Linux Virtual Server

Projekt z przedmiotu RSO

Robert Kuźmiak
R.Kuzmiak@elka.pw.edu.pl

16 czerwca 2005

Spis treści

1 Opis

1.1 Czym jest Linux Virtual Server?

Linux Virtual Server jest zrealizowaną koncepcją serwera zbudowanego na zbiorze rzeczywistych maszyn przetwarzających żądania klientów (dalej zwane Real Server bądź RS) oraz specjalnego komputera pełniącego rolę nadzorcy i pracującego pod kontrolą systemu operacyjnego Linux (load balancer, dalej zwany Linux Director bądź LD). Serwer taki jest widziany przez klienta jako jedna maszyna. Rozwiązanie to cechuje wysoka skalowalność i niezawodność.

Linux Virtual Server może służyć do zbudowania serwera różnorakich usług. Do ważniejszych należą:

- www
- poczta elektroniczna
- cache serwery
- ftp
- Voice over IP

1.2 Jak działa LVS?

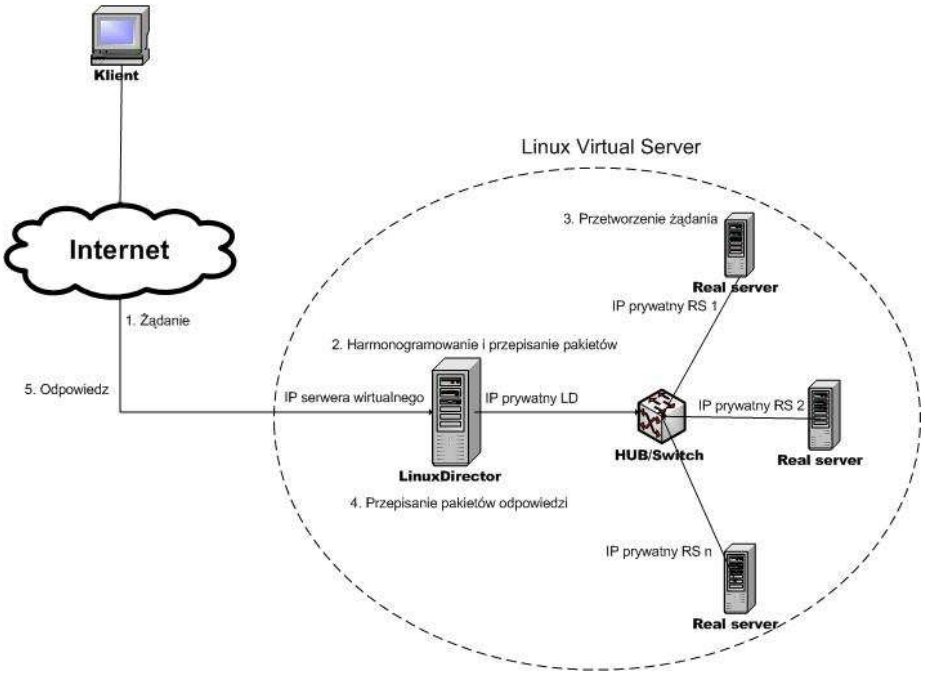
LVS opiera się na równoważeniu obciążenia przez przekierowywanie pakietów zapytań klienckich przez Linux Director.
Istnieją trzy sposoby na przekierowywanie pakietów:

1. translacja adresów sieciowych (ang. *Network Adress Translation*)
2. tunelowanie pakietów IP (ang. *IP Tunneling*)
3. ruting bezpośredni (ang. *Direct Routing*)

Poniżej znajduje się opis tych trzech sposobów.

1.2.1 NAT

Linux Virtual Server działający na NAT można przedstawić na następującym schemacie:



Rysunek 1: LVS wykorzystujący NAT

Pakiet żądania połączenia klienta dochodzi najpierw do Linux Director. Sprawdzany jest adres i port docelowy pakietu. Jeśli odpowiadają one serwerowi wirtualnemu żądanie jest obsługiwane dalej. Algorytm szeregownia decyduje, który Real Server będzie obsługiwał połączenie. Po wybraniu serwera połącznie jest zapamiętywane w tablicy haszującej znajdującej się w LD. Następnie w pakietach żądania zostaje przepisany docelowy adres IP i port na takie, jakich używa wybrany RS. Pakiet jest przekazany do RS. Kiedy od klienta przyjdą następne pakiety tego połączenia zostaną one zidentyfikowane dzięki informacji w tablicy haszującej i przekazane do tego samego RS. Pakiety odpowiedzi mają zamieniany adres i port źródłowy na takie wartości, jakie ma Virtual Server. Jeśli połącznie zostanie zerwane lub minie timeout informacja o nim zostaje usunięta z tablicy haszującej.

Zalety:

- RS mogą pracować pod kontrolą jakiegokolwiek systemu operacyjnego, który osługuje protokół TCP/IP
- RS używają adresów nierutowalnych (z klasy adresów prywatnych), co powoduje zwiększenie bezpieczeństwa systemu
- prostota konfiguracji - w podstawowej konfiguracji musimy skonfigurować jedynie LD

Wady:

- ograniczona skalowalność - ponieważ zarówno pakiety przychodzące jak i wychodzące muszą przejść przez LD(co więcej musi być na nich dokonana trans-

lacja adresów sieciowych) okazuje się, że przy skalowaniu systemu LD szybko staje się wąskim gardłem systemu. Jak pokazują wyliczenia z

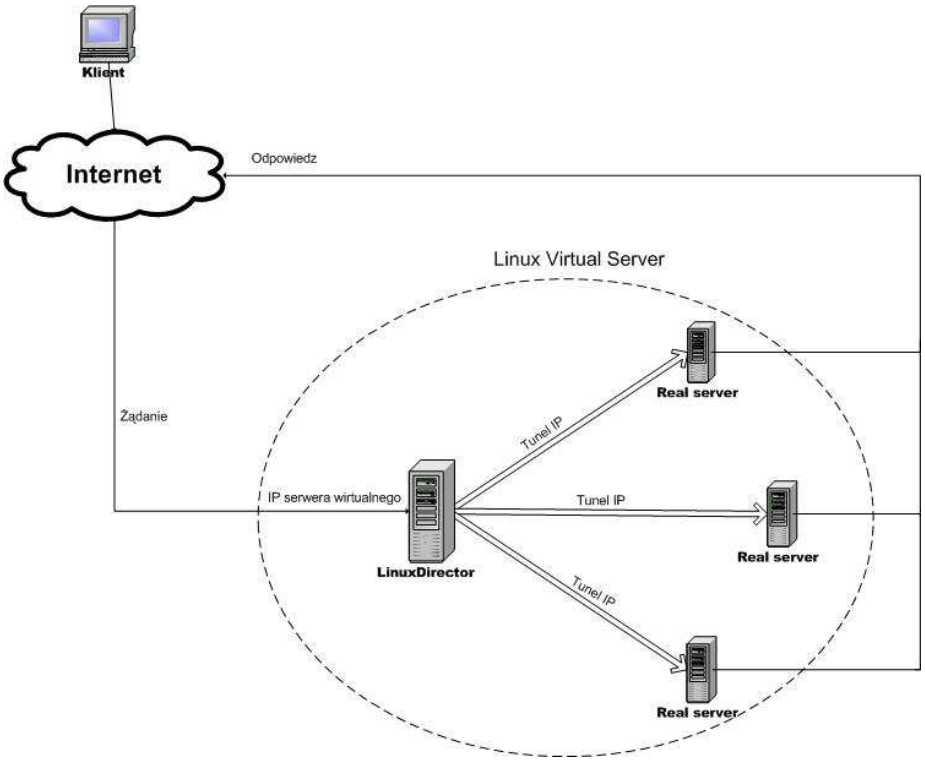
$$LVS$$

dla standardowego LD z interfejsami ethernet 100 Mbps i architekturze PC (procesor Pentium, szyna PCI 66 MHz) oraz przy założeniu, że pojedynczy RS jest w stanie wygenerować ruch na poziomie 400 kB/s, LD staje się wąskim gardłem już przy 22 RS. Można wprawdzie obejść to ograniczenie (np. stosując wiele LVS i Round-Robin DNS), ale wymaga to znacznej zmiany architektury.

- cały LVS (LD i wszystkie RS) muszą znajdować się w jednej sieci prywatnej (jedna sieć z klasy adresów prywatnych, czyli 10.*.*.*, 172.16.0.0-172.31.0.0, 192.168.*.*)

1.2.2 Tunelowanie pakietów IP

Linux Virtual Server działający z wykorzystaniem tunelowania pakietów IP można przedstawić na następującym schemacie:



Rysunek 2: LVS wykorzystujący tuneling IP

Pakiet żądania połączenia klienta dochodzi najpierw do Linux Director. Podobnie jak w poprzednim rozwiązaniu sprawdzany jest adres i port docelowy a następnie

algorytm szeregownia decyduje, który RS obsłuży żądanie i połączenie jest zapamiętywane w tablicy haszującej. Kiedy od klienta przyjdą następne pakiety tego połączenia zostaną one zidentyfikowane dzięki informacji w tablicy haszującej i przekazane do tego samego RS.
Następnie LD dokonuje hermetyzacji (ang. *encapsulation*) pakietu - oryginalny pakiet żądania jest obudowywany w nową ramkę IP z adresem docelowym będącym adresem RS. Taki pakiet jest przesyłany do RS. Tam następuje dehermetyzacja pakietu - dodatkowa ramka zostaje zdjęta i pozostaje oryginalny pakiet żądania od klienta. Żądanie zostaje obsłużone przez RS i pakiet odpowiedzi jest przesyłany przez RS bezpośrednio do klienta.
Jeśli połącznie zostanie zerwane lub minie timeout informacja o nim zostaje usunięta z tablicy haszującej LD.

Zalety:

- każdy RS może mieć jakikolwiek adres IP, niekoniecznie w tej samej sieci co LD. Zatem LVS używający tunelowania IP może być rozproszony geograficznie.
- pakiety odpowiedzi są przekazywane do klienta bezpośrednio od RS, bez udziału LD. Zmniejsza to obciążenie LD i powoduje, że rozwiązanie takie jest znacznie lepiej skalowalne, niż przy użyciu NAT.

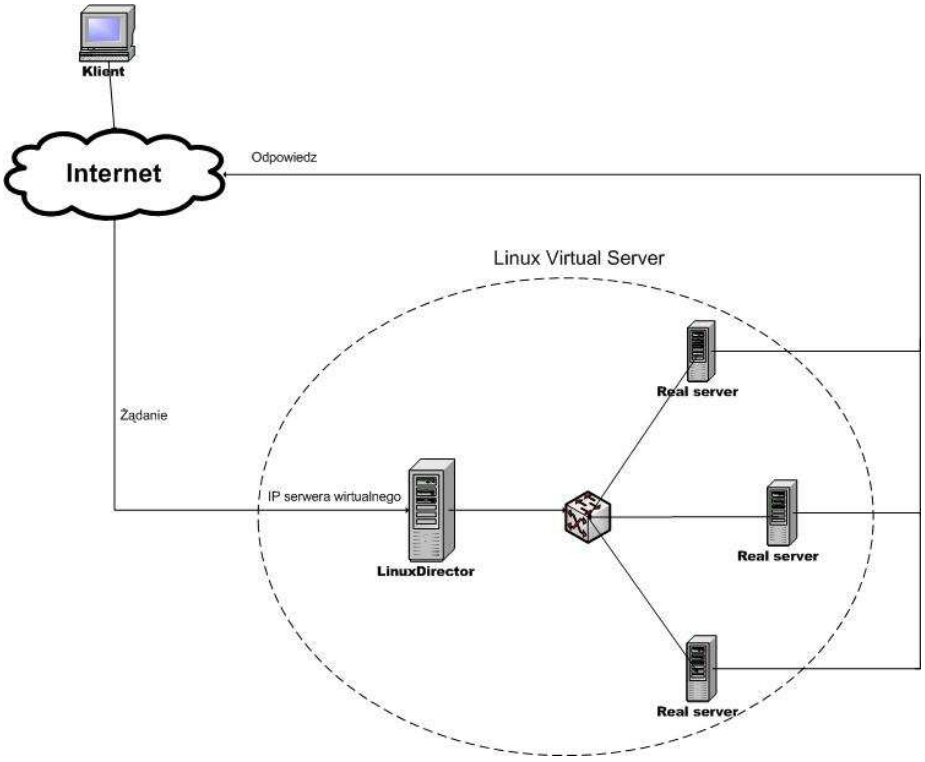
Wady:

- elementy składowe LVS (LD i RS) muszą obsługiwać tunelowanie IP. Wprawdzie na LD zawsze pracuje system Linux (który dobrze wspiera tunelowanie IP), lecz na RS - niekoniecznie (przy NAT nie było właściwie żadnych wymogów co do systemu operacyjnego RS). Używając tunelowania IP w wyborze systemu operacyjnego dla RS jesteśmy ograniczeni tylko do takich, które obsługują tunelowanie IP (np. Linux, FreeBSD).
- każdy RS wymaga dodatkowej konfiguracji tunelowania IP
- na RS musi istnieć alias o IP będącym IP serwera wirtualnego. Co więcej, musi być on skonfigurowany tak, aby nie odpowiadać na zapytania arp 1.3.

1.2.3 Ruting Bezpośredni

Linux Virtual Server działający z wykorzystaniem rutingu bezpośredniego można przedstawić na następującym schemacie:

Pakiet żądania połączenia klienta dochodzi najpierw do Linux Director. Podobnie jak w poprzednich rozwiązaniach sprawdzany jest adres i port docelowy a następnie algorytm szeregownia decyduje, który RS obsłuży żądanie i połączenie jest zapamiętywane w tablicy haszującej. Kiedy od klienta przyjdą następne pakiety tego połączenia zostaną one zidentyfikowane dzięki informacji w tablicy haszującej i przekazane do tego samego RS.
Następnie LD bezpośrednio przekazuje pakiet do RS - w oryginalnym pakiecie żądania zamieniany jest tylko adres MAC interfejsu RS docelowego. RS odbiera ten pakiet i go przetwarza (interfejs RS jako alias ma ustawiony IP całego serwera wirtualnego). Pakiet odpowiedzi jest przesyłany bezpośrednio do klienta.
Jeśli połącznie zostanie zerwane lub minie timeout informacja o nim zostaje usunięta



Rysunek 3: LVS wykorzystujący ruting bezpośredni

z tablicy haszującej LD.

Zalety:

- tak jak w tunelowaniu IP, pakiety odpowiedzi są przekazywane do klienta bezpośrednio od RS, bez udziału LD. Zmniejsza to obciążenie LD i powoduje, że rozwiązanie takie jest znacznie lepiej skalowalne, niż przy użyciu NAT.
- konfiguracja RS jest prostsza niż w rozwiązaniu z tunelowaniem. Musimy zadbać jednie o to, aby interfejs RS miał alias będacy adresem serwera wirtualnego oraz nie odpowiadał na zapytania arp.

Wady:

- cały LVS musi znajdować się w jednej sieci lokalnej (RS muszą być osiągalne z LD bez pośrednictwa ruterów, czyli na poziomie 2 warstwy sotsu ISO/OSI). Nie umożliwia to budowania LVS rozproszonego geograficznie.
- interfejs RS, do którego przywiązany jest alias IP serwera wirtualnego nie może odpowiadać na zapytania arp, co może być nietrywialnym w konfiguracji wymaganiem

1.3 Problem arp

W konfiguracjach LVS wykorzystujących tunelownie IP lub ruting bezpośredni adres IP serwera wirtualnego jest współdzielony przez wszystkie maszyny (LD i RS) serwera wirtualnego.Jeśli RS będą odpowiadały na zapytania arp o adres serwera wirtualnego i jakiś pakiet przyjdzie od klienta wystąpi wyścig - to, która maszyna odpowie na zapytanie arp (a co za tym idzie to, do kogo bezpośrednio trafi pakiet) nie będzie zdeterminowane. Ma to szczególne znaczenie w konfiguracjach, w których interfejs LD przyjmujący pakiety od klienta jest w tej samej sieci lokalnej co interfejsy RS.
Aby rozwiązać ten problem należy zagwarantować, że RS nie będą odpowiadały na zapytania arp o adres serwera wirtualnego.
Dla RS pracujących pod kontrolą systemu Linux w wersji powyżej 2.2.1 trzeba zmienić jądro systemu operacyjnego. Można to zrobić na kilka sposobów:

- kompilując jądro nałożywszy uprzednio łatę *hidden device*. Można ją pobrać z

  `http://www.ssi.bg/~ja/#hidden`

- instalując moduł jądra *noarp* autorstwa Maurizio Sartoriego. Można go pobrać z

  `http://www.masarlabs.com/noarp/`

Mając już zmienione jądro należy odpowiednio skonfugurować interfejs (opis w części opracowania dotyczącego konfiguracji).

1.4 Algorytmy szeregowania

W LVS zaimplementowano następujace algorytmy szeregowania:

1.4.1 Round-robin

Z wszystkich RS tworzona jest struktura danych będaca pierścieniem. Algorytm przydziela kolejne przychodzące żądania do kolejnych RS z pierścienia.

Przykład:
Mamy 3 RS w LVS: A, B i C. Pierwszy klient zostanie obsłużony przez A, drugi - przez B, trzeci - przez C, czwarty - przez A itd.

Zastosowania:
Najprostszy algorytm do stosowania w przypadku, gdy wszystkie RS mają podobną moc obliczeniową i wszystkie połączenia wymagają podobnego czasu przetwarzania.

1.4.2 Ważony Round-robin

Serwerom RS przypisujemy wagi proporcjonalne do ich zdolności przetważania. Wagi są liczbami naturalnymi. Ilość przekazanych do obsłużenia klientów jest wprost proporcjonalna do wagi.

Przykład:

Mamy 3 RS w LVS: A, B i C. Serwer A ma wagę 3, B - 2, C - 1. Kolejne żądania klientów zostaną obsłużone na następujących serwerach:
AABABC

Zastosowania:
Algorytm do stosowania w przypadku, gdy wszystkie połączenia wymagają podobnego czasu przetwarzania. W przeciwieństwie do zwykłego Round-Robin możemy go stosować w przpadku, gdy RS są zróżnicowane pod względem mocy przetważania. Przy stosowaniu w usługach zróżnicowanych pod względem czasu obsługi może się zdarzyć, że jeden z serwerów dostanie do obsłużenia bardziej czasochłonne zadania niż inne o tej samej wadze.

1.4.3 Najmniej-połączeń

Algorytm przydziela połączenia temu RS, z którym nawiązane jest najmniej połączeń. Algorytm musi znać nawiązane połączenia (jest to proste w LVS - wszystkie są rejestrowane w tablicy haszującej LD).
Zastosowania:
Algorytm sprawdza się, gdy czas potrzebny do obsłużenia żądania jest zróżnicowany. Nie sprawdza się jednak wtedy, gdy serwery mają zróżnicowaną moc obliczeniową. Wydawałoby się, że szybaszy serwer dostanie w jednostce czasu więcej klientów do obsłużenia, gdyż będzie przetwarzał krócej żądania. Tak jednak może się nie zdarzyć, ponieważ połączenia mogą przejść w stan TCP_WAIT i dość długo w nim pozostać nie obciążając serwera.

1.4.4 Ważony Najmniej-połączeń

Jest to zmodyfikowana wersja poprzedniego algorytmu. Serwerom RS przydziela się wagi tym większe, im większe jest ich moc obliczeniowa. Zatem wybór serwera do obsługi przychodzącego żądania zależy od otwartych połączeń i mocy obliczeniowych serwerów RS. Wybierany jest ten serwer, który ma najmniejszy iloraz $\frac{C}{W}$, gdzie C to ilość otwartych połączeń a W - waga serwera.
Zastosowania:
Algorytm zachowuje się lepiej niż poprzednie w konfiguracjach z RS o róznych mocach obliczeniowych i połączeniach o różnym czasie obsługi.

1.4.5 Najkrótszy Przewidywany Czas Obsługi

Algorytm przyporządkowuje przychodzące połączenie serwerowi RS, który ma minimalny przewidywany czas obsługi tego zadania. Wyraża się on wzorem $\frac{C+1}{U}$, gdzie C jest ilością otwartych połączeń danego serwera a U - jego wagą. Algorytm ten jest zatem bardzo podobny do *Ważonego Najmniej Połączeń*.

1.4.6 Nigdy-kolejkowy

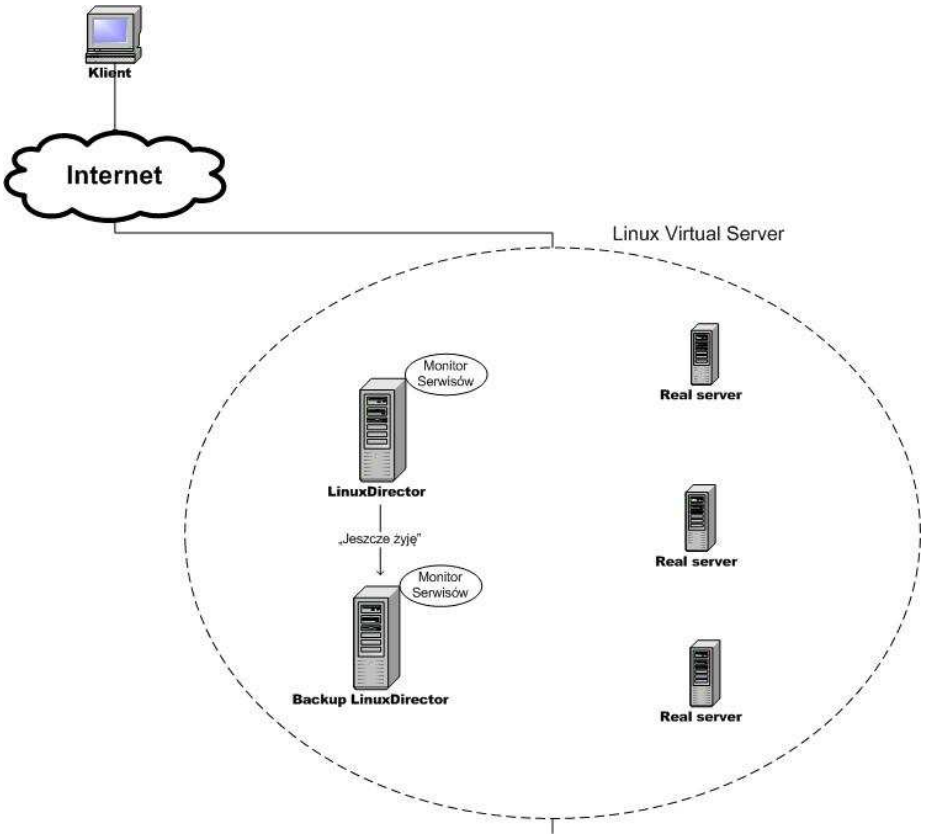
Jeśli w danej chwili istnieje RS, który nie obsługuje żadnego połączenia algorytm Nigdy-Kolejkowy (ang. *Never Queue*) przyporządkowuje przychodzące połączenie temu serwerowi. Jeśli nie ma takiego serwera połącznie zostanie obsłużone przez serwer o najmniejszym współczynniku $\frac{C+1}{U}$ (jak w poprzednim algorytmie).

1.5 Zwiekszanie niezawodności

Jeśli chcemy zwiększyć niezawodność naszego serwera witrualnego musimy wyposażyć go w redundantne zasoby, tak aby w razie awarii niektórych części systemu inne, zapasowe przejęły ich rolę. Aby stworzyć taki system musimy zatroszczyć się także o mechnizmy wykrywania awarii części systemu i przejmowania ruchu przez redundantne zasoby. W LVS zwiększanie niezawodności przez redundancję odbywa się na dwóch płaszczyznach:

1. serwery LD
2. serwery RS

Poniższy schemat przedstawia serwer o zwiększonej niezawodności LVS z redundantnymi zasobami.



Rysunek 4: Architektura LVS z zwiększoną niezawodnością

W klasycznym LVS pracuje tylko jeden LinuxDirector. Jest on krytycznym składnikiem systemu - przyjmuje wszystkie połączenia. W razie jego awarii cały system nie będzie działał. Aby zabezpieczyć się przed taką ewentualnością można wyposażyć LVS w redundantny LD. Będzie on co pewnien czas otrzymywał od głównego

LD informację typu ”Jeszcze żyję”. W razie nieuzyskania takiej informacji w wyznaczonym czasie zapasowy LD ustawi swój interfejs sieciowy tak, aby połączenia do LVS zostały przechwytywane przez niego (ustawi adres IP interfejsu na adres IP serwera wirtualnego i włączy odpowiadanie na zapytania arp). Od tej chwili będzie on LD serwera wirtualnego.
Gdy pierwszy (główny) LD powstanie po awarii możliwe są dwa rozwiązania :

1. Zapasowy LD oddaje sterowanie serwerem wirtualnym LD powracającemu do pracy.
   To rozwiązanie jest lepsze w sytuacji, gdy zapasowy LD działa na wolniejszej maszynie niż główny.

2. Powracający do pracy główny LD staje się zapasowm.
   Dzięki takiemu rozwiązaniu powrócenie głównego LD do pracy nie zakłóca pracy całego systemu.

Innym aspektem twaorzenia redundantnych serwerów LD jest zachowywanie informacji (tablicy haszującej) o otwartych połączeniach. Gdy na serwerze zapasowym LD nie ma takiej inforamcji wszystkie połączenia otwarte w momencie awarii głównego LD zostaną zerwane. Aby temu zapobiec główny LD może rozsyłać aktualną zawartość tablicy przez wielorozgłaszanie (ang. *multicast*) na protokole UDP do zapasowych serwerów LD. Przy takim rozwiązaniu zostaną zerwane tylko te połączenia, które zostały nawiązane po ostatnim uaktualnieniu tablicy otwartych połączeń w zapasowych serwerach LD.

Jeśli chcemy zabezpieczyć się przez awarią serwerów RS (z natury jest ich wiele) musimy zadbać o wykrywanie ich awarii. Po wykryciu awarii nie przekazujemy nowych połączeń do awaryjnych serwerów.
Wykrywaniem takim zajmuje się *Monitor Serwisów* (ang. *Service Monitor*) pracujący na LD. Co jakiś czas wysyła on pakiety ICMP ECHO_REQUEST do wszystkich RS w systemie. W przypadku nieotrzymania odpowiedzi od któregoś z nich usuwa ten RS z listy dostępnych serwerów. Po wykryciu powrotu funkcjonalności serwer RD zostaje z powrotem dodany do tej listy.

2 Instalacja i konfiguracja środowiska testowego

2.1 Środowisko operacyjno - sprzętowe

Do przeprowadzenia instalacji, konfiguracji i testów serwera LVS została wybrana następująca architektura:

- jeden LD pracujący pod kontrolą systemu Linux 2.6.11 z jednym interfejsem sieciowym
- od jednego do sześciu RS pracujacych pod kontrolą systemu Linux 2.6.11 z jednym interfejsem sieciowym
- od dwóch do czterech klientów pracujących pod kontrolą systemu Linux 2.6.11 z jednym interfejsem sieciowym
- wszystkie maszyny w tej samej sieci lokalnej, podłączone do jednego przełącznika ethernetowego (ang. *switch*) 100 Mbps

- jedna maszyna pracująca jako gateway
- LVS wykorzystuje ruting bezpośredni (1.2.3).
  Rozwiązanie to wybrano z uwagi na największą praktyczną stosowalność. LVS znajduje się zazwyczaj w jednej sieci lokalnej, pracuje na homogenicznych RS (tunelowanie IP nieprzydatne i skomplikowane). Nie jesteśmy także ograniczeni ilością serwerów RS (jak przy NAT).

Wszystkie komputery poza LD posiadają następującą platformę sprzętową:

- procesor główny: *Intel Pentium III* 448 MHz, $\sim$ 895 BogoMIPS
- pamięć RAM: 128 MB
- szyna PCI 2.2
- interfejs sieciowy: *3Com 3c509 (Boomerang)* , ethernet 100Mbps, podłaczony do szyny PCI
- pamięć masowa: dysk twardy EIDE

Komputer LD posiada następującą konfigurację sprzętową:

- procesor główny: *AMD Athlon Mobile* 1700 MHz, $\sim$ 3555 BogoMIPS
- pamięć RAM: 512 MB
- interfejs sieciowy: *National Semiconductor DP8381* , ethernet 100Mbps, podłaczony do szyny PCI
- pamięć masowa: dysk twardy EIDE

Sieć:

- przełącznik ethernetowy: 3Com 100 MBit/s
- kable w standardzie Cat5

Jako przedmiot testó wybrano serwer http. Wynika to z zalety jaką jest bezstanowość tego protokołu (żądania klientów będą niezależne od siebie, nie będzie problemu jeśli danego klineta obsługiwać będzie jednen RS a przy następnym połączeniu inny).

2.2 Program kliencki (benchmark)

Do przeprowadzenia testów wydajnościowych potrzebujemy rozwiązania programowego posiadającego następujące własności:

- generowanie ruchu http - uruchamianych jest wiele procesów, każdy nawiązuje połączenie z serwerem, czeka na odpowiedz i mierzy parametry takiej procedury (czy dostano odpowiedz, czas odpowiedzi)
- procesy klienckie uruchamiane na wielu maszynach - to wymaganie zapewnia, że interfejs sieciowy maszyny klienckiej nie będzie wąskim gardłem
- istnieje nadrzędny proces tworzący klientów (możliwość tworzenia różnej ilości procesów klienckich) , przydzielający im zadania o zróżnicowanych parametrach (wielkość żądanej inforamacji, ilość powtórzeń), zbierający wyniki działania klientów i tworzący z nich statystyki

Rozwiązaniem spełniającym powyższe wymagania jest *WebStone* firmy *Mindcraft* (*http://www.mindcraft.com/webstone/*).
Środowisko to składa się z programu nadrzędnego (*webserver*) i programów klienckich (*webclinet*). Model rozproszenia działania tego środowiska możemy opisać jako *farma* (jeden producent, wielu konsumentów).
Program rozpowszechniany jest na licencji typu *Public Licensce*
($http://www.mindcraft.com/webstone/license10.html$).

2.2.1 Opis instalacji i konfiguracji

Ponieważ najnowsza dostępna wersja programu *WebStone* (2.5) pochodzi z 1998 roku, występuje kilka problemów z uruchomieniem jej na współczesnym systemie linux. Wynika to z niedostosowania kodu źródłowego do specyfikacji ANSI C oraz wymogami na biblioteki rzadko obecnie spotykanego serwera *Netscape FastTrack* oraz drobnymi błędami w plikach ./confiure (np. błędnym określaniem wersji kompilatora *perl*).
W celu umożliwienia poprawnej instalacji narzędzia *WebStone* autor tego opracowania stworzył łatę (ang. *patch*) w postaci pliku typu *diff*. Łata poprawia błędy plików źródłowych związane z niezgodnością ze standardem ANSI C (np. inicjalizację zmiennych globalnych, typ zwracany przez funkcję *main*), błędy programów ./configure i wyłącza korzystanie z API serwerów *Netscape FastTrack* oraz *Microsoft IIS*
Aby zainstalować środowisko *WebStone* należy wykonać następujące kroki:

1. pobrać źródła *WebStone http://www.mindcraft.com/webstone/ws25_src.tgz* i rozpakować na maszynie na której będzie uruchamiany nadrzędny proces *webserver*

2. poprawić źródła przy pomocy dołączonego do pracy pliku $webstone/_2.5 - linux.diff$ kopiując go do katalogu z rozpakowanymi źródłami *WebStone* i wydając polecenie

   ```
   # patch -p1 < ./webstone_2.5-linux.diff
   ```

3. skompilować i zainstalować uruchamiając

   ```
   # ./webstone -setup
   ```

   Po udanej instalacji pojawi się plik konfiguracyjny *WebStone*.

4. jeśli chcemy korzystać ze środowiska graficznego i chcemy zmienić domyślnie wybraną w trakcie instalacji przeglądarkę www należy zmienić wartość zmiennej *$MOSAIC* w pliku *conf/paths.pl*

5. teraz można ustawić środowisko webstone edytując plik *conf/testbed* lub uruchamiając graficzne środowisko konfiguracyjno-uruchomieniowe poleceniem

   ```
   # ./webstone -gui
   ```

Opis najważniejszych parametrów:

- MINCLIENTS : początkowa liczba procesów klienckich
- MAXCLIENTS : końcowa liczba procesów klienckich
- CLIENTINCR : liczba o jaką zwiększa się ilość procesów klienckich w kolejnych iteracjach
- TIMEPERRUN : czas w min pojedynczego testu (dla danej liczby procesów klienckich)
- SERVER : adres testowanego serwera
- CLIENTACCOUNT : konto systemowe z którego uruchamiamy procesy klienckie
- CLIENTPASSWORD : hasło do powyższego konta
- CLINETPROGFILE : docelowy plik do ktorego zostanie skopiownay program *webclinet*
- WEBDOCDIR : katalog w którym zostaną utowrzone pliki htmldo testowania
- CLIENTS : lista maszyn na których uruchamiane będą procesy *webclinet* oddzielona spacją

6. po ustawieniu parametru WEBDOCDIR uruchomić generator plików html wydając polecenie

```
# ./webstone -genfiles
```

7. na pozostałych maszynach klienckich utworzyć konto o nazwie takiej jak parametr CLIENTACCOUNT i haśle o wartości zgodnej z CLIENTPASSWORD

8. na wszystkich maszynach uaktywnić usługi rsh, rlogin i rexec. W tym celu zainstalować pakiet *rsh-server*, wydać polecenie

```
# /sbin/chkconfig rsh on
# /sbin/chkconfig rlogin on
# /sbin/chkconfig rexec on
```

   a następnie stworzyć plik .rhosts w katalogu domowym użytkownika CLIENTACCOUNT o zawartości

```
(adres ip lub nazwa hosta serwera z procesem webserver)
```

   i prawach tylko do odczytu tylko przez właściciela.

9. wyedytować plik conf/filelist. Znajduje się w nim opis przebiegu pojedynczego klienta: zapytanie do serwera www i ilosć powtórzeń. Przykładowe pliki dołączone są do tej pracy w archiwum filelist.tar.

2.2.2 Opis działania

Webserver kopiuje program webclient na wszystkie maszyny, których ma użyć przy pomocy *rcp*. Następnie na wszystkich maszynach uruchamiana jest taka sama ilość procesów klienckich (sumaryczna liczba jest konfigurowalna).
Każdy z procesów generuje zapytanie zdefiniowane w conf/filelist. Może to być żądanie statycznej strony html albo wyników skryptu cgi-bin. Po udzieleniu odpowiedzi przez serwer klinet wykonuje następne zapytanie. Zmieniając liczby w conf/filelist możemy zmieniać ststystyczne parametry zapytań (ustalać jaki ma być stosunek plików dużych do małych, dynamicznych do aktywnych).
Wyniki działania zostają zapisane w katalogach znajdujących się w bin/runs/. Niektóre z parametrów wyników możemy przeglądać przy pomocy GUI (są pokazane tylko najważniejsze).
Pliki html są tak skonstruowane, że odpowiedz serwera html nie wymaga dużej ilości operacji dyskowych (zarówno statyczne jak i dynamiczne).

2.3 Uwagi i napotkane problemy

Podczas pracy z narzędziem *WebStone* odnotowano następujące problemy:

- obsługa rexec i rsh w dystrybucjach *RedHat 7.3* i Knoppix 3.9 jest niepoprawna. Uniemożliwia to uruchomienie *WebStone*. Jedynyą znaną dystrybucją zawierającą poprawnie działające rexec i rsh jest *Fedora Core 3*. Po zainstalowaniu tam *rsh-server* w wersji 0.17-23 *WebStone* działało poprawnie (na 2 maszynach)

2.4 Konfiguracja Linux Director

Do konfiguracji Linux Director użyto następującego skryptu:

```
#set ip_forward OFF for lvs-dr director (1 on, 0 off)
#(there is no forwarding in the conventional sense for LVS-DR)
cat    /proc/sys/net/ipv4/ip_forward
echo "0" >/proc/sys/net/ipv4/ip_forward

#director is not gw for realservers: leave icmp redirects on
echo 'setting icmp redirects (1 on, 0 off) '
echo "1" >/proc/sys/net/ipv4/conf/all/send_redirects
cat    /proc/sys/net/ipv4/conf/all/send_redirects
echo "1" >/proc/sys/net/ipv4/conf/default/send_redirects
cat    /proc/sys/net/ipv4/conf/default/send_redirects
echo "1" >/proc/sys/net/ipv4/conf/eth0/send_redirects
cat    /proc/sys/net/ipv4/conf/eth0/send_redirects

#add ethernet device and routing for VIP 192.168.1.110
/sbin/ifconfig eth0:110 192.168.1.110 broadcast 192.168.1.110 \
netmask 255.255.255.255 /sbin/route add -host 192.168.1.110 dev
eth0:110 #listing ifconfig info for VIP 192.168.1.110
/sbin/ifconfig eth0:110

#check VIP 192.168.1.110 is reachable from self (director)
```

```
/bin/ping -c 1 192.168.1.110
#listing routing info for VIP 192.168.1.110
/bin/netstat -rn

#setup_ipvsadm_table
#clear ipvsadm table
/sbin/ipvsadm -C
#installing LVS services with ipvsadm
#add telnet to VIP with round robin scheduling
/sbin/ipvsadm -A -t 192.168.1.110:80 -s rr

#forward http to realserver using direct routing with weight 1
/sbin/ipvsadm -a -t 192.168.1.110:80 -r 192.168.1.4:80 -g -w 1
/sbin/ipvsadm -a -t 192.168.1.110:80 -r 192.168.1.6 -g -w 2
/sbin/ipvsadm -a -t 192.168.1.110:80 -r 192.168.1.5 -g -w 2
/sbin/ipvsadm -a -t 192.168.1.110:80 -r 192.168.1.9:80 -g -w 1
/sbin/ipvsadm -a -t 192.168.1.110:80 -r 192.168.1.10:80 -g -w 1
/sbin/ipvsadm -a -t 192.168.1.110:80 -r 192.168.1.11:80 -g -w 1
ping -c 1 192.168.1.4
ping -c 1 192.168.1.5
ping -c 1 192.168.1.6
ping -c 1 192.168.1.9
ping -c 1 192.168.1.10
ping -c 1 192.168.1.11

#displaying ipvsadm settings
/sbin/ipvsadm
```

Podczas przeprowadzania testów należy zmieniać liczbę serwerów RS i ew. algorytm szeregowania. W powyższym skrypcie włączany jest algorytm *round robin* (rr)

```
/sbin/ipvsadm -A -t 192.168.1.110:80 -s rr
```

Możliwe są także:

- wrr : Ważony Round-robin
- lc : Najmniej-połączeń
- wlc : Ważony Najmniej-połączeń
- sed : Najkrótszy Przewidywany Czas Obsługi
- nq : Nigdy-kolejkowy

2.5 Konfiguracja Real Server

Jako RS użyto serwera *Apache* w wersji 1.3.22 działającego pod systemem Linux 2.6.11 z dystrybucji *Knoppix Live-CD 3.9*.
Konfigurację przeprowadzano w następujący sposób:

1. kopiowano wygenerowane przez *webstone -genfiles* pliki html do katalogu *www-root*

2. kopiowano plik lm/ws25_cgi (plik cgi-bin) do katalogu *ScriptAlias* serwera *Apache*. W dystrybucji *Knoppix* jest to /usr/lib/cgi-bin

3. usuchamiano następujący skrypt:

```
#installing default gw 192.168.1.254 for vs-dr
/sbin/route add default gw 192.168.1.254
#showing routing table
/bin/netstat -rn
#checking if DEFAULT_GW 192.168.1.254 is reachable
ping -c 1 192.168.1.254

#set_realserver_ip_forwarding to OFF (1 on, 0 off).
echo "0" >/proc/sys/net/ipv4/ip_forward
cat    /proc/sys/net/ipv4/ip_forward

#looking for DIP 192.168.1.1
ping -c 1 192.168.1.1

#looking for VIP (will be on director)
ping -c 1 192.168.1.110

#install_realserver_vip /sbin/ifconfig lo:110 192.168.1.110
broadcast 192.168.1.110 \
netmask 0xffffffff up #ifconfig output
/sbin/ifconfig lo:110 #installing route for VIP 192.168.1.110 on
device lo:110 /sbin/route add -host 192.168.1.110 dev lo:110
#listing routing info for VIP 192.168.1.110 /bin/netstat -rn

#hiding interface lo:110, will not arp
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
cat    /proc/sys/net/ipv4/conf/all/arp_ignore
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
cat    /proc/sys/net/ipv4/conf/lo/arp_ignore
```

2.6 Gateway

Ponieważ każdy z RS przesyła każdy pakiet odpowiedzi do domyślnego gateway (w plikach konfiguracyjnych 192.168.1.254) należy taki zapewnić, jeśli chcemy testować rozwiązanie na więcej niż jednej maszynie (jeśli tylko jeden klient to możemy jako gateway ustawić jego IP).

3 Testy

Przebieg testów:

- liczba klientów zmienna w zakresie od 100 do 500 z krokiem 100 (starać się obciążyć serwery w pełni)

- czas trwania testu 10 min (zalecany czas, wg. autorów *webStone* tylko przy takim wyniki testów są miarodajne i można porównywać je z przeprowadzonymi przez innych)
- w jednym teście lista zapytań klientów (conf/filelist) nakierowana na zapytania statyczne (dołączony plik filelist.html.standard), w drugim dynamiczne (dołączony plik filelist-cgi)

Niestety autorowi pracy nie udało się przeprowadzić zamierzonych testów. Powodem były następujące problemy:

- przy dużej ilości połaczeń na serwerach pracujących z dystrybucją *Knoppix* zaczynało brakować pamięci RAM i zawieszały się
- w laboratorium nie było dostępnych maszyn, na których działałoby poprawnie rsh i rexec (potrzebna dystrybucja *Fedora Core* - klienty działały na jednej maszynie

4 Spis dołączonych plików

W podkatalogu src katalogu lvs_rkuzmiak znajdują się:

- $webstone\_2.5 - linux.diff$: plik łaty na *WebStone 2.5*
- *filelist.tar* : archwium z przykładowymi plikami *conf/filelist*

5 Literatura

[LVS] www.linuxvirtualserver.org
[WebStone] http://www.mindcraft.com/webstone/