WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Szczegółowa koncepcja proponowanego rozwiązania i dokumentacja techniczna

Projekt – RSO MongoDB

Autorzy:

Tomasz Adamiec
Piotr Cebulski
Marek Kowalski
Mateusz Rosiewicz
Paweł Sokołowski
Marcin Wnuk

Warszawa, 2013

CRUD

FileOperations - Operacja na plikach

Klasa FileOperations odpowiada za czynności wykonywane na plikach. Jest to klasa zawierająca najczęściej powtarzane czynności, zapewniająca tworzenie i modyfikację struktury danych. Struktura bazy danych tworzona jest w folderze domowym użytkownika w folderze „exampleDB”

Zaimplementowane funkcje:

- Wczytywanie bajtów z pliku reprezentujących BSON dokument.
- Wyszukiwanie pliku zawierającego daną wartość ObjectID.
- Wyszukiwanie pliku po nazwie pliku i nazwie kolekcji.
- Tworzenie pliku na dysku.
- Zwrócenie listy plików z danego folderu.

Selector -Wybór plików z zapytania

Klasa Selector posiada funkcje zwracające listę ID plików wymienionych w zapytaniu. Porównywane są dokumenty zapisane w plikach jak i dokument stanowiący pole selektora.

Insert - Zapisywanie plików na dysku

Funkcja odpowiedzialna za dodanie danych wprowadzonych przez użytkownika do bazy przyjmuje, jako argument listę dokumentów stanowiących pole obiektu InsertMessage. Dla każdego elementu zostaje wywołana metoda zapisująca dokument na dysk w postaci bajtów. W zależności czy dana kolekcja wymieniona w InsertMessage istnieje, tworzona jest nowa, bądź aktualizowana istniejąca. Pozostałe operacje opierają się o wyżej opisaną klasę FileOperations.

Delete - Usuwanie plików z dysku

Delete dzieli się na dwa etapy. Pobranie przez Selector listy plików wybranych przez użytkownika do usunięcia oraz wyszukanie wybranych plików i ich usunięcie.

Update - Modyfikacja plików na dysku

Realizacja update polega na pobraniu nazwy pliku do zmodyfikowania z updateMessage i wyszukaniu go na dysku. Następnie dane są porównywane. Modyfikacja to stworzenie nowego dokumentu zawierającego niekonfliktowe pola oraz dodanie modyfikacji z updateMessage. Na koniec kasowany jest stary plik i tworzony nowy zawierający zmodyfikowany dokument.

Komunikacja i przetwarzanie wiadomości

Ważną cechą naszego projektu jest możliwość integracji z innymi instancjami mongoDB. Integracja ta opiera się na dostosowaniu się do protokołu sieciowego Mongo. Specyfikacja tego protokołu znajduje się http://docs.mongodb.org/meta-driver/latest/legacy/mongodb-wire-protocol/.

Aplikacja nasłuchuje połączeń na ustalonym porcie, odbiera i rozpoznaje kod wiadomości, a następnie reaguje na nie wykonując odpowiednie czynności CRUD. Treść wiadomości jest zróżnicowana i opisana w linku podanym powyżej. Jedyną cechą wspólną wszystkich wiadomości jest nagłówek, który przechowuje informację o długości wiadomości, adresacie i nadawcy wiadomości oraz o typie wiadomości. Wiadomości o następujących typach są obsługiwane w naszej bazie danych:

- Wiadomość o kodzie OP_INSERT jest komendą, odpowiada za operację Create (tworzenia) nowych plików z danymi. W nim jest zawarta cała tablica z dokumentami przeznaczonymi do przechowania w bazie.
- Wiadomość o kodzie OP_QUERY powoduje utworzenie kursora, który zwraca do klienta pewną ustaloną liczbę dokumentów, zgodnie z wzorcem zawartym w tej wiadomości.
- Wiadomość OP_GET_MORE odnosi się do konkretnego kursora, otrzymanie tej wiadomości powoduje odczytanie z kursora o pewnej ilości dokumentów i zwraca ją klientowi.
- Zarówno OP_QUERY jak i OP_GET_MORE odpowiadają operacji Read.
- Wiadomość OP_KILL_CURSOR niszczy kursor.
- Wiadomość o kodzie OP_UPDATE powoduje wykonanie operacji Update.
- Wiadomość o kodzie OP_DELETE powoduje wykonanie operacji Delete.

Opis rozwiązania

Struktura

Rozwiązanie modułu messages składa się z następujących klas:

- MessageParser: odpowiada za rozpoznanie typu wiadomości korzystając z surowej tablicy bajtowej, dostarcza również funkcje parsujące tablice bajtowe na wiadomości o konkretnym typie.
- MessageHeader: przechowuje informacje o nagłówku wiadomości.
- KillCursorsMessage, DeleteMessage, GenericMessage, GetMoreMessage, InsertMessage, QueryMessage, ReplyMessage, UpdateMessage: przechowują informacje o danych zawartych w wiadomościach.

Prasowanie wiadomości

Parsowanie wiadomości polega na odczytywaniu kolejnych zmiennych z tablicy bajtowej. Nie jest to trudne, gdyż w specyfikacji jest ściśle ustawione gdzie znajdują się liczby, gdzie napisy, i gdzie znajduje się BSON. Istnieje kilka wyjątków, gdy nie da się z góry przewidzieć treści wiadomości – występuje on w wiadomości typu OP_INSERT, OP_QUERY i OP_REPLY. Jedyne co wiadomo, to fakt że na końcu wiadomości może wystąpić pewna ilość dokumentów BSON (w przypadku OP_QUERY, maksymalnie jedna, w pozostałych przypadkach może ich być więcej). Porównuję

wówczas liczbę wczytanych bajtów z długością wiadomości. Jeżeli cała wiadomość nie została jeszcze wczytana, jest to znak że można wczytać jeszcze jeden dokument.

Dla prostoty implementacji przyjęliśmy założenie, że wszystkie wiadomości są poprawne, gdyż obsługa błędów syntaktycznych jest niekreatywna i nieciekawa, i nie pokrywa się z właściwym sensem projektu.

Wysyłanie komunikatów

Jedynym komunikatem wysyłanym przez nasz projekt jest komunikat typu OP_REPLY. Komunikat OP_REPLY jest tworzony w trakcie pracy kursora i zawiera pewną liczbę dokumentów pasujących do wzorca zapytania a także informację pomocnicze. Następnie jest on translowany do postaci binarnej za pomocą klasy ByteBuffer i wysyłany do klienta.

BSON

Czym jest BSON

BSON (skrót od Binary JSON) jest binarnym formatem stosowanym do przechowywania dokumentów. Każdy dokument składa się z zestawu elementów typu nazwa-wartość. Gdzie wartość może należeć do wielu różnych typów (takich jak int, double, string) w tym typów złożonych takich jak dokument w dokumencie czy tablica elementów.

Zastosowanie BSON

W naszym projekcie BSON stosowany jest do przesyłania, zapisywania i przeszukiwania zawartości bazy danych. Pole "_id", które każdy BSON zawiera, stosowane jest, jako klucz główny ze względu na swoją unikalność.

Ogólny zarys

Rozwiązanie modułu BSON składa się z pięciu klas:

- BSON - klasa abstrakcyjna zawierająca statyczne metody służące do parsowania tablicy bytów zawierającej BSON na BSONDocument
- BSONDocument - sparsowany dokument BSON, zawierający obiekty BSONElement różnych typów
- BSONElement - pojedyncza para nazwa - wartość
- BSONtype - wyliczenie możliwych typów wartości zawartych w BSON
- ObjectID - pole ObjectID dokumentu BSON, zawiera 4 numeryczne pola: czas utworzenia, id maszyny, id procesu, licznik

Najważniejszymi elementem rozwiązania jest klasa BSON. Zawarte w niej metody parseBSON i getBSON stanowią podstawę działania całego pakietu.

Parsowanie BSON

Parsowanie BSON (czyli przejście z tablicy bajtów na obiekt klasy BSONDocument) wykonywane jest przez metodę parseBSON. Najpierw odczytywane jest pierwsze pole dokumentu BSON, czyli jego długość.

Następnie dla każdego elementu (pary nazwa-wartość) zawartego w dokumencie proces wygląda tak samo. W pierwszej kolejności odczytywany jest jego typ i nazwa. Parsowanie wartości przechowywanej w elemencie jest zależne od typu i jest wykonywane przez metodę parse klasy BSON.

Tworzenie BSON

Tworzenie BSON (czyli tablicy bajtów z obiektu klasy BSONDocument) jest procesem odwrotnym do jego parsowania. W pierwszej kolejności tworzona jest lista, w której będą przechowywane tablice bajtów odpowiadające poszczególnym elementom zawartym w dokumencie.

Kiedy wszystkie elementy są już przetworzone na tablice bajtów (zajmuje się tym metoda getByte) obliczana jest łączna długość wszystkich tablic. W ostatnim kroku wszystkie tablice łączone są w jedną całość, do której dodawana jest na początku łączna długość i na końcu bajt zerowy.

Sharding

Podział bazy na części – shardy został zrealizowany na wzór shardingu w MongoDB. Zachowana została koncepcja shardów, serwera konfiguracyjnego oraz procesu mongos, odpowiedzialnego za równoważenie obciążenia shardów. Ze względu na znaczne uproszczenie funkcji poszczególnych elementów, niektóre aspekty zostały zmienione w stosunku do MongoDB.

Serwer Konfiguracyjny

Serwer konfiguracyjny jest centralnym elementem w bazie. W nim przechowywana jest informacja o aktualnym stanie wszystkich shardów. W implementacji serwera został wykorzystany interfejs Java RMI.

W pakiecie configserver znajdują się cztery klasy:

- Rem
- RemClient
- RemServer
- RemImpl.

Pierwsza z nich zawiera jedynie prototypy funkcji i służy do tworzenia stub'a, czyli obiektu widocznego dla klientów serwera przez rejestr RMI. Obiekt Rem pozwala klientom na wywoływanie metod: registerToConfigServer(), updateShardInfo(), registerBalancer(), getShards(). Dwie pierwsze są zarezerwowane dla shardów, a dwie ostatnie dla balancera.

Klasa RemClient jest wykorzystywana przez shardy w celu połączenia się do serwera konfiguracyjnego. Po zarejestrowaniu się w serwerze wątek klienta usypia aż do momentu wybudzenia go przez ShardMonitor. Wtedy następuje aktualizacja stanu shardu w serwerze konfiguracyjnym.

RemServer tworzy rejestr RMI z nasłuchiwaniem na domyślnym porcie 1099 i nadaje nazwę usługi widoczną przez sieć: //<serverIP>/Rem.

W klasie RemImpl znajduje się implementacja funkcji, które są zdalnie wykonywane przez shardy i balancer. Zawiera ona również informacje o shardach, w postaci mapy, której kluczami są adresy IP, a wartościami obiekty balancer.ShardInfo:

HashMap<InetAddress, ShardInfo> shards;

Dodatkowo, klasa ta korzysta z blokady java.util.concurrent.locks.ReentrantLock w celu uchronienia przed jednoczesnym zapisem i odczytem informacji o shardzie.

Shardy

Każdy shard spełnia podstawową funkcjonalność bazy – nasłuchuje na porcie 27017, tworzy wątki dla wszystkich podłączonych klientów mongo i realizuje ich zapytania. Ponadto, wykonuje on dodatkowy wątek monitorujący stan bazy. Służy do tego wspomniana wcześniej klasa ShardMonitor. Monitor skanuje zawartość folderu z bazą i zapisuje informacje o wszystkich plikach we wszystkich kolekcjach w obiekcie ShardInfo. Wylicza on przy tym obciążenie sharda, czyli łączną wielkość wszystkich plików. Kolejną funkcją ShardMonitora jest wznawianie wątku RemClient'a, w celu wysyłania zaktualizowanej informacji o stanie shardu do serwera konfiguracyjnego.

Po podłączeniu sharda do bazy rozproszonej uruchamiany jest dodatkowo wątek MigrationListener'a. Klasa MigrationListener pozwala na odbieranie żądań przeniesienia części dokumentów do innego sharda bazy. Żądania są wysyłane przez balancer tylko do sharda źródłowego. MigrationListener wysyła również odpowiednie pliki do sharda o mniejszym obciążeniu (na podstawię żądania balancera) oraz odbiera pliki z innych shardów.

Reasumując shard spałnia następujące funkcje:

- Realizowanie zapytań od klientów bazy
- Monitorowanie stanu sharda
- Przesyłanie informacji o stanie sharda do serwera konfiguracyjnego
- Odbieranie informacji o migracji danych
- Przenoszenie danych między shardami

Balancer

Podstawowym zadaniem balancera jest obliczanie polityki równoważenia. W klasie balancer.Balancer następuje wyszukanie usługi Rem serwera konfiguracyjnego i podłączenie do niego za pomocą metody registerBalancer(). Następnie rozpoczyna się wykonywanie rund balancera. W każdej rundzie pobierana jest najbardziej aktualna informacja o stanie bazy za pomocą funkcji getShards().

Gdy balancer otrzyma informacje o shardach tworzony jest obiekt BalancerPolicy. BalancerPolicy na podstawie danych zawartych w HashMap<InetAddress, ShardInfo> wyznacza politykę równoważenia i zapisuje ją w postaci obiekty MigrateInfo.

Algorytm wyznaczania polityki równoważenia przebiega w dwóch etapach. Najpierw, znajdywane są shardy o najmniejszym i największym obciążeniu. Jeżeli różnica między nimi jest większa niż zakładany próg, to następuje drugi etap – wyznaczanie dokumentów do przeniesienia. W tym etapie znajdywane są dokumenty, których zsumowana wielkość będzie niemniejsza niż połowa różnicy

obciążenia shardów. Wybierane są dokumenty o wielkości najbardziej zbliżonej do średniej wielkości dokumenty w mocno obciążonym shardzie.

Po wyznaczeniu polityki przez BalancerPolicy, w klasie Balancer następuje sprawdzenie czy należy coś przenieść – czy MigrateInfo nie jest puste. Jeśli nie, to znaczy, że zawiera ono adres sharda mocno obciążonego i mniej obciążonego i nazwy dokumentów, które należy między nimi przenieść. Na końcu rundy, informacje zawarte w MigrateInfo są przesyłane do sharda mocno obciążonego na port 28017, na którym nasłuchuje MigrationListener.

Balancer'a nie trzeba dołączać do bazy, nie jest on niezbędny do jej funkcjonowania.

Sharding:

1. Shard rejestruje się w serwerze konfiguracyjnym.
2. ShardMonitor aktualizuje informacje o shardzie.
3. RemClient wysyła informacje do serwera konfiguracyjnego.
4. Balancer pobiera informacje o wszystkich shardach.
5. BalancerPolicy wyznacza politykę i tworzy MigrateInfo.
6. MigrateInfo wysyłane jest do sharda.
7. Shard wysyła dokumenty do innego sharda.