Amazon Dynamo

Tomasz Klukowski

2.04.2009

Plan prezentacji

Lista życzeń

Potrzebujemy magazynu danych

- dostępny non-stop
  - odporność na awarie sprzętowe
  - brak przerw związanych z replikacją i uspójnianiem danych
- wysoce skalowalny
- operacje wykonywane możliwie szybko (Jak to mierzyć?)
- elastyczny
  - będą z niego korzystać różne web serwisy
  - każdy wymaga od magazynu czegoś nieco innego
- łatwy w obsłudze

Czemu relacyjna baza danych odpada?

- Słabo skalowalna
- Spójność jest ważniejsza od dostępności
- Wszystkie zapytania po kluczu głównym
- Konfiguracja wymaga dużej wiedzy i doświadczenia

Założenia i wymagania Dynamo

- Model zapytań
  - Po kluczu głównym
  - Dane trzymane w postaci binarnej
  - Każda operacja dotyczy dokładnie jednego wpisu
  - Żadnych relacji między danymi
  - Zakłada się, że porcja danych jest < 1 MB
- ACID
  - Zapewnienie ACID powodowałoby słabą dostępność
  - Nie zapewnia izolacji
  - Adresowany dla aplikacji wymagających słabej spójności

Założenia i wymagania Dynamo

- Wydajność
  - Każda usługa korzystająca z Dynamo ma inne wymagania
  - Wymagania w konkretnej instancji określone na podstawie SLA (Service Level Agreement)
- Inne założenia
  - Dynamo będzie używane tylko wewnętrznie
  - Środowisko działania jest przyjazne
  - Nie trzeba autoryzacji ani autentykacji
  - Każdy serwis działa na własnej instancji Dynamo

Service Level Agreement

- Formalny kontrakt między klientem a usługą
- Dotyczy cech związanych z funkcjonowaniem systemu
  - Oczekiwana ilość żądań dla konkretnego API
  - Oczekiwany czas realizacji żądania
  - np. usługa gwarantuje, że w ciągu 300ms odpowie na 99,9% żądań przy szczytowym obciążeniu 500 żądań na sekundę

Service Level Agreement

- SLA pełni ważną rolę w rozproszonej architekturze Amazona
  - By zrealizować żądanie usługi generują własne żądania do innych usług
  - Opóźnienie na którymś poziomie powoduje opóźnienie realizacji całości żądania
- Każdy (a nie tylko statystyczny) użytkownik ma być zadowolony z działania systemu

Inne decyzje

- Rozwiązywanie konfliktów w spójności danych przy odczytach (write ma być zawsze dostępny i nieodrzucany)
- Inkrementalna skalowalność
- Symetria - wszystkie węzły pełnią tę samą rolę
- Decentralizacja
- Heterogeniczność - jeśli jakiś serwer jest wydajniejszy, należy to wykorzystać

Mechanizmy użyte by zapewnić kluczowe cechy

Problem	**Technika**	**Zalety**
Rozpraszanie danych	Consistent hashing	Inkrementalna skalowalność
Wysoka dostępność zapisów	Zegary wektorowe z uzgadnianiem podczas odczytów	Rozmiar wersji nie zależy od częstości uaktualnienia danych
Czasowe awarie	Sloppy Quorum oraz hinted handoff	Zapewnia wysoką dostępność i trwałość kiedy część replik nie jest dostępych

Mechanizmy użyte by zapewnić kluczowe cechy

Problem	**Technika**	**Zalety**
Przywracanie po trwałej awarii	Antyentropia używająca drzew Merkle'a	Synchronizuje robieżne repliki w tle
Wykrywanie członkostwa i awarii	Protokół członkostwa i wykrywania awarii oparty na algorytmi plotkującym	Zapewnia symetrię i brak centralnego rejestru do trzymania informacji o członkostwie i żywotności poszczególnych węzłów

Interfejs

- get(key)
  - Znajduje repliki obiektu związanego z kluczem
  - Zwraca pojedynczy obiekt lub listę obiektów w konfliktujących wersjach razem z kontekstem
- put(key, context, object)
  - Na podstawie wartości klucza zapisuje replikę obiektu na dysk
  - Kontekst koduje metadane o obiekcie (wersja obiektu, itp.)
  - Kontekst jest trzymany razem z obiektem

Klucz

- Dynamo traktuje klucz i obiekt dostarczony przez wołającego jako tablicę bajtów
- Na podstawie hasha MD5 z klucza, generuje 128-bitowy identyfikator
- Identyfikator jest użyty by ustalić węzły w systemie, które są odpowiedzialne za obsługę tego klucza.

Algorytm partycjonowania

- Dynamo ma skalować się inkrementalnie
- Potrzebujemy mechanizmu dynamicznie dzielącego dane między węzły
- Consistent hashing rozkłada obciążenia między węzły

Consistent hashing

- Wyniki funkcji haszującej traktowane jako pierścień
- Najwyższa wartość haszu przechodzi na najmniejszą
- Każdy węzeł losuje wartość z przestrzeni haszów, która oznacza pozycje na pierścieniu
- Dane znajdujemy haszując klucz, znajdując pozycję tego hasza na pierścieniu i ruchu zgodnym ze wskazówkami zegara aż do napotkania pierwszego węzła

Consistent hashing

- Każdy węzeł odpowiada za dane między nim, a jego poprzednikiem
- Zmiany ilości węzłów dotyczą tylko bezpośrednich sąsiadów
- Losowość prowadzi do niejednolitego rozkładu danych i obciążeń
- Nie wykorzystuje różnej wydajności węzłów

Wariant Consistent hashing w Dynamo

- Każdy węzeł otrzymuje wiele punktów na pierścieniu
- Koncepcja węzłów wirtualnych (tokenów)
  - Wygląda jak zwykły węzeł
  - Każdy węzeł może być odpowiedzialny za wiele wirtualnych

Wariant Consistent hashing w Dynamo

- Kiedy węzeł się dezaktywuje, obciążenie jest równomiernie rozdzielane między pozostałe węzły
- Kiedy się uaktywnia, dostaje z grubsza po tyle samo tokenów do obsługi od każdego innego węzła
- Ilość tokenów na węzeł jest ustalana na podstawie jego pojemności, kosztu i fizycznej infrastruktury

Replikacja

- Każda porcja danych replikowana na N hostów
  - N - parametr ustalany per instancja Dynamo
- Zarządca porcji danych wyznaczany na podstawie haszu klucza
- Pozostałe to N-1 następników (niewirtualnych) zgodnie z ruchem wskazówek zegara

Replikacja

- Listę fizycznych węzłów odpowiedzialnych za trzymanie ustalonego klucza nazywamy listą preferencji
- Każdy węzeł potrafi określić które węzły mają być na tej liście dla określonego klucza
- Lista preferencji zawiera więcej niż N węzłów na wypadek awarii

Wersjonowanie danych

- Dynamo zapewnia tylko końcową spójność, co pozwala na asynchroniczną aktualizację replik
- Wywołanie put() może się zakończyć przed aktualizacją wszystkich replik, przez co następujące get() może zwrócić obiekt sprzed aktualizacji
- Gdy nie ma awarii, czas propagowania aktualizacji jest ograniczony
- W niektórych przypadkach (pady serwerów, podział sieci) aktualizacje mogą nie dotrzeć do wszystkich replik w założonym czasie

Wersjonowanie danych

Niektóre aplikacje w platformie Amazona mogą tolerować takie niespójności, np:

- Akcja “dodaj do koszyka” nigdy nie może zostać odrzucona, ale jeśli dodamy do starszej wersji koszyka, to operacja dalej jest znacząca i powinna zostać zachowana
- Nie powinno to wyrugować obecnie niedostępnego stanu koszyka, która również może zawierać zmiany, które powinny zostać zachowane
- Gdy klient chce dodać/usunąć obiekt z koszyka i ostatnia wersja koszyka nie jest dostępna, obiekt jest dodawany/usuwany do starszej wersji
- Konflikty wersji są rozwiązywane później

Wersjonowanie danych

- By zapis był zawsze akceptowany, Dynamo traktuje wynik modyfikacji jako nową, niezmienną wersję danych
- W tym samym czasie w systemie może być wiele wersji tego samego obiektu
- Zazwyczaj nowe wersje nakładają się na wcześniejsze i system sam potrafi określić ważną wersję (uzgadnianie syntaktyczne)

Wersjonowanie danych

- W wyniku awarii lub równoczesnych zapisów, mogą pojawić się kilka wersji będących w konflikcie
- W takim wypadku system nie może pogodzić wersji i to klient musi je uzgodnić (uzgadnianie semantyczne)
- Aplikacje korzystające z Dynamo powinny być przygotowane na otrzymanie wielu różnych wersji tego samego obiektu

Vector clocks

- Dynamo używa zegarów wektorowych, by wyłapać wynikanie między różnymi wersjami tego samego obiektu
- Zegar wektorowy to lista par (węzeł, licznik)
- Jeden zegar związany z wszystkimi wersjami wszystkich obiektów
- Na jego podstawie wiemy, czy to wersje równoległe, czy jedna powstała z drugiej

Vector clocks

- Jeśli wszystkie liczniki pierwszego obiektu są mniejsze niż odpowiadające im liczniki drugiego obiektu, pierwszy jest przodkiem drugiego i można go zapomnieć
- W p.p. mamy konflikt wersji, który trzeba rozwiązać

Konflikt wersji

- Podczas aktualizacji, trzeba podać, którą wersję aktualizujemy
- Pole context zawiera m.in. informację o zegarze wektorowym
- Jeśli powstaje konflikt wersji, jest on rozwiązywany przy odczycie

Konflikt wersji

- Logika aplikacji korzystającej z Dynamo musi na podstawie zawartości i kontekstu umieć uzgodnić wersje
- Po uzgodnieniu wersji przez klienta, jest ona zapisywana w Dynamo
- Wszystkie dotychczasowe rozgałęzienia znikają i obowiązuje od teraz wersja uzgodniona

Przykład


write
handled by Sx
D1 ([Sx,1])
write
handled by Sx
D2 ([Sx,2])
write
handled by Sy
write
handled by Sz
D3 ([Sx,2],[Sy,1])
D4 ([Sx,2],[Sz,1])
reconciled
and written by
Sx
D5 ([Sx,3],[Sy,1][Sz,1])

Zegary wektorowe

- Potencjalnie rozmiar zegara wektorowego stale rośnie, gdy zapisy są koordynowane przez wiele różnych węzłów
- W praktyce bardzo rzadko, bo koordynator zazwyczaj jest jednym z N pierwszych na liście preferencji
- Dynamo ogranicza rozmiar zegara

Zegary wektorowe

- Do pary (węzeł, licznik) dodajemy jeszcze timestamp zmiany
- Gdy zegar przekroczy ustalony rozmiar, najstarszy wpis jest usuwany
- Może to prowadzić do nieefektywności w uzgadnianiu, bo nie zauważymy wynikania między wersjami
- Problem jeszcze nigdy nie pojawił się w środowisku produkcyjnym, więc się nim nie przejmują

Operacje get() i put()

Strategie wybrania węzła - koordynatora:

- Skorzystanie z load-balancera, który nam go wskaże
  - Nie trzeba linkować do aplikacji fragmentu kodu Dynamo
- Skorzystanie z biblioteki klienckiej, świadomej podziału danych, która kieruje żądania bezpośrednio do odpowiednich koordynatorów
  - Zapewnia niższe opóźnienia, bo omijamy krok przekierowania

Operacje get() i put()

- Koordynator - zazwyczaj jeden z N pierwszych węzłów na liście preferencji
- Load balancer może skierować do węzła, który nie jest jednym z N pierwszych
- Wówczas węzeł ten przekieruje do kogoś z N pierwszych
- Operacje dotyczą N pierwszych “zdrowych” węzłów z listy preferencji

Operacje get() i put()

- Do zachowania spójności stosuje się protokół podobny do tych z quorum systems
- Dwie zmienne: R oraz W to minimalna ilość węzłów do udanego odczytu (R) i zapisu (W)
- $R + W > N$
- Zazwyczaj $R < N$, $W < N$, by zminimalizować opóźnienia

Operacja put()

- Koordynator tworzy nowy obiekt i generuje mu zegar wektorowy
- Zapisuje go lokalnie i wysyła do N pierwszych osiągalnych węzłów z listy preferencji
- Jeśli otrzyma co najmniej W - 1 odpowiedzi, uważamy zapis za udany

Operacja get()

- Koordynator prosi N pierwszych osiągalnych węzłów z listy preferencji o wszystkie wersje obiektu jakie mają
- Następnie czeka na R odpowiedzi i próbuje uzgadniać wersje
- Wysyła wszystkie wersje, których nie dało się uzgodnić do klienta
- Jeśli było ich więcej, klient sam je uzgadnia, a następnie nowa, uzgodniona wersja jest zapisywana

Awarie

- Gdyby zastosować ten system w tradycyjny sposób, byłby on niedostępny przy awarii serwerów i miałby zmniejszoną trwałość
- Stąd zamiast pierwszych N węzłów, używamy pierwszych N “zdrowych” i stosujemy tzw. “hinted handoff”

Awarie - hinted handoff

- Niech A, B, C, D -pierwsze 4 węzły z listy preferencji, N = 3, A - ma awarię, reszta działa
- Dane, które trafiłyby do węzła A trafiają do D
- W metadanych wysłanych do D jest, że ten dane były przeznaczone pierwotnie dla A
- D zapamiętuje to w oddzielnej lokalnej bazie, którą co jakiś czas skanuje
- Gdy D wykryje, że A ożył, wysyła mu te dane
- Jeśli operacja się powiedzie, D może skasować swoją kopię danych bez zmniejszania liczby replik w systemie

Awarie

- Dzięki temu system jest niepodatny na chwilowe awarie
- Aplikacje potrzebujące najwyższego poziomu dostępności mogą ustawić W = 1
- W praktyce ustala się wyższe W, by osiągnąć pożądany poziom trwałości
- Dynamo jest tak zaprojektowane, by być odporne na awarię całego centrum danych
- Repliki obiektu są trzymane w różnych centrach danych
- Są one połączone specjalnymi szybkimi łączami

Awarie

- Może stać się tak, że repliki ze wskazaniem stają się niedostępne zanim zostały przekazane właściwemu adresatowi
- Dynamo zawiera protokół synchronizacji replik, używający drzew Merkle'a

Drzewo Merkle'a

- Drzewo haszowe, w którym liście są haszami poszczególnych kluczy
- Ojciec to hasz z wartości synów
- Jeśli wartości korzeni są niezgodne, to któreś z poddrzew niezgodne
- Rekurencyjnie znajdujemy wszystkie niezgodne liście
- Zmniejszają ilość przesyłanych danych, by wykryć niespójności replik
- Zmniejszają ilość odczytów dyskowych podczas procesu antyentropii

Drzewa Merkle'a w Dynamo

Dynamo używa drzew Merkle'a w następujący sposób:

- Każdy węzeł trzyma osobne drzewo Merkle'a dla każdego przedziału kluczy, za który jest odpowiedzialny
- Pozwala to łatwiej sprawdzić, czy trzymamy aktualne wersje obiektów
- Wadą rozwiązania jest to, że przedziały kluczy zmieniają się wraz z przyłączaniem i odłączaniem wierzchołków
- Trzeba wówczas od początku wyliczać wartości na drzewie
- Odpowiedzią na ten problem jest ulepszony schemat partycjonowania, opisany później

Wykrywanie członkostwa

- Gdy węzeł nie jest dostępny, zazwyczaj oznacza to chwilową awarię
- Nie powinno się wówczas odtwarzać replik ani aktualizować podziału pierścienia haszów
- Z tego powodu włączania i wyłączania węzłów z pierścienia dokonuje ręcznie administrator

Wykrywanie członkostwa

- Węzeł z pierścienia, który obsłużył to zdarzenie, zapisuje je wraz z czasem
- Algorytm plotkujący rozsyła zmiany w historii członkostwa, co powoduje spójny obraz członkostwa przez wszystkie węzły
- Każdy węzeł kontaktuje się z losowym innym co sekundę i wymieniają się wówczas zmianami w historii członkostwa

Wykrywanie członkostwa

- Kiedy węzeł po raz pierwszy podłącza się do pierścienia, wybiera swój zbiór tokenów, a także mapuje węzły i odpowiadające im węzły wirtualne
- Mapa ta jest uzgadniana między węzłami razem z wymianą informacji o historii członkostwa
- Dlatego każdy węzeł potrafi przekazywać żądania do odpowiedniego zbioru węzłów

Wykrywanie członkostwa - problemy

- Powyższy mechanizm może powodować czasowy logiczny podział pierścienia
- Administrator przyłącza wierzchołki A i B do pierścienia. Każdy wie, że w nim jest, ale przez dłuższy czas mogą być wzajemnie nieświadomi
- By temu zapobiec, część węzłów pełni rolę “seedów”

Wykrywanie członkostwa - problemy

- Można je poznać przez zewnętrzny mechanizm
- Znane wszystkim węzłom w pierścieniu
- Wszystkie węzły uzgadniają swoje członkostwo z seedami, więc logiczny podział jest wysoce nieprawdopodobny
- Poza tym seedy są zwykłymi węzłami w pierścieniu

Wykrywanie awarii

- Nie chcemy wysyłać żądań do niedostępnych węzłów
- Węzeł A uznaje B za niedostępny jeśli ten nie odpowiedział na jego żądanie
- Wówczas używa on kolejnych węzłów z listy preferencji, by wykonały żądanie, które miał zrobić B
- Gdy A uzna B za niedostępny, co pewien czas sprawdza, czy B nie powrócił
- Węzeł nie przekazuje swojej wiedzy o awariach innym

Dodawanie i usuwanie węzłów

- Gdy węzeł jest dodawany do systemu, przydziela mu się tokeny rozsiane losowo po całym pierścieniu
- Dla każdego przedziału kluczy istnieje grupa węzłów odpowiedzialnych za obsługę tego przedziału
- Któryś z dotychczasowych węzłów nie musi już trzymać informacji o kluczach z tego przedziału
- Przesyła on całą swoją wiedzę o tym przedziale do nowego węzła, a sam o nim zapomina

Dodawanie i usuwanie węzłów

- Przy usuwaniu węzła w podobny sposób realokujemy klucze usuwanego węzła
- Z obserwacji wynika, że klucze rozkładają się równomiernie między węzłami
- Jest to ważne ze względu na założenia o opóźnieniach wykonania operacji
- Dodatkowe potwierdzenie przesłania danych o kluczach wyklucza wielokrotne transfery tych samych danych (a tym samym zmniejszenie liczby replik)

Najważniejsze komponenty

- Moduł przechowywania danych
- Moduł koordynacji żądań
- Moduł wykrywania członkostwa i awarii

Wszystko napisane w Javie

Przechowywanie danych

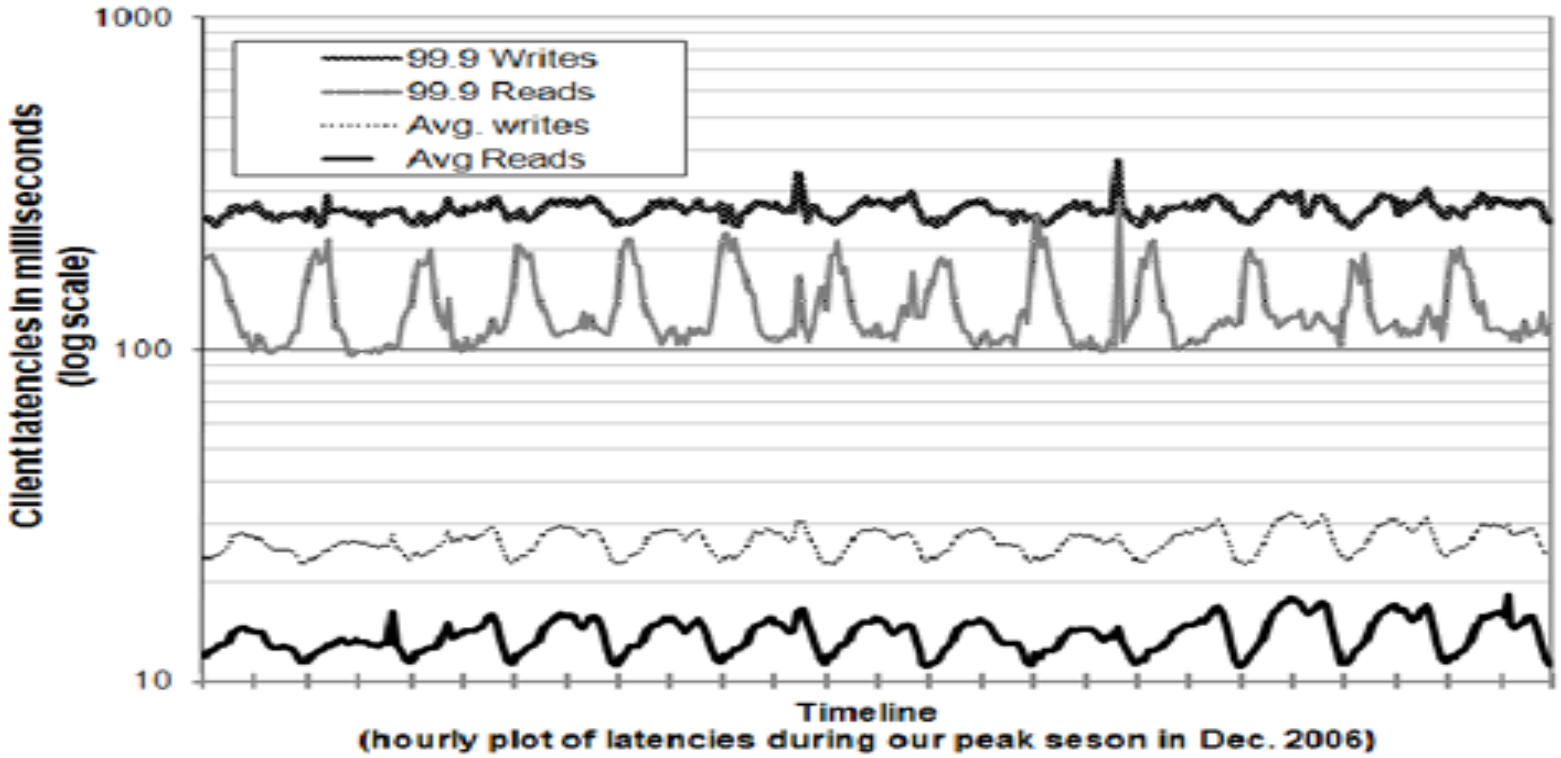
Możliwość podłączenia różnych komponentów do przechowywania danych. W użyciu obecnie są:

- Berkeley Database (BDB) Transactional Data Store - używane przez większość instancji produkcyjnych
- BDB Java Edition
- MySQL
- Bufor w pamięci, zapisujący dane w sposób trwały

Koordynacja żądania

- Komponent event-driven
- Cała komunikacja idzie przez Java NIO
- Rozkład żądań nierównomiernie rozdzielony między obiekty, więc w praktyce na koordynatora to jeden z N pierwszych węzłów listy preferencji
- Zazwyczas put() następuje zaraz po get(), więc na koordynatora put() wybiera się tego, który jako pierwszy odpowiedział na get(), co zapisuje się w context

Opóźnienia zapisu i odczytu



Typy usług korzystających z Dynamo

Wyróżniamy trzy główne typy konfiguracji aplikacji korzystających z Dynamo:

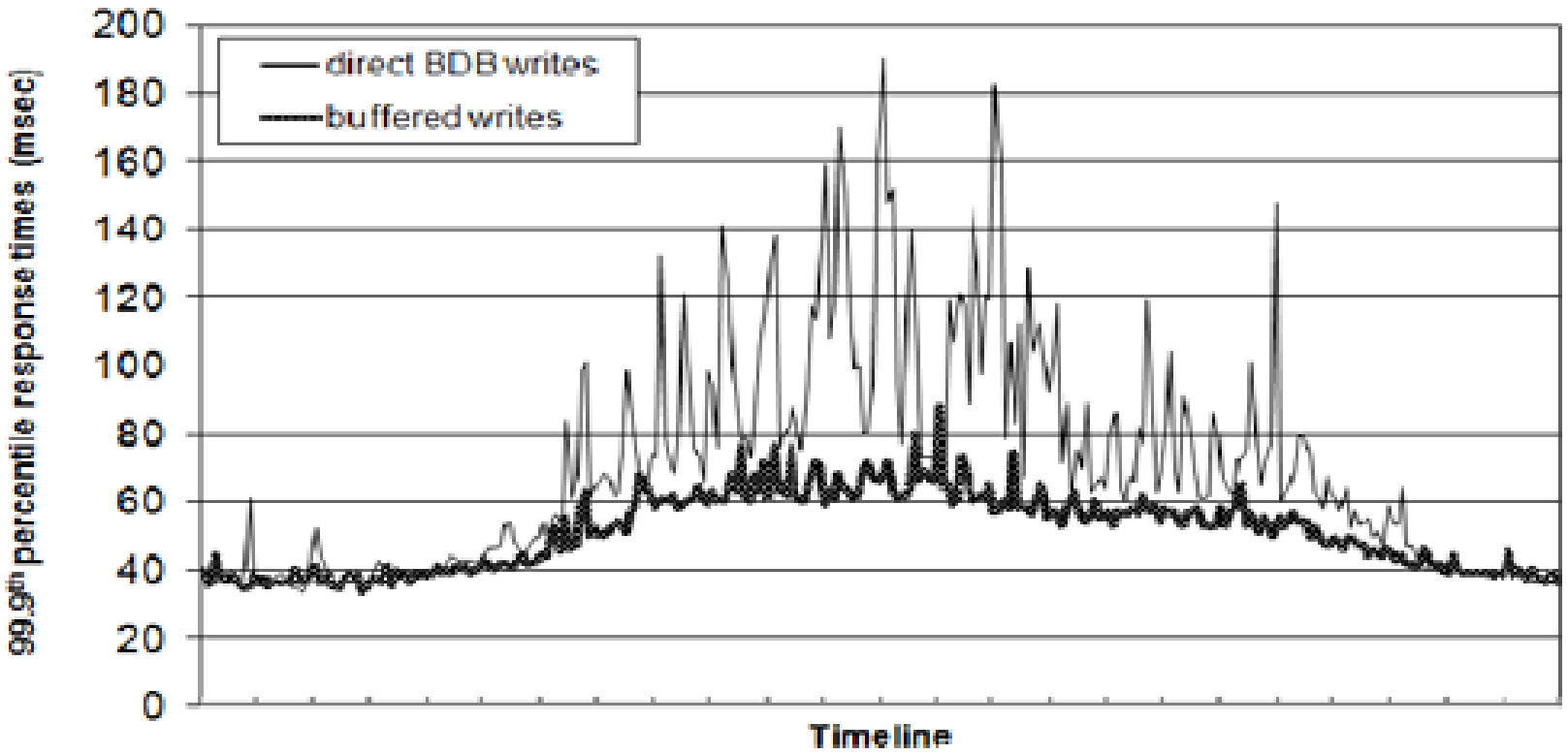
- Uzgadniające wersje obiektów w logice biznesowej
  - Uzgadnianie na poziomie aplikacji klienckiej
- Uzgadniające wersje na podstawie timestampu
  - Wykonywane przez Dynamo
  - Ostatni zapis wygrywa
- Wymagające bardzo wydajnych odczytów
  - R = 1, W = N
  - Często wykorzystywane przez cięższe bazy danych jako cache

Zazwyczaj N = 3

Buforowany zapis

- Zapisy trzymane są w lokalnym buforze w pamięci operacyjnej
- Przy odczycie najpierw sprawdzamy w buforze
- Co jakiś czas zapisujemy te dane w trwałej formie
- Powoduje obniżenie opóźnienia dla 99,9% zapisów nawet trzykrotnie w szczycie przy buforze na 1000 obiektów
- Wpływa na trwałość, gdyż w razie awarii tracimy dane z bufora
- By ograniczyć to ryzyko, koordynator przy każdym zapisie wyznacza jeden węzeł, który od razu zapisuje trwale

Buforowany zapis

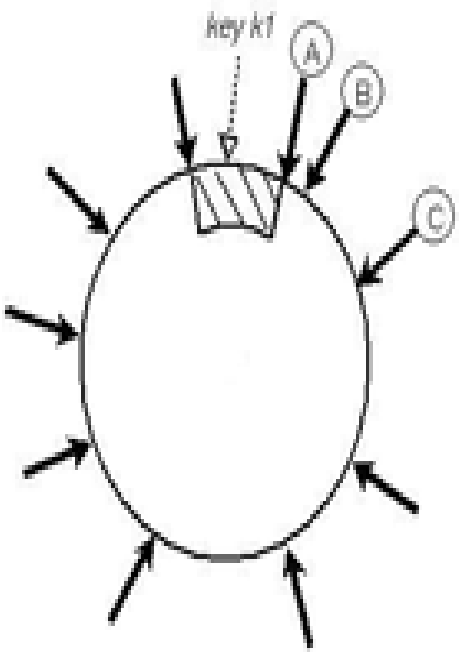


Strategie partycjonowania

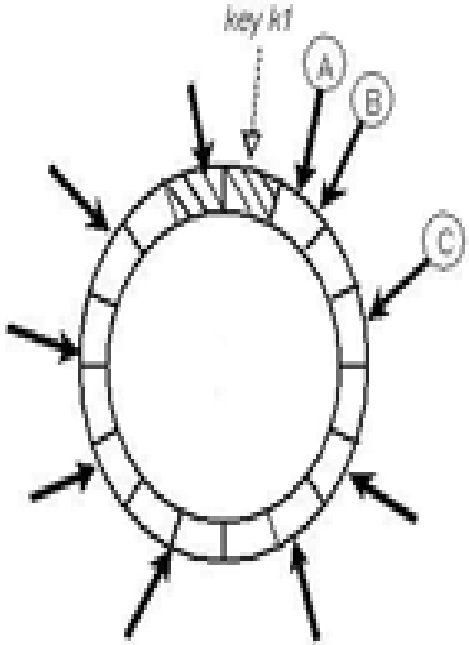
S - ilość węzłów w systemie

1. T losowych tokenów na węzeł, dzielenie po wartości tokena
   - Ciężko zrobić snapshot przestrzeń kluczy
   - Gdy węzeł ma przekazać innemu jakiś przedział, musi przeszukiwać całą swoją przestrzeń kluczy
   - Przy dodawaniu/usuwaniu węzłów trzeba przeliczać całe drzewa Merkle'a
2. T losowych tokenów na węzeł, podział na Q równych części
3. Q/S tokenów na węzeł, podział na Q równych części

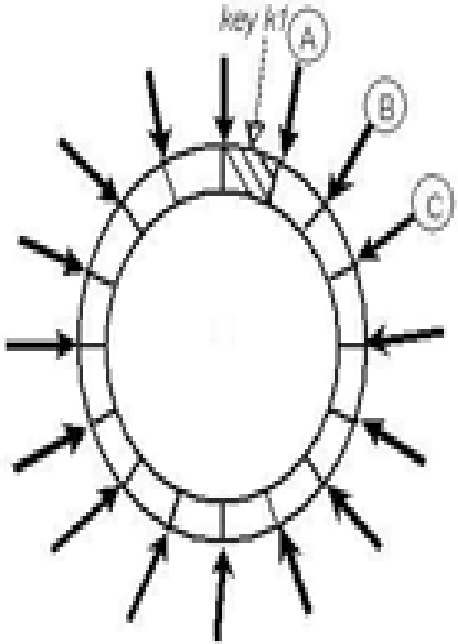
Strategie partycjonowania



Strategy 1

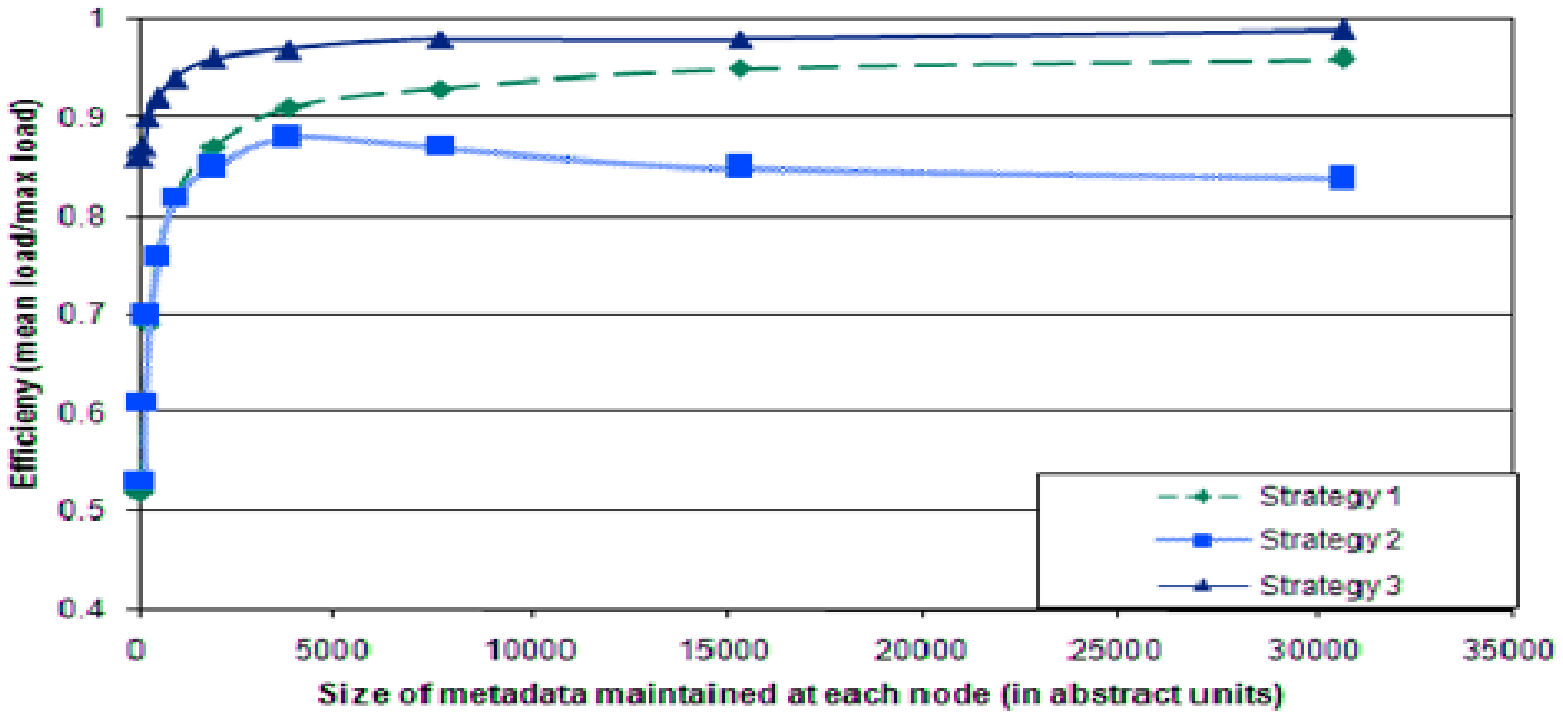


Strategy 2



Strategy 3

Strategie partycjonowania



Konflikty wersji

Aplikacja kliencka typu 1 (uzgadnianie w warstwie logiki), działała 24h:

- 99,94% żądań w tym czasie otrzymało dokładnie 1 wersję
- Konflikty wersji występują rzadko
- Konflikty są generowane głównie przez współbieżny zapis (a nie awarie)

Wybór koordynatora

Opóźnienia przy wyborze koordynatora (ms)

	Odczyt 99,9%	**Zapis 99,9%**	**Odczyt średni**	**Zapis średni**
Przez serwer	66,9	68,5	3,9	4,02
Przez klienta	30,4	30,4	1,55	1,9

Koniec

Pytania?