POLITECHNIKA WARSZAWSKA
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2013/2014

PRACA DYPLOMOWA MAGISTERSKA

Jakub Turek

[TYTUŁ]

Praca wykonana pod kierunkiem
dra inż. Jakuba Koperwasa

Ocena: .....

.....

Podpis Przewodniczącego Komisji
Egzaminu Dyplomowego

Kierunek: Informatyka
Specjalność: Inżynieria Systemów Informatycznych
Data urodzenia: 1990.01.09
Data rozpoczęcia studiów: 2013.02.20

Życiorys

Urodziłem się 9 stycznia 1990 roku w Łodzi. W 1997 roku rozpocząłem edukację w Szkole Podstawowej nr 7 w Łodzi. W latach 2003-2006 kontynuowałem naukę w Gimnazjum nr 42 im. Władysława Stanisława Reymonta w Łodzi. Od 2006 roku uczyłem się w Liceum Ogólnokształcącym nr 31 im. Ludwika Zamenhofa w Łodzi. W 2009 roku zdałem egzaminy maturalne i ukończyłem szkołę licealną z wyróżnieniem. W latach 2009-2013 studiowałem dziennie informatykę na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Ukończyłem studia z wynikiem celującym i odebrałem tytuł zawodowy inżyniera. Obecnie kończę Pracę Dyplomową Magisterską pod kierownictwem Instytutu Informatyki. We wrześniu 2012 roku rozpocząłem pracę zawodową jako programista aplikacji do zarządzania procesami biznesowymi oraz aplikacji mobilnych w firmie Xentivo, gdzie pracuję do dziś. Moją pasją jest tworzenie aplikacji mobilnych oraz internetowych, które uruchamiane są w środowisku iOS.

.....
Podpis studenta

Egzamin dyplomowy:
Złożył egzamin dyplomowy w dniu: .....
z wynikiem: .....
Ogólny wynik studiów: .....
Dodatkowe uwagi i wnioski Komisji: .....
.....

Streszczenie

Celem Pracy Dyplomowej jest stworzenie interfejsu programowania aplikacji umożliwiającego efektywne rozwiązywanie problemów w modelowaniu dziedziny danych opartych o bazę danych Apache Cassandra.

Client-server Augmented Reality applications framework for Android system

Summary

The goal of this thesis is to create a framework supporting the development of multiuser applications for Android system. The framework should consist of a client-server bus, as well as several Augmented Reality components. An additional goal is to explore the possibility of adapting commonly used programming practises dedicated to large projects during Andorid applications development. All the goals have been fully accoplished - the output is the implementation of the client-server bus as well as AR components allowing to track the device's geographic coordinates and to render a stable three-dimensional graphics on the display of the device. In order to demonstrate the features of the framework, a sample multiplayer game has been created. The thesis includes a description of the created components' design process, as well as their functionality and structure.

Spis treści

1 Wstęp **4**

2 Apache Cassandra **5**
2.1 Model danych 6
2.2 Dystrybucja danych 6
2.3 Konsekwencje dla modelowania dziedziny 7
2.4 CQL 9
2.5 Modelowanie - wzorce i antywzorce 10

3 Mapowanie obiektowe **11**
3.1 Mapowanie obiektowo-relacyjne dla Cassandry 11

Rozdział 1

Wstęp

Tu będzie wstęp

Rozdział 2

Apache Cassandra

Apache Cassandra jest bazą danych NoSQL[1], która powstała w wyniku połączenia rozwiązań wykorzystywanych w Dynamo[2] oraz BigTable[3]. Cassandra początkowo była rozwijana dla potrzeb portalu społecznościowego Facebook. Baza danych powstała z myślą o rozwiązaniu problemu pełnotekstowego przeszukiwania skrzynek odbiorczych użytkowników, w których dziennie zapisywane były miliardy wiadomości. Głównym celem, do których dążyli twórcy Cassandry była możliwość wykorzystania jej do przechowywania ogromnych ilości danych w bardzo rozproszonym środowisku, gdzie awarie pojedynczych węzłów zdarzają się na porządku dziennym. W tych warunkach baza danych musi zapewniać szybki i niezawodny dostęp do danych. [2]

Apache Cassandra wykorzystywana jest w wielu serwisach na całym świecie. Najbardziej znaczące przykłady użycia produkcyjnego to eBay[4], Instagram[5] oraz GitHub[6]. Największa światowa instalacja Cassandry obejmuje około 15000 węzłów, na których przechowywane jest łącznie ponad 4 petabajty danych. [3]

W przeciwieństwie do relacyjnych baz danych, Apache Cassandra nie zapewnia wsparcia dla reguły ACID[7]. Zamiast tego zostały zrealizowane postulaty twierdzenia CAP: „we współdzielonym systemie plików można zachować maksymalnie dwie z trzech właściwości: spójności, dostępności oraz podatności na partycjonowanie". [4] Apache Cassandra priorytetyzuje właściwości dostępności oraz partycjonowania. Spójność danych jest odwrotnie proporcjonalna i może być regulowana w zależności od czasu odpowiedzi. Wysoka spójność danych oznacza wolniejszą odpowiedź bazy.

[1]NoSQL (ang. Not Only SQL) - podzbiór baz danych, które zapewniają inne sposoby modelowania dziedziny niż tradycyjny model oparty na tabelach i relacjach.

[2]Amazon DynamoDB - zdecentralizowana, wysoce skalowalna baza danych typu klucz-wartość.

[3]Google BigTable - rozproszony system bazodanowy, który dobrze skaluje się dla ogromnych ilości danych.

[4]eBay - największy portal z aukcjami internetowymi na świecie

[5]Instagram - portal pozwalający na umieszczanie fotografii.

[6]GitHub - usługa pozwalająca na przechowywanie i wersjonowanie kodu źródłowego aplikacji.

[7]ACID (ang. Atomic, Consistency, Isolation, Durability) - zasada atomowości, spójności, izolacji i trwałości, które gwarantują poprawne przetwarzanie transakcji w bazach danych.

2.1 Model danych

Model danych Apache Cassandra jest analogiczny do BigTable. [1] Można przedstawić go jako dwuwymiarowa mapa trójek wartości:

```
Map<RowKey, Map<ColumnKey, Triple<Value, Timestamp, TTL>>>
```

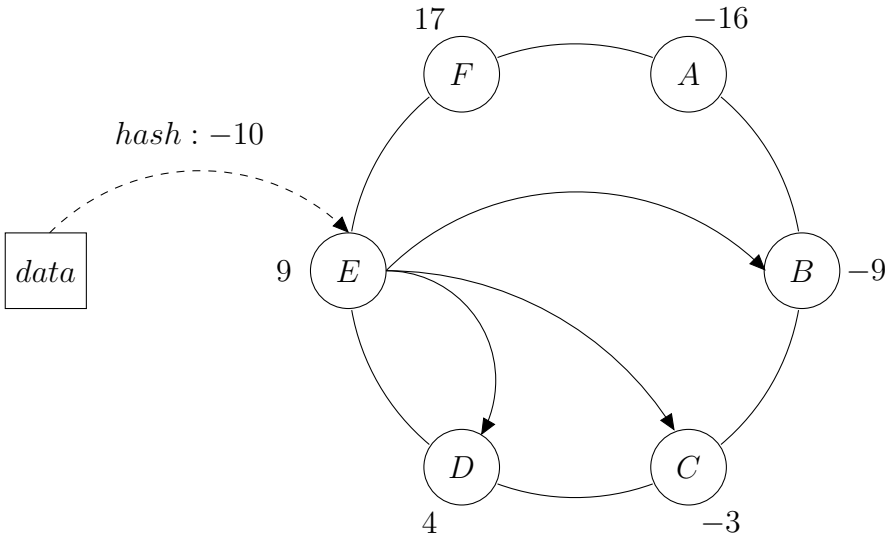
gdzie `RowKey` to identyfikator wiersza, `ColumnKey` to identyfikator kolumny, `Value` to wartość komórki, `Timestamp` to czas aktualizacji komórki, a `TTL` to czas życia danej wartości. [5] Na rysunku 2.1 przedstawiona jest schematyczna ilustracja wiersza danych. Pogrubiona wartość w lewej komórce to klucz wiersza, natomiast wyróżnione wartości w pierwszym wierszu oznaczają klucze poszczególnych kolumn. Każda komórka składa się z trzech wartości: wartości, czasu życia oraz „odcisku czasu".

	ABC	**DEF**	$\cdots$	**XYZ**
123456	test value	another test value	$\cdots$	not a test value
	20	∞	$\cdots$	∞
	1291987837942000	1291980736812000	$\cdots$	1291980736212000

Rysunek 2.1: Przykładowy wiersz modelu danych o identyfikatorze 123456. Wartość komórki (123456, DEF) to „another test value".

2.2 Dystrybucja danych

Do dystrybucji danych wykorzystywana jest funkcja skrótu, która zachowuje kolejność elementów. Węzły są przedstawiane na w topologii pierścienia. Algorytm dystrybucji zostanie omówiony na przykładzie ze schematu 2.2:



Rysunek 2.2: Schematyczna ilustracja dystrybucji danych w bazie danych Apache Cassandra.

1. Każdemu z węzłów $\{A, B, C, D, E, F\}$ przypisywany jest token, który zawiera się w zakresie wartości przyjmowanych przez funkcję skrótu. Strategię wyboru tokenu można konfigurować. Przykładową strategią jest wybór losowy. W omawianym przykładzie węzłom zostały przypisane tokeny o wartościach $\{-16, -9, -3, 4, 9, 17\}$.

2. Użytkownik bazy danych przesyła żądanie do dowolnego węzła, który pełni funkcję koordynatora dla danej operacji. Koordynator nadzoruje wpisanie danych do odpowiednich węzłów. W omawianym przykładzie rolę koordynatora pełni węzeł E.

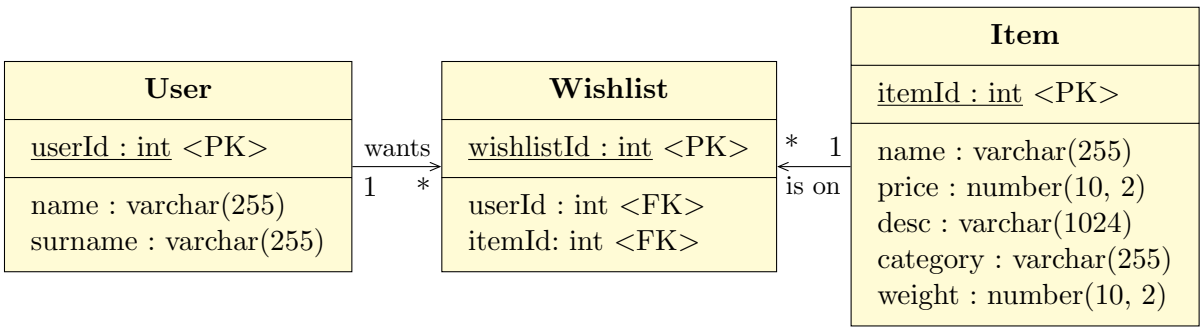
3. Każdy węzeł przechowuje dane, których funkcja skrótu zawiera się w przedziale $(token_{n-1}, token_n]$, gdzie n to numer kolejny węzła rosnący zgodnie z ruchem wskazówek zegara. W przykładzie węzeł C przechowuje wiersze o wartościach funkcji skrótu z przedziału $(-9, -3]$, natomiast węzeł D z przedziału $(-3, 4]$. Wartości funkcji obliczane są cyklicznie, stąd węzeł A przechowuje wiersze o skrócie z przedziału $(-\infty, -16] \cup (17, \infty)$. W przykładzie wiersz o kluczu z funkcją skrótu wartości -10 zostanie utrwalony na węźle B.

4. Dane replikowane są na n węzłach, gdzie n to wartość konfigurowalnego współczynnika replikacji. Poza węzłem macierzystym (wyznaczanym w punkcie 3 algorytmu) dane są replikowane na $n - 1$ kolejnych (zgodnie z ruchem wskazówek zegara) węzłach. W omawianym przykładzie dane zostaną zreplikowane na węzłach C i D.

2.3 Konsekwencje dla modelowania dziedziny

Wewnętrzna struktura i mechanizm dystrybucji danych wykorzystywany w Apache Cassandra zmienia podejście do modelowania dziedziny znane z relacyjnych baz danych. Zbudowanie efektywnego modelu danych Cassandry wymaga skupienia się w podobnym stopniu na zdefiniowaniu encji z modelowanego świata, jak również na analizie odwołań, które będą wykonywane do obiektów z tego świata. [6]

Załóżmy, że celem jest modelowanie danych dla sklepu internetowego. Zakupów dokonują użytkownicy, którzy mogą wstawić wiele przedmiotów z oferty sklepu na listę życzeń. W przypadku baz opartych o język SQL jest to klasyczny problem relacji typu wiele-do-wielu, do modelowania których wykorzystywana jest najczęściej tabela pośrednia.



Rysunek 2.3: Modelowanie listy życzeń w relacyjnej bazie danych.

Diagram prezentujący zamodelowaną relację dla listy życzeń jest przedstawiony na rysunku 2.3. W tabeli Użytkownik (*User*) przechowywane są imię, nazwisko oraz identyfikator. W tabeli Przedmiot (*Item*) znajduje się nazwa, cena, a także inne właściwości: opis, kategoria oraz waga. Tabela Lista życzeń (*Wishlist*) łączy ze sobą użytkownika i przedmiot poprzez wykorzystanie kluczy obcych.

Powyższy model jest wykorzystywany w widoku listy życzeń na profilu użytkownika. Na liście życzeń prezentowane są informacje o nazwie przedmiotu oraz jego cenie. Po kliknięciu nazwy użytkownik przenoszony jest do strony przedmiotu. Na listingu 2.4 zaprezentowano zapytanie, które wyświetla listę życzeń.

```
SELECT item.name, item.price
FROM Item item, Wishlist wishlist
WHERE wishlist.userId = 202;
```

Rysunek 2.4: Zapytanie, które pobiera wszystkie przedmioty na liście życzeń użytkownika o identyfikatorze 202.

Cassandra umożliwia utworzenie dokładnej repliki relacyjnego modelu danych. Zostało to przedstawione na rysunku 2.5.

User

	name	**surname**
123	Janusz	Kowalski

	name	**surname**
124	Marcin	Nowak

Wishlist

	userId	**itemId**
51	123	579

	userId	**itemId**
52	124	232

Item

	name	**price**	**desc**	**category**	**weight**
232	Master Chef	20.34	Cooking recipes	BOOKS	0.2

	name	**price**	**desc**	**category**	**weight**
579	Seat Hit	159.99	Wooden armchair	FURNITURE	10.8

Rysunek 2.5: Wynik błędnego przeniesienia relacyjnego modelu danych do Cassandry.

Taki model jest jednak niepoprawny. Nie umożliwia on filtrowania zawartości listy życzeń po identyfikatorze użytkownika. Wynika to z faktu, że pobranie odpowiednich wierszy listy życzeń wymaga znajomości ich identyfikatorów, podczas gdy widok dysponuje wyłącznie odniesieniem do użytkownika. Błąd ten można łatwo naprawić zastępując encję *Wishlist* encją *WishlistByUser*, co przedstawia diagram 2.6.

WishlistByUser

	579
123	-

	232
124	-

Rysunek 2.6: Definicja encji listy życzeń umożliwiająca filtrowanie względem użytkownika.

Poprawiony model można poddać dalszej optymalizacji. Wyświetlenie listy życzeń użytkownika wymaga odwołania do encji *Item*, w której znajdują się informacje o nazwie i cenie przedmiotu. Ponieważ przedmioty mogą być rozmieszczone na różnych węzłach, silnik Cassandry nie może wykonać złączenia w sposób optymalny - zapytanie o każdą pozycję listy życzeń jest wykonywane osobno. W celu przyspieszenia wykonywania operacji należy wykonać denormalizację. Dołączając do encji *WishlistByUser* informacje o nazwie i cenie produktu można uniknąć wykonywania kosztownych złączeń. Pozostałe dane przedmiotu zostaną pobrane dopiero po przejściu na jego stronę. Efekt denormalizacji jest przedstawiony na diagramie 2.7.

WishlistByUser

	579
123	(„Master Chef”, 20.43)
	232
124	(„Seat Hit”, 159.99)

Rysunek 2.7: Zdenormalizowana postać listy życzeń.

2.4 CQL

Efektywne modelowanie i obsługa danych w Apache Cassandra wymaga dobrej znajomości wewnętrznej struktury bazy danych. Dodatkowym utrudnieniem przy korzystaniu z początkowych wersji Cassandry była konieczność wykorzystania skomplikowanego interfejsu programistycznego opartego o wywoływanie zdalnych procedur Thrift[8]. Thrift jest platformą pozwalającą budować aplikacje przenośne między różnymi językami programowania. Dzięki temu rozwiązaniu baza danych dostępna była dla wszystkich platform. Niestety, skutkiem ubocznym było skomplikowanie interfejsu dostępowego.

Wraz z wydaniem 1.2 Apache Cassandry wprowadzony został nowy interfejs dostępu do tej bazy danych. Interfejs ten nosi nazwę CQL[9] i jest językiem zapytań, którego składnia wzorowana jest na SQL. Poza podobieństwami składniowymi języki te nie mają cech wspólnych. Nie są wzajemnie zgodne. W chwili obecnej CQL jest rekomendowanym standardem komunikacji z Apache Cassandra. [7] Na listingu 2.8 przedstawiono zapytanie w języku CQL, które opisuje omawianą wcześniej encję *User*. Wynikiem wykonania tego zapytania jest prosty schemat modelu danych zaprezentowany na diagramie 2.9.

```
CREATE TABLE User (
    userId uuid PRIMARY KEY,
    name text,
    surname text);
```

Rysunek 2.8: Zapytanie CQL, które tworzy encję *User*.

[8]Dokumentacja interfejsu dostępna jest pod adresem `https://wiki.apache.org/cassandra/API10`.
[9]CQL (ang. Cassandra Query Language) - język zapytań Cassandry.

User		**name**	**surname**
	userId	null	null

Rysunek 2.9: Wynik wykonania zapytania 2.8.

2.5 Modelowanie - wzorce i antywzorce

Pomimo, że CQL znacząco upraszcza modelowanie w Cassandrze stworzenie efektywnego schematu danych nie jest zadaniem prostym. Aby ułatwić ten proces twórcy i użytkownicy Cassandry zaczęli opisywać wzorce i antywzorce modelowania oraz dostępu do danych. Pełnią one analogiczną role do wzorców i antywzorców projektowych znanych z programowania. Na ich opis składa się możliwie ogólna definicja problemu, a także poprawny (lub niepoprawny) sposób jego rozwiązania.

Przykładem antywzorca modelowania jest kolejka oparta na kolumnach. [8] Kolejka to struktura danych, w której ilości wykonywanych operacji wstawiania, usuwania i odczytu są do siebie zbliżone. W przypadku Cassandry usuwanie kolumn z wiersza nie jest wykonywane natychmiast po odebraniu żądania. Zamiast tego usunięta kolumna jest oznaczana specjalnym znacznikiem *tombstone* i fizycznie usuwana dopiero po upłynięciu pewnego czasu. Takie działanie przyspiesza znacząco operację usuwania kosztem operacji odczytu. Kiedy w wierszu występuje wiele znaczników *tombstone* operacje filtrowania zakresu kolumn wykonywane są znacznie wolniej.

Przykładem wzorca dostępu do danych jest unikanie konfliktów synchronizacji poprzez uaktualnianie wyłącznie zmodyfikowanych wartości. [9] Encja *Item* z diagramu 2.5 ma 5 właściwości. Wyświetlenie ekranu aktualizacji przedmiotu wymaga pobrania zawartości całego wiersza z bazy danych. Wzorzec stanowi, że jeżeli na tym ekranie zostanie zmieniony wyłącznie opis to do Cassandry należy przesłać żądanie uaktualniające zawartość wyłącznie jednej komórki - *desc*. Pozostałe wartości z formularza mogą być już nieaktualne. Przesłanie kompletu informacji poskutkowałoby nadpisaniem aktualnych wartości. Postępowanie według wzorca pozwala wykorzystywać mechanizm rozwiązywania konfliktów wbudowany w Cassandrę.

Rozdział 3

Mapowanie obiektowe

Jedną z największych zalet relacyjnych baz danych w kontekście programowania obiektowego jest dostępność bibliotek, które umożliwiają proste mapowanie obiektowo-relacyjne[1]. Wykorzystanie ORM znacząco upraszcza proces tworzenia oprogramowania wykorzystującego bazy danych:

- Umożliwia definiowanie modelu danych poprzez tworzenie klas reprezentujących obiekty wykorzystywane w aplikacji.
- Zarządzanie połączeniami, sesjami i transakcjami bazodanowymi jest znacząco uproszczone lub dokonywane automatycznie.
- Zwiększa bezpieczeństwo aplikacji. Mechanizmy ORM dostarczają narzędzi do ochrony przez atakami typu SQL Injection[2]. [10]
- Umożliwia modyfikacje środowiska bazodanowego bez zmieniania kodu aplikacji. Mechanizmy ORM wspierają wiele silników bazodanowych, które mogą różnić się od siebie implementowanymi standardami języka SQL.

O popularności mechanizmów ORM świadczy fakt powstawania oficjalnych standardów mapowania obiektowo-relacyjnego włączanych do specyfikacji języków programowania. Przykładem takiego standardu jest JPA[3] dla języka Java.

3.1 Mapowanie obiektowo-relacyjne dla Cassandry

Wraz z pojawieniem się języka CQL pojawiła się szansa wykorzystania istniejących implementacji mechanizmów ORM do przechowywania danych z użyciem silnika Cassandry. Motywacją do stworzenia takiego rozwiązania jest zwiększenie wydajności istniejących aplikacji bez modyfikacji ich kodu. Minimalna rekonfiguracja aplikacji i podłączenie jej

[1] W dalszej części pracy używany będzie skrót ORM od angielskiego pojęcia object-relational mapping.

[2] SQL Injection (ang. wstrzykiwanie SQL) - typ ataku polegający na wykorzystywaniu specjalnie spreparowanych wartości, które dołączone do szablonu zapytania SQL powodują szkodliwe działania niezgodne z intencją programisty.

[3] Java Persistence API.

pod klaster bazodanowy skutkowałyby zwiększeniem wydajności działania aplikacji. Ponadto wykorzystanie istniejących interfejsów mogłoby umożliwić obsługę różnych typów baz danych należących do ruchu NoSQL.

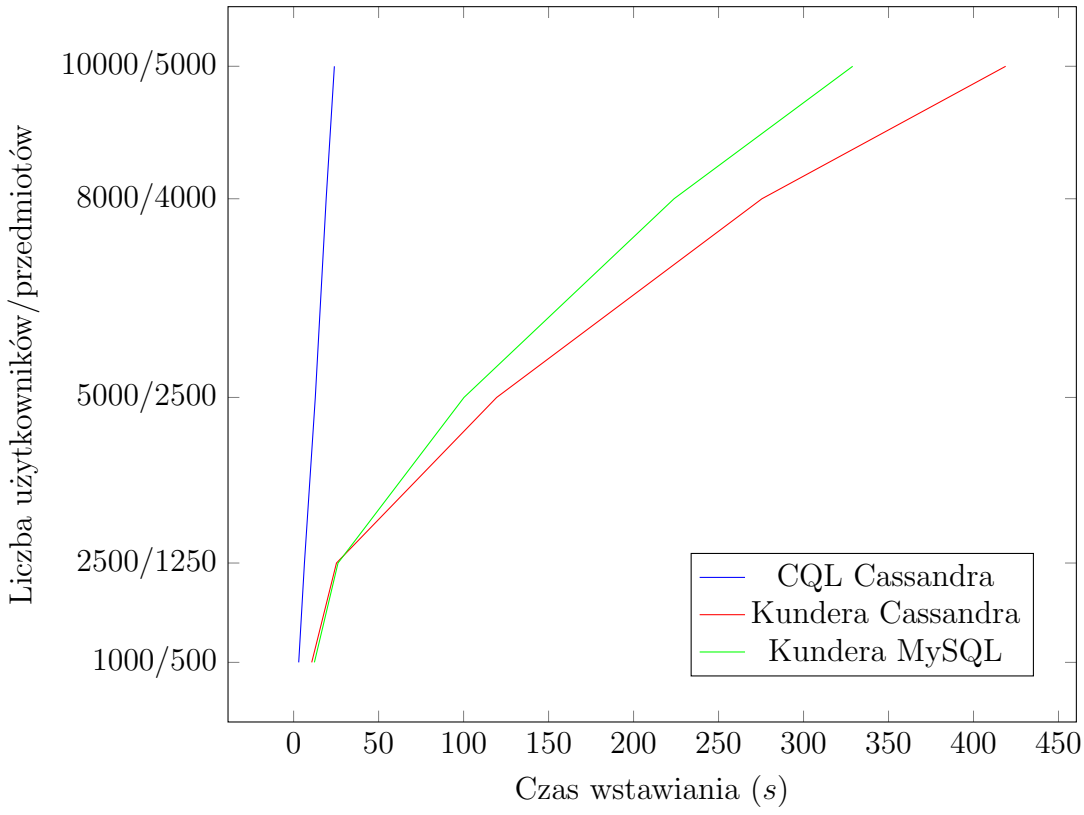
Przykładem projektu, który wykorzystuje mechanizm ORM do obsługi NoSQLowych baz danych jest Kundera.[11] Kundera zapewnia implementację mapowania obiektowego zgodną ze standardem JPA 2.0 między innymi dla Cassandry, HBase, MongoDB oraz Neo4j. Dodatkowo biblioteka wspiera obsługę wielu mechanizmów równocześnie.

3.1.1 Kundera - wydajność

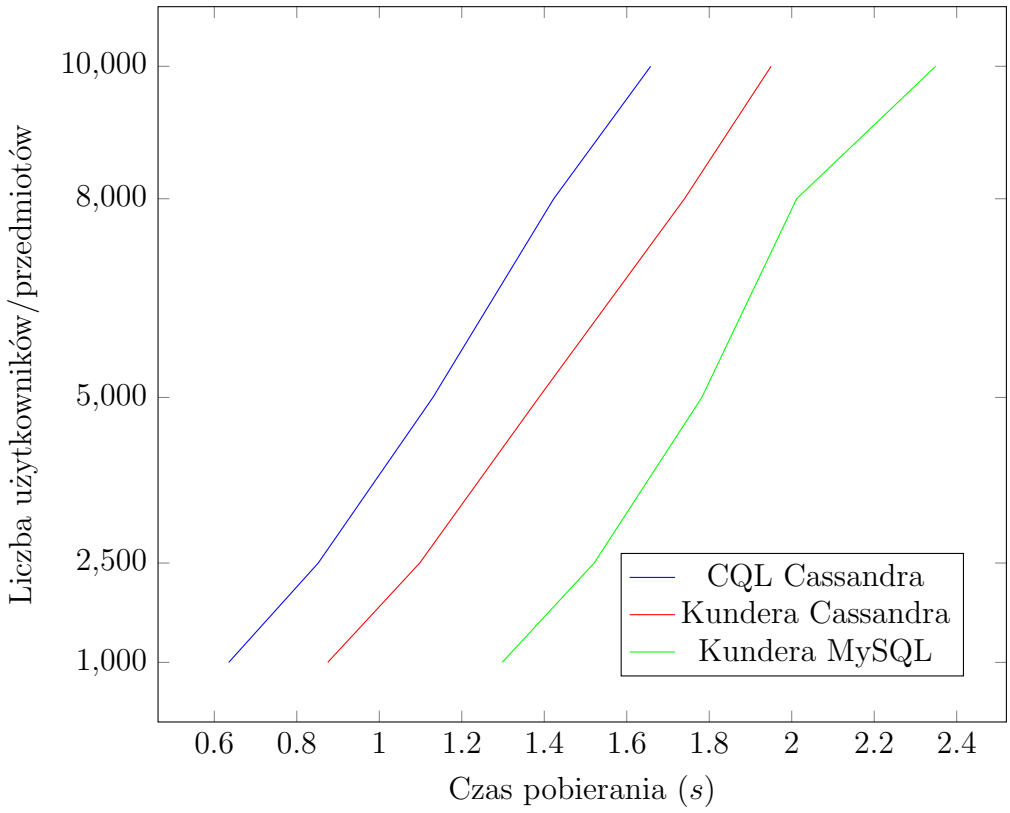
Wyniki pomiarów na stronie domowej projektu Kundera świadczą o tym, że biblioteka nie wprowadza znacznych narzutów wydajnościowych względem bezpośredniego wykorzystania interfejsu bazy danych. W przypadku wykorzystania mechanizmów mapowania obiektowo-relacyjnego w odniesieniu do Cassandry należy zbadać jaki narzut związany jest z modelowaniem dziedziny danych przy ich pomocy. Do pomiarów wykorzystano dwa modele: przedstawiony na diagramie 2.3 oraz 2.7. Pierwszy model został opisany w standardzie JPA i uruchomiony dla Cassandry oraz relacyjnej bazy danych. Drugi model bezpośrednio wykorzystuje CQL. Dla obu wariantów zostały przeprowadzone dwa testy: wstawianie oraz odczytywanie elementów z bazy danych.

W trakcie testów został zmierzony całkowity czas wykonania danej operacji. Przyjęte zostały następujące założenia:

- Liczba użytkowników i przedmiotów są parametryzowane i skalowane liniowo.
- Liczba przedmiotów, które użytkownik może mieć na swojej liście życzeń zawiera się w przedziale $[0; 10]$.
- Odczytywana jest parametryzowalna liczba użytkowników.



Rysunek 3.1: Porównanie czasu wstawiania wielu rekordów.



Rysunek 3.2: Porównanie czasu pobierania wielu rekordów.

Bibliografia

[1] http://static.googleusercontent.com/media/research.google.com/pl//archive/bigtable-osdi06.pdf

[2] http://www.datastax.com/documentation/articles/cassandra/cassandrathenandnow.html

[3] http://cassandra.apache.org

[4] Towards Robust Distributed Systems, Dr. Eric A. Brewer, PODC Keynote

[5] The data model is dead, long live the data model, Patrick McFadin

[6] http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/

[7] https://wiki.apache.org/cassandra/API

[8] http://www.datastax.com/dev/blog/cassandra-anti-patterns-queues-and-queue-like-datasets

[9] http://www.slideshare.net/davegardnerisme/cassandra-concepts-patterns-and-antipatterns

[10] http://doctrine-orm.readthedocs.org/en/latest/reference/security.html

[11] https://github.com/impetus-opensource/Kundera

Oświadczenie

Oświadczam, że Pracę Dyplomową pod tytułem *"[TYTUŁ]"*, którą kierował dr inż. Jakub Koperwas, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Jakub Turek