvedávání dotazů od ostatních modulů z fronty zpráv. Pokud je fronta zpráv prázdná, funkce se zablokuje, dokud do fronty není vložena nová zpráva. Pokud nevypršel timeout pro zpracování dotazu, tak je obsah zprávy složen zpět do struktury dotazu příslušné typu dotazu.

commun_send_answer_stat

Tato funkce zajišťuje ve statistickém modulu odesílání odpovědi zpět modulu, který poslal daný dotaz. Pokud nevypršel timeout pro zpracování dotazu, je podle jeho typu vytvořena z příslušné struktury odpověď, která je poté odeslána zpět modulu, který dotaz položil.

8.3 knihovna groups.c

Knihovna groups obsahuje funkci na převod jmen skupin nebo serveru na seznam serverů.

get_group_servers

```
int get_group_servers (t_core_groups groups,  
                      char *name,  
                      int *servers)
```

Tato funkce vrací seznam serverů podle zadaného jména skupiny nebo serveru. Parametr groups obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a neměla by je nijak měnit. Parametr name obsahuje dotazované jméno skupiny nebo serveru. V parametru servers funkce vrátí seznam serverů. Seznam je tvořen polem typu int o velikosti servers_num (počtu serverů). Funkce nastaví 1 u serverů, které se nacházejí v požadované skupině a 0 u ostatních. Pokud je jméno jen jméno serveru, nastaví se u něho 1 a u všech ostatních 0. Pokud je jméno skupiny nebo serveru neplatné, vrátí funkce -1, v případě úspěchu 0.

8.4 knihovna ping.c

Knihovna ping obsahuje funkci na testování jednotlivých služeb na farmě serverů.

ping_server

Funkce ping_server otestuje dostupnost služby dané v parametru address, která obsahuje IP adresu a port. Funkce implementuje testování několika základních protokolů na jejich standardních portech. Pokud zadaný port není implementován, vrátí funkce -1. Po otestování služby vrátí 1 pokud je služba dostupná, jinak 0. Součástí Load Balanceru je testování těchto služeb: SMTP, HTTP, MYSQL, FTP, POP a IMAP. Další služby je možné přidat jednoduchým upravením této funkce.

8.5 knihovna port.c

Knihovna port poskytuje funkce pro otevírání a zavírání pomocných datových spojení. Každé hlavní spojení má možnost voláním funkcí této knihovny pracovat s pomocnými datovými spojeními. Spojení jsou uložena v poli a každé je identifikováno svým indexem. Při ukončení hlavního spojení jsou všechna pomocná datová spojení uzavřena.

open_port

```
int open_port (t_core_dm_ports * ports,
               ENUM_DM_DIRECT direct,
               struct sockaddr_in server_addr,
               int *balancer_outcome,
               struct sockaddr_in *balancer_listen)
```

Tato funkce otevře nové datové spojení a typicky se volá z podmodulu na datové vrstvě. Nové spojení se skládá ze dvou částí, na spojení od klienta k Load Balanceru a od Load Balanceru k serveru. Klient a server je zde relativní podle toho, jestli se nové spojení otevírá ve směru od reálného klienta k serveru nebo opačně. Parametr ports obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a neměla by je nijak měnit. Parametr direct určuje směr nového datového spojení. Nabývá hodnot DM_DIR_CTOS pro směr od reálného klienta k reálnému serveru a DM_DIR_STOC pro opačný směr. Všechny další parametry jsou již relativní, a proto klient bude znamenat odchozí konec pomocného datového spojení (ať je to reálně server

nebo klient) a podobně pro server. Parametr `server_addr` obsahuje IP adresu a port serveru. Parametr `balancer_outcome` obsahuje číslo portu, ze kterého se má navázat nové datové spojení na klienta. Pokud obsahuje 0, číslo portu vybere systém. Pokud se nepodaří otevřít spojení ze zadaného portu, zkusí se otevřít z libovolného portu, který vybere systém. V tomto parametru je vráceno číslo skutečného portu, ze kterého bude navázáno spojení. Parametr `balancer_listen` obsahuje port, na který se má připojit klient. Pokud obsahuje 0, číslo portu vybere systém. Pokud se nepodaří přiřadit socketu daný port, zkusí se přiřadit libovolný port, který vybere systém. V tomto parametru je vráceno číslo skutečného portu, na který se může připojit klient. V případě úspěchu funkce vrací index nového pomocného datového spojení, jinak -1.

Funkce nejdříve nastaví stav nového spojení na `DM_LISTEN`, jeho směr na parametr `direct` a adresu serveru na parametr `server_addr`. Pak se pokusí nastavit parametry spojení od Load Balanceru k serveru. Vytvoří nový socket a pokusí se mu přiřadit odchozí port na `balancer_outcome`. Pokud se to nepodaří, přiřadí mu libovolný odchozí port a ten pak vrátí v parametru `balancer_outcome`. Poté se pokusí nastavit parametry spojení od klienta k Load Balanceru. Vytvoří nový socket a pokusí se nastavit port na poslouchání na hodnotu z parametru `balancer_listen`. Pokud se to nepodaří, přiřadí mu libovolný port na poslouchání a ten pak vrátí v parametru `balancer_listen`. Po úspěšném přiřazení se na socket zavolá funkce `listen`.

close_port

```
int close_port (t_core_dm_ports * ports,  
               int index)
```

Tato funkce zavře určené pomocné datové spojení. Parametr `ports` obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a neměla by je nijak měnit. Parametr `index` obsahuje index pomocného datového spojení. Pokud je daný index platný, spojení se uzavře. Funkce vždy vrací hodnotu 0.

close_all_ports

```
int close_all_ports (t_core_dm_ports * ports)
```

Tato funkce zavře všechna pomocná datová spojení. Parametr `ports` obsahuje informace jádra, které funkce podmodulu dostane ve své hlavičce a neměla by je nijak měnit. Funkce postupně projde seznam otevřených pomocných datových spojení a uzavře je. Funkce vždy vrací hodnotu 0.

8.6 knihovna `startup.c`

Knihovna `startup` zajišťuje spuštění modulů jádra a statistik. Knihovna komunikuje s procesem `init` pomocí komunikačního protokolu uvedeného v dodatku Komunikační protokol procesu `init`. Běh funkcí je implementován jako stavový automat, který doplňuje stavový automat v procesu `init`. Funkce knihovny `startup` se volají co nejdříve po spuštění jádra nebo statistik. Po úspěšně ukončené startovací sekvenci se proces jádra nebo statistik rozdělí. Rodičovský proces dále udržuje komunikaci s procesem `init` a synovský je výkonným procesem daného modulu. Rodičovský proces odpovídá na kontrolní zprávy o stavu, provádí restartování výkonného procesu, jeho ukončení a případně automatické kontroly běhu výkonného procesu.

`startup_init_core`

Tato funkce se volá při startu jádra a zajišťuje komunikaci s procesem `init` a vytvoření výkonného procesu.

`startup_init_core_ready`

Pomocí této funkce se rodičovský proces jádra dozví, že synovský výkonný proces ukončil svoji inicializaci a je připraven na požadavky.

`startup_init_stats`

Tato funkce se volá při startu statistik a zajišťuje komunikaci s procesem `init` a vytvoření výkonného procesu.

`startup_init_stats_ready`

Pomocí této funkce se rodičovský proces statistik dozví, že synovský výkonný proces ukončil svoji inicializaci a je připraven na dotazy.

Dodatek A

Komunikační protokol procesu init

A.1 Tabulka zpráv

Komunikační protokol jsou textové zprávy obsahující případné parametry. Na začátku každé zprávy je třímístný číselný kód a mezera. Pak následuje textový popis zprávy. Zprávy od procesu init jsou ve tvar 1xx, od jádra 2xx a od statistik 3xx. Zprávy v rozsahu x00 až x49 jsou odchozí informace, příkazy nebo dotazy. Zprávy v rozsahu x50 až x69 jsou pozitivní odpovědi, v rozsahu x70 až x99 negativní odpovědi a chyby.

Zprávy od initu

- 100 init start :CFGFILE_LEN** - Parametr CFGFILE_LEN určuje délku konfiguračního souboru.
- 101 CFGFILE** - Parametr CFGFILE obsahuje celý konfigurační soubor.
- 102 stats started :STATS_STARTED** - Parametr STATS_STARTED má hodnotu 1, pokud byl úspěšně spuštěn statistický modul, 0 opačně.
- 130 checking stats** - Zpráva kontrolující statistiky.
- 131 checking core** - Zpráva kontrolující jádro.
- 132 shutdown** - Zpráva příkazující ukončení modulu.
- 133 restart** - Zpráva příkazující restartování modulu.

Zprávy od jádra

- 200 checking stats** - Zpráva kontrolující statistiky.
- 250 core start** - Odpověď o startu jádra.
- 251 cfgfile OK** - Odpověď, že konfigurační soubor byl v pořádku přenesen.
- 252 stats started OK** - Odpověď o přijetí zprávy o statistikách
- 253 core is running** - Zpráva, že jádro je ve stavu běhu.
- 260 core check OK** - Odpověď na kontrolní zprávu.
- 261 core restartnig** - Zpráva o přijetí příkazu k restartu, případně o stavu jádra.
- 270 core expected start** - Jádro očekávalo zprávu 100.
- 271 core expected cfgfile - too short** - Jádro očekávalo zprávu 101, přijatá zpráva byla příliš krátká.
- 272 core expected cfgfile - too long** - Jádro očekávalo zprávu 101, přijatá zpráva byla příliš dlouhá.
- 273 core read cfgfile - error** - Přijatý konfigurační soubor je chybný.
- 274 core expected stats started** - Jádro očekávalo zprávu 102.
- 275 core fork failed** - Jádro se nepodařilo vytvořit výkonný proces.
- 276 core startup timeout** - Vypršel timeout pro inicializaci výkonného procesu.

Zprávy od statistik

- 350 stats start** - Odpověď o startu statistik.
- 351 cfgfile OK** - Odpověď, že konfigurační soubor byl v pořádku přenesen.
- 352 stats is running** - Zpráva, že statistiky jsou ve stavu běhu.
- 360 stats check OK** - Odpověď na kontrolní zprávu.

- 361 stats restartnig** - Zpráva o přijetí příkazu k restartu, případně o stavu jádra.
- 370 stats expected start** - Statistiky očekávaly zprávu 100.
- 371 stats expected cfgfile - too short** - Statistiky očekávaly zprávu 101, přijatá zpráva byla příliš krátká.
- 372 stats expected cfgfile - too long** - Statistiky očekávaly zprávu 101, přijatá zpráva byla příliš dlouhá.
- 373 stats read cfgfile - error** - Přijatý konfigurační soubor je chybný.
- 374 stats fork failed** - Statistikám se nepodařilo vytvořit výkonný proces.
- 375 stats startup timeout** - Vypršel timeout pro inicializaci výkonného procesu.

A.2 Sekvence příkazů

Níže jsou popsány některé bloky zpráv, které vykonávají potřebné příkazy. Sekvence jsou uvedeny v pořadí očekávaných stavů a bez výskytu chyb.

Startování jádra

100 init start :CFGFILE_LEN

250 core start - Další možné odpovědi jsou 253 a 270.

101 CFGFILE

251 cfgfile OK - Další možné odpovědi jsou 271, 272 a 273.

102 stats started :STATS_STARTED

252 stats started OK - Další možná odpověď je 274

253 core is running - Další možné odpovědi jsou 275 a 276.

Startování statistik

100 init start :CFGFILE_LEN

350 stats start - Další možné odpovědi jsou 352 a 370.

101 CFGFILE

351 cfgfile OK - Další možné odpovědi jsou 371, 372 a 373.

352 core is running - Další možné odpovědi jsou 374 a 375.

Kontrola jádra

131 checking core

260 core check OK

Kontrola statistik

130 checking stats

360 stats check OK

Restart Load Balanceru

Statistikám se posílá zpráva pouze v případě, že jsou spuštěné. Pokud byly spuštěné a po restartu nemají být, je jim poslána zpráva 132.

133 restart - Zpráva se zašle jádru i statistikám a nezáleží na pořadí odpovědí.

261 core restartnig - Pak následuje sekvence startování jádra.

361 stats restartnig - Pak následuje sekvence startování statistik.

Ukončení Load Balanceru

Statistikám se posílá zpráva pouze v případě, že jsou spuštěné.

132 shutdown - Zpráva se zašle jádru i statistikám.

Dodatek B

Přehled některých existujících balancerů

B.1 WebMux

typ	komerční, HW
platforma	nezávislé
protokoly	všechny na bázi TCP/IP
cena	4.000 USD WebMux, 9.000 USD WebMux Pro
zajímavé featury	bezdiskové řešení

B.2 Coyote Point's Equalizer

typ	komerční, HW
platforma	nezávislé
protokoly	všechny na bázi TCP/IP
cena	od 3600 USD
zajímavé featury	jádro postaveno na průmyslovém standardu BSD kernelu

B.3 Prestwood Load Balancer

typ	komerční, SW
platforma	Windows
protokoly	HTTP
cena	100 USD
zajímavé featury	-

B.4 Zeus Load Balancer

typ	komerční, SW
platforma	Solaris SPARC, Solaris x86, SGI IRIX, Compaq Tru64, Linux Intel, Linux Alpha, Linux PowerPC, IBM AIX, HP-UX, FreeBSD
protokoly	všechny na bázi TCP/IP
cena	od 4.500 Eur za balancer vyvažující 2 servery po 12.000 Eur za zdvojený balancer vyvažující neomezeně serverů
zajímavé featury	zvýšena odolnost proti chybám zdvojením balanceru, administrace přes webové rozhraní

B.5 Distributor Load Balancer

typ	nekomerční, SW
platforma	UNIX
protokoly	všechny na bázi TCP/IP
zajímavé featury	mimo sýkorky použito rozdělování pomocí hashování IP adresy, testování živosti serveru a pravidelné testování zatížení pomocí fiktivních uživatelských požadavků

B.6 Pure Load Balancer

typ	nekomerční, SW
platforma	UNIX
protokoly	HTTP, SMTP
zajímavé featury	non forking/non threading/non blocking architektura

Dodatek C

Chronologický průběh prací

1.4.2005 obhajoba projektu

17.3.2005 odevzdání projektu

1.2005 - 3.2005 intenzivní testování

10. 2003 - 12.2004 implementace

4. 2003 - 10. 2003 podrobnější plánování a rozvržení

10. 2002 - 4. 2003 plánování, návrh architektury, rozdělení úkolů

10. 2002 první schůzka projektu