

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](#_Toc354590990)

[Balancer 4](#_Toc354590991)

[1.1 Klasy Balancera 4](#_Toc354590992)

[Balancer 4](#_Toc354590993)

[BalcancerPolicy 5](#_Toc354590994)

[ShardInfo 5](#_Toc354590995)

[MigrateInfo 6](#_Toc354590996)

[ChunkInfo 6](#_Toc354590997)

[1.2 Algorytm 6](#_Toc354590998)

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*).

* 1. 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.

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)

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.

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.

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.

ChunkInfo

struct ChunkInfo

{

const BSONObj min;

const BSONObj max;

}

Min – Pierwszy dokument w kawałku, inclusive.

Max – Ostatni dokument w kawałku, non-inclusive.

* 1. Algorytm