

Rozproszone systemy operacyjne

Równoważenie obciążenia shardów

Autorzy:

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

Warszawa, 2013

Spis treści

[Mongos 4](#_Toc356166857)

[Balancer 4](#_Toc356166858)

[Klasy Balancera 4](#_Toc356166859)

[1.1 Balancer 4](#_Toc356166860)

[1.2 BalcancerPolicy 5](#_Toc356166861)

[1.3 ShardInfo 5](#_Toc356166862)

[1.4 MigrateInfo 6](#_Toc356166863)

[1.5 ChunkInfo 6](#_Toc356166864)

[Algorytm 7](#_Toc356166865)

Mongos

W bazie składającej się z klastra shardów, na każdym z serwerów uruchomiona jest instancja programu mongos. Program ten pośredniczy w komunikacji bazy mongod, z klastrem shardów. Spełnia on przy tym dwie podstawowe funkcje: kieruje żądania (zapis i odczyt) do odpowiedniego shardu – query routing oraz równoważy obciążenie wszystkich shardów - balancer. Mongos śledzi rozłożenie danych w bazie zbierając informację z serwerów konfiguracyjnych.

Balancer

Balancer to wykonujący się w tle proces, który ma na celu utrzymanie takiej samej liczby kawałków bazy na każdym serwerze należącym do klastra shardów. Każdy mongos ma uruchomionego balancera, ale tylko jeden (na jednym z serwerów) jest aktywny w danej chwili. Aby balancery nie działały jednocześnie używany jest mechanizm *DistributedLock.* Gdy któryś z serwerów dostaje sygnał zniesienia blokady, wykonuje on rundę balancera. W jednej rundzie następuje stwierdzenie czy występuje nierówność w obciążeniu serwerów i w razie potrzeby wysyłane jest żądanie przeniesienia co najwyżej jednego kawałka (*chunk*).

Klasy Balancera

Mechanizm balancera posiada dwa pliki nagłówkowe \mongo-master\src\mongo\s\balance.h i \mongo-master\src\mongo\s\balancer\_policy.h. Zdefiniowane są w nich klasy Balancer i BalancerPolicy, a także klika klas pomocniczych.

* 1. Balancer

class Balancer : public BackgroundJob

{

typedef MigrateInfo CandidateChunk;

typedef shared\_ptr<CandidateChunk> CandidateChunkPtr;

int \_balancedLastTime;

scoped\_ptr<BalancerPolicy> \_policy;

bool \_init();

void \_doBalanceRound( DBClientBase& conn, vector<CandidateChunkPtr>\* candidateChunks );

int \_moveChunks(const vector<CandidateChunkPtr>\* candidateChunks,

bool secondaryThrottle,

bool waitForDelete);

void \_ping( DBClientBase& conn, bool waiting = false );

bool \_checkOIDs();

};

\_balancedLastTime – Liczba ostatnio przeniesionych kawałków.

\_policy – Polityka, czyli wskaźnik na kawałek do przeniesienia z informacją skąd dokąd przenieść, lub NULL.

\_init() – Łączy się z serwerem konfiguracyjnym w celu otrzymania informacji o shardach. Funkcja jest wykonywana za każdym razem, gdy rozpoczyna się runda. Właściwie, wywołuje \_checkOIDs i sypie wyjątki.

\_doBalanceRound() – Wykonuje rundę balancera. conn to adres serwera konfiguracyjnego. Najpierw sprawdza czy jest jakaś kolekcja podzielona na shardy do zbalansowania, w tym celu sprawdza czy kolekcja ma przydzielony *shardkey*. Następnie pobiera listę shardów wraz z maksymalnym możliwym obciążeniem oraz aktualnym obciążeniem. Dla każdej balansowanej kolekcji sprawdza czy jest zalecane przesunięcie czegokolwiek.

\_moveChunks() – Przesuwa kawałki. candidateChunks to wektor kawałków możliwych do przesunięcia wypełniony przez funkcję \_doBalanceRound(). Przegląda kandydatów i wybiera interesujących (??) funkcją ChunkManager::findInterestingChunk, następnie próbuje go przenieść Chunk::moveAndCommit.

\_ping() – Odzywa się do serwera konfiguracyjnego i potwierdza, że balancer jest uruchomiony.

\_checkOIDs() – Pobiera listę shardów i sprawdza czy wszystkie są odrębnymi procesami (czy się nazwy nie popsuły)

* 1. BalcancerPolicy

class BalancerPolicy

{

static MigrateInfo\* balance( const string& ns, DistributionStatus& distribution, int balancedLastTime );

private:

static bool \_isJumbo( const BSONObj& chunk );

};

balance() – Główna funkcja, która wybiera kawałek do przeniesienia. Wskaźnik \_policy jest uzupełniany przez tą funkcję. ns to namespace, a DistributionStatus zawiera informacje o stanie shardów w kolekcji.

Jumbo kawałek to taki którego nie da się przenieść.

* 1. ShardInfo

class ShardInfo

{

void addTag( const string& tag );

bool hasTag( const string& tag ) const;

bool isSizeMaxed() const;

bool isDraining() const { return \_draining; }

bool hasOpsQueued() const { return \_hasOpsQueued; }

long long getMaxSize() const { return \_maxSize; }

long long getCurrSize() const { return \_currSize; }

string getMongoVersion() const { return \_mongoVersion; }

private:

long long \_maxSize;

long long \_currSize;

bool \_draining;

bool \_hasOpsQueued;

set<string> \_tags;

string \_mongoVersion;

};

isSizeMaxed() – Czy shard jest już maksymalnie obciążony.

isDraining() – Czy shard jest opróżniany, jeśli tak to wiadomo, że trzeba z niego przesuwać kawałki.

hasOpsQueued() – Czy shard ma jakieś zdania do wykonania, zazwyczaj nie można wtedy nic z niego usuwać.

getMaxSize() – Maksymalny rozmiar sharda.

getCurrSize() – Zwraca obciążenie sharda.

* 1. MigrateInfo

struct MigrateInfo

{

const string ns;

const string to;

const string from;

const ChunkInfo chunk;

};

ns – Namespace

to – Do którego sharda przenieść kawałek.

From – Z którego sharda zabrać kawałek.

Chunk – Który kawałek prznieść.

* 1. ChunkInfo

struct ChunkInfo

{

const BSONObj min;

const BSONObj max;

}

Min – Pierwszy dokument w kawałku, inclusive.

Max – Ostatni dokument w kawałku, non-inclusive.

Algorytm

Proces równoważenia shardów rozpoczyna się w funkcji Balancer::run(). Na początku tej funkcji następuje próba inicjalizacji balancera czyli \_init(). Polega ona na połączniu się z serwerem konfiguracyjnym i odebraniu najnowszych informacji o shardach (do tego służy funkcja \_checkOIDs() i Shard::getAllShards() ). W razie niepowodzenia, próba inicjalizacji jest ponawiana co 60 sekund.

Po udanej inicjalizacji następuje zarejestrowanie się balancera w mechanizmie blokad:

DistributedLock balanceLock( config , "balancer" );

Teraz następuje wejście do głównej pętli procesu. Wszystkie kolejne akcje, są powtarzane w każdej rundzie.

Na początku rundy wykonywana jest funkcja \_ping(). Po niej następuje załadowanie najświeższych posiadanych informacji o shardach - Shard::reloadShardInfo().

Dalej balancer próbuje założyć blokadę:

dist\_lock\_try lk( &balanceLock , "doing balance round" );

Jeżeli nie uda mu się, to znaczy, że inny balancer jest aktywny i wątek usypiany jest na 30 lub 6 sekund (w zależności od konfiguracji). Po upłynięciu tego czasu zaczyna się nowa iteracja głównej pętli.

Jeżeli uda się założyć blokadę, rozpoczyna się właściwa runda balancera. Tworzony jest wektor kawałków możliwych do przeniesienia:

vector<CandidateChunkPtr> candidateChunks;

Jest on wypełniany przez funkcję \_doBalanceRound( conn.conn() , &candidateChunks ). Jeśli istnieją kawałki możliwe do przeniesienia, to jest to wykonywane za pomocą funkcji:

\_balancedLastTime = \_moveChunks(&candidateChunks, . . . );

Na końcu każdej rundy ponownie wykonywany jest \_ping(), w celu poinformowania serwera konfiguracyjnego, że balancer jest aktywny i nie czekał.

Teraz przyjrzymy się bliżej funkcji \_doBalanceRound(). Na początku tej funkcji następuje sprawdzenie czy istnieją jakiekolwiek kolekcje do zbalansowania. Polega to na odpytaniu dostępnych kolekcji o to czy posiadają klucz shardingowy (shard „key”).

Dalej tworzona jest lista wszystkich shardów, dla których będzie równoważone obciążenie. Lista zawiera wszystkie potrzebne informacje (m. in. maksymalny rozmiar i aktualne obciążenie).

vector<Shard> allShards;

Shard::getAllShards( allShards );

Dla każdego sharda:

ShardStatus status = s.getStatus();

shardInfo[ s.getName() ] = ShardInfo( s.getMaxSize(),

status.mapped(),

s.isDraining(),

status.hasOpsQueued(),

s.tags(),

status.mongoVersion() );

Teraz rozpoczyna się pętla, w której dla każdej kolekcji ustalana jest polityka równoważenia. Zanim jednak nastąpi ustalenie polityki, wykonywane jest przyporządkowanie kawałków bazy do shardów, w których się znajdują:

map< string,vector<BSONObj> > shardToChunksMap;

Proces ten składa się z wielu czynności sprawdzających poprawność mapowania, które nie będą tutaj omówione.

Na końcu pętli znajdywany jest kawałek do przesunięcia:

CandidateChunk\* p = \_policy->balance( ns, status, \_balancedLastTime );

Ustalanie polityki w funkcji BalancerPolicy::balance(), przebiega w trzech etapach. Na początku sprawdzane jest czy któryś z shardów jest opróżniany. Jeśli istnieje taki shard, to ma on priorytet – trzeba zabrać z niego wszystkie kawałki. Nie ma tutaj znaczenia, który kawałek zostanie przesunięty, bo i tak trzeba zabrać wszystkie.

Jeżeli nie było żadnego opróżnianego shardu, to następuje drugi etap, w którym sprawdza się poprawność tagów każdego kawałka bazy. Tag stanowi informację o tym czy kawałek znajduje się w odpowiednim shardzie. Jeśli kawałek z niewłaściwym tagiem zostanie znaleziony, zachodzi próba przeniesienia go. Kawałek ten może być jednak duży (\_isJumbo()) i wtedy nie może zostać przeniesiony. Może się też okazać, że nie ma sharda, do którego można by go przenieść (getBestReceieverShard()).

Jeżeli nie zaszła żadna z powyższych sytuacji, to wykonywany jest etap trzeci. Na początku ustalany jest próg (treshold), powyżej którego równoważenie będzie w ogóle wykonywane. Próg oznacza różnicę w ilości kawałków najbardziej i najmniej obciążonego sharda. Ustalanie skąd dokąd wykonać przesunięcie wygląda następująco:

string from = distribution.getMostOverloadedShard( tag );

unsigned max = distribution.numberOfChunksInShardWithTag( from, tag );

string to = distribution.getBestReceieverShard( tag );

unsigned min = distribution.numberOfChunksInShardWithTag( to, tag );

const int imbalance = max - min;

if ( imbalance < threshold ) continue;

Jeśli imbalance jest niemniejszy od progu to wykonywane jest przesunięcie kawałków.

const vector<BSONObj>& chunks = distribution.getChunks( from );

return new MigrateInfo( ns, to, from, chunks[j] );

Sposób ustalania wielkości progu w balancerze wygląda następująco:

int threshold = 8;

if ( balancedLastTime || distribution.totalChunks() < 20 )

threshold = 2;

else if ( distribution.totalChunks() < 80 )

threshold = 4;