

Rozproszone systemy operacyjne

Opis interfejsu mongod

Autorzy:

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

Warszawa, 2013

Opis działania

Naszym zadaniem jest napisanie aplikacji naśladującej działanie MongoDb. Jednym z głównych składników tej bazy jest **mongod –** program reprezentujący instancje serwera odpowiedzialny za wydajne zapisywanie, aktualizacje i przechowywanie danych. Czym jest ten **mongod**?

Wszystko zaczyna się w pliku <https://github.com/mongodb/mongo/blob/master/src/mongo/db/db.cpp> w metodzie static int mongoDbMain(int argc, char\* argv[], char\*\* envp).

W dużym skrócie metoda ta odpowiada za sparsowanie parametrów wejściowych programu, oraz uruchomienie nasłuchu na porcie (domyślnym 27017, lub podanym podczas wywołania programu). Następnie odbierane są komunikaty i podejmowane odpowiednie akcje z nimi związane.

Czym są te komunikaty?

Ten paragraf zostanie opisany w oparciu o <http://docs.mongodb.org/meta-driver/latest/legacy/mongodb-wire-protocol/> . Jest to kilka rodzajów wiadomości (client request i server responses). Wszystkich z nich poprzedzone są standardowym nagłówkiem:

struct MsgHeader {  
 int32 messageLength; // total message size, including this  
 int32 requestID; // identifier for this message  
 int32 responseTo; // requestID from the original request  
 // (used in reponses from db)  
 int32 opCode; // request type - see table below  
}

Oryginalne komentarze dosyć dobrze oddają znaczenie poszczególnych pól nagłówka. Nagłówek ten zdefiniowany jest w <https://github.com/mongodb/mongo/blob/master/src/mongo/util/net/message.h>. Typ wiadomości określony jest przez pole opCode. Możliwe są następujące wartości:

|  |  |  |
| --- | --- | --- |
| opCode | Value | Comment |
| OP\_REPLY | 1 | Reply to a client request. responseTo is set |
| OP\_MSG | 1000 | generic msg command followed by a string |
| OP\_UPDATE | 2001 | update document |
| OP\_INSERT | 2002 | insert new document |
| RESERVED | 2003 | formerly used for OP\_GET\_BY\_OID |
| OP\_QUERY | 2004 | query a collection |
| OP\_GET\_MORE | 2005 | Get more data from a query. See Cursors |
| OP\_DELETE | 2006 | Delete documents |
| OP\_KILL\_CURSORS | 2007 | Tell database client is done with a cursor |

Klient może wysłać wszystkie wiadomści poza OP\_REPLY, która zarezerowana jest dla serwera w odpowiedzi na OP\_QUERY lub OP\_GET\_MORE, wszystkie pozostałe wiadomości nie otrzymują odpowiedzie. RequestID nadawany jest przez klienta lub bazę danych. Jego działanie można przedstawić na przykładzie. Klient wysyła wiadomość OP\_QUERY z RequestID = 4, baza odpowiada na nie OP\_REPLY z responseTo = 4.

Dokładna struktura pozostałych wiadomości omawiana jest we wspomnianym wcześniej <http://docs.mongodb.org/meta-driver/latest/legacy/mongodb-wire-protocol/>. Nie będę tu tego przepisywał. Co jeszcze warto dodać, dokumenty będące częścią wiadomości przesyłane są w formacie BSON. Ważnym jest zapoznanie się z jego specyfikacją: <http://bsonspec.org/#/specification>. Pozostałe dane jak np.: stringi i liczby przesyłane są w formacie zgodnym ze specyfikacją BSON.

Teraz przejdę do przykładu ilustrującego działanie protokołu. Będzie to wysłanie wiadomości, która za pomocą drivera do C# wygląda następująco:

MongoCollection<Entity> collection =   
 database.GetCollection<Entity>("entities");   
var entity = new Entity { Name = "Tom" };  
collection.Insert(entity);

Lub za pomocą shella MongoDB:

db.entites.insert({Name: „Tom”})

Szensanstkowa reprezentacja wiadomości:

46-00-00-00-04-00-00-00-00-00-00-00-D2-07-00-00-00-00-00-00-74-65-73-74-2E-65-6E-74-69-74-69-65-73-00-24-00-00-00-07-5F-69-64-00-51-75-A7-20-41-B6-76-09-20-E2-9A-08-02-4E-61-6D-65-00-04-00-00-00-54-6F-6D-00-00

Wszystkie liczby całkowite (int) zapisywane są za pomocą little indian. Tak więc pierwsze 16 bajtów jest to nagłówek wiadomości. Pierwsze cztery bajty oznaczają jej długość: 46-00-00-00 = 70. Natępny cztery bajty to id wiadomości: 04-00-00-00 = 4. Kolejne cztery to id odpowiedzi: 00-00-00-00 = 0. Jest to request klineta, więc wartość 0 nie dziwi. Kolejne cztery: D2-07-00-00 = 2002, czyli zgodnie z oczekiwaniami jest to typ wiadomości: OP\_INSERT.

Aby zrozumieć dalszy ciąg komunikatu należy zapoznać się z strukturą wiadomości OP\_INSERT. Wygląda ona następująco:

struct {  
 MsgHeader header; // standard message header  
 int32 flags; // bit vector - see below  
 cstring fullCollectionName; // "dbname.collectionname"  
 document\* documents; // one or more documents to insert into the   
 //collection  
}

A więc kolejne 4 bajty zawierają wektor flag. 00-00-00-00 – czyli żadna z flag nie została ustawiona. Następnie mamy nazwę kolekcji. Jest to ciąg znaków zakodowany za pomocą UTF-8 zakończony 0: 74-65-73-74-2E-65-6E-74-69-74-69-65-73-00 = „test.entities”. Ostatnim elementem jest BSON reprezentujący dokument. Pozwolę sobie go nie tłumaczyć.

Gdzie to jest przetwarzane

Miejscem w którym otrzymana wiadomość jest czytana z socketa jest <https://github.com/mongodb/mongo/blob/master/src/mongo/util/net/message_port.cpp>. Metoda bool MessagingPort::recv(Message& m).

Miejscem w którym posiadamy już otrzymaną wiadomość i następuje seria „ifów” rozdzielających sterowanie pomiędzy handlery odpowiednich operacji jest metoda assembleResponse w : <https://github.com/mongodb/mongo/blob/master/src/mongo/db/instance.cpp>

Przykłady przychodzących pakietów

Wszystkie wiadomości nie zawierają pierwszych 4 bajtów określających długość wiadomości, która jest wczytywana chwilę wcześniej tak aby wiadomo było ile bajtów należy wczytać z socketa. Trochę jest polsko-angielsko, w trakcie się zorientowałem i już nie chciałem zmieniać.

Dodatkowo nie są to wszystkie wiadomości wysyłane z shella mongo. Często jednej komendzie towarzyszą jeszcze inne wiadomości, np.: diagnostyczne.

* 1. Insert

Komenda z shela mongo:  
db.test.insert({Imie : „Jan”, Nazwisko : „Nowak”})  
powoduje pojawienie się wiadomości:  
08 00 00 00 ff ff ff ff d2 07 00 00 00 00 00 00 74 65 73 74 2e 74 65 73 74 00 38 00 00 00 07 5f 69 64 00 51 7e aa 16 c4 53 f2 6e 59 9d 2b 52 02 49 6d 69 65 00 04 00 00 00 4a 61 6e 00 02 4e 61 7a 77 69 73 6b 6f 00 06 00 00 00 4e 6f 77 61 6b 00 00

Teraz zagnieżdżony insert:  
db.test.insert({Imie : „Jan”, Nazwisko : „Nowak”, address : {city:”Warszawa”, street:”nowowiejska”}})  
Wiadomość:  
0b 00 00 00 ff ff ff ff d2 07 00 00 00 00 00 00 74 65 73 74 2e 74 65 73 74 00 71 00 00 00 07 5f 69 64 00 51 7e aa ba c4 53 f2 6e 59 9d 2b 53 02 49 6d 69 65 00 04 00 00 00 4a 61 6e 00 02 4e 61 7a 77 69 73 6b 6f 00 06 00 00 00 4e 6f 77 61 6b 00 03 61 64 64 72 65 73 73 00 30 00 00 00 02 63 69 74 79 00 09 00 00 00 57 61 72 73 7a 61 77 61 00 02 73 74 72 65 65 74 00 0c 00 00 00 4e 6f 77 6f 77 69 65 6a 73 6b 61 00 00 00

* 1. Find

db.test.find({“address .city” : “Warszawa”})

26 00 00 00 ff ff ff ff d4 07 00 00 00 00 00 00 74 65 73 74 2e 74 65 73 74 00 00 00 00 00 00 00 00 00 20 00 00 00 02 61 64 64 72 65 73 73 2e 63 69 74 79 00 09 00 00 00 57 61 72 73 7a 61 77 61 00 00

Przykład z projekcją:

db.test.find({“address .city” : “Warszawa”},{Imie:1, Nazwisko:1})

2a 00 00 00 ff ff ff ff d4 07 00 00 00 00 00 00 74 65 73 74 2e 74 65 73 74 00 00 00 00 00 00 00 00 00 20 00 00 00 02 61 64 64 72 65 73 73 2e 63 69 74 79 00 09 00 00 00 57 61 72 73 7a 61 77 61 00 00 25 00 00 00 01 49 6d 69 65 00 00 00 00 00 00 00 f0 3f 01 4e 61 7a 77 69 73 6b 6f 00 00 00 00 00 00 00 f0 3f 00

* 1. Update

db.test.update({Imie: “Anna”}, {$set: {“address.city”:”Krakow”}})

38 00 00 00 ff ff ff ff d1 07 00 00 00 00 00 00 74 65 73 74 2e 74 65 73 74 00 00 00 00 00 14 00 00 00 02 49 6d 69 65 00 05 00 00 00 41 6e 6e 61 00 00 28 00 00 00 03 24 73 65 74 00 1d 00 00 00 02 61 64 72 65 73 73 2e 63 69 74 79 00 07 00 00 00 4b 72 61 6b 6f 77 00 00 00

* 1. Remove

db.test.remove({Imie: “Anna”})

3b 00 00 00 ff ff ff ff d6 07 00 00 00 00 00 00 74 65 73 74 2e 74 65 73 74 00 00 00 00 00 14 00 00 00 02 49 6d 69 65 00 05 00 00 00 41 6e 6e 61 00 00

* 1. Indeks

db.test.ensureIndex({“Imie”:1})

Tworzenie indeksu korzysta z OP\_Insert.

3e 00 00 00 ff ff ff ff d2 07 00 00 00 00 00 00 74 65 73 74 2e 73 79 73 74 65 6d 2e 69 6e 64 65 78 65 73 00 51 00 00 00 07 5f 69 64 00 51 7e b4 c1 c4 53 f2 6e 59 9d 2b 56 02 6e 73 00 0a 00 00 00 74 65 73 74 2e 74 65 73 74 00 03 6b 65 79 00 13 00 00 00 01 49 6d 69 65 00 00 00 00 00 00 00 f0 3f 00 02 6e 61 6d 65 00 07 00 00 00 49 6d 69 65 5f 31 00 00

db.test.getIndexes()

42 00 00 00 ff ff ff ff d4 07 00 00 00 00 00 00 74 65 73 74 2e 73 79 73 74 65 6d 2e 69 6e 64 65 78 65 73 00 00 00 00 00 00 00 00 00 17 00 00 00 02 6e 73 00 0a 00 00 00 74 65 73 74 2e 74 65 73 74 00 00

Krótkie wnioski

Zaprezentowane przeze mnie przykłady są jedynie bardzo małym zbiorem. Wszystkie dokumenty mogą być dowolnie zagnieżdżone. Find może przyjmować wiele operatorów, typu mniejsze, większe, zawiera itp…

Sama obsługa indeksów to też niezła masakra. Uważam, że na początek dobrze by było gdyby działało nam zapisywanie, usuwanie i proste zapytania, nawet tylko działające na zasadzie operatora =.