WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Rozproszone systemy operacyjne

Projekt – RSO MongoDB

Autorzy:

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

Warszawa, 2013

1. Wstęp

MongoDB jest to otwarty i nierelacyjny system zarządzania bazą danych napisany w języku C++. Prace nad systemem rozpoczęła firma 10gen w 2007 roku. Pierwsze wydanie stabilnej wersji nastąpiło w 2009 roku. MongoDB charakteryzuje się dużą skalowalnością oraz wydajnością. Nie posiada ściśle określonej struktury obsługiwanych baz danych. Dane składowane są jako dokumenty w stylu JSON, co umożliwia aplikacjom bardziej naturalne ich przetwarzanie, przy zachowaniu możliwości tworzenia hierarchii raz indeksowania.

MongoDB charakteryzuje się następującymi właściwościami:

- jednorodne wsparcie dla standardu unicode ,
- obsługa danych w innych kodowaniach w formacie binarnym,
- duża liczba obsługiwanych typów danych,
- obsługa kursorów,
- zapytania ad-hoc,
- zapytania do zagnieżdżonych pól dokumentów,
- indeksowanie,
- wsparcie dla agregacji danych,
- możliwość składowania plików w bazie,
- architektura zaprojektowana z myślą o łatwej replikacji.

Wewnętrznym językiem do definiowania zapytań oraz funkcji agregujących jest JavaScript wykonywany bezpośrednio przez serwer MongoDB. Interfejsy programistyczne pozwalające obsługiwać bazy MongoDB powstały dla wszystkich wiodących języków programowania, w tym dla C, C++, C#, Javy, PHP, Perla, Pythona, Rubiego.

MongoDB posiada ograniczone wsparcie dla transakcji - ich zasięg jest ograniczony do zmian w pojedynczym dokumencie, aczkolwiek zmiany te mogą być bardzo skomplikowane. Z tego powodu część użytkowników ogranicza zastosowanie MongoDB do niekrytycznych danych informacyjnych, pozostawiając obsługę krytycznych operacji (np. obsługa zamówień w sklepie) relacyjnym bazom danych, gdzie takie ograniczenia nie występują.

1.1 Przykłady zastosowania MongoDB

W bazach tego typu zamiast tradycyjnych wierszy używa się pojęcia dokumentu, zawierającego pary klucz-wartość. Rozwiązanie to jest bardzo elastyczne, a co za tym idzie dzięki niemu możliwe jest bardzo wierne odtwarzanie rzeczywistych danych w systemach informatycznych. MongoDb jest w ostatnim czasie coraz częściej wdrażane. Z dobrodziejstw tej bazy korzystają już: SourceForge, diaspora, New York Times czy Wielki Zderzacz Hadronów. Ponadto MongoDb można równolegle używać z innymi SZBD, np.: MusicNation.com używa MongoDB do przechowywania plików wideo, a Server Density przeszedł z MySQL na rzecz MongoDB w aplikacji służącej do monitoringu stanu serwerów. Businessinsider.com wykorzystuje Mongo m.in. w swojej wyszukiwarce.

MongoDB najlepiej nadaje się do pewnych zadań takich jak:

- przechowywana danych serwisów internetowych: MongoDB jest w bardzo dobry sposób operuje na danych. Zapytania do bazy, tj. insert, update i select wykonywane są dużo szybciej jak w przypadku modelu relacyjnego. Ponadto MongoDB posiada świetną obsługę replikacji, co jest kluczowe w przypadku obsługi dużych serwisów.

- „Cachowanie”: MongoDB można wykorzystać jako pamięć podręczna w np. infrastrukturze intranetowej. Dzięki pamięci podręcznej po restarcie systemu lub bazy sieć nie zostanie przeciążona.
- Duża ilość danych, które nie są wartościowe: niektóre dane, tj. logi nie powinny być przechowywane w relacyjnej bazie danych. Jest to nieefektywne. Zamiast przechowywać je w plikach zewnętrznych – jako magazyn tego typu danych można użyć MongoDB.
- Zadnia, które wymagają wysokiej skalowalności: MongoDB rewelacyjnie sprawdza się w wieloserwerowych konfiguracjach.

2. Cechy MongoDB

W celu utworzenia bazy danych wystarczy polecenie `use` <nazwa bazy> , baza jest gotowa do użycia. Domyślnie przy użyciu wiersza poleceń (mongo) używana jest baza *test.*

Baza jest w stu procentach dynamiczna. Ma to swoje plusy i minusy. Bazy danych i kolekcje (odpowiedniki tabel) tworzone są dynamicznie – oznacza to, że baza pozwala na zapis dokumentów z dowolną strukturą. Nie zabezpiecza więc on programisty przed pomyłkami takimi jak literówki. Jeżeli raz napiszemy

```
db.users.insert({"name": "Student" });
```

(komenda wstawiająca dokument – odpowiednik rekordu – do kolekcji *user* z polem – kolumną - *name* która ma wartość *“Student”*) to Mongo sprawdzi czy w używanej przez nas bazie jest już kolekcja *users*, jeżeli nie to ją utworzy. Gdy za drugim razem wpiszemy

```
db.usrs.find();
```

(komenda listująca wszystkie dokumenty z kolekcji *usrs*) to Mongo nie nakrzyczy na nas, tylko wyświetli pusty zbiór wyników. Gdy wpiszemy

```
db.usrs.insert( {"name": "Student" } );
```

to Mongo stworzy kolekcję *users*, wstawi do niej dokument z polem *name* o wartości *Student* i będzie się cieszyć z dobrze wykonanego zadania.

Każdy dokument przy operacji wstawiania dostaje generowany przez bazę identyfikator. Dotyczy to też sterowników do PHP – po wywołaniu inserta obiekt który dodawaliśmy dostaje automatycznie pole _id .

2.1 API

find

```
db.collection.find();
```

Metoda *find* jest to odpowiednikiem operacji *select* w relacyjnych bazach danych. Składnia tej metody została przedstawiona poniżej.

```
db.products.find( { qty: { $gt: 25 } } );
```

Powyższy zapis należy rozumień jako: znajdź wszystkie dokumenty o nazwie products w których wartość dla pola gdy jest większa niż 25.

findAndModify

```
db.collection.findAndModify( {
                                query: <document>,
                                sort: <document>,
                                remove: <boolean>,
                                update: <document>,
                                new: <boolean>,
                                fields: <document>,
                                upsert: <boolean>
                            } );
```

Przykład wykorzystania został zamieszczony poniżej.

```
db.people.findAndModify( {
    query: { name: "Tom", state: "active", rating: { $gt: 10 } },
    sort: { rating: 1 },
    update: { $inc: { score: 1 } }
} )
```

Powyższy zapis należy rozumień jako: znajdź dokumenty z kolekcji people, gdzie wartość dla pola name jest równa Tom, dla pola state jest równa active, dla pola rating jest większa niż 10. Następnie posortuj rosnąco dokumenty z pierwszego zapytania jeśli takie istnieją i wybierz pierwszy dokument. Następnie zwiększ wartość dla pola score o 1.

findOne

```
db.collection.findOne();
```

Powyższe polecenie zwraca tylko jeden dokument spełniający warunek zadany w kwerendzie. Jeśli wiele dokumentów spełnia warunek, wyświetlony zostanie tylko pierwszy dokument zgodnie z porządkiem jaki jest on umieszczony w bazie.

Insert

```
db.collection.insert(document);
```

Przykład zastosowania:

```
db.products.insert( { item: "card", qty: 15 } );
```

Operacja ta wstawi nowy dokument do kolekcji protucts z polem item z wartością card oraz polem qty z wartością 15, oraz z unikalnym kluczem ObjectID.

Save

```
Save();
```

Metoda *save* aktualizuje istniejący dokument, lub wstawia dokument w zależności od parametru. Przykład zastosowania:

```
db.products.save( { item: "book", qty: 40 } );
```

Jeśli wartość w dokumencie dla pola _id będzie podana oraz wartość ta będzie już istniała wówczas dokument zostanie zaktualizowany z wyszczególnionymi wartościami dla danych pól. Jeśli w dokumencie nie będzie istniała wartość dla pola _id wówczas do dokumentu zostaną dodane wyszczególnione pola z ich wartościami.

Update

```
db.collection.update(query, update[, options]);
```

Metoda *update* aktualizuje istniejący dokument, lub dokumenty w kolekcji. Domyślnie *update* aktualizuje jeden dokument. Przykład zastosowania:

```
db.products.update( { item: "book", qty: { $gt: 5 } }, { $set: { x: 6 },
$inc: { y: 5} } );
```

Metoda ta aktualizuje dokument w kolekcji products, która odpowiada kryterium zapytania i ustawia wartość pola x na 6 oraz zwiększa wartość dla pola y o 5. Wszystkie pozostałe pola w dokumencie pozostają niezmienione.

2.2 Indeksy

Indeksy zostały stworzone w celu zapewnienia wysokiej wydajności najczęściej wykonywanych zapytań. Są one podobnie skonstruowane jak w relacyjnych bazach danych. Indeksy w MongoDB posiadają następujące właściwości:

- Są definiowane na poziomie kolekcji
- Możliwość tworzenie indeksów składających się z pojedynczego pola lub indeksów składających się z wielu pól[1]
- Znacznie zwiększają prędkość wykonywania się zapytać. Powodują jednak pewien narzut na operacje zapisu.
- W ich implementacji użyte są B-drzewa
- Każde zapytanie może użyć tylko jednego indeksu. Optymalizator wybiera najlepszy. Można nadpisać wybór optymalizatora używając metody `cursor.hint()`.
- Indeks pokrywa zapytanie gdy: wszystkie pola indeksu użyte w zapytaniu są częścią indeksu i wszystkie pola w dokumentach pasujących do zapytania znajdują się w tym samym indeksie. Gdy indeks pokrywa zapytanie MongoDB nie ma potrzeby przeglądania dokumentów w celu zrealizowania operacji odczytu i może zwrócić wyniki jedynie na podstawie indeksu.
- Gdy operacja aktualizowania dokumentu nie zmienia wartości pól wchodzących do indeksu i nie zmienia rozmiaru dokumentu indeks nie jest aktualizowany.
- Używanie zapytań z indeksami zmniejsza liczbę dokumentów przechowywanych w pamięci i ma duże znaczenie dla wydajności.

MongoDB oferuje następujące typy indeksów:

- **_id index –** jest to indeks unikalny i tworzony jest domyślnie dla każdej kolekcji. Domyślną wartością dla `_id` jest *ObjectId* tworzony przy każdej operacji `insert()`.
- **Secondary indexes –** są to indeksy tworzone za pomocą polecenia `ensureIndex()`. Mogą się one składać z pojedynczych pól jak i być computed indexes.
- **Indexes on Sub-documents –** są to indeksy składające się z poddokumentów
- **Indexes on Embeded Fields –** używają pól poddokumentów
- **Computed indexes –** składają się z wielu pól

[1] *Computed indexes*

- **Multikey indexes –** posiadają pola będące wartością tablicową
- **Unique indexes –** indeksy z unikalną wartością. MongoDB nie pozwala na zapis wielu dokumentów z taką samą wartością tego typu indeksu.
- **Sparse indexes -** indeks wymagający aby dokument posiadał wszystkie pola zawarte w indeksie. Dla dokumentów niezawierających wszystkich pól indeks nie jest tworzony.
- **Hashed indexes –** indeksy który przechowują *hash* pól wchodzących w skład indeksu.

Na tworzone indeksy nałożone są pewne ograniczenia.

- Kolekcja może posiadać maksymalnie 64 indeksy.
- Rozmiar indeksu nie może przekraczać 1024 bajtów.
- Nazwa indeksu łącznie z przestrzenią nazw nie może być dłuższa niż 128 znaków

Chociaż indeksy znacznie przyspieszają wykonywanie się zapytań, mają negatywny wpływ na czas zapisów, aktualizacji i usuwania danych. Przy większości tych operacji indeks musi zostać zaktualizowany dla całej kolekcji. Szczególnie ostrożnie należy rozważać tworzenie indeksów w aplikacjach w których te operacje będą miały dominującą rolę.

2.3 Replikacja danych

MongoDB oferuje dwa rodzaje replikacji danych. Są nimi:

- *Replica set replication*
- *Master-slave replicatio*

Master-slave replication jest klasycznym sposobem replikacji danych z jednym nadrzędnym węzłem i wieloma podrzędnymi. Podobne podejście oferowane jest przy *replica set replication,* jednak zawiera ono mechanizmy do automatycznego przełączania awaryjnych węzłów. Pierwszy z wymienionych sposobów uważany jest już za przestarzały i poza sytuacjami w których istnieją racjonalne wskazania na używanie go zaleca się używanie drugiego z wymienionych sposobów.

Replica set jest klastrem złożonym z co najmniej dwóch instancji *mongod*[2]**.** Co najwyżej jedna z nich uważana jest za podstawową, a pozostałe ze drugorzędne. Wszystkie zapisy danych przeprowadzane są poprzez podstawowy węzeł, dane w węzłach drugorzędnych są replikowane w sposób asynchroniczny.

Ten sposób replikacji danych w MongoDB zwiększa złożoność systemu, ale zarazem zwiększa również jego dostępność, prędkość odczytów oraz ułatwia przeprowadzenie niektórych z zadań administracyjnych takich jak kopie zapasowe danych. Jedną z najważniejszych cech *replica set* jest automatyczne przełączanie głównego węzła w przypadku jego awarii. Pozostałe węzły przeprowadzają przezroczysty dla użytkownika wybór nowego węzła nadrzędnego. Mechanizm ten wprowadza jednak pewne ograniczenia. W jednym *replica set* może być maksymalnie 12 węzłów, w tym maksymalnie siedem mających prawo głosu.

Właściwości opisujące cechy węzłów

Każdy z członków *replica set* jest opisany następującymi właściwościami:

- **Secondary-Only –** taki węzeł nie może zostać nigdy węzłem głównym
- **Hidden** – węzeł nie jest widoczny przez aplikacje klienckie

[2] **mongod** – podstawowy proces systemu MongoDB. Zarządza on formatem danych, obsługą danych oraz wykonuje operacje w tle.

- **Delayed –** operacje z głównego węzła replikowane są z określonym opóźnieniem
- **Arbiters –** węzeł nie zawierający danych, występuje jedynie w celu brania udziału w wyborach
- **Non-Voting –** węzeł nie ma prawa głosu

Przełączanie węzła głównego i głosowanie

Aby doszło do automatycznego przełączenia węzła głównego muszą być spełnione co najmniej dwa warunki:

- Nadrzędny członek klastra zaczyna nie odpowiadać
- Większość z drugorzędnych członków klastra jest połączona każdy z każdym

Jeżeli powyższe warunki są spełnione węzły drugorzędne wybierają spośród siebie nowy węzeł główny. Dzieje się to poprzez procedurę głosowania. Członkowie klastra poza członkami niegłosującymi mają jeden głos, włączając w to członków **Secondary-Only**, **Hidden** i **Arbiters** i wybierany jest pierwszy z członków który zdobędzie większość głosów (może się zdarzyć, że któryś z członków ma ustawiony wyższy priorytet, wtedy ma on większe szanse na zostanie wybranym na nowy węzeł główny. Nie jest to jednak zalecana rozwiązanie). W wyborach uczestniczą także członkowie niegłosujący.

Spójność danych

W przypadku węzła głównego wszystkie dane odczytywane są spójne z ostatnią operacją zapisu. Gdy preferencje odczytu zostaną tak ustawione, że pozwalają na odczyt z członków drugorzędnych możliwe jest odczytanie danych, które nie są całkowicie zgodne z ostatnim zapisem do głównego serwera. W normalnej, domyślnej sytuacji nie można zapewnić takiej spójności. Można jednak tak skonfigurować driver MongoDB, aby operacja zapisu była uznana za zakończoną z sukcesem dopiero w momencie gdy zapis zostanie przeprowadzony we wszystkich węzłach.

W pewnych przypadkach awarii węzła głównego i zmiany jego na jeden z węzłów drugorzędnych możliwe jest zaistnienie danych jeszcze niezreplikowanych do pozostałych członków klastra. W takiej sytuacji członek, które poprzednio był głównym członkiem i ponownie dołącza do klastra dokonuje operacji odwrócenia[3] danych w celu zachowania ich spójności. Dane takie zapisywane są do pliku *BSON* w katalogu *dbpath.* Dane te mogą być odczytane za pomocą polecania *bsondump*[4]**.** Zaktualizowanie nowego węzła głównego o te dane musi zostać wykonane ręcznie.

2.4 Auto-Sharding

Sharding jest innym podejściem do rozproszenia danych niż replikacja. W tym przypadku dane nie są powielane a składowane różnych porcji danych na różnych maszynach. Bardzo częstym i zalecanym podejściem jest połączenie możliwości jakie daje *sharding* i replikacja danych. W tym podejściu każda z maszyn będąca odpowiadająca za składowanie podzielonych danych jest replikowana.

Pojęcie *sharding* jest ściśle związane z sharding klastrami. Klaster taki jest zbiorem węzłów zajmujących się składowaniem danych. Składa się on z następujących komponentów:

- **Shards** – kontener odpowiedzialny za składowanie podzbioru danych. Każdy z takich kontenerów jest albo pojedynczą instancją *mongod* albo jednym *replica set.*

[3] rollback
[4] **Bsondump –** konwertuje dane w postaci BSON w formę czytelną dla człowiek, np.: JSON

- **Config servers –** każdy z takich serwerów jest instancją *mongod* odpowiedzialną za utrzymywanie metadanych związanych z klastrem.
- **mongos instances –** odpowiadają one za zarządzenie zapisami i odczytami wewnątrz klastra. Aplikacje nie mają bezpośredniego dostępu z zewnątrz do kontenerów w których przechowywane są dane. Dostęp ten uzyskują właśnie przez instancje *mongos.*

Włączenie opcji *sharding* na poziomie bazy danych zapewnia możliwość rozpraszania kolekcji na różne węzły. Aby zapewnić możliwość rozproszenia danych wewnątrz jednej kolekcji należy wyznaczyć te kolekcje jawnie wyznaczyć. Dla każdej takiej kolekcji należy także wyznaczyć *shard key* – jest to klucz według którego dane rozdzielane są na różne węzły klastra.

Shared key

Shard key podobnie jak indeks może składać się z jednego lub wielu pól. MongoDB tworzy zakresy kluczy według których przydzielane są one do poszczególnych węzłów. W każdym z węzłów zakresy te dzielone są na jeszcze mniejsze porcje zwane *chunks* o określonej z góry pojemności. Gdy zostanie ona przekroczona zakres kluczy zostaje zmniejszony i jedna porcja zostaje podzielona na dwie.

Wybór odpowiedniego klucza jest bardzo ważny w celu zapewniania wydajności. Ważną właściwością klucza jest jego liczność, czyli liczba możliwych wartości jaką klucz może przyjąć. Ma ona duże znaczenie przy osiąganiu dobrego rozproszenia danych.

Przy określonej z góry, małej liczbie możliwych wartości klucza możliwe jest nawet, że dane nie będą mogły być rozdzielone pomiędzy wszystkie węzły. W przypadku gdy zbiór możliwych wartości jest większy niż liczba węzłów w klastrze to niebezpieczeństwo nie istnieje; pojawia się jednak inne. Dane mogą być nierówno rozłożone względem wartości klucza. Dla jednego z nich może to być kilka egzemplarzy, z kolei dla innego kilkaset tysięcy. W takim przypadku niemożliwym stanie się równomierne rozłożenie danych pomiędzy węzły. Najlepszą sytuacją jest gdy klucz jest unikalną wartością obiektu. Wtedy MongoDB może dowolnie dopierać zakresy kluczy tak aby dane były równomiernie rozłożone.

Sytuacja gdy kluczem jest unikalna wartość obiektu ma także swoje wady. Jedną z nich jest izolacja zapytań. Gdy instancja *mongos* otrzymuje zapytanie do wykonania na podstawie metadanych zawartych w *config server* nawiguje jedynie do odpowiednich węzłów klastra. Ich liczba powinna być jak najmniejsza, w idealnej sytuacji powinien być to jeden z nich. Należy tak dopierać klucze, aby używane one były w najczęstszych i najbardziej kluczowych dla aplikacji zapytań.

Znalezienia odpowiedniego klucza spełniającego wyżej wymienione zalecenia może być ciężkie. W tym celu oblicza się klucze składające się z kilku wartości z dokumetu.