Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Projekt z RSO

Integracja bazy danych MySQL
z klastrem OpenSSI

Autorzy:
Piotr Bilski
Andrzej Kiewicz
Marek Kisiel
Krzysztof Skulski
Łukasz Tupaj

Prowadzący wykład:
dr inż. Tomasz Jordan Kruk

Prowadzący laboratoria:
mgr inż. Adam Kozakiewicz

Warszawa 2006

Spis treści

Część I

Podstawy teoretyczne

Rozdział 1

Wprowadzenie do realizacji niezawodności i wydajności poprzez replikację zasobów.

Replikacja zasobów już ze swojej natury zazwyczaj podnosi niezawodność oraz wydajność systemu. Jako przykład można podać macierze dyskowe, które nie dość, że zapewniają zachowanie informacji w przypadku awarii jednego z dysków, to dodatkowo zwiększają szybkość odczytywania i zapisywania danych.

Poniższe wprowadzenie ma za zadanie przybliżyć pojęcie replikacji pod kątem jej zastosowania w rozwiązaniach klastrowych. W dalszej części rozdziału pominięte zostały typowo sprzętowe metody replikacji[1]. W zamian przybliżone zostaną metody replikacji funkcjonujące w systemach rozproszonych. Przybliżone zostaną również metody zachowania spójności replikowanych danych.

1.1 Replikacja zasobów

W zależności od sposobu inicjowania oraz metody przeprowadzania wyróżnia się różne rodzaje replikacji. Poniżej zostały opisane najważniejsze z nich.

[1] M. in. wspomniane na początku rozdziału macierze dyskowe.

1.1.1 Rodzaje kopii

Kopie trwałe

Kopie trwałe tworzone są mniej lub bardziej statycznie. Zazwyczaj istnieje mała liczba kopii trwałych. Jako przykład kopii trwałej można podać np. zwielokrotnienie witryny WWW, poprzez jej umieszczenie na stanowisku lustrzanym (ang. *mirror sites*). Kopie trwałe stosuje się najczęściej połączonych (federacyjnych) bazach danych.

Kopie inicjowane przez serwer

Tworzone są w celu polepszenia wydajności. Zazwyczaj polegają na dynamicznym zwielokrotnieniu (rozproszeniu) plików z serwerów, na których potrzebne jest zwiększenie wydajności. Jednym z problemów kopii inicjowanych jest podejmowanie decyzji gdzie i kiedy kopie powinny być tworzone i usuwane.

Kopie inicjowane przez klienta

Kopie częściej znane pod pojęciem pamięci podręcznych (ang. *caches*). Służą głównie do polepszenia czasu dostępu do danych, poprzez pobranie danych z serwera i przetrzymywanie je np. na komputerze klienta. Zarządzanie tą pamięcią w tym utrzymywanie w niej aktualnych plików pozostaje w gestii klienta.

1.1.2 Rodzaje rozpowszechniania aktualizacji

Rozpowszechnianie zawiadomień o aktualizacji

W rozpowszechnianiu zawiadomień o aktualizacji stosuje się protokoły unieważniania (ang. *invalidation protocols*). Protokoły unieważnienia informują poszczególne kopie o wystąpieniu aktualizacji. Powoduje to, że niezaktualizowane dane w kopiach przestają być aktualne. W przypadku próby operacji na nieaktualnych danych następuje najpierw pobranie ich aktualnej wersji – w sposób odpowiedni dla zastosowanego modelu spójności[2].

Największą zaletą tej metody jest niskie wykorzystanie przepustowości łącza. Wykorzystanie protokołów unieważniających spisuje się najlepiej gdy współczynnik operacji czytanie/pisanie jest dość mały.

[2]Modele spójności zostały opisane w rozdziale 1.2, na stronie 14.

Przesyłanie danych z jednej kopii do drugiej

Rozpowszechnianiu poprzez przesyłanie danych z jednej kopii do drugiej polega na przesyłaniu zmodyfikowanych danych pomiędzy replikami.

Metoda ta sprawdza się w przypadku, gdy współczynnik czytanie/pisanie jest dość wysoki.

Rozpowszechnianie operacji uaktualniających

Metoda ta zwana także aktywnym zwielokrotnieniem (ang. *active replication*) zakłada, że kopia jest w stanie „aktywnie" utrzymywać aktualne dane, poprzez wykonywanie na nich operacji.

Główną zaletą tej metody jest możliwość wykonywania częstych aktualizacji przy minimalnych kosztach przepustowości łącza – przy założeniu, że rozmiar operacji modyfikujących dane jest dość mały. Wadą jest z kolei zwiększone zapotrzebowanie na moc obliczeniową replik, szczególnie w przypadku skomplikowanych modyfikacji.

1.1.3 Metody rozpowszechniania aktualizacji

Protokoły rozsyłania

Istnieją dwa rodzaje rozpowszechniania aktualizacji, które określają czy dane mają być ciągnięte (sprowadzane), czy pchane (wyprowadzane).

W metodzie polegającej na ciągnięciu aktualizacji, dane pobierane są przez replikę (serwer, klienta) w miarę potrzeb. Metoda ta jest efektywna, kiedy współczynnik czytanie/pisanie jest dość mały.

Metoda polegająca na pchaniu polega z kolei na rozprowadzani aktualnych danych do replik niezależnie od tego, czy repliki ich potrzebują czy nie. Metoda ta jest dość efektywna przy założeniu wysokiego współczynnika czytanie/pisanie.

Rodzaje rozsyłania

Kolejnym aspektem, poruszanym podczas wyboru sposobu replikacji danych, jest wybór pomiędzy komunikacją od punktu do punktu, a rozsyłaniem.

W komunikacji od punktu do punktu serwer wysyła uaktualnienia do N innych serwerów za pomocą N oddzielnych komunikatów.

Przy rozsyłaniu sieć tworząca zaplecze troszczy się o efektywne przesłanie komunikatu do wielu odbiorców. Rozsyłanie jest najczęściej lepszą (tańszą) opcją w przypadku uaktualniania replik[3].

[3]Rozgłaszanie jest szczególnie tanie w przypadku replik znajdujących się w tej samej

1.1.4 Architektury replikacji

Przy znajomości zasad replikacji wybór konkretnej architektury replikacji zależy wyłącznie od pomysłowości i doświadczenia projektanta.

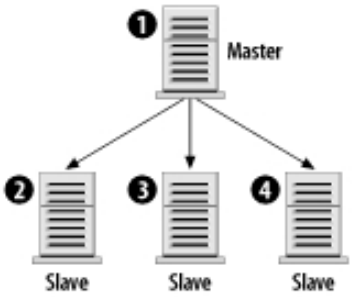
Podstawowe zasady replikacji dla serwera MySQL[4] są następujące:

- każda kopia podrzędna (ang. *slave*) musi posiadać unikalne ID,
- kopia podrzędna może podlegać tylko jednej bazie nadrzędnej (ang. *master*),
- baza nadrzędna może mieć wiele kopii podrzędnych,
- kopie podrzędne mogą pełnić funkcję bazy nadrzędnej dla innych kopii.

Poniżej przedstawiono kilka najpopularniejszych architektur wykorzystywanych podczas replikacji danych.

Master with slaves

Model *master with slaves* jest prostym i dość powszechnie stosowanym rozwiązaniem. Przydaje się w sytuacji, gdy występuje niewielka liczba za-



Rysunek 1.1: Model Master with slaves

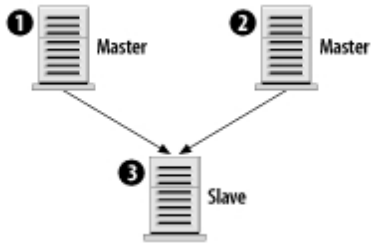
pisów, a głównym rodzajem operacji są odczyty. W takim przypadku model ten potrafi znacznie zwiększyć wydajność dostępu do danych.

sieci lokalnej, gdy istnieje możliwość wykorzystania sprzętowego rozgłaszania.

[4]Replikacja w serwerach MySQL została dokładniej omówiona na stronie 42 w rozdziale 4.2.

Slave with two masters

Architektura *slave with two masters* przydaje się w przypadku, gdy istnieje konieczność powiązanie dwóch niezależnych kopii bazy danych. Powyższa architektura umożliwia przechowywanie kopii zapasowych obu baz z zacho-



Rysunek 1.2: Model slave with to masters

waniem oszczędności sprzętu – kopie przechowywane są na jednej maszynie. W przypadku bazy MySQL istnieje ograniczenie, że kopia nie może mieć 2 baz nadrzędnych. Dlatego w powyższej architekturze, należy uruchomić 2 bazy MySQL na serwerze kopii zapasowych – niezależnie dla jednego i drugiego serwera nadrzędnego.

Dual master

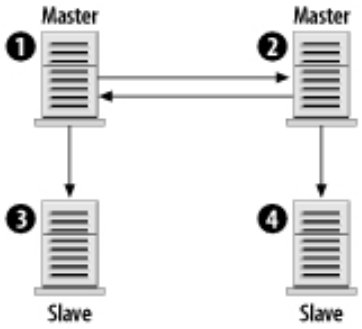
Model *dual master* wykorzystywany jest głównie w odseparowanych geograficznie częściach organizacji. Model sprawdza się w przypadku, kiedy ko-

Rysunek 1.3: Model Dual master

mórki danej organizacji zapisują głównie do „swojej" bazy danych, natomiast do drugiego serwera potrzebują wyłącznie sporadyczny dostęp. W przeciwnym wypadku narzut na komunikację pomiędzy bazami znacząco spowalnia pracę bazy.

Dual master with slaves

Architektura *dual master with slaves* dziedziczy zalety i wady po architekturze *dual master*. Dodatkowo posiada w sobie cechy architektury *master*

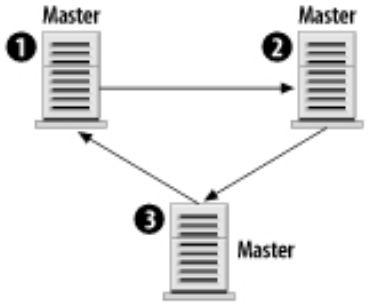


Rysunek 1.4: Model Dual master with slaves

with slaves, tak więc eliminuje pojedynczy punkt awarii (ang. SPOF – *Single Point Of Failure*) po każdej stronie głównego serwera oraz przyśpiesza operacje odczytu danych.

Multi-master (ring)

Szczególny przypadek architektury *multi-master* został omówiony podczas opisywania architektury *dual master*. W architekturze *multi-master*



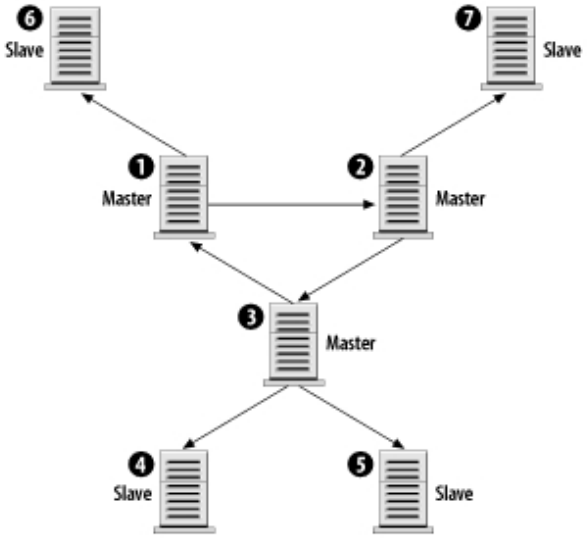
Rysunek 1.5: Model Multi-master

każdy z serwerów jest jest serwerem nadrzędnym dla swojego sąsiada oraz

kopią zapasową dla innego. Architektura ta jest dość zawodna ze względu na występujące komplikacje w przypadku awarii jednego z serwerów[5].

Ring with slaves

Rozwiązaniem problemów z awarią serwera występującego w architekturze *multi-master*, jest poszerzenie serwerów o kopie zapasowe. Powstały w



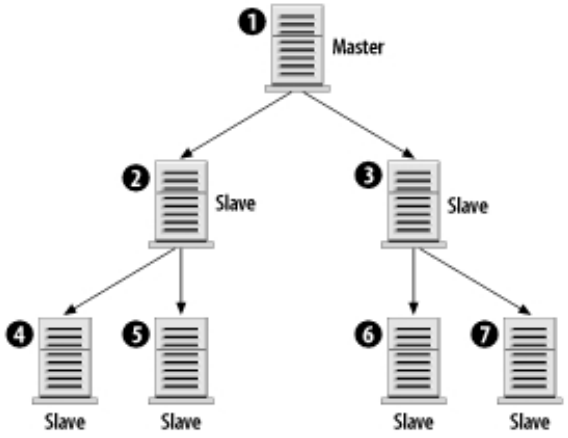
Rysunek 1.6: Model Ring with slaves

ten sposób model nosi nazwę architektury *ring with slaves*. Zalety poszerzenia architektury o kopie zapasowe są podobne jak w przypadku modelu *dual master with slaves* – zmniejsza się przypadki awarii pojedynczego serwera w pierścieniu oraz przyśpiesza się operacje odczytu z poszczególnych replik.

Pyramid

Architektura piramidy sprawdza się w przypadku znacznie rozproszonych organizacji, w których nie ma bezwzględnego nakazu (lub możliwości) do komunikowania się z głównym serwerem. W architekturze tej repliki głównego serwera przekazują swoje dane dalej do kolejnych węzłów piramidy. Architektura ta jest powszechnie stosowana w rozproszonych bazach danych. Pozwala ona znacząco odciążyć serwer główny.

[5]W takiej sytuacji zostaje przerwany przepływ informacji pomiędzy serwerami.



Rysunek 1.7: Model pyramid

1.2 Spójność zasobów

Wybór modelu spójności danych ma znaczące znaczenie na wydajność i niezawodność systemu.

Model spójności jest „umową" pomiędzy procesami, a pamięcią danych. Głosi ona, że jeśli procesy będą przestrzegać pewnych reguł, to pamięć zobowiązuje się do poprawnej pracy.

Modele nastawione na silną spójność są zazwyczaj bardzo trudne w implementacji i są wolniejsze niż modele o mniej restrykcyjnej spójności. W zamian oferuję one jednak lepszą spójność danych. Tak więc wybór odpowiedniej metody zależy wyłącznie od umiejętności i potrzeb projektanta.

1.2.1 Modele spójności nastawione na dane

Poniżej przedstawiony zostały wymienione modele spójności, nastawione na dane, wraz w warunkami koniecznymi do ich spełnienia.

Spójność ścisła

Najostrzejszy model spójności. Występuje on w jednoprocesorach. Uzyskanie spójności ścisłej w rozproszonym środowisku jest niemożliwe, ze względu na teorię względności Einsteina. Model jest spójny jeżeli spełnia warunek:

Każde czytanie danej x zwraca wartość odpowiadającą wynikowi ostatnio wykonanej operacji pisania x.

Spójność sekwencyjna

Nieco słabszy rodzaj spójności niż spójność ścisła. Model jest sekwencyjny, jeżeli spełnia warunek:

Wynik dowolnego wykonania jest taki sam, jak gdyby operacje (czytania i pisania) wszystkich procesów na pamięci danych były wykonane w pewnym porządku jedna po drugiej, przy czym operacje każdego poszczególnego procesu wystąpiły w tym ciągu w kolejności określonej przez jego program.

Spójność przyczynowa

Model słabszy niż model spójności sekwencyjnej. Osłabienie polega na tym, że rozróżnia się zdarzenia, które mogą być powiązane przyczynowo i zdarzenia nie pozostające w tym związku. Model jest spójny przyczynowo, jeżeli spełnia warunek:

Zapisy potencjalnie powiązane przyczynowo muszą być widziane przez wszystkie procesy w takim samym porządku. Zapisy współbieżne mogą być na różnych maszynach oglądane w różnej kolejności.

Spójność FIFO

Osłabienie modelu spójności przyczynowej poprzez zrezygnowanie z warunku, że procesy, które są powiązane przyczynowo muszą być widoczne dla wszystkich w jednakowym porządku. Model jest spójny FIFO, jeżeli spełnia warunek:

Zapisy wykonane przez jednej proces są oglądane przez wszystkie inne procesy w porządku, w którym powstawały, lecz zapisy pochodzące od różnych procesów mogą być przez różne procesy oglądane w różnym porządku.

Spójność słaba

Model słabszy niż model spójności FIFO. Model ma słabą spójność, jeżeli spełnia poniższe warunki:

Dostęp do zmiennych synchronizacji, skojarzonych z pamięcią danych, są spójne sekwencyjnie. Działanie na zmiennej synchronizacji jest zabronione do czasu, aż wszystkie poprzednie zapisy zostaną wszędzie ukończone. Na jednostkach danych zabrania się wykonywania operacji czytania i pisania dopóty, dopóki nie zostaną wykonane wszystkie poprzednie operacje na zmiennych synchronizacji.

Spójność zwalniana

Model ma spójność zwalniania, jeżeli spełnia poniższe warunku:

Przed wykonaniem operacji czytania lub zapisania danych dzielonych muszą się pomyślnie sfinalizować wszystkie poprzednie nabycia dokonane przez proces. Zanim będzie wolno wykonać zwolnienie, w procesie należy zakończyć wszystkie poprzednie operacje czytania i pisania. Dostępy do zmiennych synchronizacji wykazują spójność FIFO (nie jest wymagana spójność sekwencyjna).

Spójność wejścia

Model ma spójność wejścia, jeżeli spełnia poniższe warunki:

Nabycie zmiennej synchronizacji nie może w procesie nastąpić dopóty, dopóki nie zostaną wykonane wszystkie aktualizacje strzeżonych danych dzielonych dotyczących tego procesu. Przed udzieleniem procesowi zezwolenia na dostęp do zmiennej synchronizacji w trybie wykluczającym, należy zagwarantować że żaden inny proces nie będzie utrzymywał tej zmiennej – nawet w trybie niewykluczającym. Po wykonaniu wykluczającego dostępu do zmiennej synchronizacji żaden inny proces nie może wykonać do niej niewykluczającego dostępu bez uzgodnienia tego z właścicielem zmiennej.

1.2.2 Modele spójności nastawione na klienta

Poniżej przestawiono modele spójności, dla pamięci, które charakteryzują się brakiem jednoczesnych aktualizacji lub poradzeniem sobie z takimi sytuacjami, gdy do nich dochodzi.

Spójność ostateczna

Model spójności ostatecznej występuje m. in. w przestrzeni nazw DNS lub sieci WWW, gdzie nie występują konflikty z zapisem danych (najczęściej dokonuje ich jeden organ do tego uprawniony), a dane zmodyfikowane stopniowo przekazywane są do pozostałych replik.

Spójność monotonicznego czytania

Model ma spójność monotonicznego czytania, jeżeli spełnia warunek:

Jeśli proces czyta wartość danej x, to każda następna operacja, którą wykona na x, będzie zwracać tę samą wartość lub wartość nowszą.

Spójność monotonicznego pisania

Model ma spójność monotonicznego pisania, jeżeli spełnia warunek:
Operacja zapisania przez proces jednostki danych x kończy się przed wszelkimi następnymi operacjami zapisania x przez ten sam proces.

Spójność czytania swoich zapisów

Model ma spójność czytania swoich zapisów, jeżeli spełnia warunek:
Skutek wykonania przez proces operacji zapisania jednostki danych x będzie zawsze widoczny w następnych operacjach czytania x tego samego procesu.

Spójność zapisów następujących po odczytach

Model ma spójność zapisów następujących po odczytach, jeżeli spełnia warunek:
Gwarantuje się, że operacja zapisania przez proces jednostki danych x, następująca po wcześniejszym wykonaniu przez ten sam proces operacji czytania x, będzie miała do czynienia z tą samą lub nowszą wartością x, która została przeczytana.

1.3 Protokoły spójności

W poprzednim punkcie przedstawione zostały definicje poszczególnych modelów spójności. Poniżej przedstawiono ich rzeczywiste zastosowanie w protokołach spójności.

1.3.1 Protokoły oparte na kopii podstawowej

W protokołach opartych na kopii podstawowej każda jednostką danych x w pamięci danych ma przyporządkowaną kopię podstawową, odpowiedzialną za koordynowanie operacji zapisywania x.

Protokoły pisania zdalnego

Najciekawszymi przykładami protokołów pisania zdalnego są protokoły podstawa-zapas (ang. *primary-buckup protocols*).

W protokole tym proces, który chce wykonać operację zapisania jednostki danych x, przekazuję tę informację do serwera głównego x. Serwer po otrzymaniu sygnału dokonuje uaktualnienia lokalnej kopii x, po czym przekazuje uaktualnienie do serwerów zapasowych. Gdy wszystkie serwery zapasowe

uaktualnią swoje kopie lokalne, serwer podstawowy wysyła potwierdzenie do procesu, który zapoczątkował działania.

Protokoły pisania lokalnego

Odmianą protokołu pisania lokalnego jest protokół podstawa-zapis, w którym kopia podstawowa wędruje między procesami, które życzą sobie operacji pisania.

1.3.2 Protokoły zwielokrotnionych zapisów

W odróżnieniu od protokołów kopii podstawowej w protokołach zwielokrotnionych zapisów istnieje możliwość wykonania operacji pisania na wielu kopiach.

Aktywne zwielokrotnianie

W aktywnym zwielokrotnianiu z każdą kopią skojarzony jest proces, który wykonuje operacje aktualizacji. Aktualizacje są zazwyczaj rozpowszechniane za pomocą operacji pisania.

Protokół oparty jest w gruncie na komunikacji rozsyłanej z zapobieganiem, by ten sam komunikat nie był rozsyłane przed różne kopie.

Potencjalnym problemem tego protokołu jest konieczność wykonywania operacji wszędzie w tym samym porządku.

Protokoły oparte na kworum

Protokoły oparte na kworum prezentują nieco inne podejście do realizacji zwielokrotnionych zapisów.

W protokołach tych, przed czytaniem lub zapisem zwielokrotnionych danych, wymaga się uzyskania pozwolenia na przeprowadzenie danej operacji od wielu serwerów (połowy + 1). W protokole tym plikom nadawane są numery wersji. Po sprawdzeniu, że dana wersja znajduje się na połowie + 1 serwerów przyjmuje się, że jest to wersja najnowsza pliku.

1.4 Podsumowanie

Replikacja nie jest idealna, jak większość rozwiązań posiada ona swoje wady i zalety, przy czym zależą one głównie od konkretnej metody replikacji oraz protokołu wykorzystywanego do zapewniania spójności danych.

Replikacja zapewnia m. in.:

- dystrybucję danych – replikacja ułatwia rozmieszczanie kopii danych w odległych miejscach,
- rozłożenie obciążenia – poprzez replikację tych samych danych na różnych maszynach zmniejsza się obciążenie, a przez to zwiększa wydajność poszczególnych serwerów,
- zapewnienie kopii bezpieczeństwa – w przypadku bardzo obciążonych serwerów wykonywanie kopii bezpieczeństwa może być trudne do przeprowadzenia, stopniowo uaktualniana replika danych jest więc o wiele lepszym rozwiązaniem[6]
- wysoką dostępność oraz tolerowanie awarii – stosując repliki unika się pojedynczego punktu awarii, przez co znacznie zwiększa się odporność na awarię oraz dostępność rozwiązania.

Wszystko zależy jednak od wybranej architektury replikacji oraz przyjętego modelu spójności. Przy replikacji ciężko jest osiągnąć aktualny stan danych we wszystkich replikach, umożliwić zapisy w replikach, jednocześnie zakładając wysoką wydajność oraz spójność zasobów. Dlatego nie ma idealnej architektury dla wszystkich zastosowań. W zależności od potrzeb należy wybrać tą, która będzie najbliższa spełnieniu oczekiwanych potrzeb.

[6]Niż dokonywanie całościowej kopii bezpieczeństwa.

Rozdział 2

Wprowadzenie do wysokiej dostępności i niezawodności rozproszonych zasobów

Awarie systemów są niepożądanym zjawiskiem. Obecnie dużo uwagi poświęca się zapewnieniu odpowiedniej niezawodności oraz zminimalizowaniu czasu niedostępności serwerów. Powyższe aspekty są szczególnie ważne w odniesieniu do klientów, którzy coraz częściej rezygnują z usług w przypadku ich wadliwego działania.

2.1 Wysoka dostępność

Systemy wysoko dostępne (ang. *High Availability*) projektowane są w celu zapewnianie ciągłej (wysokiej) dostępności do usług.

2.1.1 Wprowadzenie

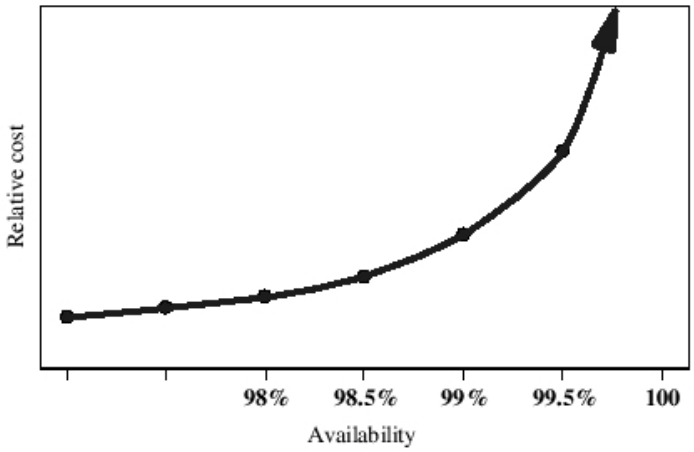
Dostępność definiuje się (w procentach) jako gotowość systemu do prawidłowego działania (ang. *uptime*). W zależności od zastosowania i posiadanych funduszy projektant może dążyć do jednej z wartości dostępności przedstawionej na rysunku 2.1. Przykładowo dla dostępności określonej na poziomie 99.999% czas niedostępności usługi wynosi 5.25 minuty w ciągu roku oraz 6 sekund w ciągu tygodnia. Z kolei w dostępności na poziomie 98% usługa jest niedostępna przez 3 godziny i 22 minuty w ciągu tygodnia. Jak łatwo się zorientować czas niedostępności, przedstawiony w drugim przypadku, jest nie do zaakceptowania w przypadku usług o charakterze krytycznym – występujących m. in. w bankowości.

Uptime in Nines	Downtime (%)	Downtime Per Year	Downtime Per Week
One (98%)	2	7.3 days	3 hours, 22 minutes
Two (99%)	1	3.65 days	1 hour, 41 minutes
Three (99.9%)	0.1	8 hours, 45 minutes	10 minutes, 5 seconds
Four (99.99%)	0.01	52.5 minutes	1 minute
Five (99.999%)	0.001	5.25 minutes	6 seconds

Rysunek 2.1: Miary dostępności

W różnych opracowaniach poszczególnym wartościom dostępności przypisuje się różne skale i nazwy. Przyjęło się jednak, że systemy określane jako HA[1] mają dostępności na poziomie 99.99%.

Na rysunku 2.2 przedstawiono związek pomiędzy kosztem, a dostępnością, którą chce się uzyskać. Jak łatwo zauważyć wysoka dostępność jest bardzo



Rysunek 2.2: Koszt dostępności

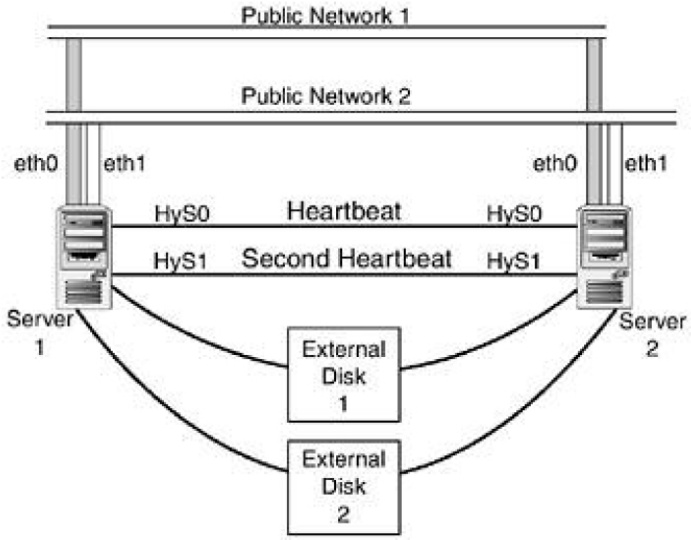
kosztowna. Przed przystąpieniem do poprawienia dostępności danej usługi warto zastanowić się jaka dostępność jest tak naprawdę dla niej potrzebna oraz czy koszt jej wdrążenia warty jest wydanych pieniędzy.

2.1.2 Optymalny model HA

Jedną z zasad, podczas budowania usług HA, jest unikanie pojedynczego punktu awarii (ang. SPOF – *Single Point of Failure*). Optymalny model HA

[1]Używany w dalszej części pracy skrót HA oznacza wysoką dostępność (ang. *High Availability*).

(według [9][2]), uzyskany poprzez redundancję „wszystkiego", przedstawiony został na rysunku 2.3. W poniższej konfiguracji wykorzystano podwojony



Rysunek 2.3: Optymalny model wysokiej dostępności

serwer. Każdy z serwerów dodatkowo połączony jest do 2 sieci publicznych, za pomocą 2 kart sieciowych oprogramowanych za pomocą VIP[3]. Każdy z serwerów jest podłączony do 2 zewnętrznych dysków. Komputery zostały połączone, między sobą, za pomocą 2 połączeń *heartbeat*, które zapewniają wymianę komunikatów o stanie pracy serwera. Dzięki zastosowaniu komunikacji *heartbeat*[4] możliwe jest automatyczne podniesienie uśpionej kopii zapasowej, w przypadku wykrycia awarii głównego serwera[5].

Zgodnie z zasadą o unikaniu pojedynczego punktu awarii, w przedstawionym powyżej rozwiązaniu, należy się zatroszczyć o dodatkowy generator

[2]Wysoką dostępność uzyskuje się głównie poprzez oprogramowanie, z kolei niezawodność, która została omówiona dalej, poprzez modyfikacje sprzętowe. Pojęcia te są dość ściśle od siebie zależne i często autorzy różnych artykułów wymieniają cechy rozwiązań niezawodnych jako pożądane w rozwiązaniach wysokiej dostępności i na odwrót – co nie do końca zgodne jest z prawdą.

[3]VIP oznacza wirtualny IP (ang. *virtual IP*), który może być przenoszony z jednego interfejsu na drugi. VIP zostanie dokładniej omówiony, w części poświęconej OpenSSI, na stronie 28 w rozdziale 3.1.3.

[4]Komunikacja *heartbeat* opiera się w dużym uproszczeniu na wysyłaniu wiadomości: „Jesteś tam?" i uzyskiwaniu odpowiedzi z drugiej strony: „Tak jestem".

[5]Przy założeniu, że jeden z komputerów był serwerem głównym, a drugi jego kopią zapasową.

prądu, na wypadek awarii sieci energetycznej.

Wszelkie awarie w klastrowych rozwiązaniach HA powinny być automatycznie maskowane i niewidzialne dla użytkownika końcowego.

2.2 Niezawodność rozproszonych zasobów

Pojęcie wysokiej dostępności (ang. *high availability*) jest obecnie tak popularne, że często opisując pożądane cechy klastrów typu HA wymienia się wśród nich m. in. cechy, które odpowiadają za niezawodność systemów.

2.2.1 Definicja niezawodności

Niezawodność to zdolność systemu do ciągłego, bezawaryjnego działania. System o dużej niezawodności powinien działać nieprzerwania w długim okresie czasu. Subtelna różnica pomiędzy niezawodnością i dostępnością polega na tym, że:

- jeśli system wyłącza się na jedną milisekundę co godzinę, to jego dostępność wynosi 99.9999%, lecz mimo tego jest bardzo zawodny
- z kolei system, który nie załamuje się nigdy, lecz jest wyłączany raz w roku na 2 tygodnie, ma dużą niezawodność, jednak jego dostępność wynosi jedynie 96%.

2.2.2 Osiąganie niezawodności

Rozpatrując aspekt niezawodności należy wziąć pod uwagę osobno niezawodność sprzętu oraz oprogramowania.

Niezawodność sprzętu

Przy wyborze niezawodnego sprzętu nie ma za wiele filozofii. Należy unikać nowinek technicznych, a oprzeć się na dojrzałych i dobrze przetestowanych rozwiązaniach. Wybierając sprzęt należy wyszukiwać rozwiązania, które same z siebie zwiększają niezawodność systemu, przykładem mogą tu być np. macierze dyskowe czy też komputery ze zwielokrotnionymi układami (np. płyty wieloprocesorowe), które kontynuują pracę pomimo awarii jednego ze zduplikowanych podzespołów.

Niezawodność oprogramowania

Przy wyborze niezawodnego oprogramowania należy stosować podobną ideologią jak w przypadku wyboru niezawodnego sprzętu. Należy unikać nowinek i oprzeć się na sprawdzonych rozwiązaniach. W przypadku oprogramowania należy jednak dodatkowo wziąć pod uwagę odporność systemu na ataki ze strony potencjalnych agresorów, dlatego system powinien zostać wyposażony m. in.:

- restrykcyjnie określone polityki bezpieczeństwa – np. prawa dostępu do zasobów, firewall itp.
- dodatkowe zwiększenie bezpieczeństwa poprzez zastosowanie łat na jądro systemu[6] – np. SELinux czy LIDS[7],
- bezkonfliktowe mechanizmy aktualizacji.

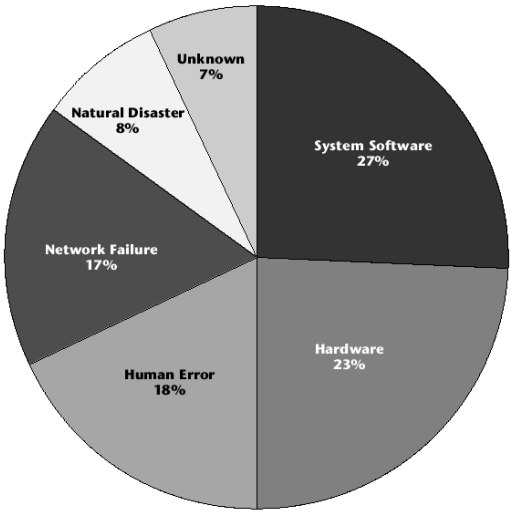
W największym skrócie można powiedzieć, że system budowany dla uzyskania wysokiej niezawodności powinien być budowany jako stabilny system-twierdza.

2.3 Podsumowanie

Opisana w powyższym rozdziale dostępność oraz niezawodność określa główne cechy, które są najczęściej wymieniane podczas projektowania klastrów HA. Opierając projekt klastra HA na niezawodnych rozwiązaniach znacząco zwiększa się jego dostępność, poprzez zmniejszenie zawodności rozwiązania. Dlatego też rozwiązania niezawodności oraz dostępności rozpatrywane są najczęściej równocześnie, jako metody służące podniesieniu jakości oferowanych usług. Na zakończenie tego krótkiego wprowadzenia warto spojrzeć na rysunek 2.4, przedstawiający rozkład czynników powodujących awarię systemów informacyjnych, które mają bezpośredni wpływ na dostępność i niezawodność systemu. Mając powyższe dane o wiele łatwiej jest podjąć konkretne kroki mające na celu podniesienie niezawodności i dostępności w projektowanym systemie.

[6] W opracowaniu ilekroć pojawia się pojęcie systemu operacyjnego, przez domniemanie należy przyjąć, że chodzi o system operacyjny oparty na jądrze Linux.

[7] Ciekawe porównanie łat na jądro Linuxa można znaleźć, w artykule Michała Piotrowskiego „Bezpieczny Linux - przegląd projektów". Artykuł został opublikowany w numerze 2/2006 magazynu hackin9, a pobrać go można ze strony *http://www.infoprof.pl/article/h206.pdf*.



Rysunek 2.4: Przyczyny awarii systemu

Rozdział 3

OpenSSI w kontekście realizowanego projektu

3.1 Specyfika rozwiązania klastrowego

3.1.1 Wstęp

Klaster komputerowy, zwany również gronem, jest to grupa połączonych jednostek komputerowych, połączonych szybką siecią komunikacyjną, współpracujących ze sobą w celu udostępnienia zintegrowanego środowiska pracy. Z punktu widzenia użytkownika sprawia wrażenie pojedynczego systemu. Klastry pracują pod nadzorem specjalnego oprogramowania, które potrafi rozdzielać zadania między wszystkie pracujące w nim maszyny. Instalacje klastrowe stosuje się w instytutach badawczych, firmach zajmujących się efektami specjalnymi w filmach, bankach, systemach bilingowych firm telekomunikacyjnych oraz wszędzie tam, gdzie niezbędne jest wsparcie dla wszelkich zadań obliczeniowych. Dotyczy to również zadań przechowywania i udostępniania dużej ilości danych.

Ze względu na szeroki zakres zastosowań, powstało wiele rozwiazań klastrowych, jednakże do najważniejszych należą Beowulf, Mosix, OpenMosix, OpenSSI, Kerrighed, DragonflyBSD, Genezy, Plurix. W tym dokumencie skupimy się na OpenSSI, na którym będzie się operało wprowadzane przez nasz zespół rozwiązanie.

Pojawiający się w nazwie skrót SSI (ang. SSI – *Single System Image*) oznacza jednolity obraz systemu. Jest to fizyczny lub logiczny mechanizm dający złudzenie, że zestaw rozproszonych zasobów (pamięć, dysk, procesor) tworzy jednolity zasób. W obecnej chwili termin SSI nie ogranicza się do jednego zasobu, ale jest coraz bardziej rozszerzany do wszystkich zasobów klastra. SSI może być wprowadzony w życie na kilku poziomach: sprzę-

tu komputerowego, systemu operacyjnego, warstwy pośredniej (middleware) oraz aplikacji. Rozwiązania klastrowe oparte na systemie operacyjnym Linux to systemy, które dojrzały juz do tego, aby być używane poza ośrodkami naukowymi. Niektóre z nich są często używane w przemyśle.

OpenSSI ukazał się w 2001 roku bazując na projekcie UnixWare Non-Stop Cluster, który powstał na bazie systemu Locus. Celem zaprojektowania OpenSSI jest dostarczenie platformy, w której mogą być zintegrowane inne technologie klastrowe oparte na otwartym oprogramowaniu. Aktualna wersja OpenSSI zawiera kilka systemów plików i systemów zarządzania dyskami opartych na otwartym oprogramowaniu (GFS, OpenGFS, Lustre, OCFS, DRSD), rozproszony mechanizm blokowania (OpenDLM – Distributed Lock Manager) i mechanizm migracji pochodzący z systemu Mosix. OpenSSI pozwala dynamicznie równoważyć obciążenie procesorów w klastrze. Mechanizm migracji OpenSSI używa procesu delegacji do obsługi IPC i wywołań systemowych po migracji procesu oraz klastrowy system plików (OCFS) do obsługi dostępu do otwartych plików.

3.1.2 Podział klastrów

Podstawowy podział klastrów ze względu na pełniona funkcje:

- *klastry wysokiej wydajności* (ang. *High Performance Computing*), zwane też klastrami do przetwarzania równoległego. Służą do masowego przetwarzania danych jednego rodzaju (np. danych naukowych lub procesów wizualizacji). Wymagają specjalnie przygotowanych programów tworzonych przy użyciu wyspecjalizowanych bibliotek programistycznych. Przykładem takiego klastra jest Beowulf.

- *klastry równoważące obciążenie* (ang. *Load Balancing Cluster*), zwane też klastrami serwerowymi. Przeznaczone są do utrzymywania bardzo obciążonych usług sieciowych (np. serwerów WWW) lub prostych zadań obliczeniowych. Ich główne zadanie polega na równoważnym dystrybuowaniu obciążenia miedzy poszczególne serwery – węzły klastra. Tego typu klaster instaluje się w systemach, w których bardzo istotny jest czas reakcji na zadanie klienta. Do tej grupy należy Mosix.

- *klastry dużej dostępności* (ang. *High Availability Computing*). Klastry tego typu nie zwiększają wydajności serwisów, a maja jedynie eliminować tzw. pojedynczy punkt awarii (ang. Single Point of Failure) – w razie uszkodzenia jednego z serwerów jego zadania są, w sposób niewidoczny dla użytkowników, przejmowane przez inny węzeł klastra.

Przykładem takiego klastra jest oprogramowanie Red Hat High Availability Server oraz oprogramowanie opracowane w ramach projektu Linux High Availability.

W praktyce rozwiązania klastrowe maja charakter mieszany i wykonują dla pewnych aplikacji funkcje wydajnościowe, przy jednoczesnej niezawodności. Często taki tryb pracy klastra dotyczy serwerów WWW, pocztowych, itp., z racji sposobu aplikacji obsługujących tego typu serwisy.

3.1.3 Komunikacja sieciowa w rozwiazaniu OpenSSI

Zagadnienie komunikacji sieciowej w rozwiązaniu OpenSSI można podzielić na dwie odrębne, choć często przeplatające się części:

- komunikację wewnętrzną,
- komunikację standardową.

Komunikacja wewnętrzna (ang. *interconnect communication*) jest związana z niskopoziomowym połączeniem pomiędzy węzłami. Jądra poszczególnych węzłów klastra używają dedykowanych połączeń do realizowania idei SSI. W komunikacji standardowej ważne jest zapewnienie widoczności klastra jako pojedynczej, wysoko dostępnej (ang. *highly available*) maszyny. W związku z tym tworzone są wirtualne adresy IP klastra (CVIP) umieszczone w sieci zewnętrznej.

Komunikacja sieciowa w przypadku rozwiązania OpenSSI w zasadniczy sposób różni się od klasycznych rozwiązań dla samodzielnych maszyn. W przypadku SSI grupa maszyn musi być widziana dla świata zewnętrznego jako jedna, skalowalna i wysoce dostępna maszyna z pojedynczym zbiorczym, adresem sieciowym. LVS zapewnia, że klaster udostępnia na zewnątrz pojedynczy adres IP, a połączenia przychodzące są rozkładane pomiędzy maszyny obsługujące odpowiednie usługi przy wykorzystaniu rozmaitych algorytmów (jest to związane z równoważeniem obciążenia sieci - *network load balancing*).

Klaster OpenSSI jest „widziany" przez klienta jak jedna maszyna. Taka przeźroczystość jest osiągana dzięki zastosowaniu CVIP (ang. *Cluster Virtual IP*). W ten sposób, łącząc się z klastrem, np. przez ssh, nie wiemy , na którym węźle jesteśmy zalogowani, na którym działają nasze procesy. Mamy do dyspozycji zasoby z wszystkich węzłów składających się na klaster.

CVIP to wirtualny adres IP klastra. Adres ten jest używany przez świat zewnętrzny do połączenia z klastrem. Należy zauważyć, że musi on być różny od zestawu adresów IP wewnątrz klastra, tzn. CVIP nie może być adresem korzenia (ang. *root node*). Węzły główne przekierowują żądania usług

obsługiwanych przez LVS do odpowiednich węzłów, zwanych prawdziwymi serwerami, na których dane usługi są faktycznie uruchomione. Węzły główne mogą dzielić funkcjonalność – mogą być zarówno serwerami głównymi, jak i prawdziwymi. W przypadku wielu adresów CVIP albo powinny być one dzielone przez wszystkie węzły główne, albo nie powinny być dzielone przez żadne węzły. Oznacza to, że więcej niż jeden CVIP może być skonfigurowany w klastrze, pod warunkiem, że wszystkie CVIP'y będą miały dokładnie te same zestawy węzłów głównych je obsługujących.

3.2 Komunikacja międzyprocesowa

3.2.1 Wstęp

Komunikacja międzyprocesowa w środowisku OpenSSI realizowana jest poprzez te same obiekty IPC, jakie występują w tradycyjnym systemie Linux. Mamy zatem do dyspozycji:

- potoki
- kolejki FIFO
- sygnały
- kolejki wiadomości
- semafory
- pamięć dzieloną
- gniazda (Internet-domain i Unix-domain)

Warto wspomnieć, że OpenSSI zawiera pakiet oprogramowania PVM oraz implementuje interfejs MPI. Obu tych mechanizmów także można użyć do komunikacji międzyprocesowej.

Rozproszona komunikacja międzyprocesowa w środowisku OpenSSI jest przezroczysta. Oznacza to, że dla semaforów, kolejek wiadomości i pamięci dzielonej jest jedna przestrzeń nazw w całym klastrze, a potoki, kolejki FIFO i gniazda są wspólne w całym klastrze. Przestrzeń nazw dla IPC jest zarządzana przez IPC Nameserver, który jest automatycznie reaktywowany po awarii węzła, na którym się on znajdował. Każdy obiekt IPC jest tworzony na tym węźle, na którym wykonywany jest proces, który go „powołuje do życia". Ale jest on również dostępny z każdego innego węzła w klastrze, skąd każdy proces może z niego korzystać w dokładnie taki sam sposób, jakby

znajdował się on na tym samym węźle, co dany obiekt IPC. Wyjątek stanowią te obiekty IPC, które zostały utworzone przez procesy wykonywane jako lokalne na danej maszynie. Ich zasięg, podobnie jak zasięg procesu jest lokalny i odwoływać się mogą do niego tylko procesy z tego węzła. Obiekty IPC nie mogą być przenoszone z jednego węzła na inny węzeł, jak np. procesy. Gdy zostaną utworzone na danym węźle, muszą już na nim pozostać aż do momentu ich usunięcia. Dlatego właśnie szybkość działania mechanizmów IPC w środowisku OpenSSI w dużej mierze zależy od tego czy proces będzie migrował, czyli przemieszczał się na inny węzeł czy też nie. Jeżeli bowiem po migracji proces odwoła się do któregoś z obiektów IPC, które wcześniej (przed migracją) utworzył (na innym węźle), to wystąpi opóźnienie związane z koniecznością przesyłania komunikatów przez sieć.

Komunikację między procesami można zapewnić na parę sposobów:

- kolejki komunikatów,
- pamięć dzielona,
- łącza nienazwane,
- łącza nazwane,

Kolejki komunikatów – mechanizm IPC (Interprocess Communication) wprowadzony w systemie UNIX w wersji V. Umożliwia on stworzenie kolejki komunikatów które mogą być odbierane przez inne procesy. Pozwala również na filtrowanie odbieranych komunikatów (określenie typów odbieranych wiadomości – pole msgtype w funkcji msgrcv). Umożliwia to adresowanie wiadomości do konkretnego odbiorcy, czyli problem zapewnienia komunikacji miedzy procesami zostaje rozwiązany. Niestety ta metoda nie umożliwia łatwego sprawdzenia czy proces, do którego wysyłamy, istnieje (wysyłamy tylko określony typ wiadomości). Kłopot może powstać zwłaszcza podczas algorytmu elekcji gdy, czekając aż wszystkie procesy "wypowiedzą" się, nie mamy możliwości sprawdzenia czy są one nadal aktywne. (funkcje w Linuxie: msgget, msgsnd, msgrcv, msgctl).

Pamięć dzielona – podobnie jak Kolejka komunikatów mechanizm wprowadzony w UNIX System V, pozwalający na określenie pewnego obszaru pamięci dostępnego dla wszystkich procesów. Pierwszym kłopotem pojawiającym się natychmiast, jest konieczność zapewnienia synchronizacji zapisów i odczytów. Ten problem można rozwiązać stosując semafory i przydział konkretnej komórki pamięci dla każdego procesu. Dalej jednak pozostaje nierozwiązany problem sprawdzania czy wszystkie procesy są aktywne. (funkcje w Linuxie: shmget, shmat, shmdt, shmctl).

Łącza nienazwane – umożliwiają jednokierunkowa komunikacje pomiędzy procesami (w ramach jednego programu). Jeden z procesów wysyła dane do łącza a inny odczytuje w takiej samej kolejności w jakiej zostały wysłane. Łącze posiada wiec organizacje kolejki FIFO (First In First Out). Sama realizacje systemowa opiera się na tworzeniu tymczasowych obiektów w pamięci jądra i udostępnienie ich poprzez interfejs systemu plików. Przy tworzeniu nowego łącza system otwiera je od razu do czytania i pisania. Ponieważ łącza nienazwane są jednokierunkowe (jeden proces zamyka odczyt, drugi zapis) aby zapewnić komunikacje dwustronna trzeba utworzyć dwa kanały dla każdego procesu. Łącza nienazwane spełniają wymaganie dotyczące komunikacji miedzy procesami, a dodatkowo umożliwiają sprawdzenie czy proces do którego wysyłamy, czy odbieramy, wiadomość jest aktywny (w przypadku zakończenia procesu zamykane są jego łącza i próba odczytania z niego wiadomości np. funkcja read kończy się zwrotem błędu). Podsumowując łącza nienazwane zapewniają:

- komunikacje pomiędzy procesami uruchomionymi na różnych węzłach,
- informacje o zakończeniu usługi na którymś węźle (funkcje w Linuxie: pipe, read, write, close).

Łącza nazwane – są pewnym rozszerzeniem łączy nienazwanych pozwalającym na komunikacje miedzy dowolnymi procesami (nie tylko w ramach jednego programu). System realizuje to poprzez tworzenie tymczasowych plików (o określonej nazwie) typu FIFO. Niestety znowu pojawiają się problemy ze sprawdzeniem istnienia innego procesu w systemie. (funkcje w Linuxie: mknod, mkfifo, open, unlink).

3.3 Zarządzanie procesami

3.3.1 Przemieszczanie procesów

OpenSSI dostarcza interfejsy dla przemieszczenia i lokacji procesów miedzy węzłami klastra. Do takich mechanizmów można zaliczyc funkcje migrate , rexec i rfork .

Funkcja migrate

Funkcja migrate() używana jest do przenoszenia jednego lub wiecej procesów na wybrany węzeł. Argumentami komendy są: węzeł oraz identyfikator procesu (pid). Jeśli podany pid jest ujemny, komenda przeniesie wszystkie procesy z grupy. Jeśli natomiast podany identyfikator procesu jest częścią

grupy wątków, to cała ta grupa będzie przeniesiona. Migracja procesu wyspecyfikowanego przez identyfikator procesu jest wykonywana przez zapisanie numeru węzła podanego jako argument do pliku /proc/pid/goto . Proces, który ma ulec migracji musi być procesem użytkownika chyba, ze użytkownik jest uprzywilejowany. Listę rodzajów procesów, które nie mogą zostać przeniesione za pomocą funkcji migrate można znaleźć w dokumentacji (man migrate).

Funkcja rexec

Funkcja rexec służy do wykonania zdalnego wywołania pliku na określonym węźle.Należy ona do rodziny funkcji, które zastępują dotychczasowy obraz procesu nowym obrazem procesu. Działa ona analogicznie do funkcji exec z ta różnicą, że specyfikuje ona węzeł, na którym obraz procesu zostanie uruchomiony. Jeśli proces, który próbujemy utworzyc działa juz na wyspecyfikowanym węźle, to nowy obraz procesu będzie działał na tym samym węźle co oryginał, a funkcja rexec da ten sam efekt co jej odpowiednik exec .

Funkcja rfork

Funkcja rfork służy do utworzenia procesu potomnego na określonym węźle. Jest ona wersja wywołania systemowego rfork. Proces potomny tworzony jest w węźle określonym przez argument funkcji. Jeśli proces macierzysty działa już na określonym węźle, to proces potomny zostaje uruchomiony na tym samym węźle co proces macierzysty, a funkcja spełnia te sama role co fork. Jeśli jako argument określający węzeł podamy CLUSTERNODE_BEST, wybrany zostanie najmniej obciążony węzeł.

3.3.2 Clusterproc

Clusterproc jest pakietem zmian, które umożliwiają zarządzanie procesami w systemach klastrowych takich jak OpenSSI. Clusterproc został stworzony aby zapewnić:

- unikalne w skali klastra numery identyfikacyjne procesów (PID‘s)
- widoczność i możliwość dostępu do procesów z jakiegokolwiek węzła klastra
- rozproszone relacje między procesami, parami parent – child, grupami procesów

- możliwość przemieszczania się wykonywanych procesów miedzy węzłami klastra (np. następna instrukcja wykonywana na innym węźle)(process migration)
- możliwość kontynuacji procesu nawet jeśli węzeł, na którym dany proces został stworzony opuści klaster
- możliwość utrzymania relacji miedzy procesami, niezależnie od tego który węzeł klastra ulegnie awarii
- pełny (i opcjonalny) /proc/¡pid¿ dla wszystkich procesów na wszystkich węzłach klastra
- możliwość wsparcia dla współdzielonego głównego systemu plików lub dla głównego systemu plików dla każdego z węzłów
- wspomaganie do 64000 węzłów w klastrze, z możliwością opcjonalnego wsparcia dla większej ilość węzłów

Ważna własnością systemów klastrowych są unikalne w skali klastra numery procesów (process id's) zgodne ze standardowym pid_t. Każdy węzeł klastra posiada własny zakres przydzielanych numerów identyfikacyjnych procesów – może zarządzać przydziałem tych numerów. Z taką strategią przydziału fork może być całkowicie lokalny i śledzenie procesów jest znacznie uproszczone ponieważ przeszukujemy tylko węzeł, który zarządza danym zasięgiem pidów.

Metoda przydziału zakresu pidów dla poszczególnych węzłów wygląda następująco: numer węzła kodowany jest na bitach wysokich a jednoznaczny identyfikator na bitach niskich numeru procesu. Przykładowo jeśli jednoznacznemu identyfikatorowi przydzielimy 16 bitów daje nam to do 65000 unikalnych procesów dla każdego węzła.

clusterproc_pid_alloc() – wstawia numer węzła do numeru pid następnie zwraca nowy pid, wykorzystuje cluster_maxnodes(zmienna konfiguracyjna określającą maksymalną ilość węzłów w klastrze) – dzięki niej jest wiadome ile bitów jest koniecznych do przydzielenia na numer węzła a co za tym idzie ile bitów zostaje na unikalny identyfikator.

3.3.3 Vproc

W celu umożliwienia stworzenia systemu rozproszonego, który pracując na wielu maszynach umożliwiałby przezroczystą dla użytkownika migrację procesów pomiędzy tymi maszynami należało stworzyć w jądrze odpowiednie struktury i mechanizmy zarządzania procesami. Mechanizmy te zostały

nazwane przez twórców OpenSSI Vproc. Rozszerzają one możliwości tradycyjnego jądra jednocześnie zapewniając jego zgodność z tymi strukturami, które nie są świadome pracy w środowisku rozproszonym.

Architektura Vproc umożliwia rozproszenie (rozdzielenie):

- Procesu rodzica i dziecka
- Grupy procesów
- Procesów należących do jednej sesji
- Procesu debugującego i debugowanego
- Procesu wysyłającego i odbierającego sygnał

Dzięki temu w systemie OpenSSI umożliwiono:

- Jeden system /proc ze wszystkimi procesami dostępny w każdym węźle klastra
- Przezroczystą migrację procesów
- Ręczna migracja procesów (pełna kontrola nad procesem migracji)
- Równoważenie obciążenia (ręczne i automatyczne)
- Przezroczysty proces debugowania
- Gwarancja poprawnej obsługi procesów w przypadku odłączania się i powtórnego dołączania do klastra jednego z węzłów.

Vproc musiał zostać tak zaimplementowany aby nie kolidował z istniejącą architekturą zarządzania procesami i aby nie trzeba było w tym celu przepisywać kodu całego jądra. Dlatego jego twórcy zdecydowali się dodać pewną wirtualną warstwę obsługi procesów , która wskazuje na tradycyjne struktury związane z procesami (listy struktur task_struct). Ta warstwa składa się z list struktur typu vproc i pvproc . Dzięki temu typowe typowe wywołania systemowe czy funkcje jądra, które nie są związane z klastrem odwołują się bezpośrednio do tradycyjnych struktur procesów, natomiast te, które są przeznaczone do pracy w klastrze odwołują się do tej wirtualnej warstwy ,która z kolei odwołuje się do tradycyjnych struktur procesów.

Struktura vproc zawiera w zasadzie tylko pid procesu, co wskazuje na odpowiednią strukturę pvproc . W strukturze pvproc natomiast zawarte są

wszelkie informacje związane z pokrewieństwem oraz dane związane z klastrem (np: na jakim węźle wykonuje się dany proces). Natomiast struktura task_struct została nieco rozbudowana, jednak wszystkie pola, które występowały w jej oryginalnym odpowiedniku występują i tutaj (dzięki temu zachowano kompatybilność). Można zauważyć, że niektóre pola ze struktury pvproc (związane z pokrewieństwem) dublują podobne pola ze struktury task_struct . Przy obsłudze systemu klastrowego takie wpisy z pvproc mają pierwszeństwo nad tymi z task_struct .

Węzeł na którym został stworzony proces nazywany jest origin node i odpowiada on za śledzenie procesu (oraz jego migrację). System gwarantuje unikalność dla numerów pid procesów. Osiągnięto to przez przydzielenie każdemu węzłowi pewnej przestrzeni dostępnych numerów pid. W ten sposób węzeł pierwszy tworzy procesy o numerach z przedziału (65536,2*65536). Drugi węzeł pokrywa przedział (2*65536,3*65536), n-ty (n*65536,(n+1)*65536). Dzięki takiemu podziałowi zawsze wiadomo, który węzeł odpowiada za jaki proces.

API użytkownika zawiera min. opisane wcześniej wywołania związane z klastrem, takie jak: rexecv(), rfork(), migrate(). Warto też tutaj opisać wywołanie rexecve, odpowiadające wywołaniu exec(), tyle, że jako czwarty argument pobiera numer węzła, na którym ma zostać wykonane. Funkcja sys_rexecve() wywołuje dvp_rexecve(), która wykonuje całą pracę.

Jak dla wywołania execve() tak i dla rexecve() istnieje szereg podobnych funkcji w bibliotece libcluster.so (rexecl, rexecle, rexeclp, rexecv, rexecvp, rexecve), które opakowują wywołanie rexecve(). We wspomnianej bibliotece są również odpowiednie funkcje opakowujące powyższe wywołania systemowe.

3.3.4 Proc

Dla użytkownika dostępny jest również rozbudowany system plików proc poprzez który w łatwy sposób można dokonywać zmian w konfiguracji samego klastra jak i poszczególnych węzłów i procesów wchodzących w jego skład. Można odnieść wrażenie, że jest to preferowany (nad wywołaniami systemowymi) interfejs programistyczny.

Tak jak w tradycyjnym systemie proc w katalogu głównym znajduje się szereg katalogów i plików związanych z fizycznym sprzętem, na którym pracuje podany system. Jednak należy pamiętać, że katalogi z numerami procesów są wspólne dla całego systemu (co nie oznacza, że każdy węzeł widzi w każdej chwili wszystkie i aktualne katalogi procesów). Dodatkowo w */proc* zostały stworzone następujące wpisy (entries) poprzez, które można kontrolować zachowanie klastra (tylko ważniejsze wpisy):

- cluster/nodeN – kontroluje zachowanie węzła o numerze N. Dostępne są pliki:
- load – obiciążenie węzła (wg. MOSIX)
- loadlevel – czy włączono równoważenie obciążenia (1 to tak)
- cluster/loadlevellist – lista aplikacji, które mogą automatycznie migrować
- cluster/events – informacje o przyłączeniach, odłączeniach węzłów
- PID – PID to numer procesu
- where – na którym węźle się proces wykonuje
- pin – przywiązanie procesu do węzła, na którym się wykonuje
- loadlevel – czy włączono równoważenie obciążenia dla tego procesu(1 to tak)
- goto – używany do migracji procesów (zapisuje się numer węzła, na który proces ma migrować)

Funkcje biblioteczne dość często korzystają z wcześniej wspomnianych (i tych nie wspomnianych) wywołań systemowych oraz danych z /proc. Znajdują się w bibliotece libcluster.so. Ważniejsze z nich to:

- `node_pid()` – zwraca numer węzła na którym wykonuje się podany proces
- `clusternode_setinfo()` – ustawia status węzła
- `clusternode_num()` – zwraca numer węzła
- `clusternode_info()` – zwraca szczegółowe informacje o węźle
- `clusternode_get_ip()` – zwraca adres ip węzła
- `clusternode_avail()` – testuje czy podany węzeł jest dostępny
- `cluster_transition()` – zwraca informacje o przyłączeniach i odłączeniach od klastra począwszy od pewnego przekazanego jako parametr numeru (transid)
- `cluster_detailedtransition()` – zwraca szczegółowe informacje dotyczące ostatnich zdarzeń na podstawie numeru transid

- `cluster_ssicong()` – testuje czy środowisko openssi jest dostępne
- `cluster_name()` – zwraca nazwę klastra
- `cluster_membership()` – zwraca numery dostępnych węzłów klastra
- `cluster_maxnodes()` – zwraca maksymalną liczbę węzłów w klastrze
- `cluster_getnodebyname()` – zwraca numer węzła na podstawie nazwy
- `cluster_getnodebyname()` – zwraca numer węzła na podstawie nazwy
- `cluster_events_register_signal()` – pozwala aplikacjom rejestrować sygnały rejestrowane przez klaster (np: o odłączaniu, dołączaniu węzłów)

3.3.5 Dostępne funkcje (podsumowanie)

- `int rexecv(const char *file, char const *argv, clusternode_t node)` – uruchamia program na wybranym węźle,
- `int rfork(clusternode_t node)` – tworzy proces potomny na wybranym węźle,
- `int migrate(clusternode_t node)` – przenosi wykonywany proces na inny wezeł (zwraca 0 jeśli proces został przeniesiony na inny węzeł)
- `int cluster_maxnodes()` – zwraca maksymalna ilość węzłów przewidziana w klastrze
- `cluster_events_register_signal` – umożliwia obsługa sygnałów przychodzących do węzła (tj. node up, node down)
- `cluster_getnodebyname` – zwraca numer węzła na podstawie jego nazwy
- `cluster_membership` – zwraca listę aktualnie aktywnych węzłów (ze stanem ustawionym na CLUSTERNODE_UP)
- `char * cluster_name()` – zwraca nazwę węzła na którym uruchomiony jest proces
- `int cluster_ssiconfig()` – zwraca czy uruchomiony jest system OpenSSI

- `int clusternode_avail(clusternode_t nodenum)` – zwraca czy dany węzeł klastra jest aktywny
- `int clusternode_info(clusternode_t nodenum, int sizeof (clusternode_info_t),clusternode_info_t *nodeinfo)` – umożliwia uzyskanie dokładnych informacji odnośnie węzła,
- `clusternode_t clusternode_num(void)` – zwraca węzeł na którym jest wykonywany proces,
- `int clusternode_setinfo(clusternode_t nodenum, int action, int sizeof (clusternode_info_t), clusternode_info_t *nodeinfo )` – umożliwia zmianę stanu węzła (wymagany jest odpowiedni poziom uprzywilejowania procesu, zmiana może dotyczyć m.in. odłączenia węzła).

3.4 Pakiety i narzędzia, które mogą okazać się pomocne do realizacji projektu

Przy realizacji projektu mogą się okazać pomocne mechanizmy zapewniające niezawodność usług w systemie OpenSSI. Do tej grupy możemy zaliczyć funkcje migrate, rexec, rfork, które opisane zostały juz wcześniej, oraz bardziej wyrafinowane narzędzia takie jak: demony spawndeamon i keepalive, itp.

3.4.1 Keepalive

Demon keepalive monitoruje procesy i demony, które sa zarejestrowane przy użyciu demona spawndaemon. Kiedy zarejestrowany proces lub demon zawodzi, keepalive zapisuje zdarzenie używajac do tego funkcji syslog i wywołuje skrypt restartujący proces lub demon. Do podstawowych cech demona keepalive można zaliczyć:

- Wprowadzenie interfejsu wiersza poleceń (spawndaemon) z wieloma opcjami dla restartowania procesów/demonów.
- Monitorowanie procesów czasu rzeczywistego
- Monitorowanie demonów
- Restartowanie procsów/demonów na dowolnym węźle w klastrze, w sposób określony przez użytkownika

Demon keepalive używa standardowych skryptów powłoki do restartowania procesów/demonów, które zostały przerwane.

3.4.2 Spawndaemon

Komenda spawndaemon uruchamia interfejs wiersza poleceń dla demnona keepalive, który monitoruje procesy i demony, restartujac je kiedy umierają. Keepalive może monitorować procesy i demony indywidualnie oraz w grupach. Spawndaemon wykonuje kilka podstawowych zadań:

- Przetwarza opcje z wiersza poleceń oraz związane z nimi pliki konfiguracyjne
- Wysyła komunikaty do demona keepalive, który następnie wykonuje wszystkie zadania związane z uruchomieniem i monitorowaniem procesów
- Czyta informacje z tablicy procesów monitorowanych przez keepalive i wyświetla te informacje dla administratora systemu.

3.4.3 Heartbeat

Zapewnienie stałej komunikacji między węzłami umożliwia wykrycie awarii komputera wchodzącego w skład klastra. Połączenie gwarantujące komunikacje nazywa się heartbeat czyli dosłownie "uderzenia serca" – węzły informują sie nawzajem, ze działają. Jeżeli któryś z węzłów przestaje odpowiadać, pozostałe przejmują jego funkcje do czasu usunięcia usterki. Pakiet heartbeat jest to publicznie dostępne oprogramowanie udostępniające podstawowa funkcjonalność pozwalająca tworzyć i zarządzać klastrami wysokiej dostępności w systemie Linux. Do głównych zadań pakietu należy:

- monitorowanie dostepności węzłów klastra,
- uruchamianie i zatrzymywanie usług działających w trybie wysokiej dostępności,
- zarządzanie zasobami dzielonymi w obrębie całego klastra (np. adres IP, zasoby dyskowe).

Jeden z komputerów w klastrze pełni role węzła głównego serwując pewne usługi, drugi (węzeł zapasowy) oczekuje bezczynnie na awarie węzła głównego. Gdy do tego dojdzie przejmuje on jego funkcje, uruchamiając wszystkie te usługi, które udostępniał węzeł główny. Oprogramowanie hearbeat nie potrafi przenosić usług i aplikacji miedzy węzłami wraz z ich stanem – wykrywa tylko awarie węzłów, przenosi zasoby na węzeł zapasowy i na nim od nowa uruchamia usługi zatrzymane w wyniku awarii. Nadaje sie do prostych instalacji np. serwerów http.

3.4.4 Fake

Fake służy do odbierania uszkodzonemu serwerowi adresu IP, dzięki czemu zapasowy serwer może udostępniać usługi. Jest to przydatne wtedy, gdy uszkodzenie wystąpi w taki sposób, że maszyna i system operacyjny będą nadal pracować (zajmując adres IP), ale dana usługa będzie niedostępna. W tym celu fake konfiguruje odpowiedni interfejs z danym adresem IP (wykorzystując IP aliasing), a następnie podszywa się pod daną maszynę wykorzystując ARP spoofing.

3.4.5 DRBD

DRBD to moduł jadra służący do mirrorowania systemów plików przez siec. Działanie DRBD polega na tym, że każdy blok danych zapisywany lokalnie na dysku jest także przesyłany do zdalnego węzła, na którym również zostaje zapisany. Odczyty są zawsze wykonywane lokalnie. Jest wiec to mechanizm ciągłej replikacji danych. W chwili obecnej tylko jeden węzeł w danej chwili może mieć dane urządzenie zamontowane do zapisu i odczytu, choć możliwe byłoby rozwiniecie tego oprogramowania tak, aby więcej niż jeden węzeł miał taki dostęp.

3.4.6 LVS (Linux Virual Server)

Jest to projekt służacy do stworzenia oprogramowania serwerowego do budowania klastra wydajnościowego. W jego skład wchodzi oprogramowanie oraz zestaw łat na jadro linuksa. Projekt ten jest rozwinięciem idei budowy klastra w oparciu o translacje adresów sieciowych. Zawiera wyspecjalizowane algorytmy rozkładania obciążenia miedzy poszczególne węzły i monitorowania stanu pracy węzłów. Klaster zbudowany jest z jednego serwera głównego, który jako jedyny jest „widziany" na zewnątrz sieci i pełni role nadrzędną nad pozostałymi węzłami klastra. Jego zadaniem jest przekazywanie zadań do kolejnych węzłów. Dzięki niemu pakiety są przekierowywane tak, żeby klient był przekonany, ze łączy sie bezpośrednio z wybranym serwerem. W rzeczywistości wygląda to w ten sposób, ze każdy pakiet przechodzi przez ten właśnie serwer główny, który decyduje do jakiego rzeczywistego serwera go przekierować. Proces przekazywania zadań jest całkowicie przezroczysty dla klientów. Ważna rola projektu jest monitorowanie stanu technicznego i obciążenia poszczególnych wezłów klastra oraz szybka reakcja na zachodzące zmiany. Architektura LVS jest trójwarstwowa, składa się z następujących elementów:

- Load balancer (dzielnik obciążenia) - jego funkcją jest rozdzielanie połączeń przychodzących od klientów pomiędzy serwery pracujące w klastrze.

- Server pool (pula serwerów) - składa się z serwerów, które udostępniają klientom usługi charakteryzujące cały klaster - są to np. ftp, poczta, www, streaming multimediów.

- Backend storage (system przechowywania danych) - najczęściej rozproszony system plików taki jak Koda lub GFS zapewniający spójność danych.

Load balancer rozdziela przychodzące połączenia pomiędzy pulę serwerów korzystając z różnych technik dzielenia obciążenia. Jego zadaniem jest też obsługa stanu tych połączeń oraz przekazywanie pakietów do puli serwerów. Cała praca wykonywana przez dzielnik obciążenia jest wykonywana w jego jądrze, co ma pozytywny wpływ na wydajność - dzięki temu LVS jest w stanie obsłużyć dużo więcej połączeń niż zwykły serwer - dzięki temu nie stanowi wąskiego gardła dla całego systemu. Węzły puli serwerów mogą być replikowane dla potrzeb skalowalności lub wysokiej dostępności.

3.4.7 Dostępne narzędzia testujące

Wraz z systemem OpenSSI dostarczone są narzędzia służące do testowania równoważenia obciążenia. Są to następujące programy:

- demo-proclb - program uruchamiający test, uruchamia procesy, przydziela im fragment zasobu, którym jest podany w argumencie plik, i odbiera od tych procesów, za pomocą kolejki wiadomości IPC, informacje o stanie przetworzenia zasobu

- demo-proclb-child - program uruchamiany przez demo-proclb jako pod proces dziecko, proces ten czyta kawałek po kawałku przydzielony zasób i zgłasza przez kolejki wiadomości IPC

- demo-proclb-monitor - program monitorujący prace programu testującego, proces czyta z pliku logów programu testującego, numery procesów i wyświetla na którym węźle klastra znajdują się procesy, wyświetla również informacje /proc/loadavg dla każdego węzła (średnie obciążenie węzła)

Rozdział 4

Rozwiązania klastrowe w MySQL

4.1 Wstęp

Rozwiązania umożliwiające realizację klastrowości, zwiększenie wydajności i niezawodności w środowisku bazy danych MySQL obejmują:

- replikację
- wykorzystanie alternatywnego systemu zarządzania pamięcią
- wykorzystujące środowiska realizującego klastrowość na poziomie systemu operacyjnego, bez dodatkowych modyfikacji samego silnika bazy danych

Opis rozwiązań dotyczy wersji 5.1 bazy danych MySQL, czyli najnowszej dostępnej na dzień przygotowania opracowania.

4.2 Replikacja w MySQL

4.2.1 Wstęp

Jest to mechanizm, który nie opiera swojego działania na klastrach. Zapewnia on jednak sposób na współistnienie i aktualizację wielu instancji tej samej bazy danych na rozłącznych komputerach. Przyczynia się tym samym do wzrostu wydajności i niezawodności systemu poprzez swoiste uczynienie go rozproszonym.

4.2.2 Opis działania

Replikacja polega na zapisywaniu zmian danych („binary logging"), które zachodzą w instancji bazy pełniącej rolę „Master" oraz asynchronicznym (nie jest narzucony żaden okres ani maksymalny czas, po którym musi dojść do pobrania zmian od instancji „Master") przeprowadzaniu tych samych zmian w instancjach bazy danych pełniących rolę „Slave". Do realizacji replikacji zaprzęgnięte są trzy wątki - jeden tworzony przez instancję „Master" a dwa tworzone przez instancję „Slave". Pierwszy z wątków, odpowiedzialny za wysyłanie zmian do instancji „Slave", zaczyna pracę od ustalenia od którego wpisu w pliku logującym zmiany należy zacząć replikację. Następnie wysyła kolejne zmiany do instancji „Slave". Tam „wątek I/O" odbiera zmiany i zapisuje je do lokalnych plików zwanych logami zmian („relay logs"). Natomiast „wątek SQL", bazując na informacjach zapisanych w logach zmian, dokonuje analogicznych zmian w lokalnej instancji bazy danych. Rozdzielenie pracy po stronie odbierającej zmiany pomiędzy dwa wątki pozwala na skrócenie do minimum czasu komunikacji pomiędzy instancjami bazy.

W powyższym opisie celowo używane było słowo „zmiana", gdyż replikacja odbywać się może na dwóch poziomach: na poziomie poleceń DML/DDL lub na poziomie wierszy. Pierwszy sposób jest zazwyczaj oszczędniejszy, ogranicza więc ilość danych, które muszą zostać przesłane pomiędzy instancjami „Master" i „Slave". Dodatkowo zapewnia on kompatybilność w wypadku, jeśli instancja „Slave" zarządzana jest przez nowszą wersję serwera MySQL, o innej strukturze wewnętrznej wierszy. Sposób ten jest jednak nieodpowiedni do zastosowań, w których polecenia DML cechują się niedeterministycznym zachowaniem, np. zawierają w sobie funkcje zwracające losowe wartości. Może również okazać się wolniejszy, jeśli polecenia DML zawierają filtry wymagające przeszukania znacznej ilości rekordów (w wypadku replikacji na poziomie wierszy, polecenie wybierające wiersze do zmodyfikowania na podstawie filtru, zostałoby zamienione na serię poleceń modyfikujących pojedyncze wiersze, bez filtru wybierającego).

4.2.3 Konfiguracja replikacji

Proces konfiguracji replikacji składa się z następujących kroków:

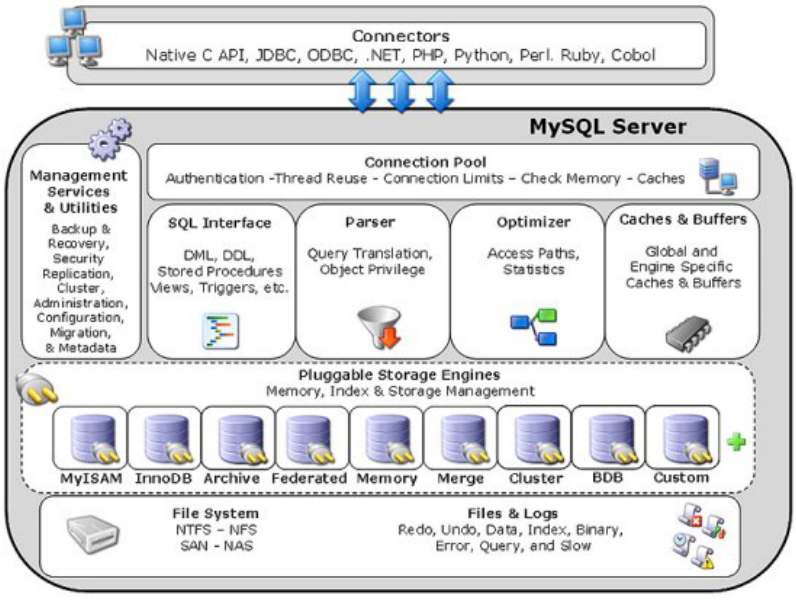
- utworzenia konta użytkownika, z którego będzie mogła korzystać instancja „Slave" podczas replikacji oraz nadania mu przywileju „replication":
- utworzenia kopii instancji „Master", będącej obrazem startowym dla instancji „Slave"

- dodania wpisów do plików konfiguracyjnych; nakazującego tworzenie pliku rejestrującego zmiany danych po stronie „Master" oraz nadającego różne numery identyfikacyjne instancjom po obu stronach
- ustawieniu w instancji „Slave" danych serwera „Master" oraz wydaniu polecenia rozpoczynającego replikację

4.3 Rozwiązanie MySQL Cluster

4.3.1 Podstawy architektury bazy danych MySQL

Rozwiązanie MySQL Cluster oparte jest na możliwości dodawania do bazy danych MySQL alternatywnych systemów zarządzania pamięcią (dwoma podstawowymi systemami zarządzania pamięcią w MySQL są MyISAM oraz InnoDB - pierwszy z nich będący prostym ale szybkim rozwiązaniem; drugi oferujący obsługę transakcji i wiążące się z tym złożone mechanizmy zapewnienia spójności odczytu, blokad i zapobiegania zakleszczeniom) w postaci wtyczek. Wtyczka taka to klasa realizująca określony interfejs, którą dołączyć można do bazy, bez potrzeby ponownej kompilacji jej samej. Struktura bazy danych MySQL przedstawiona jest na rysunku 4.1.



Rysunek 4.1: Struktura bazy danych MySQL

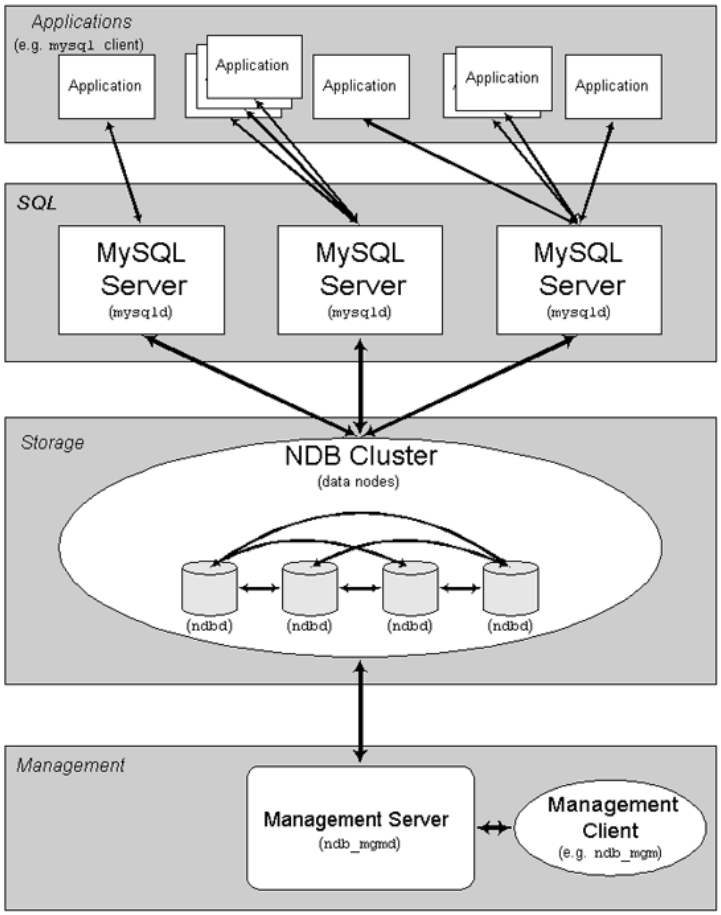
Jak widać, sam serwer bazy danych działa tak samo, niezależnie od systemu zarządzania pamięcią, z którym współpracuje. Wszystkie operacje dostępu do danych i ich struktury (tabele, przestrzenie tabel, widoki) zamieniane są po sparsowaniu na wywołania funkcji wchodzących w skład interfejsu, który każdy z systemów zarządzania pamięcią musi implementować. Np. podczas odczytywania danych z tabeli wywołane zostaną następujące funkcje:

- `store_lock()` - wywoływana przed nałożeniem blokady na obiekt
- `external_lock()` - nałożenie blokady na obiekt
- `rnd_init()` - inicjalizacja obiektu do odczytu
- `info()` - pobranie informacji o obiekcie
- `extra()` - ew. pobranie dodatkowych informacji dotyczących zapytania (nie implementowana w większości systemów zarządzania pamięcią)
- `rnd_next()` - służy do iteracji po kolejnych rekordach

4.3.2 Budowa klastra MySQL Cluster

Rozwiązanie MySQL Cluster oparte jest o system zarządzania pamięcią NDB Cluster (NDB - Network Database). Schemat działania rozwiązania przedstawiony jest na rysunku 4.2. W klastrze NDB wyróżnić można 3 podstawowe elementy:

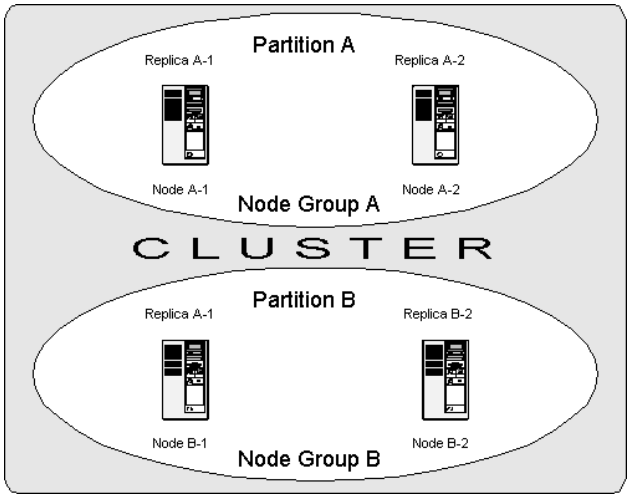
- węzły NDBD (NDBD Nodes, „węzły danych") - uruchomione procesy NDBD (NDB daemon) realizujące pojedyncze węzły systemu zarządzania pamięcią - są odpowiedzialne za przechowywanie i udostępnianie danych węzłom SQL
- węzły MGM (MGM Nodes) - węzeł lub węzły odpowiedzialne za zarządzanie pozostałymi węzłami w ramach MySQL Cluster; główne zadania to zapewnianie danych konfiguracyjnych, zatrzymywanie, uruchamianie węzłów NDBD oraz dbanie o tworzenie kopii zapasowych; bez działającego węzła MGM nie jest możliwa inicjalizacja klastra, potem klaster działać może bez niego aż do momentu wystąpienia ewentualnej awarii, kiedy to znowu węzeł MGM niezbędny jest do przywrócenia prawidłowej pracy systemu
- węzły SQL (SQL Nodes) - zwykłe serwery MySQL, skonfigurowane do korzystania z usług NDB Cluster



Rysunek 4.2: Schemat działania MySQL Cluster

Warto zaznaczyć, że wszystkie wymienione wyżej elementy sytemu uruchomione mogą być na jednym komputerze, nie ma bowiem wymagań dotyczących fizycznej niezależności jednostek wykonujących poszczególne procesy. Aby jednak korzystanie z MySQL Cluster wnieść mogło korzyści w postaci zwiększonej wydajności lub bezpieczeństwa, zalecane jest aby proces zarządczy uruchomiony był na osobnej jednostce oraz by procesy NDB uruchomione były na różnych komputerach (ewentualnie po 1 z grupy na jednej maszynie) i by w skład systemu wchodziły przynajmniej dwie repliki (kopie wszystkich danych przechowywanych w systemie). System dokonuje automatycznego partycjonowania składowanych danych dzieląc aktywne w ramach klastra procesy NDB na grupy (patrz rysunek 4.3). W ramach jednej grupy, każdy proces NDB przechowuje te same dane. Ogólnie liczba węzłów NDBD obecnych w systemie równa jest liczbie partycji pomnożonej przez

liczbę replik. Stosowanie replik pozwala na uniknięcie utraty danych i nieprzerwaną pracę całego systemu w wypadku awarii jednego z komputerów (aby system utracił możliwość normalnego działania, awarii musiałyby ulec wszystkie komputery, na których uruchomione są wszystkie procesy przynajmniej jednej grupy). Partycjonowanie danych zapewnia za to zwiększoną wydajność zapytań.



Rysunek 4.3: Partycjonowanie i podział węzłów danych na grupy

4.3.3 Wymagania sprzętowe i systemowe MySQL Cluster

Do wersji 5.1.5 system zarządzania pamięcią NDB Cluster pozwalał na składowanie danych jedynie w pamięci operacyjnej komputerów, na których uruchomione były procesy NDBD. Od wersji 5.1.6 możliwe stało się przechowywanie tych kolumn tabel, na które nie są nałożone indeksy, w pamięci trwałej. Niezależnie od miejsca przechowywania danych, istnieje mechanizm logowania zmian w logu REDO, składającym się z co najmniej dwóch plików przechowywanych w pamięci trwałej. Możliwe jest do tego zastosowanie mechanizmów replikacji i tworzenia kopii zapasowych, które działają analogicznie jak w wersji nieklastrowej bazy danych . W wypadku replikacji, jeden z węzłów danych klastra wyznaczany jest jako master dla instancji bazy slave (może to być węzeł danych innego klastra bądź osobna, nieklastrowa instancja bazy danych). Jeśli chodzi o tworzenie kopii zapasowych, to odbywa się ono na każdym węźle danych osobno - proces ten musi być zarządzany przez węzeł MGM.

System NDB Cluster korzysta z synchronicznej replikacji, co oznacza, że każde polecenie modyfikujące stan danych zawartych w systemie wykonywane jest jednocześnie na wszystkich węzłach danych, których dotyczy zmiana. Wiąże się z tym fakt, że każdy z procesów NDBD powinien mieć do dyspozycji podobną ilość zasobów, nie tylko pamięci, ale także mocy obliczeniowej. W wypadku bowiem zaistnienia dużych rozbieżności w ilości dostępnych zasobów, proces lub procesy o gorszym dostępie do nich spowalniać będą działanie całego systemu.

W obecnej wersji MySQL Cluster działać może pod kontrolą systemów operacyjnych Linux, Mac OS X oraz Solaris, przy czym różne węzły jednego systemu pracować mogą pod kontrolą różnych systemów operacyjnych pod warunkiem, że w ten sam sposób reprezentują wielobajtowe typy danych w pamięci (wszystkie są big-endian lub little-endian).

4.3.4 Konfiguracja MySQL Cluster

Konfiguracja MySQL Cluster składa się z trzech elementów:

- dodania wpisu do pliku konfiguracyjnego *.cnf, z którego korzystać będzie proces NDBD podczas uruchamiania:

  ```
  [MYSQL\_CLUSTER]
  # opis lokalizacji węzła zarządczego
  ndb-connectstring=192.168.0.1:7777
  ```

  Możliwe jest podanie adresu bez numeru portu (wtedy brany jest domyślny 1186) oraz podanie kilku adresów oddzielonych średnikami (wtedy następuje próba połączenia się z każdym, w kolejności w jakiej są zapisane, aż do udanego nawiązania połączenia).

- dodania wpisu do pliku konfiguracyjnego *.cnf, z którego korzystać będzie proces serwera bazy danych (korzystającego z NDB Cluster):

  ```
  [MYSQLD]
  # korzystaj z NDB Cluster
  ndbcluster
  # opis lokalizacji węzła zarządczego
  # uwagi odnośnie tego adresu analogiczne jak powyżej
  ndb-connectstring=192.168.0.1:7777
  ```

- stworzenia pliku konfiguracyjnego config.ini, zawierającego komplet informacji o wszystkich węzłach systemu, z którego korzystać będzie proces zarządczy przy starcie i informacje z którego roześle do pozostałych węzłów:

```
# liczba replik
[NDBD DEFAULT]
NoOfReplicas=2
# możliwe też inne ustawienia dotyczące pamięci
DataMemory=80M

[NDB_MGMD]
# nazwa hosta lub adres IP węzła zarządczego
hostname=192.168.0.1:7777
# katalog, gdzie przechowywane będą pliki logów
datadir=/var/lib/mysql-cluster

[NDBD]
# nazwa hosta lub adres IP pierwszego węzła danych
hostname=192.168.0.30
# katalog, w którym przechowywał będzie dane po zamknięciu
#systemu
datadir=/usr/local/mysql/data
[NDBD]
# nazwa hosta lub adres IP pierwszego węzła danych
hostname=192.168.0.40
# katalog, w którym przechowywał będzie dane po zamknięciu
#systemu
datadir=/usr/local/mysql/data

[MYSQLD]
# opcjonalne ustawienia dla węzła SQL
[MYSQLD]
# opcjonalne ustawienia dla drugiego węzła SQL
```

4.3.5 Ograniczenia MySQL Cluster

MySQL Cluster ma pewne ograniczenia w stosunku do bazy działającej z standardowymi systemami zarządzania pamięcią. Najważniejsze z nich to:

- obsługiwany jest jedynie poziom izolacji transakcji „read commited” (w ramach transakcji dwa identyczne zapytania zwrócić mogą inne wyniki,

jeśli w międzyczasie została zatwierdzona inna transakcja modyfikująca odczytywane dane; niedostępny jest poziom „repeatable read" zapewniający spójność odczytu w ramach transakcji)

- odczyt z tabeli posiadającej pola typu BLOB, VARCHAR lub TEXT powoduje założenie blokady odczytu
- maksymalna liczba tabel w klastrze to 1792
- maksymalna liczba atrybutów w tabeli to 128
- maksymalna liczba atrybutów wchodzących w skład klucza to 32
- brak obsługi kluczy obcych
- maksymalna liczba węzłów danych w ramach klastra to 48

4.4 Rozwiązanie wykorzystujące istniejący klaster OpenSSI, bez modyfikacji silnika bazy danych

4.4.1 Zasada działania i sposób konfiguracji

Rozwiązanie to polega na skonfigurowaniu bazy MySQL na kilku węzłach klastra tak, aby na wszystkich z nich przechowywała dane w tym samym, dostępnym ze wszystkich nich, miejscu. Następnie niezbędne jest dodanie wpisu:

```
mysql initnode Y
```

do pliku */etc/rc.nodeinfo*, co zapewni, że przy starcie klastra serwer MySQL uruchomiony zostanie tylko na węźle inicjalizującym. W wypadku jednak jego awarii, usługa ta zostanie uruchomiona na kolejnym węźle, który przejmie na siebie rolę węzła inicjalizującego.

4.4.2 Cechy rozwiązania

Rozwiązanie to jest bardzo proste i nie oferuje w żaden sposób poprawy wydajności a jedynie wzrost niezawodności.

4.5 Podsumowanie

Spośród trzech opisanych rozwiązań, najbardziej godnym uwagi wydaje się być MySQL Cluster - pozwala on nie tylko na prawdziwe rozproszenie danych, ale zapewnia także rozdzielanie zapytań do różnych węzłów i synchroniczną replikację, umożliwiającą uzyskanie sekwencyjnej spójności danych. Jednak rozwiązanie to jest też najmniej sprawdzonym i, niestety, ciągle pełnym ograniczeń (patrz punkt 1.3.5). Kolejne z rozwiązań, replikacja, to często spotykany w bazach danych mechanizm, który dosyć prostymi środkami pozwala na zwiększenie bezpieczeństwa i ew. wydajności (aby uzyskać wzrost wydajności niezbędne są dodatkowe rozwiązania umożliwiające rozkład pracy pomiędzy instancje bazy danych). Ostatni z mechanizmów, wykorzystujący mechanizm OpenSSI do podnoszenia instancji bazy danych po upadku węzła, jako nie oferujący żadnych korzyści wydajnościowych, uznać można za najmniej atrakcyjny.

Część II

Oprogramowanie pomocnicze

Rozdział 5

Subversion

5.1 Opis Subversion

Subversion jest darmowym systemem kontroli wersji danych. Dane przechowywane są w centralnym repozytorium, które zachowuje się bardzo podobnie do zwykłego serwera plików, z ta różnicą że pamięta on każdą zmianę w plikach i katalogach. Pozwala to na odzyskanie starszych wersji danych bądź na analizę historii ich zmian.

5.2 Zalety Subversion

Oprócz standardowych zalet każdego systemu wersjonowania, Subversion zapewnia także:

- wersjonowanie katalogów – w przeciwieństwie do np. CVSa – najpowszechniej stosowanego systemu tego typu – Subversion potrafi śledzić zarówno zmiany nazw plików i katalogów, jak i wszystkie zmiany w strukturze drzewa katalogów.

- niepodzielne commity – Subversion zapewnia ze zestaw zmian zostaje zapisany w repozytorium w całości, bądź w ogóle. Dzięki temu repozytorium nie może znaleźć się w stanie niespójnym.

- wersjonowanie plików binarnych – w przeciwieństwie do CVSa, Subversion radzi sobie z wersjonowaniem zarówno plików tekstowych jak i binarnych.

5.3 Po co stosować Subversion?

Jak każdy system wersjonowania, Subversion zapewnia nam dostęp do każdej wersji danych przechowywanych w repozytorium od czasu jego stworzenia. W przypadku przechowywania kodów źródłowych zabezpiecza nas przede wszystkim przed przypadkowym skasowaniem danych. Ponadto jeżeli zmiany zastosowane przez nas spowodują błędne działanie pisanej przez nas aplikacji bądź uniemożliwią jej uruchomienie, mamy możliwość powrócenia do dowolnej poprzedniej wersji kodu źródłowego.

5.4 Instalacja Subversion

Najprostszym sposobem jest pobranie odpowiedniej dla naszego systemu operacyjnego paczki ze strony Subversion (*http://subversion.tigris.org*) . W przypadku Debiana sprowadza się to do komendy:

```
apt-get install subversion
```

5.5 Użytkowanie Subversion

Od strony klienta użytkowanie ogranicza się do uruchomienia kilku poleceń w miarę potrzeby. Jako ze system działa na zasadzie pobierz-modyfikuj-zapisz, użytkownik musi wiedzieć jak stworzyć lokalna kopie interesującej go części repozytorium, w jaki dozwolony przez Subversion sposób może je modyfikować i jak zapisać modyfikacje w repozytorium.

System składa się z następujących modułów:

svn Jest to najczęściej wykorzystywany przez klienta program, składnia parametrów zostanie opisana dalej.

svnversion Jest to program raportujący stan lokalnej kopii lokalnej kopii repozytorium.

svnlook Narzędzie do inspekcji repozytorium

svnadmin Narzędzie do tworzenia, administrowania i naprawiania repozytorium.

svndumpfilter Narzędzie do filtrowania strumieni zrzutów repozytorium.

mod_dav_svn Wtyczka do serwera HTTP Apacze, służy do udostępnienia repozytorium przez protokół HTTP.

svnserve Program uruchamiany jako demon, inny sposób na udostępnienie repozytorium poprzez siec.

5.6 Opis polecenia svn

Polecenie svn służy do operowania na repozytorium oraz lokalnej kopii danych. W zależności od parametrów, potrafi dokonać następujących rzeczy:

svn help ¡polecenie¿ najważniejsze polecenie, odpowiednik unixowego man'a. Przykład użycia: svn help update

svn checkout Polecenie to pobiera z repozytorium drzewo katalogów bądź pliki i tworząc lokalna kopie, na której użytkownik będzie pracował. Przykład użycia:

```
svn checkout file://sciezka/do/rep/repozytorium/moj/proj/trunk \
moj/proj
svn checkout http://sciezka/do/rep/repozytorium/moj/proj/trunk \
moj/proj
svn checkout svn://ściezka/do/rep/repozytorium/mój/proj/trunk \
moj/proj
```

Jeżeli pominiemy ostatni parametr, drzewo katalogów zostanie zapisane w bieżącym katalogu.

svn commit Polecenie to służy do zapisania dokonanych zmian w repozytorium. Każdy zestaw zmian zostaje zapisany jako kolejna wersja, dlatego jeśli w repozytorium jest wersja nr 14 i użytkownik wywoła to polecenie, zmiany zostaną zapisane jako wersja nr 15. Przykład użycia:

```
svn commit --message "komentarz do zmian"
svn commit --file plik_z_komentarzem
```

Wywołanie tego polecenia bez parametrów spowoduje uruchomienie domyślnego edytora w celu stworzenia komentarza do zastosowanych zmian.

svn update Polecenie to aktualizuje nasza lokalna kopie danych danymi z repozytorium. Zaleca się stosowanie tego polecenia zawsze przed rozpoczęciem pracy z danymi, dzięki czemu często udaje się uniknąć dość zawiłego i często nieudanego scalania dwóch różnych wersji plików. Przykład użycia:

```
svn update
```

svn add plik_lub_katalog_lub_link Polecenie to powiadamia Subversion, iż podczas najbliższego commita do repozytorium ma zostać dodany plik/katalog/link. Przykład użycia:

```
svn add file.c
```

svn delete plik_lub_katalog_lub_link Polecenie to powiadamia Subversion, iż podczas najbliższego commita z repozytorium ma zostać usunięty plik/katalog/link.

svn copy dane_źródłowe dane_docelowe Polecenie to zachowuje się jak unixowy cp, z ta różnicą ze Subversion rejestruje informacje o nowej kopii danych. Przykład użycia:

```
svn copy file.c nowy_katalog/file.c
```

svn mkdir katalog Polecenie tworzące katalog o podanej nazwie i informujące Subversion o zdarzeniu. Przykład użycia:

```
svn mkdir nowy_katalog
```

svn moce dane_źródłowe dane_docelowe Polecenie to zachowuje się jak unixowy mv, z ta różnicą ze Subversion rejestruje informacje o nowej kopii danych. Przykład użycia:

```
svn move file.c inny_katalog/file.c
```

svn status Polecenie to pokazuje zastosowane zmiany od chwili zrobienia ostatniego checkout'a. Uwaga: Porównywane są zmiany w lokalnym repozytorium, polecenie to nie łączy się z repozytorium! Zwraca ono listę zmodyfikowanych plików wraz z ich stanem. Najczęściej występujące stany to:

M file.c – plik ma lokalne modyfikacje

? file.c – svn nie zarządza tym plikiem, stan ten może wystąpić, gdy np. stworzymy plik używając polecenia touch zamiast svn add.

! katalog – svn zarządza katalogiem, ale jest on np. niekompletny, jest to skutek używania standardowych (niezalecanych) unixowych poleceń do zarządzania plikami/katalogami.

L katalog_lub_plik – svn zablokował dostęp do tego pliku/katalogu, np. podczas commita komputer stracił zasilanie i Subversion nie zdjął lock'a z zasobu.

D /katalog/file.c – plik jest przeznaczony do skasowania (przy najbliższym commicie)

A /katalog/foo.c – plik jest przeznaczony do dodania do repozytorium

Przykład użycia:

```
svn status
svn status --verbose
```

Parametr *verbose* spowoduje pokazanie stanu wszystkich plików i katalogów, niezależnie od tego czy zaszły w nich jakieś zmiany czy tez nie.

svn diff Polecenie to pokazuje zmiany pomiędzy dwoma różnymi wersjami danych. Wywołanie go bez parametrów spowoduje wyświetlenie zmian pomiędzy lokalna kopia danych a bieżącą zawartością repozytorium (tu różnica w stosunku do svn status). Polecenie to w przeciwieństwie do svn status pokazuje dokładnie, które linie w plikach uległy zmianie i w jaki sposób. Przykład użycia:

```
svn diff
svn diff > patchfile  #gdzie patchfile to plik, w którym
#zostanie zapisany zestaw różnic pomiędzy dwoma wersjami
svn diff r12:r15  #porównuje dwie podane przez nas wersje
#danych repozytorium
```

svn revert Jedno z ważniejszych poleceń – jeśli zepsujemy cos w kodzie, mamy możliwość powrócenia do wcześniejszej wersji (domyślnie poprzedniej, jeśli nie określimy numeru wersji jawnie) Przykład użycia:

```
svn revert
svn revert catalog
svn revert plik
svn revert r15
svn revert 15-03-2006
```

5.7 Najczęściej występujące sytuacje jakie mogą wystąpić podczas pracy z Subversion

5.7.1 Scenariusz I (główny)

W repozytorium znajduje się kod źródłowy paczki mysql, który chcemy zmodyfikować.

Krok I

Pobieramy interesujący nas katalog:

```
svn checkout file:///var/svn/rso2/repository/dirname
```

Jeśli poprawnie wpisaliśmy ścieżkę, svn pokaże komunikat:

```
Pobrano wersję <nr wersji>.
```

Uwaga: W tym przypadku utworzyliśmy lokalną kopię interesujących nas danych w katalogu bieżącym.

Krok II

Dowolnie modyfikujemy zawartość pobranego katalogu:

- pliki modyfikujemy dowolnym edytorem
- operacje modyfikujące strukturę plików i katalogów wykonujemy przy pomocy opisanych wcześniej poleceń systemu Subversion (svn copy, svn delete).

Uwaga: Jeszcze raz zaznaczam, iż przy modyfikacji struktury plików i katalogów należy pamiętać o stosowaniu poleceń svn ¡operacja¿. Niezastosowanie się do tego zalecenia sprawi iż Subversion nie zarejestruje zmian i np. pliki dodane przez nas do katalogu z lokalną kopią po wykonaniu commita nie znajdą się na serwerze!

Krok III

Po zmodyfikowaniu danych możemy chcieć sprawdzić co tak naprawdę zmieniliśmy w stosunku do poprzednich wersji. Służą do tego polecenia svn status i svn diff. Przed zakończeniem pracy należy zapisać zastosowane przez nas zmiany do repozytorium. Pomocne pod tym względem może być polecenie svn commit. Uwaga: W repozytorium zapisujemy jeśli nie wersje stabilne,

to przynajmniej kompilujące się. Należy pamiętać o dodawaniu sensownych komentarzy do każdej wersji (parametr –*m*). Pomoże to każdemu z użytkowników w przypadku potrzeby pobrania jednej z wcześniejszych wersji kodu.

5.7.2 Scenariusz II

Założenia podobne jak w scenariuszu poprzednim. Pobieramy interesujące nas dane, modyfikujemy je. W tym samym czasie ktoś inny pobrał z repozytorium te same dane i również zaczął je modyfikować. Mamy sytuację, w której w repozytorium znajduje się pierwotna wersja danych, a każdy z użytkowników posiada u siebie inaczej zmodyfikowane dane. Jeżeli zapiszemy zmiany do repozytorium jako pierwsi to mamy szczęście – problem ma użytkownik drugi.

5.7.3 Scenariusz III (konflikt)

Sytuacja taka jak w scenariuszu II: Użytkownik nr 2 zapisał swoje zmiany przed nami. Mamy więc w lokalnej kopii danych naszą wersję, a w repozytorium znajduje się wersja użytkownika nr 2.

Mamy teraz trzy możliwości:

- Wywołujemy polecenie svn update, które spowoduje uaktualnienie naszej lokalnej kopii wersją z repozytorium (czyli tym co właśnie zostało zapisane przez drugiego użytkownika). Subversion zadziała w następujący sposób:
  - Każdy plik zmodyfikowany przez użytkownika 2, a nienaruszony przez nas zostanie – zapisany w naszej lokalnej kopii.
  - Pliki niezmodyfikowane przez użytkownika 2 pozostaną bez zmian.
  - Pliki zmodyfikowane przez nas i przez użytkownika 2 zostaną poddane próbie scalenia: jeśli modyfikowane zostały różne linie, to scalenie kończy się powodzeniem. Jeśli natomiast modyfikacji uległy te same linie kodu, występuje konflikt. Rozwiązanie konfliktu polega na „ręcznym” wybraniu przez nas właściwych linii kodu tak, aby całość miała sens. Po rozwiązaniu konfliktu należy powiadomić Subversion o tym fakcie poleceniem svn resolved.
- Zapisujemy swoje dane w repozytorium. Subversion zachowa się tak jak poprzednio, czyli spróbuje scalić dwie wersje danych.
- cofamy nasze zmiany poprzez svn revert.

5.8 Podsumowanie

System wersjonowania danych Subversion doskonale nadaje się do zarządania kodem modyfikowanym przez wiele osób. Jest wobec tego odpowiednim narzędziem do zastosowania w różnego rodzaju projektach informatycznych. W sytuacji, kiedy wiele osób pracuje nad jednym modułem, spójność modyfikowanych danych jest czynnikiem krytycznym. Zastosowanie Subversion pozwala na uniknięcie sytuacji w której ktoś rozwija nieaktualną wersję modułu. Należy jednak pamiętać o przestrzeganiu zasad korzystania z takiego systemu. Nieodpowiednie używanie go może wprowadzić zamieszanie wśród pozostałych członków zespołu, często powodując znaczne opóźnienia w pracach nad projektem.

Rozdział 6

Paczka MySQL

Do zbudowania pakietu zostały użyte:

- MySQL 5.1 beta (wersja binarna, statyczna)
- makeself (skrypt generujący instalator)
- zestaw własnych skryptów (*.sh)

6.1 Wstęp

Standardowo instalacja MySQL wymaga uprawnień użytkownika uprzywilejowanego, ponieważ zwykli użytkownicy nie maja praw zapisu w katalogach takich jak /bin czy /usr. Dlatego tez zrezygnowaliśmy z pakietów rpm. Wersja skompilowana pozwala na umieszczenie całej bazy w katalogu do którego użytkownik ma pełne prawa. W tej postaci jest to juz wersja możliwa do uruchomienia przez użytkownika. Wiąże się z tym jednak następujący problem, można uruchomić najwyżej jedna instancje bazy na całym klastrze. Dzieje się tak ponieważ MySQLd uruchomione bezparametrowo słucha na standardowym porcie i tworzy socketa w /tmp/MySQL.sock. Należy wiec przygotować skrypty które będą uruchamiały serwis bazy.

6.2 Opis skryptów

6.2.1 setup.sh

Skrypt instalacyjny, jest uruchamiany automatycznie przez instalator. Do jego zadań należy:

- wyświetlenie monitów o podanie katalogu instalacji, portu oraz socketa serwisu ,
- skopiowanie plików z folderu tymczasowego do katalogu wybranego przez użytkownika
- wygenerowanie pliku konfiguracyjnego (rso2MySQL.conf)
- wygenerowanie bazy danych korzystając ze skryptów MySQL
- wystartowanie serwisu korzystając ze skryptu service.sh start
- przyznanie praw do bazy

```
#!/bin/sh
```

pobieranie danych konfiguracyjnych, jeżeli użytkownik wciśnie enter zostaje przypisana domyślna wartość

```
echo Podaj katalog instalacji: '['~/rso2MySQL/']'
read installDir;
if [ ${#installDir} == 0 ]
then
    installDir=~/rso2MySQL/;
fi


echo Podaj port: '['4489']'
read port;
if [ ${#port} == 0 ]
then
    port=4489;
fi

echo Podaj socket: '['/tmp/RSO2MySQL.sock']'
read sock;
if [ ${#sock} == 0 ]
then
    sock=/tmp/RSO2MySQL.sock
fi

ddir=./data/

echo instalowanie programu
```

kopiowanie plików z katalogu tymczasowego do wybranego przez użytkownika

```
mkdir $installDir
cp -r  * $installDir
```

generowanie pliku konfiguracyjnego paczki RSO

```
echo port=$port > $installDir/rso2MySQL.conf
echo datadir=$ddir >> $installDir/rso2MySQL.conf
echo socket=$sock >>  $installDir/rso2MySQL.conf

echo instalowanie baz
```

generacja podstawowych baz MySQL jednym z dostarczonych przez MySQL skryptów

```
cd $installDir
./scripts/MySQL_install_db > /dev/null
```

startowanie zainstalowanej usługi

```
./service.sh start > /dev/null
```

sprawdzenie czy serwer uruchomił się poprawnie jeżeli nie to koniec

```
for sec in 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 20;
do
    if [ -r ./data/MySQL.pid ]; then
break;
    fi
    sleep 1;
    if [ "$sec" = "20" ]; then
echo "Nie udalo sie uruchomic uslugi"
exit 1;
    fi
done
```

przyznanie praw do bazy

```
./bin/MySQL  -u root MySQL --socket=$sock  \
< $installDir/scripts/grantAll.sql #> /dev/null
```

6.2.2 service.sh

Skrypt zarządzający serwisem MySQL. Uruchamiany jest z jednym z 3 parametrów start, stop, restart. W przypadku opcji start czytany jest plik rso2MySQL.conf i na bazie zawartych w nim danych (socket, port, katalog baz) startowany jest serwis. Przy wywołaniu opcji stop wywoływany jest skrypt myMySQLKill.sh.

```
#!/bin/sh

# Start MySQLd:
MySQLd_start() {
```

odczytanie danych konfiguracji bazy

```
   filename=rso2MySQL.conf
    #read conf
```

wyszukiwanie odbywa się na zasadzie wyszukania frazy 'port' w pliku za pomocą grep, następnie jest z niej wyciągana wartość atrybutu poprzez usunięcia 'port='

```
    port=‘cat $filename | grep port‘;
    port=${port/"port="/""};
    socket=‘cat $filename | grep socket‘;
    socket=${socket/"socket="/""};
    ddir=‘cat $filename | grep datadir‘;
    ddir=${ddir/"datadir="/""};
    echo $port $socket $ddir;
```

sprawdzamy czy plik MySQLd_safe istnieje i czy jest wykonywalny

```
    if [ -x ./bin/MySQLd\_safe ]; then
```

jeśli tak uruchamiamy serwer bazy z parametrami pobranymi z pliku konfiguracyjnego

```
./bin/MySQLd_safe --port=$port --socket=$socket \
--datadir=$ddir --pid-file=./MySQL.pid &
    fi
}

# Stop MySQLd:
```

```
MySQLd_stop()
{
  ./myMySQLKill.sh
}

# Restart MySQLd:
MySQLd_restart() {
  MySQLd_stop
  MySQLd_start
}
```

Badanie parametrow wywolania skryptu start/stop/restart

```
case "$1" in
'start')
  MySQLd_start
  ;;
'stop')
  MySQLd_stop
  ;;
'restart')
  MySQLd_restart
  ;;
*)
  echo "$0 start|stop|restart"
esac
```

6.2.3 myMySQLKill.sh

Skrypt służący do selektywnego zamknięcia serwisu. Skrypt odczytuje plik tworzony przez MySQLd przy starcie zawierający pid powołanego procesu, a następnie wywołuje kill dla tego pid-a.

```
#!/bin/sh
```

odczytanie z pliku konfiguracyjnego położenia katalogu baz (znajduje się tam również MySQL.pid)

```
ddir=‘cat rso2MySQL.conf | grep datadir‘
ddir=${ddir/"datadir="/""}
```

sprawdzenie czy taki plik istnieje

```
if [ -r $ddir/MySQL.pid ]
then
```

odczytanie pid-a z pliku

```
    pid=‘cat $ddir/MySQL.pid‘
```

wysalanie SIGTERM-a do MySQLd, proces powinien zakończyć działanie i usunąć plik MySQL.pid

```
    kill $pid
    if [ -r $ddir/MySQL.pid ]
    then
        sleep 4;
```

sprawdzamy czy napewno udało się zakończyć

```
        if [ -r $ddir/MySQL.pid ]
        then
        echo nie udalo sie;
        fi
    fi
    exit 0

else
echo nie mozna znalezc MySQL.pid
fi
```

6.3 Makeself

Jest to skrypt generujący samo-rozpakowujące się archiwum (rso2MySQL.sh). Wygenerowany plik po uruchomieniu wypakowuje swoja zawartość do katalogu tymczasowego a następnie uruchamia skrypt instalacyjny zdefiniowany przez użytkownika (setup.sh). Zasadniczym plusem makeself jest to ze nie operuje na ścieżkach bezwzględnych w przeciwieństwie do większości standardowych pakietów. Strona domowa:www.megastep.org/makeself/

6.4 Scenariusz przygotowania paczki

- pobranie zewnętrznych paczek MySQL oraz makeself
- rozpakowanie do oddzielnych katalogów

- umieszczenie skryptów setup.sh, service.sh, myMySQLKill.sh w katalogu MySQL
- nadanie wykonywalnosci skryptom (chmod +x *.sh)
- by wygenerować instalator należy wejść do katalogu makeself
- wywołać makeself.sh z następującymi parametrami:

  ```
  ./makeself.sh ../MySQLDIR/ rso2MySQL.sh \
  Instalator\_MySQL\_RSO2 ./setup.sh
  ```

  (parametry to kolejno: katalog z plikami, nazwa wynikowego archiwum, tekst wyświetlany przy rozpoczęciu instalacji, polecenie wykonywane po rozpakowaniu plików do folderu tymczasowego)

Rozdział 7

Paczka2: MySQL CLUSTER

7.1 Wstęp:

Paczka instalacyjna zawiera:

- standardowe binaria pakietu mysql pobrane ze strony producenta.
- instalator „makeself”
- zestaw skryptów konfiguracyjnych
- wstępnie wygenerowany przykładowy skrypt uruchomieniowy

7.2 Opis instalacji:

Paczka dystrybuowana jest w postaci pojedynczego samo rozpakowującego się archiwum (rso2mysql.sh). Instalacja pakietu przebiega wieloetapowo. W pierwszym kroku użytkownik instaluje tylko pliki binarne oraz generuje startową bazę danych. Określane są tutaj również parametry takie jak katalog plików binarnych, port nasłuchu serwerów mysql oraz położenie i nazwa plików *.sock. Drugi krok obejmuje konfiguracje struktury klastra. Odpowiedzialny jest za nią skrypt config.pl. Wyświetla on listę działających węzłów klastra openssi wraz z ich numerami ip oraz przypisanymi rolami. Zadaniem użytkownika jest przypisanie zadań dowolnym węzłom. Dostępne role to serwer zarządzający (mgmd), serwer ndb, serwer bazy (mysqld). Po wybraniu odpowiedniej opcji zostaną wygenerowane skrypty uruchomieniowe mysql cluster, oraz nastąpi utworzenie i skopiowanie startowej bazy danych na dyski maszyn hostujących proces mysqld. Trzeci etap to uruchomienie skonfigurowanego klastra. Służy do tego celu wygenerowany skrypt uruchomieniowy. Podnosi on kolejno serwer zarządzający, kolejne węzły NDB oraz

serwery baz danych. Prawidłowo skonfigurowany klaster powinien być gotowy do użytku. By zakończyć pracę klastra należy wywołać skrypt mainstop.sh. Do jego zadań należy jednoczesne zamkniecie procesów NDB oraz MGMD, następnie wyłączanie kolejnych węzłów MYSQLD. Rozwiązanie dostarcza więc dość prosty mechanizm uruchamiania i gaszenia struktury.

7.3 Etapy instalacji – przygotowanie – szczegółowy opis:

7.3.1 Paczka plików binarnych

Do jej przygotowania została użyta statyczna dystrybucja mysql w wersji nie rpm. Pliki programu wraz z przygotowanymi skryptami zostały umieszczone we wspólnym katalogu a następnie opakowane za pomocą programy „makeself" tworząc samo rozpakowujące archiwum. Przy generacji została wybrana opcja uruchamiania własnego skryptu instalacyjnego. „Makeself" rozpakowuje pliki programu do folderu tymczasowego tak więc skrypt instalacyjny ma za zadanie wyświetlić monit o folder docelowy instalacji a następnie skopiować pliki. Ponadto do jego zadań należy generacja pliku standardowej konfiguracji serwerów mysql. Skrypt wyświetla monit o port oraz socket a następnie zapisuje je w katalogu głównym instalacji dając użytkownikowi możliwość późniejszej zmiany ustawień. W ostatnim kroku uruchamiany jest skrypt generujący pustą bazę danych wykorzystywaną w kolejnym etapie, do tego celu użyty został dostarczony przez producenta bazy.

7.3.2 Konfiguracja klastra

Zainstalowany program nie nadaje się jeszcze do użytku gdyż nie posiada plików konfiguracyjnych niezbędnych do uruchomienia klastra. Służy do tego celu skrypt config.pl. Skrypt generuje następujące pliki config.ini, main.sh, mainstop.sh.

config.ini jest plikiem konfiguracyjnym serwera zarządzającego klastrem. Definiuje ilość węzłów klastra.

```
[NDBD DEFAULT]
```

liczba replik

```
NoOfReplicas=2
```

```
[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
[TCP DEFAULT]
```

definicja węzła MGMD

```
[NDB_MGMD]
HostName=10.0.0.3
```

specjalnie dla konfiguracji RSO komunikacja została przeniesiona na port 5560 by nie kolidować z rozwiązaniami innych zespołów

```
portnumber=5560
```

wezły NDB

```
[NDBD]
HostName=10.0.0.1
DataDir/cluster/node1/mnt/disk/rso/rso2/nod1
[NDBD]
HostName=10.0.0.2
DataDir=/cluster/node2/mnt/disk/rso/rso2/nod2
```

2 dowolne węzły mysql

```
[MYSQLD]
[MYSQLD]
```

skrypt main.sh – służy do łatwego powoływania do życia całego klastra. Wykorzystuje polecenie onnode pozwalające odpalić proces na dowolnym node-ie. Tak wię z tym przypadku będziemy mieli mgm node pracujące na węźle 3 openssi, dwa węzły ndb pracujące na 1 i 2 oraz dwa serwery mysql działające również na 1 i 2.

odpalenie mgmd na node 3 z plikiem konfiguracyjnym opisanym powyzej

```
onnode 3 ./bin/ndb_mgmd -f config.ini
```

ta linijka zapewnia nam to że serwer napewno działa (potrzebne tylko na openone)

```
onnode 1 ./bin/ndb_mgm -c 10.0.0.3:5560 -e show
```

podnesienie NDBD

```
onnode 1 ./bin/ndbd --initial -c 10.0.0.3:5560
onnode 2 ./bin/ndbd --initial -c 10.0.0.3:5560
```

rozruch serwerów mysql

```
onnode 1 ./mysqlservice.sh start 10.0.0.3  \
/cluster/node1/mnt/disk/rso/rso2/data1/data/
onnode 2 ./mysqlservice.sh start 10.0.0.3  \
/cluster/node2/mnt/disk/rso/rso2/data2/data/
```

Skrypt mainstop.sh – skrypt kończy pracę klastra. Do zatrzymania mgmd oraz ndb wykorzystana została konsola mgmd. Wykonuje ona polecenie shutdown. Serwery mysql zamykane są odzielnie.

zamkniecie mgmd i ndbd

```
onnode 1 ./bin/ndb_mgm -c 10.0.0.3:5560 -e shutdown
```

zamkniecie serwerów

```
onnode 1 ./mysqlservice.sh stop
onnode 2 ./mysqlservice.sh stop
```

7.3.3 Użytkowanie:

Paczka instalacyjna mysql cluster została zoptymalizowana specjalnie pod zajęcia rso. Została też wzięta pod uwagę specyfika samego openone. Tak więc, komunikacja sieciowa została przeniesiona ze standardowych portów. Serwery mysql słuchają na portach 4489. Została również zmieniona nazwa pliku sock z mysql.sock na RSO2.sock. Serwer zarządzający nasłuchuje na porcie 5560. Pozwala to uniknąć kolizji z innymi zespołami pracującymi na klastrze. Serwery mysql oraz ndbd korzystają ponadto z lokalnych dysków komputerów na których są uruchomione. Zwiększa to wydajność gdyż zmniejszamy czas dostępu do danych (unikamy zbędnej komunikacji przez sieć).

7.3.4 Skrypty konfiguracyjne:

setup.sh – odpalany przy rozpakowywaniu makeself, kopiuje pliki z katalogu tmp do wybranego przez użytkownika, generuje konfigurację dla serwerów mysql.

```
#!/bin/sh
```

pobranie od użytkownika katalogu instalacji.

```
echo Podaj katalog instalacji: '['~/rso2mysql/']'
read installDir;
if [ ${#installDir} == 0 ]
then
    installDir=~/rso2mysql/;
fi
```

portu pracy serwerów mysql

```
echo Podaj port: '['4489']'
read port;
if [ ${#port} == 0 ]
then
    port=4489;
fi
```

nazwy socketa i jego położenia

```
echo Podaj socket: '['/tmp/RSO2mysql.sock']'
read sock;
if [ ${#sock} == 0 ]
then
    sock=/tmp/RSO2mysql.sock
fi
```

kopiowanie plików

```
echo instalowanie programu
mkdir $installDir
cp -r  * $installDir
```

zapis pliku rso2mysql.conf

```
echo port=$port > $installDir/rso2mysql.conf
echo socket=$sock >>  $installDir/rso2mysql.conf
```

wywołanie skryptu instalującego bazy

```
echo instalowanie baz
cd $installDir
./scripts/mysql_install_db > /dev/null
```

ip.pl – jest to prosty skrypt służący do pobrania adresu ip komputera na którym został uruchomiony, jest wykorzystywany w skrypcie config.pl

```
#!/usr/bin/perl
use IO::Socket;
use Sys::Hostname;
$hostname = hostname();
my($addr)=inet_ntoa((gethostbyname($hostname))[4]);
print "$addr";
```

config.pl – jest to główny skrypt konfigurujący klaster. Wyświetla tekstowe menu, daje możliwość przypisania poszczególnym węzłom openssi. Skrypt używa polecenia cluster za pomocą którego pozyskuje informacje o uruchomionych maszynach. Następnie używając komendę onnode odpala skrypt ip.pl na badanym węźle zbierając tym samym numery ip komputerów. Na tej podstawie generuje skrypty uruchomieniowe oraz plik konfiguracyjny dla mgmd. Jego zadaniem jest również utworzenie niezbędnych katalogów i skopiowanie do nich startowych baz.

```
$dataPath="/mnt/disk/rso/rso2";
@wezl;
@ip;
@ndb;
@mysqld;
$mgm="";
```

procedura generująca konfigurację na bazie wyborów użytkownika

```
sub generuj
{
```

plik konfiguracji mgmd

```
open(CONFIGINI,">config.ini");
print CONFIGINI "[NDBD DEFAULT]\nNoOfReplicas=2";
print CONFIGINI "$noofrep\n";
print CONFIGINI "[MYSQLD DEFAULT]\n[NDB_MGMD DEFAULT]\n[TCP   \
DEFAULT]\n[NDB_MGMD]\n";
print CONFIGINI "HostName=$ip[$mgm]\n";
print CONFIGINI "portnumber=5560\n";

for($i=0;$i<=$#ndb;$i++)
{
    if($ndb[$i]==1)
    {
        system("onnode $wezl[$i]  mkdir  $dataPath");
        system("onnode $wezl[$i] mkdir  $dataPath/nod$wezl[$i]");
        print CONFIGINI "[NDBD]\n";
        print CONFIGINI "HostName=$ip[$i]\n";
       print CONFIGINI \
       "DataDir=/cluster/node$wezl[$i]/mnt/disk/rso/rso2/nod$wezl[$i]\n"
    }

}
$serverCount=0;
for($i=0;$i<=$#ndb;$i++)
{
    if($mysqld[$i]==1)
    {
        $serverCount++;
        print CONFIGINI "[MYSQLD]\n";
    }
}
if($serverCount==0)
{
    print CONFIGINI "[MYSQLD]\n";
}
close(CONFIGINI);
```

plik rozruchowy klastra

```
open(START,">main.sh");
```

```
print START "onnode $wezl[$mgm] ./bin/ndb_mgmd -f config.ini\n";
#-----------------------
#kod niezbędny do działania na opnone
if($mgm==0)
{
    print START "onnode $wezl[$#wezl] ./bin/ndb_mgm -c \
    $ip[$mgm]:5560 -e show\n";
}
else
{
    print START "onnode $wezl[0] ./bin/ndb_mgm -c \
    $ip[$mgm]:5560 -e show\n";
}
#-----------------------
for($i=0;$i<=$#ndb;$i++)
{
    if($ndb[$i]==1)
    {
        print START "onnode $wezl[$i] ./bin/ndbd \
        --initial -c $ip[$mgm]:5560\n";
    }
}
for($i=0;$i<=$#ndb;$i++)
{
    if($mysqld[$i]==1)
    {
        system("onnode $wezl[$i]  mkdir  $dataPath");
        system("onnode $wezl[$i] mkdir  $dataPath/data$wezl[$i]");
        system("onnode $wezl[$i] cp -r data  $dataPath/data$wezl[$i]");

       print START "onnode $wezl[$i] ./mysqlservice.sh start $ip[$mgm] \
       /cluster/node$wezl[$i]/mnt/disk/rso/rso2/data$wezl[$i]/data/\n";

    }
}
close(START);
```

skrypt zamykający klaster

```
open(STOP,">mainstop.sh");
```

```
    #print STOP("")
    if($mgm==0)
    {
        print STOP "onnode $wezl[$#wezl] ./bin/ndb_mgm -c \
        $ip[$mgm]:5560 -e shutdown\n";
    }
    else
    {
        print STOP "onnode $wezl[0] ./bin/ndb_mgm -c \
        $ip[$mgm]:5560 -e shutdown\n";
    }
    for($i=0;$i<=$#ndb;$i++)
    {
        if($mysqld[$i]==1)
        {
            #system("onnode $wezl[$i]  mkdir  $dataPath");
            #system("onnode $wezl[$i] mkdir  $dataPath/data$wezl[$i]");
            #system("onnode $wezl[$i] cp -r data  $dataPath/data$wezl[$i]");
            print STOP "onnode $wezl[$i] ./mysqlservice.sh stop $ip[$mgm] \
            $dataPath/data$wezl[$i]/data/\n";
        }
    }
    #print STOP "";
    close(STOP);
}
```

funkcja wypisująca menu oraz dostępne węzły wraz z przypisanymi rolami

```
sub wypisz
{
    print"Dostepne wezly:\n";
    print"MGM Node:$wezl[$mgm]\n";
    print"Node\tIP\t\tNDB\tMYSQLD\n";
    for($i=0;$i<=$#wezl;$i++)
    {

        print("$wezl[$i]\t$ip[$i]\t$ndb[$i]\t$mysqld[$i]\n");
    }
    print "\n\n1.Nowa rola:\n";
```

```
    print "2.Generuj pliki:\n";
    print "3.Koniec\n";
}
```

pobranie dostępnych węzłów

```
system("cluster > temp");

open(TMP,"temp");
$wezly = <TMP>;
@wezl=split(" ",$wezly);
close(TMP);
```

pobranie informacji o numerach ip maszyn

```
for($i=0;$i<=$#wezl;$i++)
{

    system("onnode $wezl[$i] ./ip.pl > temp");
    open(TMP,"temp");
    $ip[$i] = <TMP>;
    $ndb[$i]=0;
    $mysqld[$i]=0;

    close(TMP);
}

$wybor="";
```

pętla obsługująca menu tekstowe

```
while($wybor!=3)
{
    &wypisz(@wezl);
    $wybor=<STDIN>;
    if($wybor==1)
    {
        print "Podaj numer node-a:\n";
```

```
        $numer=<STDIN>;
        print"Podaj role (m-mgm node, n-ndbd, s-mysqld):\n";
        $wybor2=<STDIN>;
        if($wybor2=~/m/)
        {
            for($j=0;$j<=$#wezl;$j++)
            {
                if($wezl[$j]==$numer)
                {
                    $mgm=$j;
                }
            }
        }

        if($wybor2=~/n/)
        {

            for($j=0;$j<=$#wezl;$j++)
            {
                if($wezl[$j]==$numer)
                {
                    $ndb[$j]=1;
                }
            }
        }
        if($wybor2=~/s/)
        {

            for($j=0;$j<=$#wezl;$j++)
            {
                if($wezl[$j]==$numer)
                {

                    $mysqld[$j]=1;

                }
            }

        }
    }
    if($wybor==2)
```

```
    {
        &generuj();
    }
```

mysqlservice.sh – do zadań tego skryptu należy uruchamianie i zatrzymywanie serwera mysql. Odczytuje on konfigurację z pliku rso2mysql.conf utworzonego podczas instalacji. Przyjmuje następujące parametry : start—stop adres_serwera_mgmd ścieżka_katalogu_baz

```
#!/bin/sh

dataPath="";
mgmHost="";

# Start mysqld:
```

uruchomienie mysqld

```
mysqld_start() {
    filename=rso2mysql.conf
```

odczyt konfiguracji

```
    port=‘cat $filename | grep port‘;
    port=${port/"port="/""};
    socket=‘cat $filename | grep socket‘;
    socket=${socket/"socket="/""};
```

powołuje do życia serwer mysql

```
    if [ -x ./bin/mysqld_safe ]; then
        ./bin/mysqld_safe --port=$port --socket=$socket  \
        --datadir=$dataPath --pid-file=./mysql.pid --ndbcluster \
        --ndb-connectstring=$mgmHost:5560 &
    fi

}
```

zamyka mysql wysyłajac do niego SIG_TERM

```
# Stop mysqld:
mysqld_stop()
{

    kill `cat $dataPath/mysql.pid`;
}


mgmHost=$2;
dataPath=$3;
case "$1" in
'start')
  mysqld_start
  ;;
'stop')
  mysqld_stop
  ;;
*)
  echo "$0 start|stop"
esac
```

Rozdział 8

Paczka3: MySQL CLUSTER

8.1 Wstęp:

Paczka instalacyjna trzeciego etapu nie przeszła większych zmian w stosunku do etapu 2. Związane jest to z faktem oparcia rozwiązania o rozwiązania zewnętrzne nie zmieniające i nie modyfikujące działania MySQL.

8.1.1 Instalacja

Instalacja Pakietu składa się z 3 etapów.

- instalacji plików wykonywalnych i generacja bazy wzorcowej
- konfiguracji klastra mysql i generacji plików konfiguracyjnych
- uruchomienia klastra

instrukcja instalacji:

- uruchomić mysqlpaczka3.sh
- podać port oraz socket lub nacisnąć $< enter >$ dla wartości domyślnych (port 4456 , socket RSO2.sock)

konfiguracja:

- wejść do katalogu $< katalog\_instalacji/ >$
- uruchomić ./config.pl
- korzystając z menu tekstowego przypisać role node-om. (opcja 1)

- po przypisaniu ról wybrać opcję 2 która to wygeneruje konfigurację oraz skrypty uruchamiające
- zakończyć działanie (3)

uruchamianie:

- uruchomić ./main.sh - skrypt uruchomi klaster
- ny zakończyć działanie należy uruchomić skrypt ./mainstop.sh

8.2 Przygotowanie MySQL do współpracy z CVIP

Ponieważ wpółpraca z cvip wymaga połączeń klienta z serwerem MySQL poprzez sieć baza musi udostępniać taką możliwość. Domyślnie tabele uprawnień wygenerownej bazy blokują taką możliwość. Dostęp do bazy danych odbywać może się jedynie poprzez socket (/tmp/RSO2.sock). Tak więc proces instalacji należy rozbudować o wygenerowanie baz ze zmienionymi uprawnieniami. Problem ten ostał rozwiązany w następujący sposób:

- Skrypt kopiuje pliki instalacji binarnej
- Generuje wzorcową bazę danych bez uprawnień
- Uruchamia lokalnie serwer
- Przy pomocy klienta mysql, odpala na wygenerowanej bazie pomocniczy skrypt przyznający uprawnienia wszystkim do wszystkiego.
- Zamyka serwer

8.2.1 Zmieniony skrypt instalujący:

setup.sh - odpalany przy rozpakowywaniu makeself, kopiuje pliki z katalogu tmp do wybranego przez użytkownika, generuje konfigurację dla serwerów mysql.

```
#!/bin/sh
```

pobranie od użytkownika katalogu instalacji.

```
echo Podaj katalog instalacji: '['~/rso2mysql/']'
read installDir;
if [ ${#installDir} == 0 ]
then
    installDir=~/rso2mysql/;
fi
```

portu pracy serwerów mysql

```
echo Podaj port: '['4456']'
read port;
if [ ${#port} == 0 ]
then
    port=4489;
fi
```

nazwy socketa i jego położenia

```
echo Podaj socket: '['/tmp/RSO2mysql.sock']'
read sock;
if [ ${#sock} == 0 ]
then
    sock=/tmp/RSO2mysql.sock
fi
```

kopiowanie plików

```
echo instalowanie programu
mkdir $installDir
cp -r  * $installDir
```

zapis pliku rso2mysql.conf

```
echo port=$port > $installDir/rso2mysql.conf
echo socket=$sock >>  $installDir/rso2mysql.conf
```

wywołanie skryptu instalującego bazy

```
echo instalowanie baz
cd $installDir
./scripts/mysql_install_db > /dev/null
```

Uruchomienie serwera w celu przyznania uprawnień

```
./bin/mysqld_safe --port=$port --socket=$sock \
--datadir=./data --pid-file=./mysql.pid
for sec in 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 20;
do
    if [ -r ./data/mysql.pid ]; then
            break;
    fi
    sleep 1;
    if [ "$sec" = "20" ]; then
            echo "Nie udalo sie uruchomic uslugi"
            exit 1;
    fi
done
```

Uruchomienie skryptu grantującego

```
./bin/mysql   -u root mysql --socket=$sock  < $installDir/scripts/grantAll.sql \
#> /dev/null

kill ‘cat ./data/mysql.pid‘
```

rebootnode.sh - ponieważ czasami (często) na openone dochodzi do zawieszenia się serwerów mysql konieczne było napisanie skryptu restarującego node-a. Ponieważ skrypt mysqlservice.sh startuje proces mysqld-max przy użyciu skryptu mysqld_safe można wykorzystać ten fakt do prostego restatu serwera. Zadaniem mysqld_safe jest uruchomienie i utrzymaniestanu mysqld-max. Jeśli proces serwera otrzyma sygnał TERM , zamknie się prawidłowo. Natomiast gdy otrzyma SIGKILL system operacyjny zamknie w tym przypadku zawieszony proces. Teraz z kolei zadziała zabezpieczenie i proces serwera zostanie podniesiony na nowo z tą samą konfiguracją co poprzednio.

```
#!/bin/sh

kill -9  ‘ onnode   $1 cat /mnt/disk/rso/rso2/data$1/data/mysql.pid‘
```

8.3 Pozostałe skrypty- listingi

Poniższe skrypty (konfiguracyjny i zarządzający serwerem mysql) nie uległy zmianie od etapu drugiego.

8.3.1 config.pl

- jest to główny skrypt konfigurujący klaster. Wyświetla tekstowe menu, daje możliwość przypisania poszczególnym węzłom openssi. Skrypt używa polecenia cluster za pomocą którego pozyskuje informacje o uruchomionych maszynach. Następnie używając komendę onnode odpala skrypt ip.pl na badanym węźle zbierając tym samym numery ip komputerów. Na tej podstawie generuje skrypty uruchomieniowe oraz plik konfiguracyjny dla mgmd. Jego zadaniem jest również utworzenie niezbędnych katalogów i skopiowanie do nich startowych baz.

```
$dataPath="/mnt/disk/rso/rso2";
@wezl;
@ip;
@ndb;
@mysqld;
$mgm="";
```

procedura generująca konfigurację na bazie wyborów użytkownika

```
sub generuj
{
```

plik konfiguracji mgmd

```
 open(CONFIGINI,">config.ini");
 print CONFIGINI "[NDBD DEFAULT]\nNoOfReplicas=2";
 print CONFIGINI "$noofrep\n";
 print CONFIGINI "[MYSQLD DEFAULT]\n[NDB_MGMD DEFAULT]\n[TCP    \
 DEFAULT]\n[NDB_MGMD]\n";
 print CONFIGINI "HostName=$ip[$mgm]\n";
 print CONFIGINI "portnumber=5560\n";

 for($i=0;$i<=$#ndb;$i++)
```

```
    {
        if($ndb[$i]==1)
        {
            system("onnode $wezl[$i]  mkdir  $dataPath");
            system("onnode $wezl[$i] mkdir  $dataPath/nod$wezl[$i]");
            print CONFIGINI "[NDBD]\n";
            print CONFIGINI "HostName=$ip[$i]\n";
            print CONFIGINI \
    "DataDir=/cluster/node$wezl[$i]/mnt/disk/rso/rso2/nod$wezl[$i]\n"
        }

    }
    $serverCount=0;
    for($i=0;$i<=$#ndb;$i++)
    {
        if($mysqld[$i]==1)
        {
            $serverCount++;
            print CONFIGINI "[MYSQLD]\n";
        }
    }
    if($serverCount==0)
    {
        print CONFIGINI "[MYSQLD]\n";
    }
    close(CONFIGINI);
```

plik rozruchowy klastra

```
    open(START,">main.sh");
    print START "onnode $wezl[$mgm] ./bin/ndb_mgmd -f config.ini\n";
    #-----------------------
    #kod niezbędny do działania na opnone
    if($mgm==0)
    {
        print START "onnode $wezl[$#wezl] ./bin/ndb_mgm \
-c $ip[$mgm]:5560 -e show\n";
    }
    else
    {
```

```
        print START "onnode $wezl[0] ./bin/ndb_mgm \
-c $ip[$mgm]:5560 -e show\n";
    }
    #-----------------------
    for($i=0;$i<=$#ndb;$i++)
    {
        if($ndb[$i]==1)
        {
            print START "onnode $wezl[$i] ./bin/ndbd \
    --initial -c $ip[$mgm]:5560\n";
        }
    }
    for($i=0;$i<=$#ndb;$i++)
    {
        if($mysqld[$i]==1)
        {
            system("onnode $wezl[$i]  mkdir  $dataPath");
            system("onnode $wezl[$i] mkdir  $dataPath/data$wezl[$i]");
            system("onnode $wezl[$i] cp -r data  $dataPath/data$wezl[$i]");

           print START "onnode $wezl[$i] ./mysqlservice.sh start $ip[$mgm] \
   /cluster/node$wezl[$i]/mnt/disk/rso/rso2/data$wezl[$i]/data/\n";

        }
    }
    close(START);
```

skrypt zamykający klaster

```
   open(STOP,">mainstop.sh");
    #print STOP("")
    if($mgm==0)
    {
        print STOP "onnode $wezl[$#wezl] ./bin/ndb_mgm \
-c $ip[$mgm]:5560 -e shutdown\n";
    }
    else
    {
        print STOP "onnode $wezl[0] ./bin/ndb_mgm \
-c $ip[$mgm]:5560 -e shutdown\n";
```

```
    }
    for($i=0;$i<=$#ndb;$i++)
    {
        if($mysqld[$i]==1)
        {
            #system("onnode $wezl[$i]  mkdir  $dataPath");
            #system("onnode $wezl[$i] mkdir  $dataPath/data$wezl[$i]");
            #system("onnode $wezl[$i] cp -r data  $dataPath/data$wezl[$i]");
            print STOP "onnode $wezl[$i] ./mysqlservice.sh stop \
    $ip[$mgm] $dataPath/data$wezl[$i]/data/\n";
        }
    }
    #print STOP "";
    close(STOP);
}
```

funkcja wypisująca menu oraz dostępne węzły wraz z przypisanymi rolami

```
sub wypisz
{
    print"Dostepne wezly:\n";
    print"MGM Node:$wezl[$mgm]\n";
    print"Node\tIP\t\tNDB\tMYSQLD\n";
    for($i=0;$i<=$#wezl;$i++)
    {

        print("$wezl[$i]\t$ip[$i]\t$ndb[$i]\t$mysqld[$i]\n");
    }
    print "\n\n1.Nowa rola:\n";
    print "2.Generuj pliki:\n";
    print "3.Koniec\n";
}
```

pobranie dostępnych węzłów

```
system("cluster > temp");
```

```
open(TMP,"temp");
$wezly = <TMP>;
@wezl=split(" ",$wezly);
close(TMP);
```

pobranie informacji o numerach ip maszyn

```
for($i=0;$i<=$#wezl;$i++)
{

    system("onnode $wezl[$i] ./ip.pl > temp");
    open(TMP,"temp");
    $ip[$i] = <TMP>;
    $ndb[$i]=0;
    $mysqld[$i]=0;

    close(TMP);
}

$wybor="";
```

pętla obsługująca menu tekstowe

```
while($wybor!=3)
{
    &wypisz(@wezl);
    $wybor=<STDIN>;
    if($wybor==1)
    {
        print "Podaj numer node-a:\n";
        $numer=<STDIN>;
        print"Podaj role (m-mgm node, n-ndbd, s-mysqld):\n";
        $wybor2=<STDIN>;
        if($wybor2=~/m/)
        {
            for($j=0;$j<=$#wezl;$j++)
            {
                if($wezl[$j]==$numer)
                {
```

```
                $mgm=$j;
            }
        }
    }

    if($wybor2=~/n/)
    {

        for($j=0;$j<=$#wezl;$j++)
        {
            if($wezl[$j]==$numer)
            {
                $ndb[$j]=1;
            }
        }
    }
    if($wybor2=~/s/)
    {

        for($j=0;$j<=$#wezl;$j++)
        {
            if($wezl[$j]==$numer)
            {

                $mysqld[$j]=1;

            }
        }

    }
}
if($wybor==2)
{
    &generuj();
}
```

8.3.2 mysqlservice.sh

- do zadań tego skryptu należy uruchamianie i zatrzymywanie serwera mysql. Odczytuje on konfigurację z pliku rso2mysql.conf utworzonego podczas instalacji. Przyjmuje następujące parametry : start—stop adres_serwera_mgmd

ścieżka_katalogu_baz

```
#!/bin/sh

dataPath="";
mgmHost="";

# Start mysqld:
```

uruchomienie mysqld

```
mysqld_start() {
    filename=rso2mysql.conf
```

odczyt konfiguracji

```
    port=‘cat $filename | grep port‘;
    port=${port/"port="/""};
    socket=‘cat $filename | grep socket‘;
    socket=${socket/"socket="/""};
```

powołuje do życia serwer mysql

```
    if [ -x ./bin/mysqld_safe ]; then
        ./bin/mysqld_safe --port=$port --socket=$socket  --datadir=$dataPath \
--pid-file=./mysql.pid --ndbcluster --ndb-connectstring=$mgmHost:5560 &
    fi

}
```

zamyka mysql wysyłajac do niego SIG_TERM

```
# Stop mysqld:
mysqld_stop()
{
```

```
    kill ‘cat $dataPath/mysql.pid‘;
}


mgmHost=$2;
dataPath=$3;
case "$1" in
’start’)
  mysqld_start
  ;;
’stop’)
  mysqld_stop
  ;;
*)
  echo "$0 start|stop"
esac
```

Rozdział 9

Testy funkcjonalności MySQL Cluster.

9.1 Cel testów:

Celem przeprowadzanych testów jest sprawdzenie, czy MySQL Cluster na OpenSSI zachowuje się tak, jak jest to od niego oczekiwane. Analiza funkcjonalności jest niezbędna do kontynuacji projektu, gdyż wyniki testów będą miały bezpośredni wpływ na architekturę prototypu rozwiązania.

9.2 Środowisko testowe:

MySQL Cluster będzie testowany na węzłach klastra OpenSSI. Dostępnych jest piętnaście węzłów, z czego pierwsze trzy są dostępne przez cały czas działania systemu. Utrudnieniem jest konfiguracja tych węzłów – każdy jest skonfigurowany inaczej, przez co mogą wystąpić problemy z komunikacją pomiędzy poszczególnymi modułami. Do testów została użyta wygenerowana wcześniej paczka MySQL Cluster.

9.3 Przygotowanie środowiska:

Na węźle nr 3 został uruchomiony serwer ndb_mgmd nasłuchujący na porcie 5560. Na węzłach nr 1, 2 oraz 4 zostały uruchomione serwery mySQL oraz baza danych ndb.

9.4 Przeprowadzone testy:

9.4.1 Test replikacji:

Na węźle nr 2 została uruchomiona konsola mysql, po czym stworzona została przykładowa tabela i wstawiony jedna krotka:

- use testowa;
- CREATE TABLE testtable (i INT) ENGINE=NDBCLUSTER;
- INSERT INTO testtable () VALUES (1);

Wynikiem SELECTA

- SELECT * FROM testtable;

była wstawiona przed chwilą krotka. Ten sam SELECT wykonany na węzłach nr 1 oraz 4 zwrócił identyczny wynik.

9.4.2 Test spójności danych.

Na węźle nr 2 została wstawiona kolejna krotka. Następnie ndb na węzłach nr 2 i 4 zostały wyłączone (killall -9), po czym uruchomione ponownie. SELECT pokazał, iż wstawiona wcześniej krotka nadal znajduje się w bazie danych.

9.4.3 Test stabilności podstawowej architektury.

Poprzedni test wykazał, iż wyłączenie jednego bądź większej ilości ndb nie ma wpływu na zawartość bazy danych, dopóki istnieje co najmniej jedna dodatkowa baza ndb. Poniższy test pokaże co się stanie, jeśli wyłączony zostanie demon mgmd. W pierwszym kroku wyłączony został demon mgmd. Następnie na jednym z węzłów została dodana do tabeli testtable kolejna krotka. Ndb na węźle 4 został wyłączony, a następnie włączony ponownie. Próba odczytania danych z tego węzła skutkowała błędem „Cluster failure”, gdyż włączony ndb nie został zarejestrowany przez demona zarządzającego. Jednakże po ponownym uruchomieniu demona mgmd system powrócił do stanu spójnego.

9.5 Podsumowanie:

MySQL Cluster zachowuje się zgodnie z oczekiwaniami, replikując dane pomiędzy poszczególnymi węzłami. Pierwszy test pokazał, iż dane replikowane są na wszystkich węzłach ndb, co pozwala wnioskować, iż nie utracimy żadnych danych, dopóki będzie działać choć jeden ndb. Przy działającym demonie mgmd wyłączenie i włączenie ndb na danym węźle powoduje przywrócenie bazy danych do stanu pierwotnego. Test nr 3 pokazał jednak, iż wymagany jest działający co najmniej jeden demon zarządzający, aby zarejestrować dołączane repliki, zapewniając spójność danych.

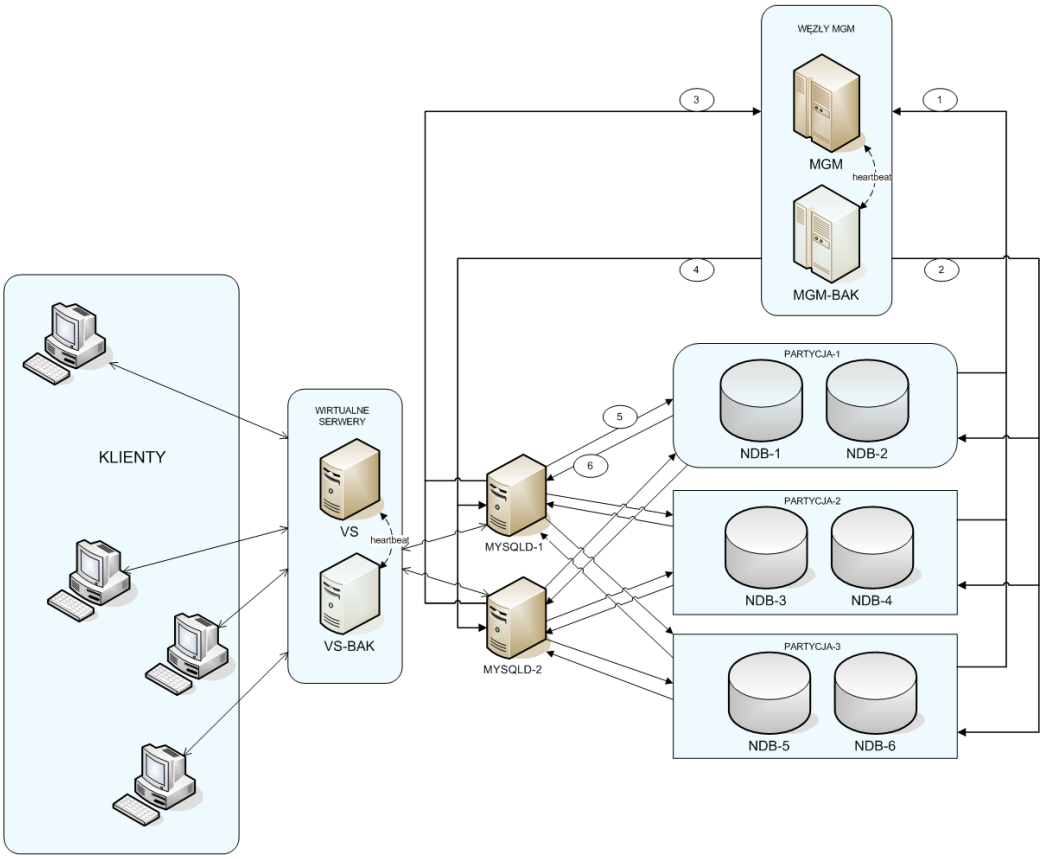
Część III

Projekt rozwiązania

Rozdział 10

Prototyp rozwiązania

Projekt rozwiązania został oparty o klastrowe rozwiązanie MySQL. Schemat rozwiązania przedstawiony jest na rysunku 10.1. Na rysunku 10.1 przed-



Rysunek 10.1: Schemat rozwiązania

stawione zostały kanały komunikacyjne w ramach rozwiązania MySQL Cluster. Numery oznaczają kolejność, w jakiej muszą zostać wymienione komunikaty. Jako pierwsze węzły NDBD pobrać muszą konfigurację klastra z serwera MGM. Następnie to samo wykonują węzły MYSQLD wykorzystujące klaster jako „storage engine". Kolejnym krokiem jest już wykonywanie przez węzły MYSQLD operacji odczytu i zapisu na danych składowanych przez węzły NDBD.

10.1 Elementy standardowe

Jak wynika z rysunku w projekcie wykorzystano podstawowe elementy klastra NDB[1], takie jak:

- węzły NDBD – odpowiedzialne za przechowywanie i udostępnianie danych węzłom SQL
- węzły MGM – odpowiedzialne za zarządzanie pozostałymi węzłami w ramach MySQL Cluster
- węzły SQL – zwykłe serwery MySQL, skonfigurowane do korzystania z usług NDB Cluster.

10.2 Elementy klastrowe OpenSSI

Rozwiązanie zostało rozszerzone o elementy dostępne w OpenSSI, takie jak:

- wirtulany serwer
- pakiet Mon
- komunikacji heartbeat.

10.2.1 Wirtualny serwer

Zbudowany przy pomocy CVIP[2] Umożliwia zamaskowanie awarii MYSQLD poprzez rozprowadzanie połączeń do pozostałych, działających serwerów MYSQLD.

[1]Zostały one dokładniej omówione w rozdziale 4.3.2, na stronie 45.
[2]Omówionego w rozdziale 3.1.3, na stronie 28.

10.2.2 Pakiet Mon

Pakiet Mon jest demonem monitorującym pracę demonów, w tym między innymi serwer MySQL. Pakiem Mon co jakiś czas łączy się z poszczególnymi demonami MYSQLD i w przypadku wykrycia nieprawidłowości w działaniu bazy uruchamia skrypt (program), którego celem jest naprawienie usterki.

10.2.3 Komunikacja heartbeat

W prezentowanym rozwiązaniu występują 2 potencjalne punkty SPOF[3]. Są nimi serwery krytyczne pod względem działania projektu, takie jak: wirtualny serwer oraz węzeł MGM. W celu uniknięcia pojedynczego punktu awarii wirtualny serwer oraz węzeł MGM wyposażony został w uśpioną kopię zapasową, komunikującą się z działającym serwerem poprzez połączenie heartbeat i w razie potrzeby przejmującą na siebie obowiązki podstawowego serwera[4].

10.3 Zyski z połączenia

Tworząc opisaną powyżej architekturę zyskujemy:

- zwiększoną dostępność rozwiązania,
- zwiększoną szybkość odczytów z bazy danych,
- zwiększoną odporność na awarie jednego lub kilku węzłów przechowujących dane (NDBD).

10.4 Planowane rozszerzenia

- Stworzenie programu, który w razie awarii będzie diagnozował pracę klastra i podejmował próbę jego rekonstrukcji (podnoszenie/zabijanie węzłów itp.).
- Optymalizacja rozsyłania zapytań na podstawie aktualnego obciążenia danych węzłów.

[3] Omówione w rozdziale 2.1.2, na stronie 21.

[4] W związku ze specyfiką laboratorium, w którym realizowany jest projekt oraz z faktem, że nie posiadamy odpowiednich uprawnień rozwiązanie to pozostanie jedynie w fazie projektu.

10.4.1 Optymalizacja rozsyłania zapytań

Możliwość poprawy obecnie wykorzystywanego mechanizmu w celu zwiększenia wydajności występuje w kroku 4[5]. Otóż zarówno przy odczycie jak i przy zapisie, węzeł MYSQLD wybiera węzeł NDBD, któremu powierzy wykonanie polecenia, wykorzystując mechanizm „round robin". Takie rozwiązanie nie uwzględnia możliwych różnic w zasobach dostępnych dla poszczególnych węzłów NDBD oraz zróżnicowania poziomu skomplikowania zapytań. Mechanizm wyboru węzła, któremu zostanie powierzone wykonanie polecenia, zaimplementowany jest we wtyczce NDB wchodzącej w skład mysqld (implementującej interfejs storage engine'u). W ramach opisywanego rozwiązania zostanie zidentyfikowany fragment kodu wtyczki odpowiedzialny za wybór węzła NDB. Następnie mechanizm wyboru węzła zostanie zmodyfikowany tak, aby uwzględniać obciążenie maszyn, na których uruchomione są poszczególne węzły NDBD. Zostanie to zaimplementowane poprzez sprawdzanie co zadaną liczbę rozesłanych poleceń obciążenia maszyn, na których uruchomione są węzły NDBD, i budowania na tej podstawie wag dla każdego z węzłów NDBD. Na podstawie wag połączenia do niektórych węzłów będą trafiać rzadziej, do niektórych częściej. Sprawdzanie obciążenia węzłów zaimplementowane zostanie przy wykorzystaniu mechanizmów dostarczanych przez OpenSSI.

10.5 Zachowanie w sytuacjach awaryjnych

- Awaria pojedynczego serwera MYSQLD – poza ew. zmniejszoną wydajnością nie ma wielkiego znaczenia na pracę rozwiązania, pod warunkiem, że pozostał co najmniej 1, działający serwer MYSQLD.
- Awaria MGM – podnoszona jest uśpiona kopia zapasowa.
- Awaria pojedynczego NDBD – podobnie jak w przypadku MYSQLD, awaria spowoduje obniżoną wydajność, ale dopóki pozostały jakieś działające węzły NDBD, dane mogą być nadal dostarczane do MYSQLD.
- Awaria wirtualnego serwera – podnoszona jest uśpiona kopia zapasowa.
- Podział sieci – w przypadku wykrycia podziału sieci uruchomiony zostaje program (poprzez demon Mon), którego celem jest próba odczytania konfiguracji z węzła MGM oraz odczyt informacji o konfiguracji

[5]Opis dotyczy przedstawionego rysunku 10.1, przedstawionego na stronie 97.

startowej na podstawie uzyskanych informacji program podejmie decyzję czy próbować reaktywować martwe procesy, czy też zakończyć działanie całego klastra – według zasady, że do dalszej pracy potrzebne jest działanie połowy + 1 procesów.

Rozdział 11

Gotowe elementy

11.1 Konfiguracja CVIP

W rozwiązaniu postanowiliśmy „podpiąć się" się pod standardową konfigurację HA-LVS na klastrze OpenOne (OpenSSI) – */etc/cvip.conf*:

```
<?xml version="1.0"?>
<cvips>
<routing>NAT</routing>
        <cvip>
                <ip_addr>194.29.167.56</ip_addr>
                <gateway>10.0.0.1</gateway>
                <director_node>
                        <node_num>1</node_num>
                        <garp_interface>eth1</garp_interface>
                        <sync_interface>eth0</sync_interface>
                </director_node>
                <real_server_node>
                        <node_num>1</node_num>
                </real_server_node>

                ...

                <real_server_node>
                        <node_num>15</node_num>
                </real_server_node>

        </cvip>
</cvips>
```

W celu „rozrzucenia" połączeń przychodzących na porcie, na którym nasłuchuje MySQL wywołany zostanie następujące polecenie (umieszczone w skrypcie):

```
setport_weight --start-port=4456 --end-port=4456 --weight=1
```

11.2 Konfiguracja Mon

Konfiguracja demonów Mon nastąpi według następującego schematu – */etc/mon/mon.cf*::

```
hostgroup mysql-node 127.0.0.1

#
# watch definitions
watch mysql-node
    service mysql
        interval 10s
        monitor msql-mysql.monitor --mode mysql --database=your \
        database --password=yourpasswordgoeshere
        period wd {Mon-Sun}
            alert stop-heartbeat.alert
```

Rozdział 12

Planowane testy wydajności i narzędzia do tego wykorzystywane.

12.1 Wybrane narzędzie.

Do testowania budowanego w ramach projektu rozwiązania planujemy użyć pakietu OSDB (*Open Source Database Benchmark*). OSDB jest narzędziem dedykowanym dla developerów systemów i baz danych udostępniającym szeroki wachlarz testów (w postaci kodu źródłowego), który może być dostosowany indywidualnie, w zależności od lokalnych potrzeb. OSDB wywodzi się z AS3AP (*ANSI SQL Standard and Portale Benchmark*), najważniejszą zmianą jaka w nim zaszła jest podzielenie benchmark'ów, co pozwala twórcom danego rozwiązania skupić się na rezultatach najbardziej oddających istotę ich pracy.

12.2 Opis narzędzia.

Aby korzystać z OSDB musimy posiadać co najmniej dwa pliki:

osdb–vx_y_z.tgz – (60kB) – główna część z kodami źródłowymi OSDB

oraz: osdb-data-4mb.tgz – (2.7MB) – dane do testowania i tworzenia oprogramowania (mogą być użyte do porównania pewnych rezultatów, ale bazy danych zazwyczaj nie są rozmiarów 4.8MB, więc twórcy OSDB doradzają używać bazy danych o rozmiarze 48MB do testowania wydajności)

albo: osdb–data–40mb.tgz – (27MB) – wspomniana wcześniej paczka danych do testowania wydajności

OSDB składa się z trzech części. Użytkownik może ich używać zgodnie z własnymi potrzebami, lub też uruchomić jako całość. Jednak przy pierwszym uruchomieniu musimy zezwolić OSDB na wykonanie części odpowiedzialnej za stworzenie struktury bazy danych. Istnieje też opcja uruchomienia OSDB bez tworzenia indeksów, kluczy głównych i kluczy obcych.

12.2.1 DB structure creation.

Część ta jest odpowiedzialna za stworzenie struktury bazy danych. Tworzenie jej zawiera:

- tworzenie tabeli – tworzenie tabeli w bazie danych o nazwie OSDB, która stanowi podstawę do późniejszych testów wydajnościowych
- generacja danych – kopiowanie danych z przygotowanych plików do wszystkich tabel wcześniej utworzonej bazy
- tworzenie kluchy głównych i indeksów – stworzenie kluczy głównych i indeksów oraz sprawdzenie danych w poszczególnych kolumnach. Są dwa typy indeksów
  - B–tree – reprezentowane symbolem "_bt"
  - hash – reprezentowane symbolem "_h"
- formowanie kluczy obcych – tworzenie kluczy obcych, przy czym mogą się pojawić komunikaty typu: *"NOTICE: ALTER TABLE ... ADD CONSTRAINT will create implicit trigger(s) for FOREIGN KEY check(s)"*, bądz podobne;

Rezultaty tej części testu są przedstawione za pomocą: populateDataBase() linii na sekundę[s].

12.2.2 Single - user test.

Ta część testów zawiera proste operacje na danych, takie jak:

- *select* – zwyczajne zapytanie o rekord (krotkę) w tabeli z podaniem warunku
- *join* – zapytanie o rekord(y) z tabel (dwóch lub trzech) złączonych razem pod pewnym warunkiem
- *aggregation* – liczenie średnich, maksimów, minimów, oraz sum

- *view* – tworzenie widoków (z agregatami)
- *projection* – zapytanie za pomocą SQL'a o różne opcje jednej lub wielu tabel
- *add* – dodanie rekordu do tabeli
- *update* – uaktualnienie jednej lub większej liczby wartości w tabeli
- *delete* – usuniecie rekordu z tabeli z podaniem warunku
- *bulk operation* – usuniecie/dodanie wielu rekordów z/do tabeli
- *index update* – uaktualnienie indeksów po zmianie danych
- *integrity test* – sprawdzenie spójności danych w tabelach z kluczami obcymi

Rezultat tej części testu jest przedstawiony jako: Single User Test linii na sekundę [s].

12.2.3 Multi-user test.

Test wielu użytkowników jest dedykowany do testowania transakcji i zachowania DBMS'a w sytuacji kiedy wielu użytkowników próbuje jednocześnie uzyskać dostęp do danych. Składa się z dwóch głównych części:

- *Mixed IR* – zapytywanie o pola w tabeli z zadanym warunkiem, który jest losowo generowany z wszystkich możliwych wartości zawartych w tabeli; wynik jest podawany w jednostce [tup/sec]
- *Mixed OLTP (On-Line Transaction Processing)* – uaktualnianie pól z warunkiem losowo wybranym w tabeli; wynik jest podany w [tup/sec]

12.3 Podsumowanie.

Przeprowadzone testy mają nam pomóc w odpowiedzi na pytania typu:

- Jak dobrze system został zbudowany?
- Jakie jest dopuszczalne obciążenie systemu?
- Jakie aspekty są ważne dla naszej bazy danych i użytkowników końcowych?

OSDB wydaje się być narzędziem elastycznym i umożliwiającym różnorakie modyfikacje. Z pewnością ta cecha pozwoli na dopasowanie testów do systemu klastrowego i specyfiki tego środowiska. Będzie też pomocne w ocenie jakości rozwiązania zaimplementowanych przez nas pomysłów.

Rozdział 13

Testy wydajności.

13.1 Wprowadzenie

Celem przeprowadzonych testów było porównanie wydajności rozproszonej bazy danych pracującej na klastrze w odniesieniu do rozwiązania umieszczonego na jednej maszynie. Porównywane było rozwiązanie MySQL przygotowane w ramach pierwszego etapu projektu. Jest to wersja skompilowana, pozwalająca na umieszczenie całej bazy danych w katalogu do której użytkownik ma pełne prawa. Następnie przystąpiliśmy do testowania owoców prac z kolejnych etapów, czyli do testowania rozwiązania MySQL Cluster.

Narzędziem wykorzystanym do przeprowadzenia testów jest OSDB (*Open Source Database Benchmark*). Z dwóch dostępnych pakietów z danymi wykorzystywanymi do testów, wybraliśmy większą, 40MB bazę danych (ale dla porównania zrobiliśmy też przykładowy test z małą 4MB porcją). Wyniki otrzymane w teście to czas potrzebny na np. stworzenie poszczególnych struktur w bazie danych, jak również jednostka [tup/sek], która określa ilość tworzonych, bądź modyfikowanych krotek na sekundę.

Pierwszym elementem wymaganym do przystąpienia do procesu testowania było stworzenie bazy danych o nazwie "osdb". Następnie, po odpowiednim skonfigurowaniu i przekompilowaniu mogliśmy przystąpić do testów wykorzystując plik binarny "osdb-my" .

13.2 Testy.

Przykladowy (*logfile*) otrzymany w wyniku testu:

```
Compile-time options: none
Restrictions: insert_into rollback subqueries views
```

```
                create_tables() 0.14 seconds return value = 0
                        load() 39.14 seconds return value = 0
     create_idx_uniques_key_bt() 13.28 seconds return value = 0
     create_idx_updates_key_bt() 13.45 seconds return value = 0
     create_idx_hundred_key_bt() 11.31 seconds return value = 0
      create_idx_tenpct_key_bt() 12.19 seconds return value = 0
 create_idx_tenpct_key_code_bt() 19.96 seconds return value = 0
        create_idx_tiny_key_bt() 0.13 seconds return value = 0
      create_idx_tenpct_int_bt() 19.93 seconds return value = 0
   create_idx_tenpct_signed_bt() 24.98 seconds return value = 0
     create_idx_uniques_code_h() 18.11 seconds return value = 0
   create_idx_tenpct_double_bt() 22.77 seconds return value = 0
   create_idx_updates_decim_bt() 16.82 seconds return value = 0
    create_idx_tenpct_float_bt() 32.43 seconds return value = 0
     create_idx_updates_int_bt() 15.70 seconds return value = 0
    create_idx_tenpct_decim_bt() 7.74 seconds return value = 0
     create_idx_hundred_code_h() 3.40 seconds return value = 0
      create_idx_tenpct_name_h() 8.65 seconds return value = 0
     create_idx_updates_code_h() 4.67 seconds return value = 0
      create_idx_tenpct_code_h() 10.20 seconds return value = 0
  create_idx_updates_double_bt() 6.28 seconds return value = 0
    create_idx_hundred_foreign() 4.07 seconds return value = 0
             populateDataBase() 266.07 seconds return value = 0

Logical database size 40.0000MB (38.1470MiB, 100000 tuples/relation)

                    sel_1_cl() 0.00 seconds return value = 1
                   join_3_cl() 0.00 seconds return value = 0
                 sel_100_ncl() 0.02 seconds return value = 100
                  table_scan() 0.32 seconds return value = 0
                    agg_func() 0.65 seconds return value = 100
                    agg_scal() 0.00 seconds return value = 0
                   sel_100_cl() 0.02 seconds return value = 100
                   join_3_ncl() 0.00 seconds return value = 1
                sel_10pct_ncl() 1.78 seconds return value = 10000
          agg_info_retrieval() 0.15 seconds return value = 0
                    join_2_cl() 0.00 seconds return value = 0
                       join_2() 0.69 seconds return value = 1000
       sel_variable_select_low() 0.00 seconds return value = 0
     sel_variable_select_high() 4.64 seconds return value = 25000
                    join_4_cl() 0.01 seconds return value = 0
```

```
                    proj_100() 1.24 seconds return value = 10000
                   join_4_ncl() 0.01 seconds return value = 1
                   proj_10pct() 5.74 seconds return value = 100000
                    sel_1_ncl() 0.00 seconds return value = 1
                   join_2_ncl() 0.01 seconds return value = 1
           drop_updates_keys() 16.83 seconds return value = 0
                    bulk_save() 0.04 seconds return value = 0
                  bulk_modify() 1.27 seconds return value = 0
        upd_append_duplicate() 0.01 seconds return value = 0
        upd_remove_duplicate() 0.00 seconds return value = 0
               upd_app_t_mid() 0.00 seconds return value = 1
               upd_mod_t_mid() 0.01 seconds return value = 0
               upd_del_t_mid() 0.00 seconds return value = 0
               upd_app_t_end() 0.00 seconds return value = 1
               upd_mod_t_end() 0.00 seconds return value = 0
               upd_del_t_end() 0.00 seconds return value = 0
    create_idx_updates_code_h() 3.36 seconds return value = 0
               upd_app_t_mid() 0.00 seconds return value = 1
               upd_mod_t_cod() 0.00 seconds return value = 0
               upd_del_t_mid() 0.00 seconds return value = 0
    create_idx_updates_int_bt() 4.32 seconds return value = 0
               upd_app_t_mid() 0.00 seconds return value = 1
               upd_mod_t_int() 0.00 seconds return value = 0
               upd_del_t_mid() 0.00 seconds return value = 0
                 bulk_append() 0.21 seconds return value = 0
                 bulk_delete() 1.65 seconds return value = 0

Single User Test 42.98 seconds (0:00:42.98)

Executing multi-user tests with 10 user tasks
Mixed IR (tup/sec) 169.12 returned in 1.00 minutes (10147 tuples)
                    sel_1_ncl() 0.01 seconds return value = 1
crossSectionTests(Mixed IR) 0.01
Mixed OLTP (tup/sec) 0.70 returned in 1.00 minutes (42 tuples)
                    sel_1_ncl() 0.32 seconds return value = 1
crossSectionTests(Mixed OLTP) 0.32

Multi-User Test 260.91 seconds (0:04:20.91)
```

Jak widzimy log z przeprowadzonego testu zawiera szereg informacji, które dają nam możliwość porównywania przyjętych rozwiązań.

13.2.1 DB structure creation – tworzenie struktury bazy danych .

Część ta jest odpowiedzialna za stworzenie struktury bazy danych. Tworzenie jej zawiera:

- tworzenie tabeli – tworzenie tabeli w bazie danych o nazwie OSDB, która stanowi podstawę do późniejszych testów wydajnościowych
- generacja danych – kopiowanie danych z przygotowanych plików do wszystkich tabel wcześniej utworzonej bazy
- tworzenie kluczy głównych i indeksów – stworzenie kluczy głównych i indeksów oraz sprawdzenie danych w poszczególnych kolumnach. Są dwa typy indeksów
  - B–tree – reprezentowane symbolem "_bt"
  - hash – reprezentowane symbolem "_h"
- formowanie kluczy obcych – tworzenie kluczy obcych, przy czym mogą się pojawić komunikaty typu: *"NOTICE: ALTER TABLE ... ADD CONSTRAINT will create implicit trigger(s) for FOREIGN KEY check(s)"*, bądz podobne;

Wyniki testu krótkiego:

	short_1 [sek]	short_2 [sek]	short_2 [sek]
create_tables()	0.04	0.22	0.14
load()	5.00	50.28	39.14
create_idx_uniques_key_bt()	1.79	15.05	13.28
create_idx_updates_key_bt()	1.56	12.41	13.45
create_idx_hundred_key_bt()	1.48	9.21	11.31
create_idx_tenpct_key_bt()	1.22	7.26	12.19
Create_idx_tenpct_key_code_bt()	2.38	21.54	19.96
create_idx_tiny_key_bt()	0.07	0.14	0.13
create_idx_tenpct_int_bt()	2.49	18.84	19.93
create_idx_tenpct_signed_bt()	3.00	12.20	24.98
create_idx_uniques_code_h()	2.08	2.65	18.11
create_idx_tenpct_double_bt()	3.43	5.06	22.77
create_idx_updates_decim_bt()	1.69	2.74	16.82
create_idx_tenpct_float_bt()	4.29	3.70	32.43
create_idx_updates_int_bt()	2.07	2.34	15.70

create_idx_tenpct_decim_bt()	4.18	4.21	7.74
create_idx_hundred_code_h()	2.17	2.11	3.40
create_idx_tenpct_name_h()	4.72	4.81	8.65
create_idx_updates_code_h()	2.82	2.75	4.67
create_idx_tenpct_code_h()	5.61	5.24	10.20
Create_idx_updates_double_bt()	3.51	3.23	6.28
create_idx_hundred_foreign()	2.15	2.41	4.07
populateDataBase()	52.71	137.90	266.07

Oznaczenia:

- short_nr – pierwsze słowo oznacza typ przeprowadzonego testu (w przypadku opcji "long" jest przeprowadzana tzw. rozgrzewka bazy danych, która wydłuża znacznie czas testu, ale usprawnia wykonywanie niektórych operacji na bazie danych, co zresztą widać w tabelach)

- nr – czyli drugi symbol oznaczenia, informuje nas z którego etapu rozwiązanie było testowane

W pierwszej kolumnie mamy dane pochodzące z testów rozwiązania MySQL w wersji 5.1. Kolejna kolumna to wyniki z testów MySQL Cluster przy 2 serverach mysqld, a ostatnia to MySQL Cluster przy 4 serwerach.

Jak widać już samo tworzenie bazy danych w przypadku rozwiązania klastrowego wymaga więcej czasu, wiąże się to oczywiście z replikacją/tworzeniem identycznych struktur na innych serwerach zarządzanych przez MySQl Cluster. Wynik ładowania danych do powstałej bazy danych jest bardzo logiczny, im więcej serwerów mysql'a, tym więcej czasu potrzebujemy na załadowanie bazy danymi.

Wynik testu długiego:

	long_1 [sek]	long_2 [sek]	long_2 [sek]
create_tables()	0.01	0.07	0.10
load()	3.48	3.45	3.46
create_idx_uniques_key_bt()	2.46	2.35	2.26
create_idx_updates_key_bt()	1.67	1.38	1.41
create_idx_hundred_key_bt()	2.23	1.47	1.48
create_idx_tenpct_key_bt()	1.60	1.50	1.34
Create_idx_tenpct_key_code_bt()	3.14	2.04	2.62
create_idx_tiny_key_bt()	0.06	0.06	0.07
create_idx_tenpct_int_bt()	2.23	2.13	2.31
create_idx_tenpct_signed_bt()	3.20	3.04	2.77

create_idx_uniques_code_h()	1.72	1.52	2.01
create_idx_tenpct_double_bt()	3.75	3.59	3.20
create_idx_updates_decim_bt()	1.69	1.50	1.95
create_idx_tenpct_float_bt()	4.29	3.91	3.69
create_idx_updates_int_bt()	2.49	1.85	1.84
create_idx_tenpct_decim_bt()	4.40	4.33	4.27
create_idx_hundred_code_h()	2.18	2.04	2.34
create_idx_tenpct_name_h()	4.74	4.71	4.90
create_idx_updates_code_h()	2.59	2.76	2.68
create_idx_tenpct_code_h()	5.86	5.22	4.88
Create_idx_updates_double_bt()	3.52	3.25	2.72
create_idx_hundred_foreign()	2.54	2.42	2.75
populateDataBase()	56.36	51.07	51.49

Oznaczenia i kolejność wyników jest identyczna jak w poprzednim przypadku.

Z otrzymanych wyników wynika przede wszystkim, że rozgrzewka (*ang. database warm up*) w znaczny sposób usprawnia działanie bazy danych. Jednak to, że np. ładowanie do bazy danych ma bardzo zbliżone czasy w przypadku MySQL Cluster i zwykłego mysql'a, bardziej może wynikać z faktu, że testy pierwszego etapu były przeprowadzone w momencie, gdy działały tylko 3 elementy klastra, a nie jak w drugim przypadku, kiedy było ich 8.

Należy też mieć na uwadze, że testy poprzez długi czas trwania pojedynczego testu, nie mogły być przeprowadzone kilkakrotnie, czyli nie możemy wykluczyć że posiadanych pomiarach istnieją takie, które mają znaczne odchylenia od możliwej do uzyskania wartości średniej.

13.2.2 Single - user test.

Ta część testów zawiera proste operacje na danych, takie jak:

- *select* – zwyczajne zapytanie o rekord (krotkę) w tabeli z podaniem warunku
- *join* – zapytanie o rekord(y) z tabel (dwóch lub trzech) złączonych razem pod pewnym warunkiem
- *aggregation* – liczenie średnich, maksimów, minimów, oraz sum
- *view* – tworzenie widoków (z agregatami)
- *projection* – zapytanie za pomocą SQL'a o różne opcje jednej lub wielu tabel

- *add* – dodanie rekordu do tabeli
- *update* – uaktualnienie jednej lub większej liczby wartości w tabeli
- *delete* – usuniecie rekordu z tabeli z podaniem warunku
- *bulk operation* – usuniecie/dodanie wielu rekordów z/do tabeli
- *index update* – uaktualnienie indeksów po zmianie danych
- *integrity test* – sprawdzenie spójności danych w tabelach z kluczami obcymi

Wyniki testu krótkiego:

	short_1 [sek]	short_2 [sek]	short_2 [sek]
sel_1_cl()	0.03	0.28	0.00
join_3_cl()	0.00	0.00	0.00
sel_100_ncl()	0.01	0.04	0.01
table_scan()	0.18	0.15	0.16
agg_func()	0.35	0.29	0.29
agg_scal()	0.00	0.00	0.00
sel_100_cl()	0.01	0.01	0.01
join_3_ncl()	0.02	0.00	0.00
sel_10pct_ncl()	3.65	1.88	1.80
agg_info_retrieval()	0.11	0.10	0.08
join_2_cl()	0.00	0.00	0.00
join_2()	0.40	0.33	0.33
sel_variable_select_low()	0.00	0.00	0.00
sel_variable_select_high()	9.57	4.95	4.28
join_4_cl()	0.00	0.00	0.00
proj_100()	1.19	0.83	0.87
join_4_ncl()	0.00	0.01	0.00
proj_10pct()	7.56	5.10	5.12
sel_1_ncl()	0.00	0.01	0.00
join_2_ncl()	0.00	0.00	0.00
drop_updates_keys()	9.29	8.32	8.27
bulk_save()	0.03	0.01	0.01
bulk_modify()	1.24	1.26	1.10
upd_append_duplicate()	0.01	0.00	0.00
upd_remove_duplicate()	0.02	0.01	0.00
upd_app_t_mid()	0.00	0.00	0.00

upd_mod_t_mid()	0.00	0.00	0.00
upd_del_t_mid()	0.00	0.01	0.00
upd_app_t_end()	0.00	0.00	0.00
upd_mod_t_end()	0.00	0.00	0.00
upd_del_t_end()	0.00	0.00	0.01
create_idx_updates_code_h()	2.03	1.56	1.82
upd_app_t_mid()	0.00	0.00	0.00
upd_mod_t_cod()	0.00	0.01	0.00
upd_del_t_mid()	0.00	0.00	0.00
create_idx_updates_int_bt()	2.48	2.48	2.22
upd_app_t_mid()	0.00	0.00	0.00
upd_mod_t_int()	0.00	0.00	0.01
upd_del_t_mid()	0.00	0.00	0.00
bulk_append()	0.21	0.20	0.19
bulk_delete()	1.63	1.60	1.53

Widzimy, że czasy operacji na danych w przypadku rozwiązania klastrowego i rozwiązania zwykłego są do siebie zbliżone, a niekiedy czasy uzyskiwane przez MySQL Cluster są bardziej korzystne. Szczególnie jest to widoczne dla operacji zapytań do bazy danych.

Wyniki testu długiego:

	long_1 [sek]	long_2 [sek]	long_2 [sek]
sel_1_cl()	0.01	0.00	0.00
join_3_cl()	0.00	0.01	0.01
sel_100_ncl()	0.01	0.00	0.00
table_scan()	0.17	0.16	0.15
agg_func()	0.35	0.29	0.30
agg_scal()	0.00	0.00	0.00
sel_100_cl()	0.01	0.01	0.00
join_3_ncl()	0.00	0.00	0.00
sel_10pct_ncl()	3.73	1.68	0.29
agg_info_retrieval()	0.09	0.08	0.08
join_2_cl()	0.00	0.00	0.00
join_2()	0.40	0.33	0.34
sel_variable_select_low()	0.00	0.00	0.00
sel_variable_select_high()	9.95	4.22	0.60
join_4_cl()	0.00	0.00	0.00
proj_100()	1.20	0.78	0.51
join_4_ncl()	0.00	0.01	0.00

proj_10pct()	7.63	4.64	3.17
sel_1_ncl()	0.01	0.00	0.00
join_2_ncl()	0.00	0.00	0.00
drop_updates_keys()	8.97	8.50	8.24
bulk_save()	0.02	0.02	0.02
bulk_modify()	1.60	1.27	1.53
upd_append_duplicate()	0.00	0.00	0.01
upd_remove_duplicate()	0.00	0.00	0.00
upd_app_t_mid()	0.00	0.00	0.01
upd_mod_t_mid()	0.00	0.00	0.00
upd_del_t_mid()	0.00	0.01	0.00
upd_app_t_end()	0.00	0.00	0.00
upd_mod_t_end()	0.00	0.00	0.01
upd_del_t_end()	0.00	0.00	0.00
create_idx_updates_code_h()	1.99	1.91	1.82
upd_app_t_mid()	0.00	0.00	0.00
upd_mod_t_cod()	0.00	0.00	0.00
upd_del_t_mid()	0.00	0.00	0.00
create_idx_updates_int_bt()	2.14	2.27	1.95
upd_app_t_mid()	0.01	0.00	0.00
upd_mod_t_int()	0.00	0.00	0.00
upd_del_t_mid()	0.00	0.00	0.01
bulk_append()	0.43	0.21	0.45
bulk_delete()	1.55	1.50	1.48

Zgodnie z przewidywaniami uzyskaliśmy nieznaczną poprawę wyników względem poprzedniego rodzaju testów. Ogólne tendencje są takie same jak dla testu krótkiego.

Jako że czasy pojedynczych operacji umieszczonych w tabeli, poniżej przedstawiam sumacyjne wartości testu Single User Test:

Krótkie testy:

```
MySQL: 40.02 sec
MySQL Cluster (2 węzły): 27.90 sec
MySQL Cluster (4 węzły): 28.11 sec
```

Długie testy:

```
MySQL: 40.27 sec
MySQL Cluster (2 węzły): 27.90 sec
MySQL Cluster (4 węzły): 20.98 sec
```

Po sumie jeszcze bardziej widać, że rozwiązanie klastrowe wspomaga proste operacje na danych.

13.2.3 Multi-user test.

Test wielu użytkowników jest dedykowany do testowania transakcji i zachowania DBMS'a w sytuacji kiedy wielu użytkowników próbuje jednocześnie uzyskać dostęp do danych. Składa się z dwóch głównych części:

- *Mixed IR* – zapytywanie o pola w tabeli z zadanym warunkiem, który jest losowo generowany z wszystkich możliwych wartości zawartych w tabeli; wynik jest podawany w jednostce [tup/sec]
- *Mixed OLTP (On-Line Transaction Processing)* – uaktualnianie pól z warunkiem losowo wybranym w tabeli; wynik jest podany w [tup/sec]

Testy zostały wykonane z 10-ma użytkownikami.
Krótkie testy:
MySQL:

```
Mixed IR (tup/sec) 58.77 returned in 1.04 minutes (3655 tuples)
                     sel_1_ncl() 0.00 seconds return value = 1
crossSectionTests(Mixed IR) 0.00
Mixed OLTP (tup/sec) 0.23 returned in 1.02 minutes (14 tuples)
                     sel_1_ncl() 0.21 seconds return value = 1
crossSectionTests(Mixed OLTP) 0.21

Multi-User Test 287.27 seconds (0:04:47.27)
```

MySQL Cluster (2 wezły):

```
Mixed IR (tup/sec) 179.73 returned in 1.00 minutes (10784 tuples)
                     sel_1_ncl() 0.01 seconds return value = 1
crossSectionTests(Mixed IR) 0.01
Mixed OLTP (tup/sec) 0.06 returned in 1.20 minutes (4 tuples)
                     sel_1_ncl() 10.01 seconds return value = 1
crossSectionTests(Mixed OLTP) 10.01

Multi-User Test 282.48 seconds (0:04:42.48)
```

MySQL Cluster (4 wezły):

```
Mixed IR (tup/sec) 178.72 returned in 1.00 minutes (10723 tuples)
                     sel_1_ncl() 0.01 seconds return value = 1
crossSectionTests(Mixed IR) 0.01
Mixed OLTP (tup/sec) 0.02 returned in 2.02 minutes (2 tuples)
                     sel_1_ncl() 0.02 seconds return value = 1
```

```
crossSectionTests(Mixed OLTP) 0.02

Multi-User Test 321.71 seconds (0:05:21.71)
```

Długie testy:
MySQL:

```
Mixed IR (tup/sec) 156.52 returned in 5.00 minutes (46964 tuples)
                     sel_1_ncl() 0.00 seconds return value = 1
crossSectionTests(Mixed IR) 0.01
Mixed OLTP (tup/sec) 1.21 returned in 5.21 minutes (378 tuples)
                     sel_1_ncl() 0.10 seconds return value = 1
crossSectionTests(Mixed OLTP) 0.10

Multi-User Test 2451.79 seconds (0:40:51.79)
```

MySQL Cluster (2 wezły):

```
Mixed IR (tup/sec) 176.71 returned in 5.00 minutes (53012 tuples)
                     sel_1_ncl() 0.01 seconds return value = 1
crossSectionTests(Mixed IR) 0.01
Mixed OLTP (tup/sec) 0.06 returned in 5.61 minutes (21 tuples)
                     sel_1_ncl() 17.29 seconds return value = 1
crossSectionTests(Mixed OLTP) 17.29

Multi-User Test 2474.32 seconds (0:41:14.32)
```

MySQL Cluster (4 wezły):

```
Executing multi-user tests with 10 user tasks
Mixed IR (tup/sec) 182.71 returned in 5.00 minutes (54812 tuples)
                     sel_1_ncl() 0.00 seconds return value = 1
crossSectionTests(Mixed IR) 0.00
Mixed OLTP (tup/sec) 0.00 returned in 15.38 minutes (1 tuples)
                     sel_1_ncl() 0.02 seconds return value = 1
crossSectionTests(Mixed OLTP) 0.02

Multi-User Test 3047.66 seconds (0:50:47.66)
```

Po otrzymanych wynikach widać, że w przypadku rozwiązania klastrowego otrzymujemy w tym teście nieco gorsze wyniki. Wnioskujemy po tym, że utrzymanie spójności rozproszonej bazy danych jednak kosztuje trochę czasu.

Jednak pomimo tego, dalej możemy MySQL Cluster rozwiązaniem skalowalnym (oczywiście, w założeniach zgodnych z rozsądkiem), gdyż zwiększenie liczby serwerów nie wpłynęło bardzo niekorzystnie na czas wykonywania testu, ale należy pamiętać, że ten fakt jednak miał znaczenie.

13.3 Podsumowanie.

Niestety przeprowadzenie testów naszego autorskiego rozwiązania nie było możliwe, gdyż w momencie uruchomienia testów połączenie z nim było zrywane. Może to mieć związek z niestandardowym zachowaniem CVIP'a, na zastosowaniu którego opierało się nasze rozwiązanie. Diagnostyka działania CVIP wykazała iż w pewnym momencie przestał on działać prawidłowo - podczas przekierowywania zadań na różne węzły, potrafił on zawiesić połączenie bądź próbował połączyć się z nieistniejącym węzłem. Diagnostyka polegała na uruchomieniu na wszystkich dostępnych węzłach serwera nasłuchującego na używanym przez CVIP porcie i logowaniu wszelkich prób połączeń. Problemu tego nie udało nam się naprawić ze względu na brak uprawnień *roota* i kłopotami związanymi z ew. modyfikacjami go wymagającymi – np. pakiem Mon, CVIP, heartbeat itp.

Jak widać z przeprowadzonych testów istnieje uzasadnienie dla stosowania rozwiązań klastrowych baz danych. Jak każde rozwiązanie ma ono swoje plusy i minusy, więc decydując się na takie rozwiązanie powinniśmy dokładnie wiedzieć na jakich aspektach zależy nam najbardziej przy korzystaniu z BD.

Dodatki

Dodatek A

Skład zespołu

Funkcja	**Kierownik zespołu**
Osoba	Andrzej Kiewicz
Obowiązki	Zarządzanie projektem.
Funkcja	**Członek zespołu**
Osoba	Piotr Bilski
Obowiązki	Opieka nad subversion.
Funkcja	**Członek zespołu**
Osoba	Marek Kisiel
Obowiązki	Przeprowadzanie testów poszczególnych rozwiązań.
Funkcja	**Członek zespołu**
Osoba	Krzysztof Skulski
Obowiązki	Ujednolicanie dokumentacji, opisy spotkań/ustaleń.
Funkcja	**Członek zespołu**
Osoba	Łukasz Tupaj
Obowiązki	Prezentacja rozwiązania, tworzenie paczek z oprogramowaniem.

Dodatek B

Spis ustaleń

DATA	**7.03.2006** – spotkanie organizacyjne
Ogólnie	Nastąpił podział obowiązków, podjęta została decyzja o utworzeniu listy dyskusyjnej oraz ustalenie kolejnego terminu spotkania.
DATA	**21.03.2006** – wstępna wizja rozwiązania
Ogólnie	Spotkanie miało na celu podzielenie się informacjami na temat realizowanego projektu.
Osoba	Andrzej
Opis	Przedstawił zasadę działania rozwiązań klastrowych w MySQL.
Osoba	Łukasz
Opis	Przedstawił sposób tworzenia paczek z gotowym oprogramowaniem.
Osoba	Krzysiek
Opis	Przedstawił podstawowe założenia replikacji oraz zachowania spójności danych.
Osoba	Marek
Opis	Przedstawił główne założenia OpenSSI.
Osoba	Piotrek
Opis	Przedstawił podstawowe założenia pracy z subversion.
DATA	**28.03.2006** – propozycje projektowe[1]
Ogólnie	Dyskusje na temat projektu rozwiązania. Ustalenie terminu oddania prototypu rozwiązania.
Osoba	Andrzej

[1] Plus ustalenia z rozmów przeprowadzonych na liście dyskusyjnej.

Opis	Zaproponował oparcie rozwiązania na klastrowym rozwiązaniu dla MySQL. Przedstawił listę pytań i poprawek do opracowanego prototypu projektu.
Osoba	Krzysiek
Opis	Przygotował prototyp rozwiązania oraz listę pytań, na które należy odpowiedzieć przed kolejnym etapem projektu. Wysunął propozycję oparcia projektu na dotychczasowych rozwiązaniach integracji OpenSSI z MySQL i nastawienie projektu wyłącznie na próbę dopisania brakujących mechanizmów w MySQL.
Osoba	Marek
Opis	Przedstawił mechanizmy OpenSSI, które mogą być pomocne w rozwiązaniu zadania.
DATA	**8.05.2006** – spotkanie w laboratorium
Ogólnie	Testowanie paczki z rozwiązaniem klastrowym MySQL, omówienie projektu.
Osoba	Andrzej i Krzysiek
Opis	Przedyskutowanie ostatecznej formy projektu. Rozpatrzenie sytuacji awaryjnych oraz postępowania w razie wystąpienia awarii.
Osoba	Łukasz i Piotrek
Opis	Przetestowanie klastrowej paczki MySQL.
Osoba	Marek
Opis	Przedstawił możliwe opcje testowania aplikacji.
DATA	**25.05.2006**
Ogólnie	Testowanie paczki, testy.
Osoba	Andrzej i Krzysiek
Opis	Testowania i konfiguracja CVIP i Mon.
Osoba	Łukasz i Piotrek
Opis	Dopracowanie paczki.
Osoba	Marek
Opis	Przygotowywanie testów.

Dodatek C

Wkład w projekt

DATA	**7.03.2006** – pomocnicze
Osoba	Krzysiek
Opis	Utworzył listę dyskusyjną dla projektu.
DATA	**20.03.2006** – paczka
Osoba	Łukasz
Opis	Przedstawił paczkę z MySQL.
DATA	**21.03.2006** – subversion
Osoba	Piotrek
Opis	Skonfigurował subversion.
DATA	**29.03.2006** – dokumentacja
Osoba	Krzysiek
Opis	Utworzył wstępną, całościową wersję dokumentacji.
DATA	**1.04.2006** – dokumentacja
Osoba	Andrzej
Opis	Oddał ostateczną wersję opracowania na temat MySQL.
DATA	**1.04.2006** – dokumentacja
Osoba	Krzysiek
Opis	Oddał ostateczną wersję opracowania na temat replikacji, niezawodności oraz wysokiej dostępności.
DATA	**1.04.2006** – dokumentacja
Osoba	Krzysiek
Opis	Uaktualnił opis postanowień ze spotkań oraz z grupy dyskusyjnej.
DATA	**2.04.2006** – dokumentacja
Osoba	Marek
Opis	Oddał ostateczną wersję opracowania na temat OpenSSI.

DATA	**2.04.2006** – dokumentacja
Osoba	Łukasz
Opis	Oddał ostateczną wersję opracowania na temat utworzonej paczki z MySQL.
DATA	**3.04.2006** – dokumentacja
Osoba	Piotrek
Opis	Oddał ostateczną wersję opracowania na temat subversion.
DATA	**3.04.2006** – dokumentacja
Osoba	Krzysiek
Opis	Zamknął dokumentację do pierwszej części projektu.
DATA	**1.05.2006** – dokumentacja
Osoba	Łukasz i Piotrek
Opis	Uruchomili paczki z klastrową wersją MySQL.
DATA	**14.05.2006** – dokumentacja
Osoba	Andrzej i Krzysiek
Opis	Dostarczyli prototypu ostatecznego rozwiązania.
DATA	**15.05.2006** – dokumentacja
Osoba	Marek
Opis	Dostarczył prototypu testów gotowego rozwiązania.
DATA	**16.05.2006** – dokumentacja
Osoba	Piotrek
Opis	Dostarczył dokumentację testów MySQL Cluster.
DATA	**16.05.2006** – dokumentacja
Osoba	Łukasz
Opis	Dostarczył dokumentację paczki MySQL Cluster.
DATA	**6.06.2006** – konfiguracja
Osoba	Krzysiek
Opis	Konfiguracja CVIP
DATA	**7.06.2006** – konfiguracja
Osoba	Piotrek
Opis	Napisanie programu do detekcji CVIP.
DATA	**12.06.2006** – dokumentacja
Osoba	Łukasz
Opis	Dostarczył kolejną dokumentację paczki MySQL Cluster.
DATA	**16.06.2006** – dokumentacja
Osoba	Marek
Opis	Dostarczył dokumentację testów.
DATA	**16.06.2006** – dokumentacja
Osoba	Krzysiek

Opis	Złożył w całość dokumentację projektu.

Bibliografia

[1] Dokumentacja bazy danych MySQL 5.1, *Phttp://dev.mysql.com/doc/refman/5.1/en/index.html*

[2] Forum użytkowników MySQL dotyczące rozwiązań klastrowych, *http://forums.mysql.com/list.php?25*

[3] Forum użytkowników MySQL dotyczące alternatywnych systemów zarządzania pamięcią, *http://forums.mysql.com/list.php?94*

[4] Kris Buytaert: „Building a High-Availability MySQL Cluster", *http://www.oreillynet.com/pub/a/databases/2006/02/16/ha_mysql_cluster.html?page=1*

[5] Alex Davies: „HOWTO set up a MySQL Cluster for two servers (three servers required for true redundancy)", *http://dev.mysql.com/tech-resources/articles/mysql-cluster-for-two-servers.html*

[6] MySQL Clustering, *http://wiki.openssi.org/go/MySQL_Clustering*

[7] okumentantacja ze strony przedmiotu RSO, *http://www.ia.pw.edu.pl/ tkruk/openssi*

[8] okumentacja OpenSSI, *http://www.openssi.org/*

[9] Evan Marcus, Hal Stern: „Blueprints for High Availability Second Edition", Wydawnictwo Wiley Publishing, Indianapolis 2003

[10] Karl Kopper: „Linux Enterprise Cluster", Wydawnictwo No Starch Press, 2005

[11] Derek J. Balling, Jeremy Zawodny: „High Performance MySQL", Wydawnictwo O'Reilly 2004

[12] Andrew S. Tanenbaum, Maarten van Steen: „Systemy rozproszone: Zasady i paradygmaty", Warszawa 2006

[13] Collins-Sussman, Fitzpatrick, Pilato: „Version Control with Subversion (For Subversion 1.2)", 2006 O'Reilly, Stanford, California, USA

[14] Subversion documentation *http://subversion.tigris.org/*

[15] Dokumentacja projektu Makeself, *http://www.megastep.org/makeself/*