Problematyka rozproszonych systemów - przypadek *Apache Cassandra*

Marek Lewandowski
Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska
marek.m.lewandowski@gmail.com

Streszczenie. *Artykuł omawia nierelacyjne bazy danych i twierdzenia z nimi związane takie jak* CAP, BASE*. W szczególności omawia cechy szczególne i budowę bazy* Apache Cassandra. Apache Cassandra *jest kanoniczną nierelacyjną bazą danych, której zrozumienie znacząco ułatwia zrozumienie większości pozostałych nierelacyjnych baz danych.*

1. Motywacja. Nierelacyjne bazy danych powstały, w celu rozwiązania nowego rodzaju problemów, którym nie były w stanie sprostać tradycyjne bazy danych. Problemy to bardzo duża, szybko przyrastająca ilość danych oraz potrzeba obsługi równie dużego obciążenia systemu.

Ilość danych, wyrażana może być, aż peta bajtach ($1PB = 1000TB$), co może sprawiać problemy jeśli chcielibyśmy obsłużyć taką ilość danych jednym komputerem. W związku z tym poruszamy się po dziedzinie klastrów serwerów, a nie pojedynczych maszyn. Pojedyncza maszyna to również pojedynczy punkt awarii. Takie i inne problemy adresowane są przez nierelacyjne bazy danych.

Często podczas omawiania nierelacyjnych baz danych przytacza się przypadek firmy *Amazon*. *Amazon* oświadczył, że wzrost czasu odpowiedzi w ich serwisie o 0.1 sekundy powoduje spadek sprzedaży o 1%. Wysoka wydajność i niski czas dostępu są możliwe dzięki nierelacyjnym bazom danych oraz innym pobocznym technikom. Warto zatem pamiętać o biznesowej wartości tych rozwiązań, wiedzieć jak działają i jakim prawom podlegają.

2. Spójność i dostępność W klasycznych bazach danych, często słyszymy o tym, że baza jest spójna. Taka spójność oznacza, że wszystkie mechanizmy w bazie takie jak więzy integralności, klucze obce, ograniczenia na kolumny są spełnione. Chcę przedstawić inną definicję spójności.

Spójność oznacza jedną wersję danych, a więc wszystkie takie same dane, są zawsze w identycznej wersji. Dostępność rozumiana jest jako zdolność systemu do obsługi żądania zapisu/odczytu danych i zwrócenia odpowiedzi.

ACID charakteryzuje klasyczne bazy danych, które posiadają bardzo wysoką spójność. *BASE* to akronim, który należy omówić.

- większość danych dostępna przez cały czas (ang. Basically Available)
- dane wystarczająco świeże (ang. Soft state)
- osiągnięcie spójności odsunięte w czasie, ale osiągalne (ang. Eventually consistent)

ACID i *BASE* reprezentują dwa podejścia projektowe po przeciwnych końcach spektrum spójność-dostępność. Naturalnie *ACID* znajduje się po stronie spójności i reprezentuje klasyczne podejście do baz danych. *BASE* znajduje się po przeciwnej stronie spektrum i reprezentuje systemy z dużą dostępnością. Dzięki tym modelom wybór pomiędzy dostępnością, a spójnością jest jawny.

3. Twierdzenie CAP. Brewer opublikował w 1999 roku [5] twierdzenie przedstawiające wybór dwóch z trzech cech w rozproszonych bazach danych. Dwie cechy już znamy. Należy wytłumaczyć czym jest odporność na partycje.

Partycja w rozprosznym systemie występuje, wtedy gdy część węzłów w klastrze nie ma łączności z co najmniej jednym węzłem, inaczej mówiąc klaster jest podzielony na dwie części, które nie mogą się ze sobą komunikować. Taka sytuacja może wystąpić w co najmniej dwóch przypadkach. Część maszyn może ulec awarii i tym samym zniknie z klastra na długi czas, a więc będzie niedostępna z punktu widzenia pozostałych maszyn. Inną możliwością jest brak łączności spowodowany awarią sieci. Z punktu widzenia maszyn w klastrze obie sytuacje są identyczne i nie da się ich rozróżnić. W obu tych sytuacjach występuje partycja rozdzielająca maszyny na dwie grupy. Twierdzenie Brewera mówi, że można mieć jedynie dwie cechy z poniższych:

- spójność,
- dostępność,
- odporność na partycje.

W przypadku rozproszonych systemów brak odporności na partycje to pojedyczny punkt awarii. Wystarczy sobie wyobrazić, że pojedyncza maszyna ulega awarii, a baza nie jest w stanie dalej funkcjonować. Z tego powodu rozważa się wybór pomiędzy dostępnością, a spójnością. Odporność na partycje powinna być zawsze. Można zatem mówić o binarnym wyborze.

Binarny wybór został zaprezentowany w oryginalnej postaci twierdzenia. Przez ponad dekadę powstało wiele typów systemów, które skupiają się na innych kombinacjach właściwości.

3.1. Twierdzenie dzisiaj. Wokół twierdzenia pojawiło się wiele nieporozumień. Przez lata odkryto sporo niuansów. Właściwości nie są binarne, a bliżej im do wartości ciągłych. Dostępność jest w zakresie procentowym. Istnieje wiele poziomów spójności. Węzły w klastrze mogą nie zgadzać się co do tego czy partycja faktycznie występuje. Autor opublikował 12 lat później artykuł [1], który rozwiewa większość wątpliwości.

Dzisiaj możemy przedstawić twierdzenie w lepszy sposób. Podczas wystąpienia partycji niemożliwa jest idealna spójność i idealna dostępność. Dzięki specjalnej obsłudze systemu podczas partycji, można mieć więcej niż tylko jedną cechę. Wymaga to użycia odpowiednik technik. Jedną z technik jest ograniczenie możliwych operacji, tylko do tych, które nie zniszczą spójności. Przykładem rozprosznego systemu, który stosuje taką technikę jest *Google Docs*, który podczas braku połączenia (partycji), udostępnia tylko część narzędzi tesktowych.

Jeśli partycja nie występuje system może mieć idealną dostępność i spójność. Dopiero podczas wystąpienia partycji należy dokonać wyboru pomiędzy spójnością, a dostępnością lub mieć specjalny tryb operacji podczas partycji. Nie dokonanie wyboru w krótkim czasie to wybór spójności, ponieważ żądanie nie zostanie obsłużone w odpowiednim czasie, a więc z perspektywy klienta system nie będzie dostępny.

4. Parametry nierelacyjnych baz danych. Nierelacyjne bazy danych często posiadają trójkę parametrów, których manipulacja pozwala osiągnąć różne tryby pracy systemu. Repliką nazywamy węzeł odpowiedzialny za przechowywanie tych samych danych.

- N - liczba replik, do których zapisuje się dane,
- W - liczba potwierdzeń zapisu potrzebna do poprawnego zapisu,
- R - liczba replik, z których się czyta.

Tabela 1 pokazuje przypadki charakterystyczne ustawień parametrów. Jak widać, może uzyskać bardzo różne zachowanie bazy danych w zależności od wartości powyższych parametrów.

R	W	Charakterystyka
1	N	Wysoka spójność, bardzo szybki odczyt
N	1	Wysoka spójność, bardzo szybki zapis
1	1	Niska spójność, bardzo szybki odczyt i zapis

Tabela 1. Przypadki charakterystyczne wartości parametrów N,R,W.

4.1. Reguła spójności. Istnieje reguła dotycząca parametrów, która jeśli spełniona gwarantuje zachowanie spójności. Jeśli nierówność $R + W \quad N + 1$ jest spełniona, to każdy odczyt otrzyma najnowszą wersję danych. Ze wzoru widać, że zapis i odczyt zazębia się na większości replik, dlatego też zawsze będą dostępne najświeższe wersje danych.

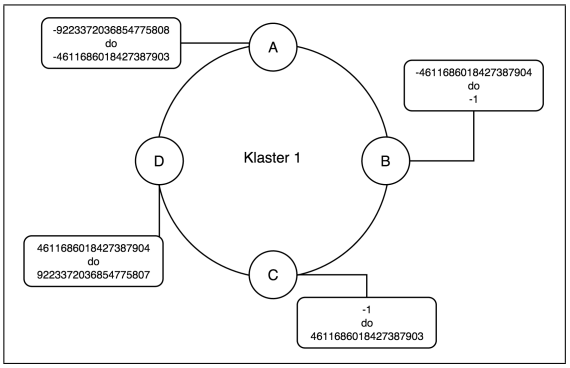
Można zadać pytanie w jaki sposób reprezentowana jest świeżość danych. Istnieje kilka technik [7] [4], które to umożliwiają. Poznamy jedną z nich użytą w bazie *Apache Cassandra*.

5. Apache Cassandra. Nierelacyjna baza danych, zapewniająca wysoką dostępność i odporność na partycje. Cechuje ją liniowa skalowalność. Jeśli mając 6 węzłów możemy obsłużyć 200000 żądań zapisu na sekundę to mając 12 obsłużymy 400000 żądań. Architektura *Cassandry* spełnia założenia *Dynamo* [3], a więc węzły są równorzędne i symetryczne. Dane są replikowane na wielu węzłach. Skaluje się horyzontalnie, poprzez dodawanie kolejnych węzłów w trybie pracy ciągłej. Wykorzystuje takie same techniki jak *Dynamo* dla problemów takich jak przywrócenie węzła ze stanu awarii, wykrycie nowego węzła w klastrze, wykrycie awarii węzła w klastrze, obsługa tymczasowej awarii. *Cassandra* od *Dynamo* różni się modelem danych. *Dynamo* posiada model danych, który w dużym uproszczeniu jest mapą klucz-wartość. *Cassandra* posiada model danych oparty o *Google BigTable* [2], w którym to dane są przechowywane w kolumnach.

5.1. Liniowa skalowalność. *Cassandra* posiada liniową skalowalność dzięki mechanizmowi spójnego mieszania kluczy. [6]. Węzły tworzą pierścień. Pierścień mieści w sobie wszystkie wartości kluczy funkcji mieszającej. Węzły odpowiedzialne są za kolejne partycje kluczy. W przypadku pierścienia z 4 węzłami, pierścień podzielony jest na 4 partycje, gdzie każda partycja przypada na jeden węzeł. Sytuacja ta jest zilustrowana na rysunku 1. Wartości kluczy w zakresie od -2^{63} do $2^{63} - 1$ rozłożone są na pierścieniu.

Aby odwołać się do danych w *Cassandrze*, należy podać klucz partycjonujący. Z wartości tego klucza liczony jest skrót funkcją mieszająca. Typowo jest to funkcja mieszająca *Murmur3*. Wynikiem jest wartość, która znajduje się na pierścieniu, a więc wiadomo, który węzeł odpowiada za te dane. Dostęp do danych ma zatem złożoność $\mathcal{O}(1)$.

Każdy węzeł jest równorzędny, można zatem pytać o dane dowolny węzeł, a ten przekieruje żądanie do odpowiedniego węzła.



Rysunek 1. Pierścień dla 4 węzłów

5.2. Replikacja danych W *Apache Cassandra* dane są zorganizowane w tabelach, które w kolei należą do przestrzeni kluczy. Przestrzeń kluczy stanowi odpowiednik schematu bazy danych, w którym znajduja się tabele. Przestrzeni kluczy można mieć dowolnie dużo, podobnie jak tabel w każdej przestrzeni kluczy. Przy tworzeniu przestrzeni kluczy podaje się współczynnik replikacji N. Domyślną wartością jest $N = 3$.

Wracając do pierścienia z rysunku 1, dane z danej partycji replikowane są na kolejnych partycjach zgodnie ze wskazówkami zegara. Mianowicie, zakładając domyślny współczyn-

nik replikacji, dane z węzła *A* zreplikowane zostaną na węzłach *B* i *C*.

5.3. Natywny model danych. *Cassandra* posiada model danych oparty o *BigTable*. W tym modelu danych dane są w kolumnach, które należą do wierszy, które należą do tabel. Nie istnieje żaden schemat danych, ponieważ każdy wiersz może mieć dowolne różne kolumny od innych wierszy. Kolumny w wierszach są posortowane po nazwach.

Kolumna w *Apache Cassandra* różni się jednak od zwykłej kolumny bazo-danowej. Kolumna, inaczej zwana komórką posiada:

- nazwę - kolumny są sortowane po nazwach
- wartość - miejsce na 2*Gb* danych
- znacznik czasowy - ostatnia data modyfikacji, bądź wstawienia komórki
- czas życia - opcjonalna wartość; wyrażona w sekundach, po upływie komórka zniknie z bazy danych

Znacznik czasowy służy do określenia, która komórka jest najnowsza, a więc jest to mechanizm pozwalajacy na spójny odczyt danych.

Każdy wiersz identyfikowany jest kluczem wiersza. Wynik funkcji skrótu od klucza wiersza określa pozycję na pierścieniu. Każdy wiersz może posiadać maksymalnie 2 miliardy komórek. Wiersz musi się fizycznie mieścić na jednym węźle. Choć każda komórka może posiadać maksymalnie 2*Gb* danych, w praktyce nie zaleca się przechowywania większej ilości danych niż kilka *Mb* z racji tego, że dostęp do danych nie jest dostępem strumieniowym. Cała wartość jest odczytywana do pamięci i przekazywana dalej, a więc duże wartości mogą szybko wykorzystać pamięć, co doprowadzi do degradacji ogólnej wydajności węzła.

Czas życia jest bardzo przydatnym parametrem, ponieważ ma dużo praktycznych zastosowań. Pierwszym z brzegu jest ważność hasła. Jeśli hasło może być ważne 1 miesiąc, to można ustawić odpowiedni czas życia dla komórki przechowującej hasło. Jeśli przy odczycie nie będzie tej komórki to znaczy, że hasło wygasło. Podobny model można zastosować w wielu innych biznesowych sytuacjach.

5.4. Model CQL. Model ten został stworzony jako dodatkowa warstwa abstrakcji nad natywnym modelem danych. *Cassandra Query Language* pozwala na organizację danych w komórkach fizycznego wiersza, która przypominać będzie relacyjną bazę danych. Każdy fizyczny wiersz zawierać będzie wiele wierszy *CQL*. Wiersze *CQL* identyfikowane są poprzez klucz złożony z dwóch części. Cześć partycjonująca klucza jest dokładnie tym samym kluczem w przypadku modelu natywnego i identyfikuje wiersz w pierścieniu. Cześć sortująca identyfikuje wirtualny wiersz reprezentowany przez zestaw komórek w wierszu fizycznym. Cały mechanizm polega na odpowiednim nazewnictwie komórek. Klucz sortujący dołączony jest do nazwy każdej komórki fizycznej wraz z jej właściwą nazwą. Inaczej mówiąc, wartość klucza sortującego stanowi prefiks nazwy kolumny. Dzięki temu, że komórki posortowane są po nazwie, wiersz *CQL* to zestaw kolejnych komórek o tym samym prefiksie.

W obecnej chwili model *CQL* jest bardzo rozwijany i wszystko wskazuje na to, że *CQL* będzie głównym medium modelowania danych. Należy jednak pamiętać o prawdziwej, natywnej reprezentacji danych.

Choć *CQL* brzmi podobnie do *SQL* należy pamiętać, że *CQL* nie posiada większości funkcjonalności, którą oferują relacyjne bazy danych. Nie istnieją żadne złączenia, zagnieżdżone zapytania, sumowania oraz większość innych rzeczy. *CQL* oferuje jedynie zapytanie typu *SELECT ... FROM ... WHERE ....* Każde zapytanie odczytu powinno dotyczyć jednego lub maksymalnie kilku wierszy. Inaczej tracona jest wydajność, ponieważ wymagane będzie zapytanie o dane kilka innych replik. Fizyczne wiersze rozsiane są na innych węzłach. Można się zatem zastanawiać w jaki sposób projektować systemy, które są banalne w małej skali w przypadku tradycyjnych baz danych.

5.5. Modelowanie danych Poznaliśmy już modele danych. Warto powiedzieć w jaki sposób, można budować typowe aplikacje operate o *Cassandrę*. Przy projektowaniu aplikacji wiemy w jaki sposób chcemy pytać o dane. Jeśli ktoś wybrał tą bazę to znaczy, że ma z tym problem. Przy modelowaniu tabel należy się zatem skupić na zapytaniach. Za przykład weźmy system, w których zbieramy dane o użytkownikach. Jednym z częstych zapytań jest zapytanie o użytkowników w danym mieście. Chcemy otrzymać listę użytkowników z danego miasta wraz z ich imieniem i nazwiskiem.

Bazę można traktować inaczej jako fabrykę indeksów. Dla użytkoników i wszystkich ich danych zbudowana zostanie jedna tabela. Dla powyższego zapytania zbudowana zostanie druga tabela, której kluczami wierszy będą nazwy miast. Przy dodawaniu użytkownika, będzie on również dodawany do drugiej tabeli pod odpowiednim miastem. Oprócz samego identyfikatora użytkownika pozwalającego na dostęp do jego wiersza skopiowane zostaną także imię i nazwisko. Tym sposobem odpowiedź na zapytanie przy takim modelu polega na odczycie jednego wiersza i wyświetleniu listy.

Podsumowując, modeluje się dane poprzez denormalizację i budowę wielu indeksów pokrywających wszystkie przypadki użycia danych, a więc w praktyce wszystkie zapytania odczytu.

6. Przypadki użycia. *Cassandra* idealnie nadaje się do przechowywania wszelkiego rodzaju danych ze znacznikami czasowymi, takimi jak logi, zdarzenia, pomiary. Poprzez użycie odpowiednik technik można sobie poradzić z obsługą bardzo dużej liczby żądań zapisu na sekundę. *Cassandra* wykorzystywana jest do budowy silników rekomendacji, dynamicznych, wielowymiarowych rankingów. Przechowywane są w niej dane geograficzne.

6.1. Przypadek *I20 Water* Firma ta wspiera różne firmy, które są dostawcami wody na świecie. Wspiera je poprzez inteligente rozwiązania monitorujące ciśnienie i wiele innych czynników w wodociągach, które odpowiednio zarządzają całą siecią wodociągową. Wodociąg posiada bardzo dużo fizycznych mierników, które mierzą te parametry i przesyłają je poprzez sieć *GPS*, do głównego systemu. *Cassandra* pozwala na zapis olbrzymiej ilości danych oraz szybki odczyt i analizę. Do analizy wykorzystywane są narzędzia, dobrze zintegrowane z *Cassandrą*. Dzięki mierzeniu różnych parametrów i obserwację awarii sieci wodociągówej oraz szybką reakcję na takie sytuacje firma ta oszczędza 100 milionów litrów wody dziennie oraz obniżyła zużycie energii potrzebnej do pompowania wody o 20%.

Literatura

[1] E Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012.

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[4] C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, 10(1):56–66, 1988.

[5] Armando Fox and Eric A. Brewer. Harvest, yield, and scalable tolerant systems.

In *In HotOS-VII*. Society Press, 1999.

[6] David R Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web Caching with Consistent Hashing. *Computer Networks ()*, pages 1203–1213, 1999.

[7] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM ()*, 21(7):558–565, 1978.