Podsumowanie I fazy projektu z RSO.

Grupa RSO4

Andrzej Grudzień
Grzegorz Lepionka
Maciek Nowak
Marcin Rudowski
Paweł Słupczyński

4 kwietnia 2006

Spis treści

1 Wstęp

Niniejszy dokument łączy opracowania przygotowane przez członków zespołu RSO4, wymagane w pierwszej fazie projektu. Realizacja części praktycznej (dystrybucja serwera MySQL) jest umieszczona w repozytorium svn (/var/svn/rso4/).

2 Przygotowanie paczki z instalacją serwera MySQL na koncie użytkownika

Opracował: Paweł Słupczyński

Dystrybucja serwera MySQL działającego na koncie użytkownika, bez potrzeby posiadania uprawnień root-a, czy dostępu do plików spoza katalogu z serwerem, powstała na bazie najnowszej (wersja 5.1.7) paczki dostępnej na stronie
http://dev.mysql.com/downloads/mysql/5.1.html
Po jej rozpakowaniu, uruchomiony został skrypt tworzący strukturę bazy danych bin/mysql_install_db, poprawiono prawa dostępu do niektórych katalogów (data, tmp) oraz stworzono skrypt kontrolujący silnik serwera. Paczka z silnikiem bazy danych serwera MySQL jest dostępna na repozytorium grupy (/var/svn/rso4/) pod nazwą *mysql-5.1.7-rso4-f1.tgz* . Wystarczy ściągnąć podany plik, rozpakować paczkę (polecenie: *tar -xzf mysql-5.1.7-rso4-f1.tgz*), skonfigurować, i serwer jest gotowy do pracy. Wszelkich ustawień należy dokonywać w skrypcie uruchumiającym *mysqlctl* - znajduje się on w katalogu z dystrybucja. W tymże katalogu znajduje się również plik README, zawierający podstawowe informacje o korzystaniu z paczki.

3 Opis i analiza już istniejących rozwiązań w ramach MySQL umożliwiających realizację klastrowości, zwiększenia niezawodności i wydajności serwera bazy danych

Opracował: Andrzej Grudzień

3.1 Wstęp

W systemie zarządzania bazą danych MySQL istnieją trzy istotne rozwiązania umożliwiające zwiększenie niezawodności i wydajności świadczonych usług.

Podstawowym udogodnieniem jest partycjonowanie, umożliwiające podział tabel według ustalonych przez administratora reguł i rozmieszczenie tak określonych fragmentów w różnych miejscach w systemie plików. Pozwala to na zwiększenie wydajności zapytań ze względu na możliwość wstępnej eliminacji przeszukiwania niektórych partycji na podstawie warunków zadanych w zapytaniu, a także, ze względu na możliwość zrównoleglenia operacji dyskowych, w przypadku wykorzystania operacji agregujących.

Drugim dostępnym rozwiązaniem jest replikacja. W postaci oferowanej w MySQL pozwala na utrzymywanie wielu kopii jednej bazy. W naturalny sposób zwiększa to bezpieczeństwo danych, gdyż kopie mogą znajdować się w fizycznie różnych lokacjach, natomiast zysk wydajności będzie uzyskany, w wypadku dużej liczby użytkowników, gdy zastosuje się równoważenie obciążenia rozdzielając nadchodzące zapytania na wszystkie kopie.

Najbardziej zaawansowanym rozwiązaniem jest jednak MySQL Cluster, który korzysta z dwóch powyższych mechanizmów. Jest to rozwiązanie klastrowe w którym w wielu węzłach przechowywane są repliki fragmentów tabel. Jest to architektura bez pojedynczego punktu awarii, zdolna do re-konfiguracji w wypadku uszkodzenia łącz lub węzłów. Dostęp do danych realizowany jest, z punktu widzenia użytkownika, przez ten sam interfejs co w przypadku zwykłego scentralizowanego SZBD.

Dokładniejszy opis i analiza przedstawionych rozwiązań jest tematem niniejszej pracy.

3.2 MySQL Cluster

3.2.1 Elementy klastra

Podstawowym pojęciem w architekturze klastra MySQL jest węzeł, pod którym w tym kontekście rozumie się jeden z trzech procesów: ndbd, msqld lub ndb_mgm. W związku z tym na jednym komputerze może znajdować się więcej niż jeden węzeł – w takim przypadku określa się go mianem cluster host. Jednak takie rozmieszczenie węzłów, z wyjątkiem przypadku który zostanie omówiony przy opisie NDB, jest niezalecane ze względów wydajnościowych i bezpieczeństwa. Każdy węzeł, niezależnie od typu jest identyfikowany unikalnym w obrębie klastra numerem ID.

Rozwiązanie klastrowe MySQL składa się z węzłów trzech rodzajów:

- danych (data node);procesy ndbd, których powinno być nie mniej niż dwa. Są one odpowiedzialne za przechowywanie danych oraz za wykonywanie podstawowych operacji dostępu do nich.

- zarządzających; procesy ndb_mgm w klastrze musi być dostępny minimum jeden węzeł tego typu w trakcie startu klastra. Istnieje możliwość stosowania większej ilości węzłów tego typu, ale wymaga to zachowania pełnej zgodności konfiguracji wszystkich tych węzłów.
- SQL; procesy mysqld, komunikujące się z procesami ndbd; ich zadaniem jest przyjmowanie zapytań, ich optymalizacja, rozbijanie na operacje pierwotne i przesyłanie żądania ich wykonania do węzłów danych. W klastrze może występować dowolna liczba węzłów SQL.

3.2.2 Komunikacja między węzłami

Każdy z węzłów powinien mieć łączność z pozostałymi wchodzącymi w stan klastra. Komunikacja może odbywać się poprzez jedno z trzech mediów:

- sieć komputerową udostępniającą protokół TCP
- pamięć dzieloną
- SCI (Scalable Coherent Interface)

W wypadku zastosowania sieci komputerowej wszystkie węzły wchodzące w skład klastra powinny znajdować się w ramach tej samej sieci. Dodatkowo, ze względu na brak zabezpieczeń przy komunikacji między węzłami, zalecane jest wykorzystanie do tego celu odrębnej sieci, niedostępnej z zewnątrz. Minimalna wymagana przepustowość to 100Mbit/s a zalecana 1Gbit/s. Istnieje także wymóg, by wszystkie komputery używały kart sieciowych o tej samej przepustowości (np. wszystkie 100Mbit/s).

Pamięć dzielona wykorzystywana jest w przypadku gdy na jednym komputerze znajduje się więcej niż jeden węzeł. Zapewnia to znacznie szybszą komunikacje między tymi węzłami, niż przy zastosowaniu np. mechanizmu pętli zwrotnej i protokołu TCP.

Rozwiązanie SCI jest najszybszym z dostępnych rozwiązań pozwalających łączyć węzły znajdujące się na odrębnych maszynach. Jest to połączenie typu point-to-point w którym uczestniczące maszyny łączy się w pierścień lub też wykorzystuje się switch SCI. Oprócz specjalnego sprzętu wymaga także odpowiednio przystosowanego sytemu operacyjnego, z odpowiednimi elementami wkompilowanymi w jądro (w syst. Linux możliwe jest zastosowanie modułów). Dodatkowo zalecane jest przydzielenie węzłowi jednego procesora, ponieważ węzeł wykorzystujący SCI nigdy nie przechodzi w stan uśpienia, co w praktyce oznacza konieczność stosowania maszyny co najmniej dwuprocesorowej. Zyskiem z użycia SCI jest znaczący wzrost wydajności względem sieci Ethernet – według autorów dokumentacji rzędu 100% – jeżeli łączone są w ten sposób węzły danych. W wypadku połączenia węzłów SQL różnica jest nieznaczna, a dla węzłów zarządzających stosowanie tego łącza stanowi marnotrawstwo, gdyż te węzły nie są wykorzystywane w trakcie normalnej pracy klastra.

Różnice pomiędzy różnymi rodzajami połączeń ukryte są w modułach transporterów. W pozostałych częściach węzłów operuje się tylko pojęciem sygnału i przekazywania sygnałów – co jest równoznaczne z przekazywaniem wiadomości.Dodatkowo warstwa transportera buforuje polecenia do wysłania – korzysta przy tym z algorytmu adaptacyjnego którego zadaniem jest maksymalizacja porcji danych wysyłanych przez łącze przy minimalizacji opóźnień.

3.2.3 Klaster NDB

Elementem odpowiedzialnym za fizyczne przechowywanie bazy danych w klastrze MySQL jest klaster NDB. Pod pojęciem tym rozumiany jest zbiór wszystkich węzłów danych w klastrze. Jego wyróżniającą cechą jest to, że może on istnieć w oderwaniu od pozostałych węzłów – oznacza to, że możliwa jest konfiguracja klastra składająca się tylko z węzłów danych (oraz węzła danych na czas startu). W takim wypadku jednak dostęp do danych odbywa się przez niestandardowe API, co znacząco utrudnia zadanie osób tworzących aplikacje mające korzystać z danych znajdujących się w tak zbudowanej bazie.

Tak określony klaster NDB dzieli się na grupy węzłów. Zadaniem każdej z grup jest przechowywanie replik jednej partycji każdej tabeli. Minimalna ilość węzłów w grupie wynosi jeden, a maksymalna cztery. Wszystkie grupy składające się na jeden klaster NDB muszą mieć taką samą liczbę węzłów. Przydział węzłów do grup odbywa się automatycznie na podstawie identyfikatorów węzłów. Przykładowo, jeśli w klastrze występują cztery węzły danych o ID 2, 4, 5, 6 oraz zadana liczebność grup wynosi dwa, to węzły 2 i 4 utworzą jedną grupę, a 5 i 6 drugą.

Przydział partycji do grup nie jest bezpośrednio konfigurowalny. Tablice są dzielone horyzontalnie według wartości funkcji haszującej z klucza głównego na liczbę fragmentów odpowiadającą zadeklarowanej liczbie grup. Wynikowe partycje są przydzielane do grup.

W każdej grupie określana jest główna replika, to jest węzeł do którego kierowane są zapytania o dane z partycji przynależnych grupie. Wybór dokonywany jest na podstawie numeru ID węzła, przy czym niższa wartośc oznacza wyższy priorytet. W wypadku awarii aktualnej głównej repliki jej następca określany jest na drodze elekcji zgodnie z algorytmem tyrana.

3.2.4 NDB Storage Engine

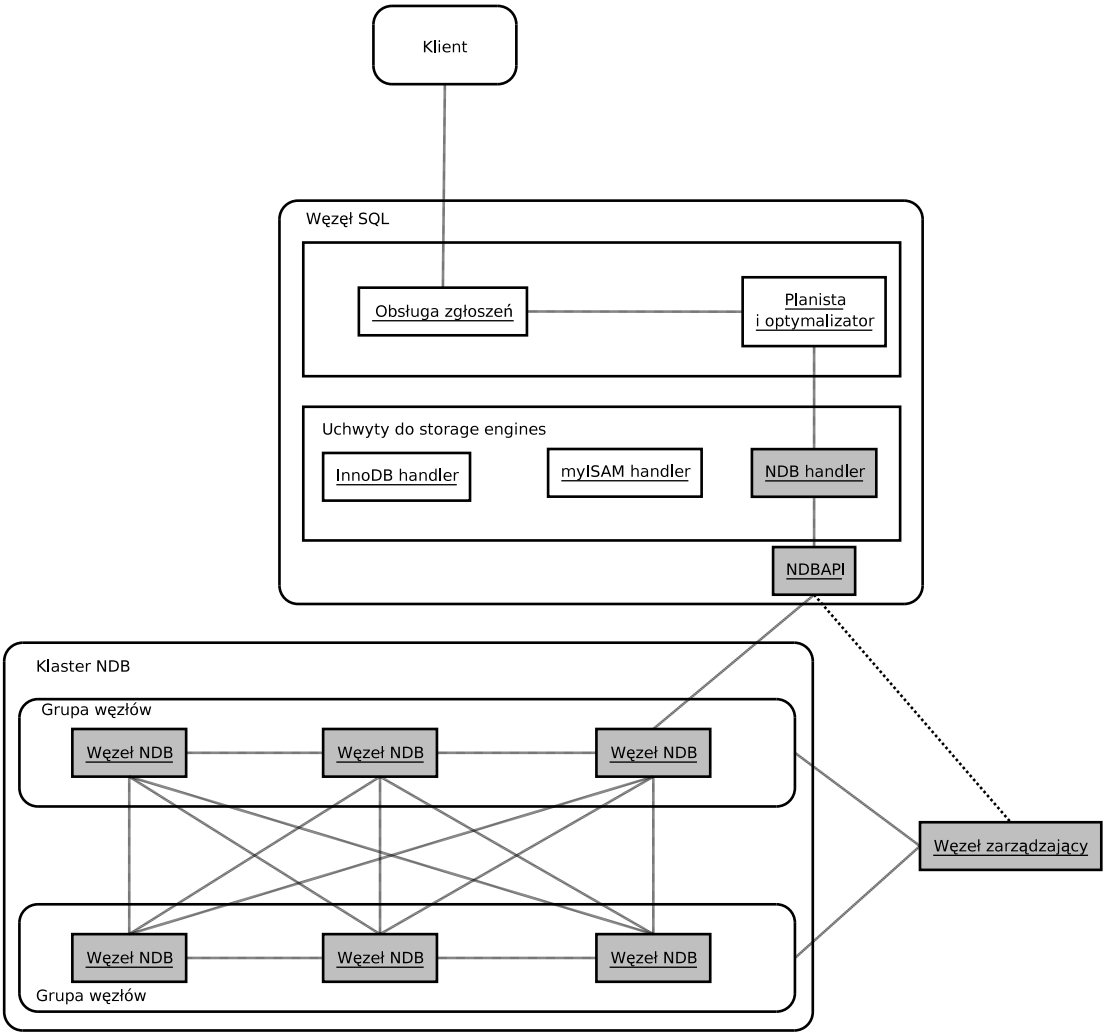
W SZBD MySQL za fizyczne przechowywanie danych odpowiadają różne storage engine's których interfejs i sposób interakcji z wyższą warstwą określony jest przez „Pluggable Storage Engine Architecture". Ze względu na to, iż w klastrze MySQL dostęp do danych odbywa się zgodnie z założeniami tej architektury, warto ją krótko przedstawić.

Storage engine jest reprezentowany w systemie przez pojedynczy obiekt nazywany handlerton'em (handler singleton). Zawiera on dane i metody dotyczące storage engine jako całości – w szczególności znajdują się w nim metody związane z przeprowadzaniem transakcji.

Wśród zadań handlerton'u znajduje się też tworzenie uchwytów (handler). Za każdą tablicę w systemie odpowiada jeden obiekt typu uchwyt. Do uchwytów wyższe warstwy delegują następujące zadania:

- fizyczne tworzenie tablicy.
- zakładanie blokad.
- przygotowanie podpowiedzi dla optymalizatora oraz zarządzanie indeksami.
- przygotowanie i przeprowadzenie dostępu do danych.

W przypadku NDB metody uchwytów i handlerton'a korzystają z obiektów określonych w NDBAPI. Obie te warstwy przedstawiają zbliżony poziom abstrakcji,



Rysunek 1: Budowa prostego klastra MySQL.

przy czym NDBAPI nieznacznie niższy, więc jedno wywołanie metody uchwytu przekładane jest na wywołanie kilku metod NDBAPI oraz pewną ilość pracy związaną ze zbieraniem statystyki i dopasowaniem działania do wymaganej semantyki, w tym konwersje danych z wewnętrznego formatu MySQL na używany przez NDB.

NDBAPI jest biblioteką obsługującą połączenia z klastrem NDB. Jak wszystkie składniki klastra MySQL wiedzę o lokalizacji (adresach) węzłów danych czerpie z węzła zarządzającego. Co prawda „wie" ona o istnieniu wielu węzłów danych, lecz zlecenia wysyłane są tylko do jednego, wybranego węzła. Biblioteka ta obejmuje także lokalny cache wierszy uzyskanych od klastra NDB.

Poniżej warstwy NDBAPI znajdują się węzły danych. Są one tymi elementami NDB do których ostatecznie trafiają wszelkie prośby o wykonanie operacji na danych. One też mają największy wpływ na charakterystyki NDB jako storage engine'u. Te cechy to:

- przechowywanie danych w pamięci. Oznacza to dużą wydajność, lecz także duże wymagania sprzętowe – komputer na którym znajduje się węzeł musi mieć dość pamięci by pomieścić n-tą cześć całej bazy (gdzie n – ilość partycji), struktury stworzone na potrzeby indeksów oraz bufory logów. W praktyce oznacza to niemożliwość zastosowania tego rozwiązania do przechowywania zbiorów danych o bardzo dużych rozmiarach (liczonych w gigabajtach). Środkiem zaradczym ma być wprowadzona w wersji 5.1.6 możliwość przechowywania kolumn na których nie są założone indeksy na dysku. Jest to jednak rozwiązanie bardzo młode i póki co niezbyt eksponowane, nawet przez samych autorów.

- separacja transakcji na poziomie READ COMMITED.

- ziarnistość blokowania na poziomie wierszy.

- ograniczenia rozmiarów metadanych. Żadna tablica nie może mieć więcej niż 128 atrybutów, nazwy kolumn mogą liczyć co najwyżej 31 znaki a maksymalna łączna długość nazwy tablicy i bazy w której się ona znajduje to 122 znaki.

- ograniczenia rozmiarów przechowywanych danych. Oprócz wspomnianego już ograniczenia wynikającego ze sposobu przechowywania istnieją dodatkowe wynikające z założeń implementacyjnych. Maksymalny rozmiar jednego wiersza to 8 kilobajtów z wyłączeniem obiektów BLOB. Poza tym atrybuty o typach o zmiennym rozmiarze (takich jak np. VARCHAR), przechowywane są i zajmują tyle samo miejsca co atrybuty o rozmiarze stałym, odpowiadającym maksymalnej długości typu o zmiennej długości. W wersji 5.17 wprowadzono typy o faktycznie zmiennej długości, lecz sprawiają one problemy chociażby z obliczaniem rozmiaru tabel, więc stosuąc je trzeba zachowywać dużą ostrożność.

- brak wsparcia dla konstrukcji SQL FOREIGN KEY. Jest to cecha powszechna wśród storage engine'ów MySQL (jedynym wyjątkiem jest InnoDB).

- konieczność posiadania przez każdą tablicę klucza głównego. Jeżeli nie zostanie on zdefiniowany jawnie przez użytkownika to zostaje automatycznie stworzony klucz ukryty.

Jak już wcześniej wspomniano, pod pojęciem węzła rozumiany jest jeden proces. W wypadku węzła danych nie jest to jednak w pełni ścisłe stwierdzenie. Podstawowe

działania związane z obsługą danych są co prawda wykonywane w jednym procesie, lecz dodatkowo aktywny jest proces nadzorujący (angel). Jego zadaniem jest monitorowanie stanu procesu głównego i w wypadku wykrycia jego braku, ponowne jego uruchomienie.

W samym procesie głównym działają dwa wątki. Pierwszy odpowiada za obsługę komunikacji oraz dostęp do danych. Możliwość obsługi wielu żądań jest zrealizowana przez używanie tylko wywołań nieblokujących i model komunikacji oparty na asynchronicznym przekazywaniu komunikatów.

Zadaniem drugiego jest pełnienie roli watchdog'a. Okresowo kontroluje on stan wątku głównego, sprawdzając czy przebywa on w którejś z pętli próbując wykonać jedno działanie. W wypadku gdy trzy razy zauważy taką sytuacje zabija cały proces. W połączeniu z działaniem procesu-opiekuna owocuje to ponownym startem węzła i próbą dołączenia się do klastra.

Z opisanych wyżej elementów budowy wynikają dwa istotne fakty:

- optymistyczne założenie o dostępności co najmniej jeszcze jednego działającego węzła w grupie. W innym wypadku rozwiązywanie problemów z działaniem przez restartowanie węzła może doprowadzić do utraty części danych. Dodatkowo podważa to wartość deklaracji o zbędności węzła zarządzającego po wystartowaniu klastra.

- ograniczenie maksymalnej ilości zajętych procesorów do dwóch. Ma to znaczenie w wypadku komputerów o liczbie procesorów cztery lub więcej. W takim przypadku zalecane jest uruchamianie drugiego węzła danych, ale należącego do innej grupy.

3.2.5 Start klastra

Przed uruchomieniem klastra niezbędne jest przygotowanie plików konfiguracyjnych dla węzłów wchodzących w jego skład.

Stosowany jest tu scentralizowany model przechowywania informacji o klastrze. Wszystkie ustawienia, włączając w to przydział numerów węzłów (unikalnych w obrębie klastra) oraz ilość replik, znajdują się w jednym pliku znajdującym się na komputerze na którym wykonywany jest węzeł zarządzający. Pozostałe węzły konfigurowane są jedynie przez podanie lokacji węzła zarządzającego. Przed startem w pliku konfiguracyjnym musi być także określona liczba węzłów każdego typu wchodząca w skład klastra. Warto tutaj wspomnieć, że z wyjątkiem węzła zarządzającego, węzły nie muszą mieć zadeklarowanej w pliku konfiguracyjnym lokalizacji (adresu) – w takim przypadku przyjęte do klastra będą te węzły, które pierwsze się zgłoszą.

Z przyjęcia powyższego modelu wynika kolejność uruchamiania węzłów. Warto zaznaczyć, że każdy węzeł musi być uruchamiany osobno na komputerze na którym ma pracować.

Pierwszym uruchamianym węzłem jest węzeł zarządzający. Dopuszczalne jest użycie więcej niż jednego takiego węzła, ale wtedy muszą one posiadać identyczne dane o konfiguracji klastra.

Następnie należy wystartować węzły danych. Nie ma żadnych wbudowanych mechanizmów zabraniających zrobienia tego przed uruchomieniem węzła zarządzającego, ale w takiej sytuacji będą one czekały, aż taki węzeł stanie się dostępny. Po uzyskaniu konfiguracji następuje załadowanie danych do pamięci z logów (więcej na temat przechowywania danych w czasie, gdy klaster jest wyłączony w punkcie

3.2.7) oraz tworzone są indeksy. Następuje tu też nawiązanie połączeń między węzłami w klastrze NDB.

Po uruchomieniu klastra NDB możliwe jest uruchamiane węzłów SQL. Aby można było przeprowadzać transakcje wymagane jest by były one w stanie połączyć się z węzłem zarządzającym i co najmniej jednym węzłem danych spośród zdefiniowanych w konfiguracji.

3.2.6 Przeprowadzanie transakcji

W klastrze MySQL inicjowanie transakcji i jej nadzór należy do zadań węzłów SQL. Korzystając z NDBAPI przygotowują lokalnie planowany przebieg transakcji, to znaczy zbiór operacji jakie mają być w jej ramach wykonane.

W ramach tych czynności przygotowawczych może mieć także miejsce zmiana żądanego poziomu blokowania. MySQL w obecnej wersji zawsze żąda blokowania całej tablicy, tak więc każdy storage engine obsługujący blokowanie na poziomie wierszy (tak jak ma to miejsce w NDB) musi sam zdecydować o zmianie poziomu.

Blokowanie dostępne w NDB jest typu pesymistycznego, z rozróżnieniem blokowania do zapisu i blokowania do odczytu. Blokady propagowane są wśród wszystkich replik obejmujących dany wiersz, poczynając od repliki głównej.

W trakcie realizacji transakcji węzeł SQL wymienia wiadomości z jednym z węzłów danych. Jest on określany mianem koordynatora transakcji. Jako że każdy z węzłów jest w stanie operować tylko na swoich lokalnych danych, koordynator, w zależności od typu żądania, kieruje żądania do pozostałych węzłów.

W wypadku gdy operacją jest wyszukanie danych i zachodzi konieczność przeszukania całej tabeli (full scan), operacja ta jest zlecana wszystkim głównym replikom partycji danej tabeli. Jeżeli natomiast potrzebne jest wyszukanie danych według klucza głównego, to koordynator zleca to zadanie tylko jednemu z węzłów.

Operacje zapisu i usuwania danych natomiast są wykonywane synchronicznie we wszystkich replikach. Ostateczny rezultat transakcji określany jest z użyciem protokołu 2PC (2 Phase Commit). O ostatecznym rezultacie transakcji decyduje głosowanie węzłów biorących w niej udział. Jeśli choć jeden z nich zgłosi sprzeciw, to do pozostałych wysyłane jest polecenie cofnięcia zmian, a klientowi zwracana jest informacja o niepowodzeniu transakcji.

Ważnym problemem związanym z działaniem klastra jest brak mechanizmów odkrywania zmian w metadanych przez węzły SQL. Przykładowo efekty wykonania polecenia ALTER TABLE są widoczne tyko w węźle w którym zostało ono wydane. Może to prowadzić do niespójnego widoku na bazę danych klientów korzystających z różnych węzłów SQL. Zalecanym rozwiązaniem jest tymczasowe zawieszenie działania wszystkich węzłów SQL oprócz jednego. Zadanie to możne być wykonane zdalnie za pośrednictwem węzła zarządzającego.

3.2.7 Obsługa awarii

Klaster MySQL uwzględnia w swojej budowie możliwość awarii dowolnego z węzłów. Od typu węzła natomiast zależy jak bardzo skomplikowany jest proces ponownego dołączenia go do klastra.

Najprostszym przypadkiem jest awaria węzła zarządzającego. Dane w nim przechowywane nie zmieniają się w czasie a pozostałe węzły korzystają z niego tylko w specjalnych przypadkach (restart, konieczność arbitrażu). Dlatego w wypadku awarii wystarczy ponowne uruchomienie.

Podobna jest sytuacja węzła SQL. W nim także nie są przechowywane dane, a najgorszy skutek awarii, to przerwanie konwersacji z użytkownikiem. Ze względu na zasadę przesyłania do klastra NDB operacji dopiero na zakończenie transakcji, upadek węzła SQL nie grozi pozostawieniem danych w klastrze w stanie niespójnym.

Najgorszym przypadkiem jest awaria któregoś z węzłów danych wchodzących w skład klastra NDB. Dodatkowo sytuacja jest komplikowana przez fakt przechowywania danych przez węzły w pamięci.

Klaster NDB jest w stanie pracować bez utraty danych dopóki w każdej z grup pozostaje aktywny co najmniej jeden węzeł. Jeżeli ten warunek jest zachowany to efektem awarii pojedynczego węzła może być co najwyżej konieczność wycofania transakcji. Ubocznym skutkiem będzie też pewien spadek czasu wykonywania operacji modyfikujących dane, następujących po awarii a przed ponownym dołączeniem węzła, ze względu na zmniejszenie liczby replik w których zapis musi mieć miejsce.

W takiej sytuacji także ponowne włączenie węzła danych nie stanowi znaczącego problemu. Zgłasza się on wtedy do głównej repliki z prośbą o przesłanie danych. Tak otrzymywane dane przesyłane są całymi fragmentami i stają się „widoczne" dla transakcji dopiero w chwili otrzymania całego fragmentu tablicy, co zabezpiecza przed niespójnością, gdyż węzeł albo już posiada dany fragment i może dokonać w nim modyfikacji lub też dopiero go otrzyma już z prowadzonymi zmianami. Negatywne efekty tej procedury to zwększenie ruchu w sieci oraz opóźnienie zakończenia transakcji nakładających się z nią czasowo, choć problem ten zachodzi tylko w przypadku modyfikacji danych, gdyż odczyty wykonywane są z repliki głównej, a dołączający węzeł z definicji nie może nią być.

W wypadku gdy awarii ulegną wszystkie węzły z grupy istnieje ryzyko utraty części danych. Środkiem zaradczym jest mechanizm opierający się na logowaniu operacji oraz lokalnych i globalnych punktach kontrolnych.

Lokalny punkt kontrolny zachodzi co okres czasu określony w konfiguracji. Wykonywany jest odrębnie dla każdego węzła bez komunikacji z pozostałymi. W ramach jego wykonania tworzona jest spójna pod względem transakcyjnym migawka wszystkich przechowywanych danych. Przechowywana jest ona dalej na dysku. Dodatkowo usuwana jest niepotrzebna już część logu.

Globalny punkt kontrolny określa chwile w których wszystkie logi transakcji zostają przeniesione na dysk. Podczas wykonywania operacji logowanie w węzłach odbywa się w pamięci operacyjnej, ponieważ zakłada się, że utrzymywanie kopii w dwóch odrębnych węzłach stanowi dostateczne zabezpieczenie, a po drugie wykonywanie operacji dyskowych w ramach transakcji spowodowałoby dalsze ich spowolnienie (ponieważ aktualizacje wykonywane są synchronicznie we wszystkich węzłach grupy).

Dzięki tym mechanizmom w przypadku poważnej awarii możliwe jest przywrócenie danych do ostatniego globalnego punktu kontrolnego, przez łączenie w poszczególnych węzłach migawek i logów. Oznacza to, że tracone są dane z okresu czasu nie większego niż odległość między dwoma globalnymi punktami kontrolnymi. Odpowiednie ustawienie tego parametru pozwala osiągnąć kompromis między bezpieczeństwem a wydajnością potrzebnymi w danym zastosowaniu.

Prawidłowe funkcjonowanie systemu zależy nie tylko od mechanizmów przywracania danych, ale także od sprawnego wykrywania awarii węzłów. W klastrze NDB wykorzystywane są dwie metody detekcji awarii węzła: błąd połączenia i sygnał bicia serca.

W trakcie normalnego funkcjonowania łączność między węzłami danych jest typu

„każdy-z-każdym". Jeżeli którykolwiek z węzłów zauważy utratę połączenia z którymś z węzłów, rozgłasza to do pozostałych i wszystkie klasyfikują dany węzeł jako nieaktywny, niezależnie od stanu ich własnego połączenia z wskazanym węzłem. Rozwiązanie to pozwala na szybkie wykrycie poważnych błędów, w szczególności sprzętu komunikacyjnego.

W celu wykrycia błędów takich jak problemy z dyskami, pamięcią, czy też nadmiernym obciążeniem procesora, które sprawiają, że węzeł przestaje poprawnie funkcjonować, ale nie naruszają połączenia wykorzystywany jest sygnał bicia serca. Węzły w klastrze NDB są zorganizowane logicznie w krąg, w którym każdy węzeł wysyła okresowo wiadomość do swoich sąsiadów, informującą o jego działaniu. Okres co jaki jest on wysyłany, a także jakie są kryteria uznania węzła za niedziałający można ustawić niezależnie dla każdego węzła. Właściwe ustawienie tych wartości jest bardzo istotne, gdyż przy nierozsądnym dobraniu tych parametrów można doprowadzić do sytuacji w której w pełni sprawny węzeł będzie po krótkiej chwili od startu klastra uznany za nieaktywny.

Ważnym problemem z jakim musi radzić sobie każde rozwiązanie klastrowe jest możliwość podziału klastra w wypadku np. awarii połączeń. W takiej sytuacji każda z części musi podjąć decyzję, czy kontynuować, czy zakończyć działanie. Podjęcie błędnej decyzji o kontynuacji przez dwie lub więcej części może doprowadzić do utraty spójności danych.

Klaster NDB wykorzystuje tutaj algorytm partycjonowania sieci uwzględniający jego specyfikę, to jest istnienie grup węzłów. Decyzja o kontynuowaniu działania podejmowania jest według następujących reguł:

- posiadanie wszystkich węzłów z jednej z grup, oraz co najmniej po jednym węźle z pozostałych.
- zasady większości, czyli posiadania ponad połowy węzłów z każdej grupy.
- posiadania co najmniej jednego węzła w każdej grupie i zezwolenia arbitra na kontynuowanie pracy.

Występującym w tych regułach arbitrem mogą być węzły zarządzające lub SQL. Użytkownik ma możliwość ustawienia preferencji co do tego, który z węzłów może zostać arbitrem. W wypadku awarii aktualnego arbitra węzły danych dokonują elekcji jego następcy spośród zgłaszających się kandydatów. Ważną uwagą jest to, że elekcja nie może odbyć się w trakcie podziału. Oznacza to, że może zajść sytuacja w której brakuje arbitra, co prowadzi do zakończenia działania przez wszystkie części, chyba, że jedna z nich spełnia warunek nie wymagający zezwolenia arbitra.

Kolejnym istotnym problemem jest wykrywanie zakleszczeń. Jak już wspomniano NDB stosuje zasadę pesymistycznego blokowania, więc istnieje zagrożenie ich wystąpienia. Sytuacje tego typu rozwiązywane są na zasadzie przeterminowania, czyli jeżeli dana operacja nie może być wykonana przez zadany w konfiguracji klastra okres czasu, to uznaje się, że nastąpiło zakleszczenie i wycofuje się całą transakcję do której ona należy.

3.3 Replikacja

3.3.1 Cechy replikacji w MySQL

Replikacja w MySQL posiada dwie podstawowe cechy:

- asynchroniczność, co oznacza, że uaktualnianie danych w serwerach podległych następuje niezależnie od operacji wykonywanych w serwerze głównym.
- jednokierunkowość, oznaczająca, że zmiany w danych przekazywane są tylko od serwera głównego do podległych i nigdy w drugą stronę.

Cechy te są też przyczynami podstawowych różnic pomiędzy funkcjonowaniem systemu opartego na replikacji a bazującego na klastrze MySQL.

Pierwszą różnicą jest szybkość dokonywania modyfikacji danych. W rozwiązaniu klastrowym musi być ona wykonana na wszystkich replikach w trakcie transakcji, a w modelu bazującym tylko na replikacji zmiany wprowadzane są tylko na serwerze głównym.

Po drugie przy zastosowaniu mechanizmu rozdziału zapytań na wszystkie repliki uzyskuje się większą wydajność dużej liczby zapytań niż w wypadku klastra, gdzie wszystkie zapytania wykonywane są na tym samym zestawie węzłów danych (głównych replikach). W klastrze natomiast większa jest wydajność pojedynczego zapytania w wypadku gdy wymaga przeszukania dużej tablicy, gdyż jest ono wykonywane równolegle we wszystkich jej fragmentach.

Nie bez znaczenia też jest fakt centralizacji operacji modyfikujących w jednym serwerze. Pozwala to uniknąć problemów związanych z wykonywaniem rozproszonych transakcji.

Ceną jaką płaci się za wskazane wyżej zalety replikacji, jest możliwość odczytu nieaktualnych danych w wypadku korzystania z serwera podrzędnego, co w niektórych zastosowaniach może wykluczyć rozwiązanie z rozdziałem zapytań. Dodatkowo w wypadku awarii serwera głównego ryzykuje się utratą większej ilości danych niż w przypadku klastra, oraz dłuższym czasem niedostępności usługi, związanym z czasem potrzebnym na przełączenie na jeden z serwerów podrzędnych.

Warta wspomnienia jest też możliwość replikowania klastra, przy czym docelowy serwer podrzędny nie musi być klastrem MySQL.

3.3.2 Realizacja replikacji

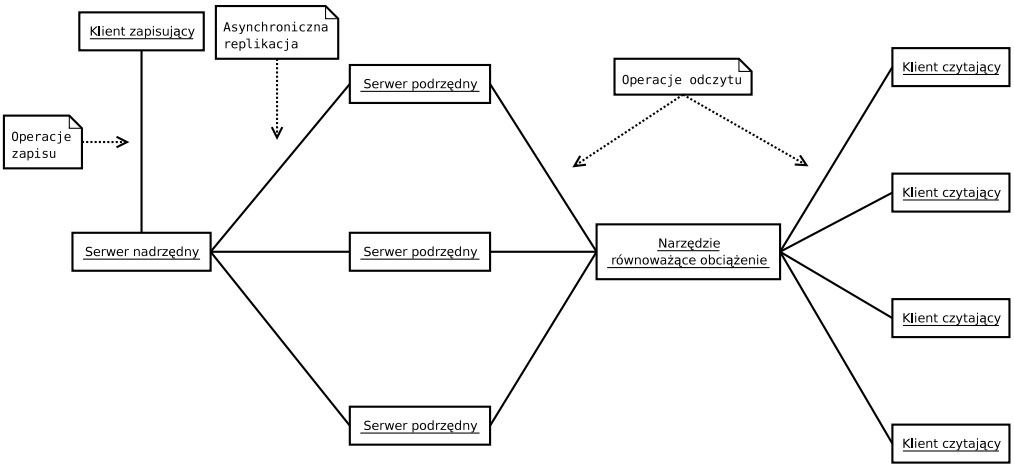
Replikacja jest zorganizowana zgodnie z zasadą „serwer nadrzędny – serwer podrzędny" (master-slave). Oznacza to, że jeden z serwerów w systemie jest wyznaczony jako nadrzędny i to w nim dokonywana jest modyfikacja danych a pozostałe (serwery podrzędne) starają się utrzymywać kopię danych znajdujących się na serwerze nadrzędnym.

Dodatkowo w SZBD MySQL replikacja jest zorganizowana tak, że serwer nadrzędny nie musi być świadomy ilości ani stanu serwerów podrzędnych, co zwiększa elastyczność tego rozwiązania (np. można dodawać nowe serwery podrzędne w trakcie działania systemu).

Replikacja może być przeprowadzana w dwóch trybach. Różnią się one postacią informacji zapisywanych w tak zwanym logu binarnym. Możliwe tryby to:

- wierszowy – w tym przypadku serwer nadrzędny zapisuje w logu jak zmieniają się poszczególne wiersze tabel.
- wyrażeniowy – zapisywane są całe wyrażenia modyfikujące.

Sposób przeprowadzania replikacji może być zmieniany w trakcie działania systemu. Dodatkowo serwery podrzędne mogą automatycznie zmienić swój tryb pracy w odpowiedzi na zmianę w serwerze nadrzędnym.



Rysunek 2: Prosty schemat replikacji.

Zastosowanie właściwego trybu ma duże znaczenie dla wydajności. W przypadku operacji modyfikujących znaczną liczbę wierszy bardziej opłacalny jest tryb wyrażeniowy, natomiast gdy wyrażenia są skomplikowane, ale modyfikują niewielką ilość wierszy korzystniejszy jest tryb wierszowy.

Istotnym szczegółem związanym z realizacją replikacji jest konieczność „ręcznego" przeniesienia danych do nowodołączanych węzłów podrzędnych. Wynika to z organizacji logu na serwerze nadrzędnym, który przechowuje tylko nowe zmiany (dokładnej jest zorganizowany w pierścień, tj. po osiągnięciu ustalonego rozmiaru zastępowane są najstarsze wpisy). Operacja ta może być wykonana albo poprzez sieć łączącą serwer z serwerem nadrzędnym, albo z wykorzystaniem kopii zapasowych.

3.4 Partycjonowanie

W MySQL dostępne jest partycjonowanie horyzontalne, co oznacza, że poszczególne wiersze tablicy mogą należeć do innych partycji. Jest one zaimplementowane odrębnie od mechanizmów storage engine, co pozwala stosować je w połączeniu z dowolnym z nich (oczywiście w zakresie ograniczonym zdrowym rozsądkiem – np. nie ma sensu stosowanie partycjonowania i NDB).

Podział na partycje wykonywany jest według funkcji partycjonującej, wybranej przez użytkownika spośród następujących: moduł, funkcja haszująca, dopasowanie do zakresów lub zbioru wartości. Dodatkowo użytkownik może określić wyrażenie dla którego wejściem jest wiersz tablicy, przekazywane do wybranej funkcji. Na to wyrażenie narzucony jest zbiór ograniczeń:

- musi być deterministyczne,
- nie może zwracać wartości stałej,
- nie może zawierać zapytań.

Poszczególne partycje mogą znajdować się w dowolnym miejscu drzewa katalogów, w szczególności na odrębnych nośnikach.

Stosowanie partycjonowania nie wpływa znacząco na bezpieczeństwo danych, choć oczywiście rozdzielenie tablicy na kilka dysków może ograniczyć skutki awarii do utraty tylko części wierszy.

Możliwe jest natomiast uzyskanie poprawy wydajności. Przy odpowiednim zdefiniowaniu podziału poprawia się wydajność zapytań, ze względu na następujące czynniki:

- zrównoleglenie operacji dyskowych,
- rezygnacje z przeszukiwania części partycji na podstawie warunku zapytania.
- zrównoleglenie wykonania zapytań zawierających funkcje agregujące.

Warto też zauważyć, że w rozważanej serii MySQL (5.1) mechanizm partycjonowania przechodzi znaczące zmiany i sami autorzy ostrzegają, że nie jest on jeszcze gotowy do praktycznych zastosowań.

3.5 Słowniczek pojęć

Klaster MySQL określenie na kompletne rozwiązanie klastrowe MySQL, obejmuje sobą klaster NDB oraz węzły SQL i zarządzające.

Klaster NDB zbiór węzłów danych (procesów ndbd) przechowujący i przetwarzający dane w sposób rozproszony. Może funkcjonować samodzielnie lub jako składnik klastra MySQL – dostępny jest wtedy przez NDBAPI.

Węzeł(node) w kontekście MySQL jest to proces ndbd, mysqld lub ndb_mgm. Jeżeli na komputerze działają np. dwa procesy ndbd to każdy z nich jest odrębnym węzłem.

Grupa węzłów(node group) w klastrze NDB zbiór węzłów danych przechowujących te same fragmenty tablic.

NDBAPI zbiór obiektów zebrany w bibliotekę, umożliwiający dostęp do klastra NDB. Może być wykorzystywany bezpośrednio przez aplikację, z pominięciem serwera MySQL.

Główna replika(primary replica) w klastrze NDB jest to węzeł danych do którego kierowane są wszystkie zapytania dotyczące danych przechowywanych w grupie do której należy. W wypadku jego awarii automatycznie wyznaczany jest następca.

Arbiter w klastrze MySQL, węzeł zarządzający lub SQL odpowiedzialny, w sytuacji podziału klastra, za rozstrzyganie która cześć ma prawo dalej funkcjonować, gdy jest możliwe klika alternatyw.

Serwer nadrzędny(master) w replikacji jest to serwer z którego dane są kopiowane. Musi być świadomy uczestnictwa w procesie replikacji (aby prowadzić log binarny), lecz nie musi znać stanu ani liczby serwerów podrzędnych.

Serwer podrzędny(slave) w replikacji jest to serwer przechowujący kopie danych serwera nadrzędnego. Odpowiada za asynchroniczne pobieranie zmian od serwera nadrzędnego i uaktualnianie swojej wersji.

3.6 Źródła

- Dokumentacja MySQL 5.1 ze strony www.mysql.org
- „MySQL Cluster technical whitepaper", dostępny po zamówieniu przez e-mail.
- Kod źródłowy wraz z zawartymi w nim komentarzami wersji 5.1.7 MySQL.
- Archiwa list dyskusyjnych „MySQL Cluster" i „MySQL Internals"

4 Opis Subversion

Opracował: Grzegorz Lepionka

4.1 Wstęp

Subversion jest systemem kontroli wersji. Głównym zadaniem systemu jest zarządzanie *repozytorium* z kolejnymi wersjami projektu. Zasadniczą różnicą między Subversion a zwykłym serwerem plików jest to, że Subversion pozwala zapamiętuje wszystkie zmiany wprowadzane od początku założenia projektu. Dzięki temu możliwe jest porównanie różnych wersji projektu albo przywrócenie poprzedniej wersji.

Subversion umożliwia cztery sposoby dostępu do repozytorium:

- *file* jeśli repozytorium znajduje się w lokalnym systemie plików wystarczy podać ścieżkę do niego
- *http* możliwy jest dostęp do repozytorium przez protokół http, jednym z serwerów który może udostępniać repozytoria Subversion jest Apache
- *svn* w skład pakietu Subverison wchodzi program svnserve, który jest serwerem subversion, służy do udostępniania repozytorium, wykorzystuje własny protokół
- *svn+ssh* podobny do svn tylko że wykorzystuje połączenia szyfrowane

Podstawowym celem powstania Subversion było poprawienia drobnych wad CVS. Celem twórców było utrzymanie możliwie dużego podobieństwa interfejsu do CVS. Dlatego używanie podstawowych funkcji Subversion jest bardzo podobne jak w CVS.

Funkcje oferowane przez Subversion:

- Wersjonowanie katalogów
  CVS śledzi tylko zmiany plików, w CVS przeniesienie plik do innego katalogu nie jest zapisywane. Subversion zapisuje również zmiany drzewa katalogów. Rejestrowane są takie operacje jak dodawania, usuwanie, kopiowanie i zmiana nazwy pliku i katalogu. Umożliwia to np. odzyskanie skasowanego pliku, z poprzedniej wersji projektu.
- Transakcyjne zapisywanie zmian
  Subversion gwarantuje że albo wszystkie zmiany zostaną zapisane albo zawartość repozytorium w ogóle się nie zmieni. Pozwala to dokonywać wielu zmian przed zapisaniem ich do repozytorium bez obawy że tylko część zostanie zapisana.
- Spójność danych
  Subversion wyznacza zmiany w plikach za pomocą jednego algorytmu zarówno dla plików binarnych jak i tekstowych. Oba rodzaje plików są przechowywane w formie skompresowanej w repozytorium.

4.2 Używanie Subversion

4.2.1 Problem z prawami

W czasie testowania konfiguracji repozytorium wystąpił problem z prawidłowym ustawianiem praw przy korzystaniu z repozytorium przez kilku użytkowników. Problem polega na tym że przy zapisie do repozytorium przez nowo tworzone pliki mają prawa do zapisu tylko dla użytkownika, przez którego zostały utworzone (grupa ma prawa tylko do odczytu). Rozwiązaniem tego problemu było napisanie skryptu, który wywołaniem svn zmienia domyślną maskę dla nowo tworzonych plików na 770 (umask 007). Skrypt umieszczony jest w katalogu /var/svn/rso4. Najwygodniejszym rozwiązaniem jest utworzenie w katalogu domowym pliku o nazwie .bash_profile (lub dopisanie do niego jeśli już istnieje) linii "export PATH=/var/svn/rso4:$PATH". Po dodaniu takiej linii do pliku i ponownym zalogowaniu można korzystać z Subversion wywołując po prostu svn.

4.2.2 Podstawowe polecenia

Klientem Subersion jest program *svn*. Wszystkie działania użytkownik wykonuje właśnie za pomocą tego programu, wywołując go z odpowiednimi opcjami. Aby rozpocząć używania Subversion wystarczy opanować klika poleceń svn. Zamieszczona ponieważ lista zawiera polecenia w formacie w jakim mają być wpisane w konsoli (łącznie z nazwą programu *snv*). Przykłady dotyczą repozytorium stworzonego na potrzeby projektu na klastrze openone. Powinny działać jeśli będą uruchamiane przez członków grupy rso4 na klastrze openone. Niektóre polecenia wymagają podania ścieżki do repozytorium, na którym mają działać. Na zamieszczonej poniżej liście poleceń ścieżka ta będzie symbolizowana przez *URL*.

Może ona mieć następujące postaci:

- file://ścieżka na dysku
- http://adres:port ścieżka
- svn://adres:port ścieżka
- svn+ssh://adres:port ścieżka

Jeśli polecenie nie wymaga podania URL (w spisie URL nie występuje jako argument wywołania), oznacza to że powinno być wykonane w katalogu z *lokalną kopią projektu*. Katalog taki jest tworzony przez polecenie svn checkout.

Lista poleceń:

- *svn checkout URL*
  Pobranie najnowszej wersji całego projektu z repozytorium. W katalogu z którego wywołano to polecenie zostanie utworzony katalog o nazwie takie jak nazwa katalogu z projektem w repozytorium. Nowo utworzony katalog zawiera *lokalną kopię projektu*
  Przykład: svn checkout file:///var/svn/rso4/projekt
- *svn info*
  Wypisuje informacje o projekcie

- *svn update*
  Uaktualnienie wersji projektu przez pobranie zmian z repozytorium. Różni się od svn checkout tym, że pobiera tylko różnice oraz musi być wykonane w katalogu z projektem. Po wykonaniu polecenia wypisywana jest lista zmian wykonanych na lokalnej wersji projektu. Opis kodów używanych do opisu zmian znajduje się w dalszej części rozdziału.

- *svn commit*
  Zapisanie zmian dokonanych lokalnie do repozytorium. Tworzy nową wersję projektu w repozytorium.

- *svn add ścieżka*
  Dodanie pliku lub katalogu do lokalnej wersji projektu. Aby zmiana została zapisana na serwerze musi zostać wykonane svn commit.
  Przykład: svn add main.c

- *svn delete ścieżka*
  Usunięcie pliku lub katalogu z lokalnej wersji projektu. Aby zmiana została zapisana na serwerze musi zostać wykonane svn commit.
  Przykład: svn delete oldsrc

- *svn copy źródło przeznaczenie*
  Skopiowanie pliku lub katalogu w lokalnej wersji projektu. Aby zmiana została zapisana na serwerze musi zostać wykonane svn commit.
  Przykład: svn copy plik1.c src

- *svn move źródło przeznaczenie*
  Przeniesienie pliku lub katalogu w lokalnej wersji projektu. Aby zmiana została zapisana na serwerze musi zostać wykonane svn commit.
  Przykład: svn move old.c new.c

- *svn log*
  Wyświetla dziennik zmian projektu

- *svn status*
  Wyświetla listę wykonanych zmian wykonanych na lokalnej kopi projektu od chwili ostatniej synchronizacji z repozytorium. Nie wyświetla zawartości zmienionych plików. Status poszczególnych plików i katalogów opisywany jest z pomocą jednoliterowych kodów. Opis kodów używanych do opisu statusu znajduje się w dalszej części rozdziału.

- *svn diff*
  Podobnie jak svn status wyświetla zmiany w porównaniu z pobraną wersją projektu. Różni się tym, że wyświetla zawartość zmienioną w plikach. Wyjście jest wyświetlane w formacie diff. Może zostać użyte do patchowania źródeł za pomocą programu patch.
  Przykład: svn diff $>$ patchfile

- *svn revert*
  Odwraca wszystkie zmiany dokonane na lokalnej kopii projektu.

- *svn cleanup*
  Należy wykonać jeśli otrzymamy komunikat że część projektu jest zablokowana (”locked”), jeśli svn status wypisuje jakieś pliki poprzedzone literą L

4.2.3 Kody svn update

Znaczenie jednoliterowych kodów opisujących zmiany w lokalnej wersji projektu wykonane przez svn update:

- A - dodane (ang. *Added*)
- D - usunięte (ang. *Deleted*)
- U - uaktualnione (ang. *Updated*)
- C - w konflikcie (ang. *Conflict*)
- G - połączone (ang. *merGed*)

4.2.4 Kody svn status

Znaczenie jednoliterowych kodów opisujących zmiany wykonane na lokalnej wersji projektu od chwili ostatniej aktualizacji:

- A - do dodania (ang. *Addition*)
- D - do usunięcia (ang. *Deletion*)
- M - zmodyfikowane (ang. *Modified*)
- R - zamienione (ang. *Replaced*)
- C - w konflikcie (ang. *Conflict*)
- I - ignorowane (ang. *Ignored*)
- ? - nie znajduje się pod kontrolą Subversion, np. utworzony ale nie dodany do repozytorium za pomocą svn add
- ! - brakuje pliku. np. usunięty lub przeniesiony w systemie plików ale nie wykonano odpowiednio svn delete lub svn move
- ~ - zmienił się typ obiektu np. w miejsce katalogu stworzono plik o identycznej nazwie
- * - nowsza wersja obiektu istnieje w repozytorium serwera

4.3 Konflikt

Konflikt może wystąpić przy próbie zapisania zmian wykonanych na lokalnej kopii do repozytorium. Jeśli repozytorium zostało zmienione od pobraniem kopii, na której wykonane były zmiany, system sprawdza czego dotyczyły zmiany. Konflikt występuje gdy zmiany dotyczyły tych samych linii, w którymś ze zmienianych plików. Występowanie konfliktu chroni przed omyłkowym usunięciem czyichś zmian. Komunikat otrzymywany gdy wystąpi konflikt.

svn: Zatwierdzenie nie powiodło się (szczegóły poniżej):
svn: Nieaktualne: '/trunk/main.c' w transakcji z '3-1'

Jeśli wystąpił konflikt należy wykonać svn update, który:
- zmienia plik w którym wystąpił konflikt

- dodaje pliki z rozszerzeniami .mine, .rX, .rY

W pliku w którym wystąpił konflikt zmiany opisywane są w następujący sposób
<<<<<<< .mine
W tym miejscu wpisywane są zmiany dokonane na lokalnej kopi
=======
W tym miejscu wpisywane są zmiany dokonane przez kogoś na serwerze
>>>>>>> .rX
gdzie X - numer aktualnej wersji na serwerze

Znaczenie rozszerzeń plików:

- *.mine - plik zawierający tylko zmiany wprowadzone lokalnie
- *.rX - plik przed zmianami lokalnymi (X - numer wersji poddawanej edycji lokalnie)
- *.rY - aktualny plik na serwerze (Y - numer aktualnej wersji na serwerze)

4.4 Dodatkowe opcje

Program svn oprócz komendy, którą ma wykonać może przyjmować również dodatkowe opcjonalne parametry. Komenda musi jest parametrem obowiązkowym. Najważniejsze komendy zostały opisane w rozdziale "Podstawowe polecenia". W tym rozdziale zostaną opisane najbardziej przydatne dodatkowe parametry wywołania. Ważniejsze opcje wywołania:

- -v (–verbose)
  przydatna zwłaszcza przy svn status i svn log
- -h (–help)
  może być podana bez argumentów do wypisania listy poleceń jeśli podamy nazwę polecenia jako parametr to wypisze pomoc do polecenia
  Przykład: svn -h commit
- -m (–message)
  przy commit pozwala podać opis zmian w linii poleceń (w przeciwnym wypadku trzeba edytorze)
  Przykład: svn commit -m "Poprawiona funkcja sin()"
- -r (–revision)
  Jest zdecydowanie najważniejszym dodatkowym parametrem.
  Pozwala określić wersje (lub zakres wersji), na których ma działać polecenie. Wersja może być podana w postaci numeru lub daty, z przed której ma pochodzić zakres wersji podaje się oddzielając jego krańce ':' HEAD oznacza aktualna wersje

  Przyklady:
  svn log -r 1729
  svn log -r 1729:HEAD
  svn log -r 1729:1744
  svn log -r {2001-12-04}:{2002-02-17}
  svn log -r 1729:{2002-02-17}

Inne przydatne wywołania z opcją -r:

- *svn diff -r 1730:1700* -wypisze różnice między wersjami 1730:1700 projektu
- *svn update -r {2002-02-17}* -pobranie najnowszej wersji pochodzącej z przed 2002-02-17

Odzyskanie poprzedniej jednego pliku z wersji 1700 uzyskamy przez wydanie dwóch komend:

svn del filename
svn -r 1700 copy URL/filename filename

5 Wprowadzenie do rozwiązania OpenSSI

Opracował: Marcin Rudowski

5.1 Wstęp

Klaster składa się ze zbioru komputerów (węzłów) połączonych szybką siecią komputerową. Główną idea budowy takich systemów jest m.in. zapewnienie skalowalności, zwiększenie wydajności lub zapewnienie podwyższonej dostępności. W zależności od zastosowania różny może być nacisk na powyższe wymagania odnośnie systemu obsługującego klaster.

W rozwiązaniach typu SSI (Single System Image) użytkownik widzi cały klaster jako jedną dużą maszynę zarządzaną i użytkowaną tak jak by to był pojedynczy komputer. Takie rozwiązanie pozwala w łatwy sposób uruchamiać istniejące oprogramowanie, które nie było projektowane z myślą o klastrze. Ponieważ całość składa się z grupy fizycznych serwerów, nie jest możliwe zapewnienie pełnej przezroczystości dla użytkownika.

5.2 Rozwiązanie klastrowe OpenSSI

5.2.1 System plików

W systemie OpenSSI system plików jest wspólny i zmiany są automatycznie widoczne na pozostałych węzłach klastra. Urządzenia danego węzła są dostępne poprzez odpowiednie wpisy /dev w lokalnym systemie plików. Urządzenia innych węzłów niż aktualny są dostępne w odpowiednio numerowanych podkatalogach wewnątrz /dev zapewniając tym samym dostęp do zdalnych urządzeń.

Pamięć typu Swap nie jest współdzielona w obrębie klastra i każdy węzeł posiada własną pamięć tego typu.

5.2.2 Procesy

Ważną cechą OpenSSI jest udostępnianie jednolitego obrazu procesów uruchomionych w systemie. Osiągnięto to poprzez nadawanie unikalnych numerów procesów (pid) i zapewnianie przezroczystości w zarządzaniu procesami przy użyciu tradycyjnych narzędzi i funkcji systemowych (zgodnie z ideą SSI).

Użytkownicy, administratorzy czy procesy mają taki sam dostęp do wszystkich procesów jakby były uruchomione na jednej dużej maszynie. Procesy mogą być uruchamianie na innych węzłach, a nawet podlegać migracji w trakcie działania bez wpływu na sposób działania aplikacji. W razie awarii węzła utracone zostają jedynie procesy na nim uruchomione, oraz aktualnie podlegające migracji obejmujących usunięty węzeł, dzięki czemu cały system może dalej funkcjonować, mimo awarii niektórych komputerów składowych

5.2.3 Programowanie aplikacji

Dzięki idei SSI dowolny program napisany dla pojedynczego systemu może być uruchomiony bez dodawania specyficznych wywołań. Jednakże możliwe jest użycie specyficznych dla klastra funkcji z biblioteku libcluster.so, co umożliwia m.in.:

- uruchamianie procesów na danych węzłach

- migrację
- pobranie informacji o węzłach w klastrze
- uzyskanie informacji o przynależności do węzła

5.2.4 Komunikacja

W klastrze OpenSSI dostępne są tradycyjnie używane mechanizmy komunikacji między procesami (IPC) znane w systemie UNIX. Aplikacje mogą również skorzystać z sieci wewnętrznej łączącej węzły.

Komunikację sieciową możemy podzielić na dwie kategorie:

- komunikacja wewnątrz klastra - do zapewnienia SSI oraz do komunikacji między procesami
- zewnętrzny interfejs sieciowy całego klastra - udostępnia adres sieciowy tak, aby klaster był widoczny na zewnątrz jako pojedyncza maszyna

Aby zapewnić obraz jednolitego systemu (SSI) jądra węzłów komunikują się ze sobą szybką siecią wewnątrz klastra. Oprócz tego, aby zapewnić, że klaster widziany jest z zewnątrz jako pojedynczy, wysokodostępny system, istnieje CVIP (Cluster Virtual IP), który jest aliasem do zewnętrznej karty sieciowej. CVIP może być przypisany innemu węzłowi, jeśli aktualnie przypisany opuszcza klaster.

5.3 Metody komunikacji

Główną zaletą systemów typu SSI jest udostępnianie standardowych mechanizmów komunikacji znanych w systemach UNIX opartych na jednej maszynie. Dzięki temu nie trzeba modyfikować istniejących aplikacji by móc je uruchomić na klastrze.

5.3.1 IPC

Rozproszona komunikacja między procesami wewnątrz klastra może być przeprowadzona w oparciu o tradycyjne obiekty IPC:

- potoki
- kolejki FIFO
- sygnały
- kolejki wiadomości
- semafory
- pamięć dzieloną
- gniazda (Unix Domain)

Obiekty IPC tworzone są na węźle, na którym był proces je tworzący, ale mogą być używane przez procesy na dowolnych węzłach w obrębie klastra. Dzięki temu procesy używają standardowych metod komunikacji bez ”wiedzyó klastrowej naturze systemu, na którym są uruchomione.

Obiekty IPC nie podlegają jednak migracji, tak jak jest to w przypadku procesów, co może mieć wpływ na wydajność komunikacji w zależności od migracji i położenia procesów. Najlepsze rezultaty uzyskuje się, jeśli obiekt IPC został utworzony na węźle na którym znajduje się proces z niego korzystający.

5.3.2 Sieć wewnątrz klastra

Oprócz IPC procesy mogą wykorzystywać do komunikacji wewnętrzną sieć łączącą węzły. W tym przypadku jednak, procesy używające gniazd Internet Domain nie podlegają migracji. Migracja taka wiązała by się z przenoszeniem adresu IP i wieloma komplikacjami i nie została zaimplementowana w OpenSSI właśnie ze względu na narzuty wydajnościowe.

5.4 Zarządzanie procesami

System OpenSSI zapewnia uruchamianie i migracje procesów między węzłami przezroczyście dla aplikacji i użytkowników. Zapewnione zostało:

- unikalne w obrębie klastra numery identyfikacyjne procesów (PID)
- dostęp do procesu z dowolnego węzła
- rozproszone relacje między procesami (parent-child, grupy procesów)
- migracje procesów w trakcie wykonywania
- możliwość kontynuacji procesu mimo awarii węzła na którym został utworzony
- utrzymanie relacji niezależnie od awarii dowolnego węzła
- wspólna struktura /proc dla wszystkich węzłów

5.4.1 Unikalne PID

Aby umożliwić migrację procesów między węzłami należało zapewnić unikalność identyfikatorów. Uzyskano to przydzielając węzłom rozłączne pule numerów, które mogą używać przy tworzeniu procesów. Uprościło to również śledzenie procesów, gdyż przeszukiwany jest węzeł zarządzający danym zakresem PID. W razie awarii węzła, obsługę śledzenia procesów na nim utworzonych przejmuje węzeł zastępczy.

5.4.2 Struktura /proc

Podobnie jak w tradycyjnym systemie UNIX, w katalogu /proc znajduje się szereg katalogów i plików związanych z zarządzaniem procesami. W przypadku OpenSSI katalogi z numerami PID są jednak wspólne dla całego klastra, jednak nie oznacza to, że każdy węzeł widzi wszystkie i aktualne katalogi wszystkich procesów.

Dodatkowo w strukturze plików /proc zostały dodane wpisy specyficzne dla rozwiązania klastrowego:

- cluster/ - zawiera informacje o całym klastrze, np.:
- events - do pobierania powiadomień o zdarzeniach (przynależność węzłów do klastra)

- lvs /ip_vs_.. - związane z zapewnieniem aliasu IP klastra
- loadlevellist - lista aplikacji podlegających automatycznej migracji
- icsstat - statystyki dotyczące struktur komunikacji między jądrami
- node* - kotrola obciążenia danego węzła w klastrze
- PID - informacje o procesie PID, m.in. o:
- where - węzeł na którym proces się wykonuje
- loadlevel - czy podlega równoważeniu obciążenia
- goto - używany do migracji (poprzez zapisanie numeru docelowego węzła)

5.4.3 Biblioteka libcluster.so

Oprócz dostępu poprzez strukturę /proc, procesy mogą skorzystać z wywołań bibliotecznych charakterystycznych dla klastra, np.:

- 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_ssiconfig() - 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)

5.4.4 Usługi niezawodne

W systemie OpenSSI możliwe jest wykorzystanie demona keepalive do zapewnienia ciągłości usługi. Keepalive monitoruje zarejestrowane procesy i w razie awarii któregoś z nich, uruchamia skrypt restartujący zatrzymaną usługę. Interfejsem do demona keepalive jest spawndeamon, który jednak wymaga odpowiednich uprawnień do konfiguracji (zapis do odpowiednich plików konfiguracyjnych).

Aby móc przywrócić działanie usługi po awarii jednego z węzłów bez uprawnień root'a, należy zaproponować własną implementację funkcjonalności demona keepalive. Wymagałoby to wykrywania aktywności procesów oraz uruchamianie ponownie zatrzymanego procesu na innym wolnym węźle klastra. Również uzyskanie informacji o dołączeniu nowego węzła do klastra (poprzez cluster_events_register_signal()) może zostać wykorzystana do uruchomienia nowego procesu i tym samym przywrócenie pełnej wydajności po przywróceniu węzłów które uległy awarii.

Do wykrycia niedostępności procesu można wykorzystać potoki, gdyż udostępniają informację o zakończeniu połączenia, co może oznaczać awarię węzła lub zatrzymanie procesu, z którym przeprowadzana była komunikacja.

Jeden z procesów na węzłach klastra pełnił by dodatkowo funkcję podobną do keepalive i nadzorował dostępność pozostałych procesów realizujących implementowaną usługę. W przypadku awarii, proces nadzorczy odpowiedzialny byłby za ponowne uruchomienie procesu na wolnym węźle. Mógłby również czekać na zdarzenia informujące o nowych węzłach w klastrze i zapewniając uruchomienie nowych procesów, jeśli istniejąca pula byłaby mniejsza niż ustawiona w konfiguracji. W przypadku awarii węzła z procesem nadzorującym, należało by przeprowadzić elekcję wśród pozostałych procesów realizujących usługę.

6 Wprowadzenie do zagadnienia realizacji niezawodności i wydajności poprzez replikację zasobów

Opracował: Maciek Nowak

6.1 Wstęp

Wzrastająca moc obliczeniowa komputerów i szybkość sieci umożliwia zastąpienie jednego superkomputera klastrem złożonym z tradycyjnych komputerów, zmniejszając koszty osiągnięcia wysokiej wydajności. Niestety rozproszony klaster jest podatny na częściowe uszkodzenia sprzętu. Prawdopodobieństwo że cały klaster nie będzie działał wynosi p^n gdzie p - prawdopodobieństwo awarii pojedynczego komputera, n - ilość komputerów w sieci. Działanie klastra komputerów musi być wspierane przez rozwiązania zapewniające wysoką dostępność. Rozwiązaniem zapewniającym wysoką dostępność a jednocześnie zwiększającą efektywność jest replikacja.

Cele stosowania replikacji:

- *zwiększenie dostępności* danych/systemu . awaria pojedynczego komputera nie powoduje konieczności wyłączenia usług,
- *zwiększenie efektywności* możliwe dzięki podziale pracy pomiędzy wiele komputerów, umożliwienie przetwarzania współbieżnego.

Zastosowanie replikacji powoduje także problemy:

- synchronizacja i spójność danych,
- przydział zadań, przejmowanie obsługi w przypadku uszkodzenia komputera w klastrze.

Rozwiązanie powyższych problemów jest zadaniem nietrywialnym, powodującym wysokie koszty implementacji.

6.2 Rodzaje replikacji

Replikacja może być realizowana przy użyciu różnych strategii:

6.2.1 replikacja w architekturze klient/serwer

Trywialny przypadek. Najprostsza w implementacji forma replikacji: istnieje jedna kopia główna „serwer", pozostałe „kopie" pełnią funkcję „klientów - pośredników". Operacje odczytu i zapisu danych są przekierowywane przez repliki „klienty" do repliki „serwer". Awaria repliki „serwer" powoduje awarię całego klastra.

Zalety:

- trywialne w implementacji.

Wady:

- rozwiązanie mało wydajne,
- nieodporne na wyłączenie komputera serwera,
- nieodporne na problemy z połączeniem do serwera.

Propozycje algorytmów potrzebnych do zaimplementowania replikacji w architekturze klient/serwer:

- algorytmy blokowania dostępu do danych w czasie modyfikacji.

6.2.2 replikacja pasywna

W replikacji pasywnej są wykorzystywane repliki: główna (ang. *primary*) i zapasowe (ang. *backup*). Przetwarzanie transakcji wymagających tylko i wyłącznie operacji odczytu odbywa się lokalnie - w ramach wybranej repliki zapasowej. Natomiast operacje zapisu - modyfikacje danych - są przekazywane do repliki głównej a następnie rozpropagowane do replik zapasowych.

Zalety:

- proste w implementacji,
- szybka, jeśli główną operacją jest odczyt danych,
- skalowalne,
- rozwiązanie będzie działać nawet w przypadku wyłączenia komputera z repliką główną (inna replika może przejąć jej rolę).

Wady:

- rozwiązanie mało wydajne w przypadku gdy głównymi operacjami są operacje zapisu/aktualizacji danych. Jest gorsze (pod względem wydajności) od rozwiązania replikacji w architekturze klient/serwer,
- występuje konieczność synchronizacji zmienionych danych między replikami serwera, a wszystkimi replikami klienckimi, powoduje to znaczne chwilowe obciążenie sieci (wszystkie repliki „klienty" muszą zostać jak najszybciej zaktualizowane - otrzymać dane z jednego źródła . serwera,
- nieodporne na upadki komputera z repliką „serwer", o ile nie zaimplementowano algorytmów umożliwiających dynamiczne przełączanie ról,

Propozycje algorytmów potrzebnych do zaimplementowania replikacji w architekturze klient/serwer:

- algorytm elekcji - wybór repliki pełniącej funkcję serwera
- algorytmy blokowania dostępu do danych w czasie ich modyfikacji.

6.2.3 aktywna replikacja

W replikacji aktywnej każda replika jest równoważna. Operacje zapisu i odczytu są wykonywane lokalnie na każdej replice. W celu zachowania spójności danych operacje są rozpropagowywane bezpośrednio (nie występuje nikt 'uprzywilejowany) do innych replik

Zalety:

- nie ma uprzywilejowanej repliki - wyłączenie jednego lub więcej komputerów spowoduje co najwyżej upadek transakcji którą aktualnie były przeważana, natomiast nie będzie miało wpływu na pracę reszty klastra,

- wydajne w operacjach odczytu,
- wydajniejsze od replikacji pasywnej i klient/serwer przy modyfikacji danych ze względu na rozłożenie obciążenia sieci oraz (dla rozłącznych danych) możliwość przetwarzania równoległego,
- rozwiązanie odporne na wyłączenie grupy komputerów . replik lub utratę połączenia między grupami - grupa w której jest ponad połowa komputerów może kontynuować pracę.

Wady:

- trudne w implementacji ze względu na konieczność zachowania spójności danych oraz blokowania dostępu w trakcie ich modyfikacji.

6.3 Strategie replikacji

Niezależnie od rodzaju replikacji omówionego w poprzednim rozdziale możemy wyróżnić następujące strategie replikacji:

6.3.1 replikacja pesymistyczna

Polega na blokowaniu dostępu do replik modyfikowanych danych.

Zalety:

- zachowuje spójność danych.

Wady:

- ogranicza dostępność do danych (gdy prowadzony jest odczyt),
- wymaga lepszej komunikacji między komputerami zawierającymi repliki danych,
- ograniczona skalowalność.

6.3.2 replikacja optymistyczna

Opiera się na założeniu że nie zawsze występuje spójność danych między poszczególnymi replikami.

Zalety:

- te same dane mogą być równolegle odczytywane i zapisywane - są bardziej dostępne,
- połączenie między komputerami nie musi być trwałe,
- bardziej skalowalna niż replikacja pesymistyczna.

Wady:

- możliwość równoległego zapisu tych samych danych w różnych węzłach - pojawia się konieczność rozwiązywania konfliktów/ustalania danych ostatecznych,
- nie gwarantuje ciągłej spójności danych.

Często wykorzystuje się strategie „hybrydowe”.

rezygnuje z wysokiej dostępności na rzecz spójności danych. Jednocześnie wymaga dobrej jakości połączeń sieciowych w celu synchronizacji.

Literatura

[BCS05] C. Michael Pilato Ben Collins-Sussman, Brian W. Fitzpatrick. *Version Control with Subversion*. 2005. http://svnbook.red-bean.com/en/1.1/svnbook.pdf.

[mysa] *Dokumentacja MySQL 5.1*. www.mysql.org.

[mysb] *MySQL Cluster technical whitepaper*.

Podsumowanie II fazy projektu z RSO.

Grupa RSO4

Andrzej Grudzień
Grzegorz Lepionka
Maciek Nowak
Marcin Rudowski
Paweł Słupczyński

16 maja 2006

1 Wstęp

Niniejszy dokument zawiera dokumentację końcowego projektu, opis modułów i podział zadań drugiej części projektu z przedmiotu RSO.

2 Główne pomysły

W toku dyskusji nad możliwymi rozwiązaniami wyklarowały się dwa, które uznaliśmy za godne dalszego rozważenia:

- Rozwiązanie z warstwą pośredniczącą w PSEA (podobne do rozwiązania NDB)
- Pomysł punktu centralnego i zespołu nakładek jako warstwy pośredniczącej między klientem a serwerem bazy MySQL.

Po długich naradach zostało wybrane rozwiązanie z punktem centralnym i zespołem nakładek.

3 Opis wybranego rozwiązania

3.1 Zarys ogólny

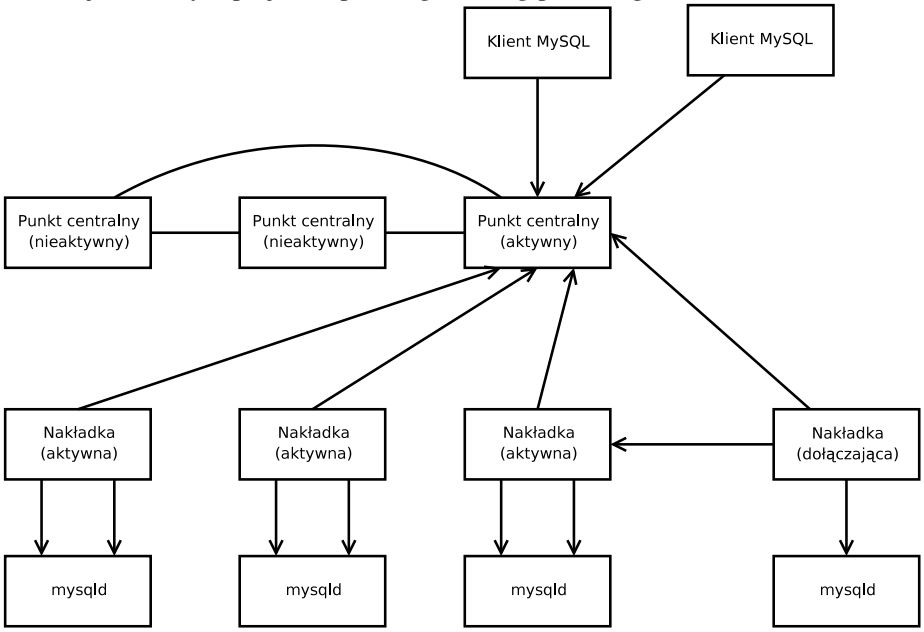
Pomysł polega na wprowadzeniu warstwy pośredniej pomiędzy klienta a serwer MySQL. Warstwa ta składałaby się z punktu centralnego, z którym komunikowałby się klient, oraz sieci nakładek połączonych z silnikami baz danych. Zdefiniowano system o następujących cechach:

- replikacja pełna, synchroniczna,
- journaling jako sposób kontroli spójności bazy,
- wszelkie operacje (klienta) wykonywane przez punkt centralny,
- każdy serwer MySQL ma przypisana do niego dokładnie jedną nakładkę,

Proponowane rozwiązanie nie ingeruje w żaden sposób w kod silnika bazy.

3.2 Komunikacja

Komunikacja w naszym projekcie przebiega według poniższego schematu:



Przekazywanie danych w punktach centralnych i nakładkach jest realizowane w jednym wątku w sposób synchroniczny. Zastosowano model oparty na zdarzeniach generowanych przez odczyty lub błędy i zlecaniu zapisów, które są realizowane w dogodnym momencie.

3.2.1 Komunikacja klient MySQL – punkt centralny

W momencie poprawnego połączenia się klienta punkt centralny przesyła do wszystkich aktywnych nakładek polecenie otwarcia połączenia do serwera mysqld w imieniu nowego klienta.

W standardowej instalacji MySQL komunikacja między klientem a serwerem odbywa się według ustalonego protokołu opartego na pakietach.

Każdy pakiet opatrzony jest czterobajtowym nagłówkiem. Pierwsze trzy bajty zawierają długośc właściwej treści pakietu. Czwarty bajt to numer kolejny pakietu w dialogu.

Dodatkowo w wypadku pakietów wysyłanych przez klienta pierwszy bajt właściwej wiadomości zawiera informacje o typie polecenia.

W naszym projekcie wykorzystujemy tą budowę komunikatów w celu uniknięcia zbędnego buforowania. W zależności od typu polecenia stosuje się następujące reguły:

1. w wypadku zapytań (typ QUERY) buforuje się nagłówek pakietu oraz dostateczną ilość danych by możlwie było określenie charakteru polecenia. W praktyce oznacza to wykrycie w zgromadzonych danych jednego ze słów kluczowych SQL: insert, select, etc. Ta informacja wystarcza by wyznaczyć punkt docelowy dla pakietu, zatem dalsze fragmenty pakietu są przekazywane w miarę nadchodzenia bez buforowania.

2. dla pozostałych typów poleceń do wyznaczenia punktu lub punktów docelowych wystarcza znajomość nagłówka.

Jak już zostało wspomniane na tym etapie wykonywane jest także wyznaczanie punktów docelowych dla pakietu. Ze względu na przyjęte założenie o utrzymywaniu otwartego połączenia między nakładką a mysqld dla każdej nakładki dla każdego klienta, jedynie polecenia będące odczytami są kierowane do pojedyńczych nakładek. Pozostałe są rozsyłane do wszystkich.

Polecenie zakończenia pracy serwera są ignorowane.

Odpowiedzi do klienta przesyłane są na bierząco bez buforowania i analizy.

3.2.2 Komunikacja punkt centralny – nakładka

Przy komunikacji punktu centralnego z nakładką konieczne było rozszerzenie standardowego protokołu używanego w komunikacji klient-serwer w MySQL. Nagłówek został wzbogacony o czterobajtowe pole długości podające długość oryginalnego pakietu (włącznie z jego nagłówkiem) i czterobajtowy identyfikator nadawcy. Nadawcą może być jeden z klientów lub punkt centralny.

Modyfikacje te wynikają z potrzeby przetransportowania przez jedno łącze poleceń i odpowiedzi dla wielu klientów.

Szczególnym przypadkiem komunikacji punkt centralny – nakładka są zgłaszanie się nowouruchomionej nakładki w celu dołączenia do klastra i powiadamianie nakładki przez nowowybrany aktywny punkt centralny o uzyskaniu tego stanu.

W pierwszym przypadku punkt centralny podaje nowoprzybyłej nakładce lokalizacje aktywnej nakładki posiadającej aktualną bazę danych i oczekuje na informacje o gotowości do przyjmowania zleceń.

W drugim aktywowany punkt kontrolny informuje wszystkie aktywne nakładki o fakcie zmiany aktywnego punktu kontrolnego oraz o konieczności zgłoszenia się do niego.

Oba te typy komunikacji odbywają się w ramach odrębnych połączeń od służących do transmisji poleceń i danych. Połączenia te są zamykane od razu po zakończeniu przesyłania danych.

3.2.3 Komunikacja punkt centralny – punkt centralny

Komunikacja między punktami centralnymi służy trzem celom:

1. wzajemnej kontroli istnienia.
2. wyborowi nowego aktywnego punktu centalnego w wypadku awarii poprzedniego.
3. przekazywaniu wiadomości o aktywnych nakładkach

Pierwszy cel realizowany jest poprzez wykrywanie awarii łącza oraz mechanizm „heartbeat" czyli okresowe wysyłanie małych porcji danych. W wypadku wykrycia awarii punkt centralny rozsyła informacje o tym fakcie do pozostałych punktów. W wypadku gdy awarii uległ aktywny punkt centralny powoduje to także rozpoczęcie procedury wyznaczania następcy.

Drugi cel osiągany jest z wykorzystaniem algorytmu „tyrana". Każdy z puntków centralnych rozsyła do pozostałych swój numer identyfikacyjny uzyskany przy starcie. Posiadacz najniższego ID uznawany jest za najsilniejszego i informuje nakładki o swojej nowej roli.

Trzecie zadanie realizowane jest przez rozgłaszanie przez aktywny punkt centralny zgłoszeń nakładek i ich gotowości do pozostałych punktów centralnych.

3.2.4 Komunikacja nakładka – nakładka

Komunikacja między nakładkami zachodzi w przypadku dołączenia się do klastra nowego węzła. Po uzyskaniu od punktu centralnego informacji o lokalizacji nakładki z aktualnymi danymi nawiązuje z nią połączenie. Następnie następuje transfer poleceń brakujących nowej nakładce. Zakończeniem procesu jest zakończenie połączenia z nakładką-źródłem i poinformowanie punktu centralnego przez nową nakładkę o gotowości do świadczenia usług.

3.2.5 Komunikacja nakładka – serwer mysqld

Pomiędzy nakładką a serwerem mysqld utrzymywane jest jedno połączenie dla każdego klienta korzystającego z klastra. Jest ono otwierane i zamykane na rządanie punktu centralnego.

Transfer danych przebiega na zasadach zbliżonych do tych stosowanych w komunikacji między klientem a punktem centralnym, czyli minimalizujących buforowanie do niezbędnego minimum. Różnica polega na wprowadzeniu analizy odpowiedzi serwera mysqld przed przekazaniem ich dalej. Wynika to z konieczności określania przez nakładkę statusu zakończenia operacji modyfikujących bazę.

3.3 Spójność bazy

3.3.1 Obsługa utraty spójności danych

Utrata spójności danych pomiędzy różnymi replikami może wystąpić w następujących sytuacjach

- dołączenie się nowego hosta do klastra, baza dołączającego się hosta jest pusta ponieważ nie był on wcześniej w klastrze. Należy wysłać mu zrzut całej bazy z jednych ze stacji posiadających aktualną kopię.
- dołączanie się hosta po awarii. Przez awarię hosta rozumiany jest stan, w którym nie może on aktualizować swojej bazy danych. Obejmuje to takie sytuacje jak (awarię łącza pomiędzy hostem a punktem centralnym, zaprzestanie działania któregoś z lokalnego serwera bazy danych lub procesu pośredniczącego w komunikacji między punktem centralnym a serwerem bazy danych. Ponieważ host był wcześniej w klastrze więc doprowadzenie jego bazy do spójności może być wykonane poprzez wykonanie na wszystkich zapytań modyfikujących bazę, które były wykonane po jego awarii.

3.3.2 Przesyłanie całej bazy

Przesyłanie całej bazy będzie realizowane przez skopiowanie plików zawierających bazę danych. Host który dołączył się do klastra musi jedynie otrzymać od punktu centralnego adres hosta który posiada aktualną kopię bazy. Następnie może przy pomocy programu scp skopiować odpowiednie pliki i uruchomić proces serwera bazy danych.

3.3.3 Niewykonane zapytania

W przypadku gdy baza jest duża a w czasie awarii hosta wykonanych było niewiele zapytań modyfikujących bazę lepszym rozwiązaniem jest wykonanie tych zapytań. Aby to było możliwe wszystkie zapytania modyfikujące bazę muszą być logowane. Dodatkowo aby jednoznacznie identyfikować zapytania oraz określić ich kolejność każde zapytanie zapisywane jest w dzienniku razem z aktualną wartością zagra logicznego. Po podłączeniu się do klastra host odpytuje punkt centralny o host posiadającego aktualną bazę. Następnie komunikuje się z tym hostem wysyłając mu wartość zegara dla ostatniego wykonanego zapytania. Host posiadający aktualną kopię bazy wysyła w odpowiedzi wszystkie zapytania mające wyższą wartość zegara logicznego niż przysłana. Wykonanie otrzymanych zapytań doprowadza bazę do stanu spójności.

3.3.4 Blokowanie przy odtwarzaniu

W czasie odtwarzania bazy w którykolwiek sposób musi być zablokowane wykonywanie zapytań modyfikujących bazę.

3.4 Niezawodność

Punkt centralny odpowiedzialny jest za zarządzanie połączeniami z klientami i rozdzielaniem ich pomiędzy repliki wchodzące w skład bazy danych. W takim układzie jego awaria spowodowałaby zatrzymanie całej usługi. Aby zapobiec niedostępności bazy z powodu awarii tylko jednego węzła, zastosowaliśmy zwielokrotnienie procesu punktu centralnego. Procesy te są uruchomione na różnych węzłach klastra, tak aby przy awarii aktualnego punktu centralnego, inny mógł przejąc jego zadania.

3.4.1 Zbiór procesów zapasowych

Oprócz procesu centralnego aktualnie obsługującego wszystkie połączenia z klientami mysql oraz nakładkami na właściwe repliki, w trakcie działania systemu utrzymywany jest zbiór procesów zapasowych, nieaktywnych w trakcie prawidłowego działania. Wszystkie procesy punktu centralnego cały czas utrzymują ze sobą połączenie, tak aby móc wykryć awarię.

W przypadku wykrycia zatrzymania aktywnego procesu punktu centralnego inicjowany jest proces elekcji nowego procesu, który będzie obsługiwał nowe połączenia z klientami. Wszystkie procesu utrzymują ze sobą połączenia i znają swoje numery identyfikacyjne i elekcja sprowadza się do określenia czy dany proces ma najwyższy numer. Taki proces uaktywnia się poprzez powiadomienie nakładek na serwerach mysqld o przejęciu przywództwa oraz rozpoczęciu nasłuchiwania na połączenia przychodzące od klientów mysqld.

W konfiguracji punktów centralnych była by ustawiona maksymalna liczba procesów punktów centralnych. Jeśli osiągnięto tą liczbę, nowe procesy centralne nie będą dołączane do zbioru. Aby zbiór procesów mógł udostępniać usługę, jego liczność musi być większa niż połowa maksymalnej liczby, tak aby nie dopuścić do powstania dwóch odrębnych zbiorów punktów centralnych konkurujących o świadczenie usługi.

W konfiguracji ustawiany jest również poziom optymalny. W przypadku liczności procesów w zbiorze poniżej tej liczby, okresowo proces o najniższym id inicjuje uruchomienie nowego procesu na jednym z węzłów klastra (pula węzłów ustalona w konfiguracji systemu), o ile nie jest już zajęty przez inny proces centralny.

3.4.2 Współpraca z nakładkami

Nowe nakładki mają w pliku konfiguracyjnym podaną pulę adresów procesów punktów centralnych, do których próbują się połączyć. W przypadku połączenia z procesem nieaktywnym, uzyskują od niego adres aktywnego punktu centralnego.

W trakcie działania systemu informacje o nowych nakładkach są rozgłaszane do wszystkich procesów punktu centralnego. Dzięki temu w przypadku aktywacji nowego innego procesu, może on szybko zlokalizować nakładki z aktualną wersją bazy.

W przypadku awarii aktualnego punktu centralnego, nakładki z nim połączone anulują niedokończone operacje na bazie i przez ustalony czas czekają na zgłoszenie się nowego punktu centralnego. Po tym czasie następuje ponowne przeprowadzenie operacji dołączania z procesami podanymi w pliku konfiguracyjnym.

3.4.3 Możliwe modyfikacje projektu

Wykorzystując usługę LVS (Linuks Virtal Server), byłoby możliwe rozłożenie obciążenia na wszystkie procesy punktu centralnego. W takim układzie awaria jednego z

węzłów powodowała by jedynie utratę połączeń przez niego obsługiwanych i nie była by zauważona przez pozostałe klienty mysql.

Modyfikacji podlegało by zarządzanie blokadami przy modyfikacjach, tak aby zablokować na wszystkich procesach punktu centralnego.

Inną modyfikacją było by przeniesienie części zakresu obowiązków punktu centralnego na nakładki obsługujące serwery mysql, tak aby punkt centralny był odpowiedzialny jedynie za rozdział połączeń oraz przechowywanie informacji o blokadach bazy. Analizą typu zapytań zajmowała by się wtedy nakładka i w przypadku modyfikacji, prosiła punkt centralny o przyznanie prawa do zapisu.

4 Szkielet implementacji

4.1 Komunikacja

Proponowany interfejs modułu komunikacyjnego:

- sendSQL(klientHandler, sqlcode, isWrite); //PC-¿N; isWrite jedynie aby nakładka nie musiała ponownie analizować
- sendSQL2(klientHandler, sqlCode); // N-¿Mysqld; wykonanie polecenia na bazie
- sendSQLReply(mysqldReply, status); // N-¿PC;
- sendSQLReply2(mysqldReply, klientHandler); // PC-¿Klient; wysłanie wyników do klienta
- sendNowyKlient(nakladka, klientHandler); // PC-¿N; powiadomienie nakładki o nowym kliencie w systemie; ew. niepotrzebne, patrz uwaga 5.2
- sendDisconnectKlient(nakładka, klientHandler); // PC-¿N: powiadomienie nakładki o zakończeniu obsługi danego gościa
- sendTimeStamp(nakladka, nakladkaAktywna, timestamp); // PC-¿N; wysłanie stempla czasowego i namiarów na nakładkę, która zajmie się uzupełnieniem danych nowej
- sendActivePC(nakladka, adresPortPC); // PC-¿N; nieaktywny PC podaje namiary na aktywnego nowo przyłączanej nakładce
- sendReqSynchrBlokuj(); //N-¿PC; zablokowanie bazy na czas synchronizacji.
- sendAckSynchrBlokuj(); // PC-¿N; potwierdzenie blokady
- sendSynchrOdblokuj(); // N-¿PC; zdjęcie blokady i prośba o przyłączenie
- sendWelcomeActive(klientHandlers[]); //PC-¿N; potwierdź przyłączenie do puli nakładek aktywnych i wyślij zbiór aktywnych klientów (ew nie wysyłaj wg uwagi 5.2)
- sendNowaNakladka(PC, nakladka); // PC-¿PC; powiadomienie nieaktywnych PC o nowej nakładce
- sendUsunNakladke(PC, nakladka); // PC-¿PC; powiadomienie o usunięciu nakładki
- sendPing(kogo); // PC-¿PC, PC-¿N; można ew rozdzielić na dwa pingi dotyczące obu warstw, bo ?kogo? trochę za ogólnie
- sendPong(kogo); // PC-¿PC, N-¿PC; jw
- sendGetJournal(nakladka,timeStamp); // N-¿N; chęć pobrania nowych zmian
- sendJournal(nakladka, journal);//N-N; wysłanie ostatnich zmian do nowej nakładki
- sendDBDump(nakladka, dbDump); // N-¿N wysłanie do nowej N całej bazy

- sendMeGod(nakladka); // PC-¿N; powiadomienie nakladki o nowym PC głównym
- inne (elekcja, przyłączanie PC, ...)

Przy komunikacji, główną rolę będą pełniły funkcje select(). Z tego względu trzeba niejako emulować wielowątkowość przy użyciu struktur przypisanych połączeniom/zadaniom: bufory nadawczo-odbiorcze, reakcje na zdarzenia zależne od stanu obiektów, którego dotyczą.

4.2 Punkt centralny

Budowa punktu centralnego w oparciu o funkcję select() wygladalaby następująco:

```
zdarzenie = opakowaneSelect(); // wysylanie z buforow gdy sie da i czytanie do momentu przeczytania calego komunikatu
switch(zdarzenie){
    case nowyKlient:
        onNowyKlient(); // obsluga przylaczenia nowego klienta
    case nowaNakladka:
        onNowaNakladka();
    case nowyPC:
       onNowyPC();
    case klienCommand:
        onClientCommand(); // analiza i obsluga nowego polecenia
    case internalCommand:
            onInternalCommand(); // obsluga wewnetrznej komunikacji: dalsze switch/case
    case lostConnection:
        onLostConnection(); // obsluga sytuacji wyjatkowych (odlaczenia)
    case onHeartTimer:
         doHeartBeat(); // sprawdz czy odebrano wszystkie heartBeat / inicjuj nowy heartbeat
    case onExit:
        onExit();
}
```

i powyższe w pętli while(not killed/finished)

Funkcja opakowaneSelect() powinna mieć zbiory deskryptorów :

- read : zwrócenie zdarzenia przez funkcję opakowaneSelect() dopiero po odebraniu całego pakietu.
- write : powinna istnieć kolejka nadawcza z pakietami do wysłania dla każdego deskryptora.
- accept : nasluchiwanie na porcie dla nowych nakładek/nowych pc/ nowych klientów.

Elementy zbiorów nadawczo-odbiorczych powinny mieć możliwość dezaktywowania czasowego, aby nie dopuścić do przypchania nowymi zleceniami, jeśli nie nadążamy z obsługą. Dodatkowo funkcja opakowaneSelect() musi kontrolować timer przypisany

heartBeat. Nie wiem jeszcze co z zatrzymywaniem. Chyba sygnały są najrozsądniejsze (wystarczy kill i program powinien się ładnie położyć). Ew. przez sieć wysyłając odpowiedni komunikat.

4.3 Zarządzanie klastrem

Konfiguracja nakładki:

- port nakładki
- adres:port podległego mysqld lub ścieżka skryptu uruchamiającego i port
- adresy:porty pewnej puli PC do próbowania dołączenia się
- inne

Konfiguracja punktu centralnego:

- port nasluchiwania
- adresy:porty puli innych PC
- rozmiar kworum (minimalnej liczby PC lub N do świadczenia usługi)
- id ew. można by użyć PID, w końcu są unikalne, ale przydział PC centralnego będzie preferować komputery o niższej/wyższej puli przydzielanych PID. Ew id byłby przydzielany inkrementacyjnie po przyłączeniu do puli aktywnych PC
- parametry różnakie (ograniczenia połączeń, czas heartbeat, ...)
- ścieżki do skryptów uruchamiających nowe N/PC (w ramach keepalive)

4.4 Nakładki

Należy uwzględnić w pętli while selektującej:

- onSqlCommand() : wykonanie SQL, i jeśli modyfikacja, poczekanie z commit aż potwierdzenie od PC.
- onModifyCommit/Rollback : zatwierdzenie z zapisem w logu / wycofanie ostatniej modyfikacji
- onSQLReply() : odebrano odpowiedz od mysqld i nalezy ja odeslac do PC
- onClientDisconnect() : odłączenie połączenia w ramach danego klienta
- onNewClient() : nawiązanie połączenia z mysqld na rzecz klienta
- onPCConnect() : w zasadzie to tylko dodanie do kolejki czytania
- onPCDisconnect() : jesli PC był głównym, to rozłącz klientów i anuluj wszelkie akcje z mysqld, które nie powinny być zatwierdzone. W przeciwnym wypadku usun tylko z kolejki czytania

- onPCMeGod() : dany PC ogłosił swoje przywództwo, ustaw go jako gościa do wysyłania przyszłych odpowiedzi. Wyślij mu własny stempel czasowy (aby mógł się osadzić w czasie).
- onPing() wysłanie Pong();
- onNowaNakladka() : inicjuje wymiane danych w celu synchronizacji...

5 Przygotowanie paczki z instalacją serwera MySQL zoptymalizowaną pod klaster

Dystrybucja serwera MySQL działającego na koncie użytkownika, bez potrzeby posiadania uprawnień root-a, czy dostępu do plików spoza katalogu z serwerem, powstała na bazie paczki w wersji 5.1.7 dostępnej w repozytorium na serwerze openone.
Po pobraniu paczki z prepozytorium uruchamiany jest skrypt ./prepare dokonujący rozpakowania paczki, a nastepnie rozdysponowania jej (przy użyciu skryptów w katalogu głównym paczki) do poszczególnych katalogów w klastrze. Za rozdysponowanie plików na klaster 'n' odpowiada skrypt ./mysql-5.1.7/setup z parametrem ¡n¿. Uruchomienie/zatrzymanie/reset/reinstalacja odbywa się przy pomocy skryptu ./mysql-5.1.7/mysqlgo z parametrami start—stop—reset—setup. W katalogu ./mysql-5.1.7 znajduje się również plik README, zawierający podstawowe informacje o korzystaniu z paczki.

Podsumowanie III fazy projektu z RSO.

Grupa RSO4

Andrzej Grudzień
Grzegorz Lepionka
Maciek Nowak
Marcin Rudowski
Paweł Słupczyński

17 czerwca 2006

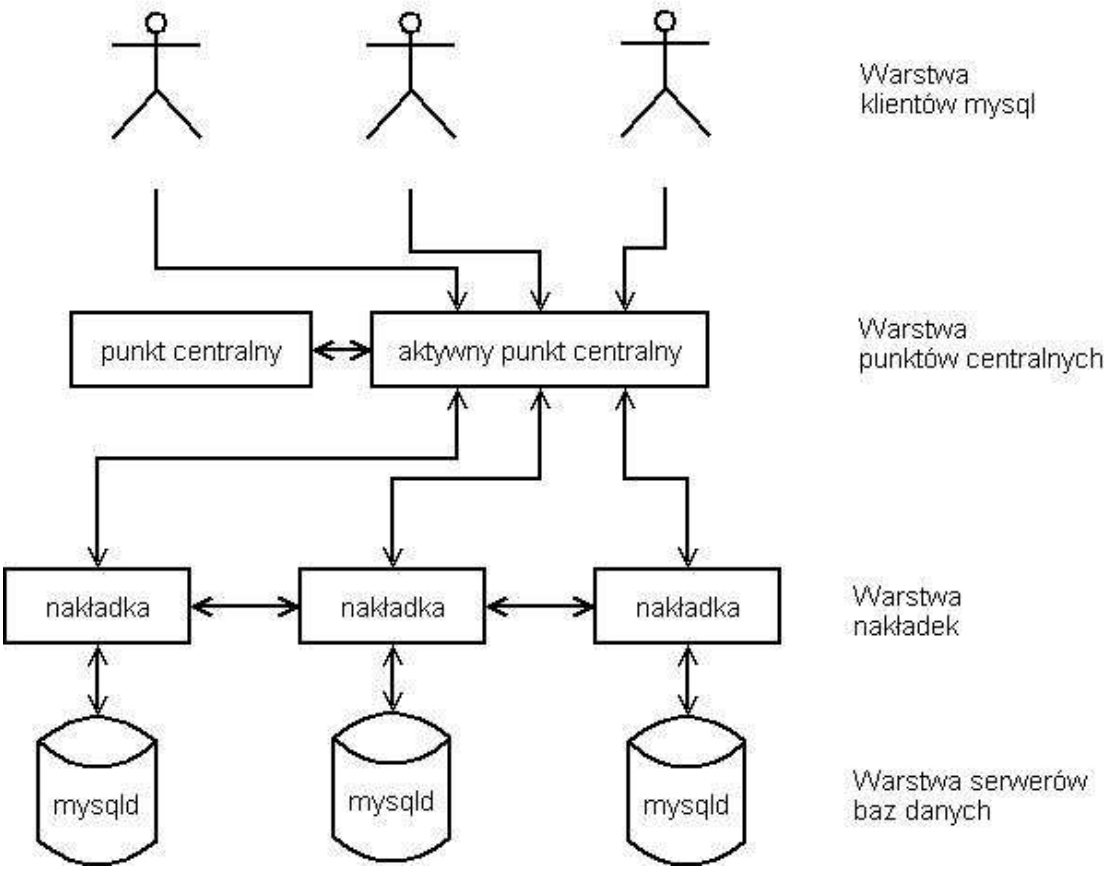
1 Wstęp

Niniejszy dokument zawiera dokumentację końcowego projektu, wyniki testów i podział zadań ogólnych i trzeciej części projektu z przedmiotu RSO.

2 Opis rozwiązania

2.1 Architektura systemu

W celu realizacji projektu - zwiększenia wydajności i niezawodności bazy danych MySQL 5.1.7 beta w stosunku do rozwiązania standardowego, wprowadziliśmy architekturę wielowarstwową. W ramach architektury wydzieliliśmy następujące warstwy:



Rysunek 1: Architektura systemu

- Warstwa klientów mysql - warstwa standardowych klientów bazy danych MySQL.

- Warstwa punktów centralnych - warstwa procesów odpowiedzialnych za zarządzanie połączeniami od klientów bazy danych. Tylko jeden proces punktu centralnego jest aktywny, pozostałe monitorują system, i w razie awarii aktywnego punktu centralnego przejmą jego rolę.
- Warstwa nakładek - warstwa odpowiedzialna za utrzymywanie połączeń z serwerami bazy danych. Każdy jeden proces nakładki współpracuje z pojedynczym procesem mysqld.
- Warstwa serwerów baz danych.

2.1.1 Warstwa klientów mysql

Jest to warstwa standardowych klientów bazy danych MySQL. Nie wprowadziliśmy żadnych wymagań dotyczących klienta, jedynym ograniczeniem jest fakt, że klient nie może wykorzystywać wyrażeń "Prepared statementsźe względu na ograniczenia związane z ponownym przyłączaniem się do serwera bazy danych w trakcie aktywnej sesji klienta (po rozłączeniu uchwyty "Prepared statementssą tracone, po ponownym przyłączeniu - trudne do odtworzenia).

2.1.2 Warstwa punktów centralnych

Warstwa punktów centralnych jest grupą procesów rozproszonych po węzłach klastra, których głównymi zadaniami są:

- Odbieranie połączeń od klientów bazy danych (tylko przez aktywny punkt centralny).
- Zarządzanie zapytań skierowanymi od klientów bazy danych - aktywny punkt centralny ustala rodzaj zapytania, w przypadku gdy jest to zapytanie o dane, zostaje ono przekierowane do najmniej obciążonej nakładki,
- Szeregowanie zapytań modyfikujących dane - w przypadku zapytania modyfikującego dane - aktywny punkt centralny jest odpowiedzialny za zablokowanie innych znadchodzących apytań, i po wykonaniu wszystkich istniejących zapytań - przekierowanie go do wszystkich nakładek.
- Monitorowanie innych punktów centralnych - punkty centralne wzajemnie siebie monitorują, dzięki czemu awaria węzła z aktywnym punktem centralnym powoduje, przy wykorzystaniu algorytmu elekcji, wybór kolejnego aktywnego punktu centralnego, który przejmuje rolę węzła dostępowego dla warstwy klientów mysql.
- Synchronizacja informacji o aktualnym stanie systemu od aktywnego punktu centralnego do pozostałych punktów centralnych w celu umożliwienia natychmiastowego przejęcia jego zadań w przypadku awarii.

Punkty centralne dysponują informacjami o liście pozostałych punktów centralnych w systemie, oraz portach i adresach ip na których maja prowadzić nasłuch. Uruchomienie ich na różnych węzłach klastra zapobiega awarii systemu na skutek wyłączenia jednego lub grupy węzłów.

2.1.3 Warstwa nakładek

Warstwa nakładek, jest to warstwa procesów bezpośrednio połączonych z serwerami bazy danych. Każdy proces nakładki (nakładka) jest połączona z jednym procesem serwera bazy danych mysqld. Nakładki są odpowiedzialne za:

- Przy włączeniu - zapewnienie synchronizacji między bazami danych, poprzez transfer brakujących poleceń od nakładki wskazanej przez punkt centralny jako zawierającej aktualny stan bazy danych (synchronizacja nie została niestety zrealizowana do końca).
- Ustawianie stanu połączenia z serwerem bazy danych i monitorowanie wykonywania poleceń przez niego - na rzecz punktu centralnego.
- Przekazywanie pozostałej komunikacji przychodzącej od punktu centralnego do serwera bazy danych
- Monitorowanie stanu serwera bazy danych związanego z daną nakładką.

Poszczególne nakładki wraz z procesami warstwy serwerów baz danych są uruchamiane na różnych węzłach klastra.

2.1.4 Warstwa serwerów baz danych

Warstwa niezmodyfikowanych, skonfigurowanych do pracy lokalnie serwerów baz danych MySQL 5.1.7 beta, otrzymujących połączenia od nakładki współpracującej z danym serwerem, uruchomionych na tym samym węźle co dana nakładka (by fizyczna awaria węzła powodowała jednoczesną awarię procesu serwera baz danych oraz nakładki).

2.2 Realizacja zwiększonej wydajności i niezawodności

Dzięki zastosowaniu wyżej opisanych rozwiązań w ramach architektury wielowarstwowej, uzyskaliśmy:

- Zwiększenie wydajności.
- zwiększenie niezawodności.

2.2.1 Zwiększenie wydajności

Zwiększenie wydajności nastąpiło poprzez wprowadzenie warstwy punktów centralnych, odpowiedzialnych za zarządzanie połączeniami z klientami. Połączenia nie wymagające modyfikacji bazy danych są przekierowywane do wolnej lub najmniej obciążonej nakładki (pod względem liczby aktualnie przetwarzanych zadań). Dzięki temu wiele jednoczesnych zapytań jest rozpraszanych pomiędzy wiele węzłów, dzięki czemu obciążanie węzła jest mniejsze. Rozwiązanie to pozwoliło zwiększyć wydajność o około 10% względem rozdzielania zadań przy użyciu algorytmu Round-Robin. Względem paczki nr 1 wydajność zapisów spadła w najgorszych warunkach (o 50%), natomiast wydajność odczytów wzrosła o 50% (maksymalnie, dla dużej ilości małych zapytań). Test był przeprowadzany przy 2 klientach i 2-3 serwerach baz danych.

2.2.2 Zwiększenie niezawodności

- Dzięki zastosowaniu warstwy punktów centralnych, z aktywnym punktem centralnym i grupą zapasowych punktów centralnych, awaria któregokolwiek z nich nie powoduje awarii całego systemu.
- Dzięki wzajemnemu monitorowaniu się punktów centralnych, awaria któregokolwiek z nich jest szybko zauważana.
- Nakładki znają listę możliwych punktów centralnych, na podstawie tej listy wybierają aktywny punkt centralny, dzięki czemu awaria aktywnego punktu centralnego nie prowadzi do awarii całego systemu
- Informacje przechowywane w poszczególnych serwerach baz danych, monitorowanych przez nakładki są między sobą zsynchronizowane, dzięki temu awaria jednego lub grupy serwerów baz danych lub nakładek nie powoduje awarii całego systemu, a nawet nie ma wpływu na przebieg transakcji wykonywanej przez klienta.
- Warstwa klientów mysql może (i powinna) dysponować listą możliwych aktywnych punktów centralnych, dzięki temu awaria punktu centralnego nie powoduje awarii całego systemu.

3 Testy

3.1 Wstęp

Architektura zaimplementowanego rozwiązani była przygotowana pod kątem zapewnienia dostępności pomimo awarii poszczególnych węzłów składowych. Optymalizacja wydajności objęła równoważenie operacji odczytów i tutaj należy oczekiwać wzrostu wydajności względem pojedynczego serwera MySQL.

3.2 Narzędzia

Do testów użyte zostały skrypty dostępne w ramach 'The MySQL Benchmark Suite'. Testy te są udostępniane razem z mysql, w katalogu sql-bench. Do uruchomienia wymagane są dodatkowe biblioteki perl

- DBI : interfejs dostępu do baz danych
- DBD-mysql : sterownik dostępu do bazy danych typu mysql

W nowszej wersji DBM-mysql (począwszy od 3.0002_1) domyślnie wykorzystywane są polecenia wstępnie przygotowywane po stronie serwera (prepared statements), które nie są obsługiwane przez nasz system. Konieczne było zatem użycie trybu emulacji po stronie sterownika DBD przez użycie dodatkowej opcji uruchomienia skryptów testujących: –connect-option=mysql_emulated_prepare=1

Do oceny rozwiązania użyto podstawowego skryptu analizującego głównie odczyty, aby zweryfikować efekty równoważenia obciążenia. W trakcie rozwoju aplikacji wykorzystywane były również pozostałe testy do analizy wąskich gardeł systemu oraz analizy błędów.

Do analizy kody używane były popularne narzędzia dla systemu Linuks:

- cachegring - profiler kodu
- vallgrind - analiza zarządzaniem pamięcią
- ethereal - wizualny analizator pakietów sieciowych

3.3 Rozmieszczenie elementów składowych

Węzły openone zostały podzielone na trzy kategorie:

- bazodanowe - każdy zawiera właściwy serwer mysql oraz własną aplikację 'nakladka, która umożliwia punktom centralnym dostęp do danej kopii danych
- punkty centralne - zbiór węzłów, które zawierają punkty centralne. W ramach testów uczestniczył jedynie jeden węzeł tego typu, gdyż zadaniem pozostałych jest jedynie zastąpienie głównego w przypadku jego awarii.
- aplikacje klienckie - węzły, na których uruchamiano skrypt testowy

Na węzłach bazodanowych uruchomiony serwer mysql miał ustawiony katalog z danymi w lokalnym katalogu węzła:

```
/cluster/node\$(ID)/mnt/dsk/rso/rso4/\$(USER)/data.\$(ID)
```

gdzie $(ID) jest numerem węzła. Proces nakladka jest uruchamiany na tym samym węźle i komunikuje się z serwerem poprzez gniazdo rodziny Unix.

Ze względu na ograniczoną liczbę węzłów uruchomionych na raz w ramach dostępnego klastra, w testach wzięto pod uwagę przypadek 1, 2 lub 3 węzłów bazodanowych.

Punkt centralny znajduje się na oddzielnym węźle i komunikuje się z nakładkami poprzez trwałe połączenie TCP/IP. Testy wydajności przeprowadzone na wstępnym etapie budowy systemu pokazały, że w przypadku komunikacji między węzłami takie połączenie było wydajniejsze niż oferowane przez OpenSSI potoki lub gniazda rodziny Unix.

Skrypty testujące były uruchamiane na pozostałych węzłach. Równolegle było uruchamiane 1-3 skryptów.

Systemem porównawczym był pojedynczy serwer mysql w wersji podstawowej.

3.4 Testy porównawcze

Porównanie z pojedynczym mysql

W poniższej tabeli zebrano średni wynik w sekundach z wykonania poszczególnych testów w ramach skryptu 'test-select' z katalogu sql-bench:. Podane wyniki wyrażone są w sekundach i określają czas wykonania danego etapu testu. Nazwy etapów zgodne ze użytymi w skrypcie testującym.

Typ test:	3K0S	3K2S	3K3S	2K0S	2K2S	2K3S
insert	21	55	93	13	49	47
select_cache	326	206	123	200	116	115
select_cache2	313	201	126	200	117	117
select_big	1	1	1	1	1	0
select_range	273	166	81	173	78	81
min_max_on_key	32	210	194	26	178	166
cnt_on_key	555	369	333	382	309	301
cnt_group_on_key_parts	61	33	25	44	25	25
cnt_distinct_key_prefix	32	20	15	28	15	14
cnt_distinct	45	20	18	35	18	18
cnt_distinct_2	57	24	24	41	23	23
cnt_distinct_grp_on_key	46	27	22	41	22	23
cnt_distinct_grp_on_key_parts	33	27	22	35	22	21
cnt_distinct_grp	36	25	23	35	21	22
cnt_distinct_big	9	16	14	10	13	12
Total:	1840	1400	1114	1264	1007	986

iK jS oznacza, że uruchomiono i równoległych sesji skryptu, a obsługą zajmowało się j serwerów mysql. Kolumna w której j==0 oznacza wersję standardowego serwera mysql, bez naszego systemu.

Proponowane rozwiązanie dobrze radzi sobie w przypadku dużych zapytań, jakie miały miejsce w przypadku testów select_cache oraz select_range.W przypadku mniejszych zapytań różnica jest już mniejsza, a dla testu min_max_on_key, standardowa dystrybucja mysql jest blisko 10x szybsza. W tym teście było bardzo dużo (70000 iteracji) małych zapytań i narzuty związane z obsługą i transportem między poszczególnymi węzłami systemu spowodowały drastyczną różnicę wydajności. Dla porównania pozostałe testy obejmowały 1000-10000

iteracji.

Próby uruchomienia testu na wersji klastrowej serwera mysql powodowały zatrzymanie na pierwszym etapie odczytów (select_cache). Analiza interfejsu sieciowego ukazywała nasycenie ruchu (ok 7MB/s) oraz maksymalne wkorzystanie procesora przez procesy klastra mysql. Test został przerwany po ok 15 minutach.

Operacje modyfikacji

W przypadku zapisów, polecenia są wykonywane na wszystkich kopiach. W związku z tym, wydajność w takich operacjach z zasady będzie gorsza niż pojedynczego serwera mysql. Poniżej znajdują się wybrane wyniki dla skryptu test-insert. Ze względu na ograniczone zasoby klastra openone (liczba węzłów aktywnych) i interferencje poszczególnych zespołów, testy udało się w sposób powtarzalny wykonać jedynie dla przypadku do 2 węzłów z danymi. Mimo wszystko testy pozwalają określić pewne ogólne tendencje.

Tabela poniżej przedstawia czas (w sekundach) wykonania k iteracji (ilość iteracji podana w nawiasie) procedur modyfikujących.

Typ test:	1K0S	1K1S	1K2S	3K0S	3K1S	3K2S
delete_all_many_keys (1)	5	5	6	11	12	8
delete_big_many_keys (128)	5	5	6	11	12	8
delete_key (2000)	0	1	1	10	20	7
insert (61000)	17	35	40	46	63	139
insert_duplicates (20000)	6	10	13	10	19	45
insert_key (20000)	15	20	23	48	48	106
update_big (10)	3	2	3	8	11	20
update_of_key (10000)	4	7	8	7	15	46
update_of_key_big (491)	2	3	2	11	10	7
update_of_pri_key_many_keys (256)	3	3	3	7	3	4
update_with_key (60000)	23	37	40	55	71	136
update_with_key_prefix (20000)	7	12	14	16	23	45

Porównując wydajność dla systemu z pojedynczym węzłem bazodanowym oraz z serwerem mysqld widać, że zaimplementowana warstwa pośrednia wnosi pewne ograniczenia. Wynikają one z pozostałych wąskich gardeł, które wymagałyby dodatkowego usunięcia i dostrojenia w przypadku dalszego rozwoju aplikaci. Dodatkowo wydajność przy zwiększaniu liczby węzłów z danymi jest ograniczona m.in. przez konieczność dokonywania modyfikacji na wszystkich replikach. Operacje te wykonywane są jednak na wszystkich węzłach równolegle, dlatego wnoszone opóźnienie niewiele wzrasta wraz ze zwiększeniem liczby węzłów z danymi.

3.5 Wąskie gardła systemu

Na podstawie testów prowadzonych równolegle z implementacją udało się zredukować główne wąskie gardła. W późnych fazach powstawania, narzuty obsługi klienta przez implementowaną warstwę pośredniąbyły znaczne. Nasze rozwiązanie okazywało się być nawet 10-20 krotnie wolniejsze od zwykłego serwera mysql. Porównanie dotyczyło przypadku jednego węzła bazodanowego, jednego punktu centralnego oraz jednego klienta. W takim układzie analizie podlegają

tylko narzuty implementacji na transport poleceń między serwerem i klientem, bez udziału równoważenia obciążenia.

Analiza kodu wykazała rozrzutność w alokacji pamięci. Przyjęty model natychmiastowego przetwarzania odbieranych danych uwzględniał dużą fragmentację, aby maksymalnie skrócić czas transportu pakietów. Spowodowało to jednak zwiększenie liczby alokacji pamięci na dane cząstkowe. Optymalizacje kodu spowodowały zmniejszenie zużycia procesora do minimum, jednak bez poprawy wydajności. Znowu, winna była fragmentacja pakietów. wysyłanie każdego pakietu oddzielnie wprowadzało opóźnienia na poziomie wejścia-wyjścia. Wprowadzenie buforowania i odbierania maksymalnej liczby danych w jednym odczycie oraz wysyłania w jednym zapisie maksymalnej liczby pakietów z bufora zredukowało opóźnienie do maksymalnie dwukrotnego pogorszenia wydajności względem standardowej dystrybucji.

Aktualnie, przy zapytaniach zwracających duże ilości danych, wydajność jest maksymalnie ograniczona przez wcześniej wspomniany narzut ok 2x. Dla wielu równoległych zapytań, średni czas obsługi jest odpowiednio mniejszy niż dla pojedynczego serwera ze względu na równoważenie odczytów. Analiza na poziomie sieciowym wskazała, że miedzy punktem centralnym a nakładką maksymalnie osiągany transfer jest rzędu 1MB. Wydaje się, iż wynika to ze znacznej fragmentacji pakietów na wielu etapach transportu danych. Rozwiązanie to zostało wprowadzone, aby skrócić czas odpowiedzi wynikający z dodatkowych punktów pośrednich. Dostrojenie fragmentacji, przez zwiększenie minimalnego rozmiaru fragmentu podlegającego wysłaniu dalej mogło by poprawić wydajność zarówno transmisji jak i procedur obsługi zdarzeń w punkcie centralnym i nakładce.

Jak widać było w teście min_max_on_key, dla zapytań wykonujących się bardzo krótko, narzut związany z obsługą przez poszczególne węzły systemu może drastycznie obniżyć korzyści płynące z równoważenia obciążenia. Możliwe, że przeprowadzenie testu z większą liczbą klientów ukazało by zalety rozproszenia zapytań również w tych przypadkach.

4 Uruchamianie i konfiguracji systemu

System składa się z dwóch aplikacji:

- pc - punkty centralne
- nakladka - aplikacja nakładki na serwer mysql

Parametry konfiguracyjne mogą być podane w pliku konfiguracyjnym lub z linii poleceń. Plik konfiguracyjny może być wspólny dla obu aplikacji lub dla każdego węzła oddzielny.

4.1 Konfiguracja

Konfiguracja systemu zapisana jest w pliku lub plikach konfiguracyjnych. Format pliku jest tekstowy i każda opcja zajmuje jedną linię. Linie puste lub rozpoczynające się od znaku '#' są ignorowane. Opcje nieznane również lub nie istotne dla danej aplikacji również nie powodują błędów.

Parametry w pliku są zapisane wg schematu:

```
nazwa_parametru wartosc_parametru
```

Nazwa parametru nie może zawierać białych znaków, a wartość zaczyna się od pierwszego nie białego znaku po nazwie.

Dostępne parametry konfiguracyjne dotyczące komunikatów: Komunikaty generowane przez aplikacje są wielopoziomowe i każdy opatrzony jest nazwą działu którego dotyczy.

Dostępne poziomy logowania:

- global - ogólny stan węzła
- do_pc - komunikacja nakładki z punktem centralnym
- multiplex - obsługa poleceń wymagających wykonania na wszystkich replikach
- pc_obsluga - stan obsługi zdarzeń w punkcie centralnym
- nk_obsluga - stan obsługi zdarzeń na nakładce
- mysql - komunikacja nakładki z serwerem mysql
- raw_data - dodatkowe wyświetlanie zawartości binarnej pakietów
- polaczenia - informacje dotyczące wejścia wyjścia oraz komunikacji
- protokoly - wyniki działania analizatora zapytań i protokołu mysql

Dla każdego poziomu komunikaty mogą być różnej istotności. Wydzielone zostało 6 podstawowych poziomów i nie wszystkie są użyte. Dodatkowo obsługa logowania zakłada jeszcze 3 zapasowe, dla chwilowych potrzeb przy testowaniu i poprawianiu kodu.

Włączanie danego poziomu jest poprzez odpowiednią maskę podawaną w konfiguracji. Maska składa się z maksymalnie 9 znaków i kolejne znaki odpowiadają kolejnym poziomom komunikatów informacyjnych. Pierwsze 3 poziomy(High, Med, Low) dotyczą komunikatów o zachowaniach istotnych dla użytkownika, lub dotyczące błędów. Kolejne 3 poziomy(D_High, D_Med, D_Low) dotyczą debugowania systemu i powinny być wyłączone przy normalnym używaniu.

Przykładowa maska włączająca tylko komunikatów użytkowych: +++——

W pliku konfiguracyjnym wydzielono następujące pozycje sterujące komunikatami generowanymi przez aplikację:

- log_file - położenie pliku dla logów (/dev/null)
- log_mask - maska wyłączenia poszczególnych poziomów wszystkich logów
- log_[dział] - maska poziomów dla danego działu
- log_show_pre - flaga określająca czy komunikaty mają być poprzedzane nagłówkiem z numerem poziomu i nazwą działu, którego dotyczą (wartość parametru: 0 lub 1)

konfiguracja dodatkowa:

- daemon - flaga określająca, czy aplikacja ma być uruchomiona w trybie daemona (0/1)
- moj_id - id danego węzła. W przypadku korzystania z wspólnego pliku, należy użyć opcji linii poleceń

pula punktów centralnych i nakładek:

W pliku konfiguracyjnym podana jest lista namiarów na punkty centralne oraz nakładki. Listy te są używana przez wszystkie węzły do określenia wzajemnego położenia na podstawie numerów identyfikacyjnych.

Elementy są numerowane kolejnymi numerami, które służą jedynie określeniu kiedy kończy się pula. Program pobierając dane z konfiguracji próbuje kolejne numery do znalezienia aktywnego węzła.

Dostępne parametry (# oznacza numer wpisu na liście):

- pc#_id - id danego PC
- pc#_ip - adres ip
- pc#_pc_port - port dla innych PC
- pc#_nk_port - port dla nakładek
- pc#_mysql_port - port dla klientów mysql
- nk#_id - id danej nakładki
- nk#_pc_ip - ip na ktorym czuwa nakladka
- nk#_pc_port - port na ktorym czuwa nakladka
- nk#_def_pc_id - domyślny pc, z ktorym probuje połączyć sie nakladka

- nk#_mysql_ip - ip serwera mysql (wymiennie z socketem)
- nk#_mysql_port - port serwera mysql (wymiennie z socketem)
- nk#_mysql_s - socket serwera mysql

Przykładowy plik konfiguracyjny

Poniższy przykładowy plik jest dla przypadku układu dwóch nakładek (na węźle 2 i 12) oraz dwóch punktów centralnych (na węźle 1 i 3). Przy uruchamianiu należy tylko ustalić id danego węzła w linii poleceń (opcja -i ID) oraz ustawić punkt centralny na węźle 1 jako główny (opcja -f).

```
####################################
#       Wspolne
####################################

## ustawienia loggera
## plik na logi (domyslnie /dev/null); flaga [ -l log_file ]
#log_file        ./log_nk.txt
# poziomy gadatliwosci (+/- dla odpowiedniego poziomu 0-9)
#               nnnddduuuu (normal_lev, debug_level, user_specific)
# log_nazwa     0120123789  - im wyzszy numerek tym mniej istotne dane
# maska pozwalajaca wylaczyc wszystko
log_mask        ++++++++++
# poszczegolne poziomy, mozna komentowac, lub wstawiac plusy w srodku
log_global      ++++++++++
log_do_pc       ++++++++++
log_multiplex   ++++++++++
log_pc_obsluga  ++++++++++
log_nk_obsluga  ++++++++++
log_mysql       ++++------
log_raw_data    +++-------
log_polaczenia  +++-------
log_protokoly   +++-------


## czy pokazywac dzial i poziom logowania na poczatku linii(0/1)
## generalnie przydatne, do okreslenia co wylaczyc
log_show_pre    1

## czy daemon (1/0), flaga [ -d ] dla daemona (experymentalna) i [ -D ] dla niedaemona (domys
daemon          0


## id jednostki operacyjnej (pc/nk); flaga [ -i moj_id ]
## wskazane podanie z linii polecen
#moj_id         10

####################################
#       Punkt centralny
####################################
```

```
# generalnie jeden powinien byc first
# flaga [ -f ] dla first oraz [ -F ] dla pobocznego
im_first        0


####################################
#       Informacje o puli pc
####################################
# pc#_id          id danego PC
# pc#_ip          adres ip
# pc#_pc_port     port dla innych PC
# pc#_nk_port     port dla nakladek
# pc#_mysql_port  port dla klientów mysql

pc0_id          1
pc0_ip          10.0.0.1
pc0_pc_port     1113
pc0_nk_port     1112
pc0_mysql_port  1111

pc1_id          3
pc1_ip          10.0.0.3
pc1_pc_port     1113
pc1_nk_port     1112
pc1_mysql_port  1111

# dla localhosta i testowania poza openone:
pc2_id          102
pc2_ip          127.0.0.1
pc2_pc_port     2113
pc2_nk_port     2112
pc2_mysql_port  2111

pc3_id          103
pc3_ip          127.0.0.1
pc3_pc_port     3113
pc3_nk_port     3112
pc3_mysql_port  3111




####################################
#       Informacje o puli nk
####################################
# nk#_id                id
# nk#_pc_ip             ip na ktorym czuwa nakladka
# nk#_pc_port           port na ktorym czuwa nakladka
# nk#_def_pc_id         domyslny pc, z ktorym probuje polaczyc sie nakladka
```

```
# nk#_mysql_ip          ip serwera mysql (wymiennie z socketem
# nk#_mysql_port        port serwera mysql (wymiennie z socketem
# nk#_mysql_s           socket serwera mysql

nk0_id          2
nk0_pc_ip       10.0.0.2
nk0_pc_port     1114
nk0_def_pc_id   1
nk0_mysql_s     /cluster/node2/mnt/disk/rso/rso4/mrudowsk/mysql.sock.2

nk1_id          12
nk1_pc_ip       10.0.0.69
nk1_pc_port     1114
nk1_def_pc_id   1
nk1_mysql_s     /cluster/node12/mnt/disk/rso/rso4/mrudowsk/mysql.sock.12


# dla localhosta i testowania poza openone:
nk2_id          202
nk2_pc_ip       127.0.0.1
nk2_pc_port     2114
nk2_def_pc_id   102
nk2_mysql_s     /tmp/marcom/mysql.sock.2

nk3_id          203
nk3_pc_ip       127.0.0.1
nk3_pc_port     3114
nk3_def_pc_id   102
nk3_mysql_s     /tmp/marcom/mysql.sock.3
```

4.2 Uruchamianie

Aplikacje w ramach systemu umożliwiają nadpisanie konfiguracji pobranej z pliku konfiguracyjnego przy pomocy parametrów linii polecen. Aplikacje uruchamia się zgodnie ze składnią:

```
pc/nakladka [ OPCJE ]
```

Gdzie opcje mogą być postaci:

- –nazwa_parametru wartość_parametru - ustawienie danej wartości (dostępne nazwy takie jak w pliku konfiguracyjnym
- -i ID - przypisanie id węzła (równoważne –moj_id ID)
- -f -F - ustawienie danego punktu centralnego na początku jako główny (-f) lub podrzędny (-F). Przy uruchomieniu pierwszy uruchomiony PC musi być mieć podaną opcję -f
- -d -D - uruchomienie aplikacji w trybie daemon (-d) lub interakcyjnym (-D). W trybie daemon aplikacja przechwytuje sygnał kill i w odpowiedzi kończy działanie.

- -c plik_konfiguracyjny - podanie ścieżki do pliku konfiguracyjnego. Można podać opcję wiele razy i ustawienia będą wczytane w podanej kolejności (nadpisywanie ew. wspólnych parametrów przez ostatni plik).
- -l plik_log - wskazanie pliku do logowania komunikatów

Dla podanego przykładowego pliku konfiguracyjnego, skrypt uruchamiający system może wykonać poniższe polecenia:

```
onnode -P 1  ./pc -i 1 -f -d -c config.txt -l pc1.log
onnode -P 3  ./pc -i 3 -F -d -c config.txt -l pc3.log
onnode -P 2  ./nakladka -i 2 -d -c config.txt -l nk2.log
onnode -P 12 ./nakladka -i 12 -d -c config.txt -l nk12.log
```

4.3 Tryb interaktywny

W trybie interaktywnym (opcja -D lub ustawienie –daemon 0) możliwe jest kontrolowanie ustawień węzła oraz pobranie statystyk. Polecenia wpisuje się pojedynczo i zatwierdza Enterem.

Dostępne polecenia:

- help - wyświetla krótką pomoc oraz wszystkie parametry konfiguracyjne (para nazwa, wartość)
- set nazwa_parametru wartość_parametru - nadanie nowej wartości danemu parametrowi. Część parametrów, które wykorzystywane są przy uruchomieniu aplikacji (numery portów, tryb daemon,..) nie zmieniają stanu węzła.
- log off - wyłączenie wyświetlania komunikatów na ekranie. Polecenie nie wyłącza logowania do pliku
- log on - przywrócenie wyświetlania komunikatów na ekranie
- stats - wyświetlenie statystyk dotyczących węzła oraz połączeń jakie aktualnie posiada
- quit - wyłączenie węzła

Przykładowy wynik działania polecenia stats

Poniższe statystyki wykonano w trakcie testowania i stąd tak duże liczby przesłanych bajtów (prawie 800MB) i obsłużonych zdarzeń wewnętrznych (prawie 30 milionów).

W statystykach połączeń w nawiasie podawane jest typ operacji, jaką dane połączenie aktualnie obsługuje (RW). Wartości UniCount i MultiCont oznaczają odpowiednio liczbę poleceń odczytu (podlegają równoważeniu) oraz operacji wymagających uruchomienia na wszystkich węzłach.

```
Statsy Polaczen(8):
  (R-): Nasluch Nakladek: 5
     Bufs (out: 0, in: 0)
     Total Out: 0 (sr: 0.00)
     Total In: 0 (sr: 0.00)
  (R-): Nasluch Kolegi: 6
     Bufs (out: 0, in: 0)
     Total Out: 0 (sr: 0.00)
     Total In: 0 (sr: 0.00)
  (R-): Nasluch czasu
     Bufs (out: 0, in: 0)
     Total Out: 0 (sr: 0.00)
     Total In: 0 (sr: 0.00)
  (R-): Nasluch Klientow: 9
     Bufs (out: 0, in: 0)
     Total Out: 0 (sr: 0.00)
     Total In: 0 (sr: 0.00)
  (R-): Klawiatura
     Bufs (out: 0, in: 6)
     Total Out: 0 (sr: 0.00)
     Total In: 12 (sr: 6.00)
  (R-): PolaczenieNakladki: 203
     Bufs (out: 0, in: 0)
     Total Out: 93075402 (sr: 107.07)
     Total In: 417558208 (sr: 483.18)
     Load: 1
  (R-): PolaczenieNakladki: 202
     Bufs (out: 0, in: 0)
     Total Out: 78607106 (sr: 110.33)
     Total In: 269955367 (sr: 377.92)
     Load: 0
  (R-): PolaczenieKlienckie: 18
     Bufs (out: 0, in: 0)
     Total Out: 87117320 (sr: 339.39)
     Total In: 18318439 (sr: 72.25)

  Lacznie In: 784822033 (sr: 284.77)
  Lacznie Out: 611418784 (sr: 220.49)

Statystyki obslugi zdarzen
  Blokada: 0
  Liczba odwleczonych 0
  numerSesjiModyfikacji: 738832
  uniCount 438596
  multiCount 738903

Statystyki PunkuCentralnego 102:
  Przetworzono zdarzen: 28881037
  Jestem Glownym
```

5 Opis prezentacji systemu

W dniu 13.06.2006 miała miejsce prezentacja finalnej wersji projektu. Zademonstrowano następujące właściwości systemu:

- rozdział zadań pomiędzy instancje MySQL na podstawie analizy zapytań. Zaprezenowano różnice w zachowaniu przy operacjach czytania i zapisu oraz spójność zapisów.
- odporność na awarię jednej z instancji MySQL. Pokazano reakcje systemu na upadek pojedynczej nakladki (ze wzgl. na ścisłe powiązanie równoważne z upadkiem bazy MySQL) polegającą na propagacji informacji o awarii i kontynuacji działania.
- odporność na awarię punktu centralnego. Wykazano, że punkty centralne potrafią wyłonić między sobą nowego lidera w drodze elekcji, oraz zdolność nowego lidera do przejęcia nakładek i przyjmowania klientów.
- odporność na błąd w wykonaniu głównego punktu centralnego powodujący przerwanie funkcjonowania bez zrywania połączeń. Zasymulowano taką sytuację zawieszając proces i wykazano poprawne działanie mechanizmu „heartbeat".

Spośród elementów zaplanowanych do realizacji do dnia prezentacji nie udało się opracować następujących elementów:

- zdolności synchronizacji przyłączających się baz danych. W systemie zademonstrowanym uwzględniono możliwość przyłączania nakładek w trakcie pracy, jednak bez uaktualniania ich stanu. Synchronizacja taka musi być wykonana ręcznie – w przeciwnym przypadku system będzie dawał niespójne odpowiedzi.
- nakładki na klienta MySQL automatyzującej proces odpytywania punktów centralnych w celu znalezienia pełniącego rolę głównego. W chwili obecnej klient musi ręcznie próbować łączyć się z kolejnymi punktami centralnymi.

W ramach prezentacji wykonano też uproszczone testy wydajnościowe potwierdzające wzrost wydajności względem rozwiązania zcentralizowanego przy odczytach oraz spadek wydajności zapisów na akceptowalnym poziomie. Pełne testy zostały wykonane w terminie późniejszym, a ich rezultaty znajdują się w niniejszej dokumentacji.

6 Rozdział zadań

W tym dziale wymieniono rozdział ról poszczególnych osób w zespole, oraz przydziały zadań w poszczególnych fazach projektu.

6.1 Przydział ról

Maciek Nowak - kierownik projektu

Grzegorz Lepionka - osoba zarządzająca repozytorium projektu i odpowiedzialna za przechowywanie aktualnego backupu projektu

Andrzej Grudzień - osoba odpowiedzialna za przygotowanie demonstracji prototypu oraz przygotowanie prezentacji końcowej

Paweł Słupczyński - osoba odpowiedzialna za zarządzanie dokumentacją projektu, zbieranie i ujednolicanie dokumentacji technicznej pisanej przez innych członków zespołu

Marcin Rudowski - osoba odpowiedzialna za testowanie rozwiązania

6.2 Przydział zadań przy projekcie

Poniższa lista zawiera przydzielone zadania: osoba - faza projektu - przydzielone zadanie

Maciek Nowak:

- I - Wprowadzenie do zagadnienia realizacji niezawodności i wydajności poprzez replikację zasobów
- II - Przygotowanie paczki nr 2 z instalacją serwera klastrowego MySQL na koncie użytkownika
- III - Opis systemu do ostatecznej dokumentacji

Grzegorz Lepionka:

- I - Praktyczne wprowadzenie i instrukcja użytkownika systemu kontroli wersji subversion
- II - Program: Opracowanie journalingu i synchronizacja procesu przy "wstawianiu"
- III - Program: synchronizacja baz (mysqldump + scp + replikacja sesji klienckich)

Andrzej Grudzień:

- I - Opis i analiza już istniejących rozwiązań w ramach MySQL umożliwiających realizację klastrowości, zwiększenia niezawodności i wydajności serwera bazy danych
- II - Program: analiza zapytan SQL i przenoszenie ich miedzy nakladkami, komunikacja
- III - Program: wielosc PC + informowanie klienta o aktualnym adresie (aktywnego PC)

Paweł Słupczyński:

- I - Przygotowanie paczki nr 1 z instalacją serwera MySQL na koncie użytkownika
- II - Program: realizacja blokowania (nowych) zapytan/zapisow
- III - Program: replikacja sesji klienckich

Marcin Rudowski:

- I - Opis rozwiązania OpenSSI w kontekście realizowanego projektu
- II - Program: keep-alive + podmiana procesu nakladki
- III - Program: analiza SQL, debug, config, zarzadzanie (uruchamianie, itp)

6.3 Ustalenia spotkań roboczych

Poniżej zawarte są ustalenia spotkań roboczych ("minutki").

Oznaczanie osób:
AG - Andrzej Grudzień
GL - Grzegorz Lepionka
MN - Maciej Nowak
MR - Marcin Rudowski
PS - Paweł Supczyńki

Jeeli nie zaznaczono konkretnej osoby oznacza to, że ustalenie/propozycję/decyzję podjęto wspólnie. Generalnie wszelkie dyskusje i wniesiony do nich wkład były bardzo zrównoważone, czy?to przy dynamicznej dyskusji nie da się jednoznacznie stwierdzi?kto by autorem danego pomysłu.

07.03.2006 (spotkanie)
- rozdzia ról w zespole
- rozdzia zadań z I fazy
- ustalenie wstępnego terminu dla paczki MySQL i systemu SVN na 21.03
- załoenie przez kierownika na potrzeby projektu grupy mailowej rso4@yahoo.com

07-21.03.2006 (drogą mailową)
- wymiana kontaktów (numery GG, telefony)
- przekazywanie informacji o postępach prac

04.04.2006 (spotkanie)
- dyskusja na temat celu projektu, odroczona do czasu uzupenienia wiedzy o rozwizaniu klastrowym MySQL

11.04.2006 (spotkanie)
- kontynuacja zawieszonej dyskusji
- wyklarowanie dwóch głónych idei na realizacje projektu: żozwiązanie z warstwą pośredniczącą w PSEA"[AG] oraz źespół nakadek na serwery MySQL z centralnym punktem sterujcym"[PS]

25.04.2006 (spotkanie)
- podjęcie decyzji o rozwoju rozwizania opartego o źespół nakładek z centralnym punktem sterowania"(wartstwa pośrednia)
- zdefiniowanie systemu o następujcych cechach:
  replikacja pełna, synchroniczna,
  journaling jako sposób kontroli spójności bazy,
  wszelkie operacje (użytkownika) wykonywane przez punkt centralny,

26.04-15.05.2006 (drogą mailową)
- wymiana ustalen i specyfikacji budowanego systemu, dyskusja o napotkanych problemach (dokladny opis w drugi sprawozdaniu)

16.04.2006 (spotkanie)
- dyskusja o przyszlym projektu
- ustalenie: kontynuowanie prac wedlug wczesniejszych ustalen

16.05-12.06.2006 (drogą mailową)
- ustalanie detali implementacyjnych systemu