

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI

---

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA  
MAGISTERSKA

Analiza i ocena wydajności mechanizmów  
replikacji w systemie bazodanowym  
wykorzystującym PostgreSQL

Analysis and performance evaluation of  
replication mechanisms in the  
PostgreSQL-based database system

AUTOR:

Adam Zimny  
PROWADZĄCY PRACĘ:

dr inż. Robert Wójcik, W4/K-9

OCENA PRACY:

---

WROCŁAW, 2018

# Spis treści

<b>Spis rysunków</b> . . . . .	<b>4</b>
<b>Spis tabel</b> . . . . .	<b>6</b>
<b>Spis listingów</b> . . . . .	<b>7</b>
<b>1. Wstęp</b> . . . . .	<b>8</b>
1.1. Cel i zakres pracy . . . . .	8
1.2. Metody replikacji w systemach bazodanowych . . . . .	9
1.3. Istniejące wyniki badań wydajności systemów baz danych . . . . .	10
<b>2. Analiza wymagań systemu</b> . . . . .	<b>12</b>
2.1. Architektura systemu . . . . .	12
2.2. Wymagania funkcjonalne . . . . .	12
2.2.1. Aplikacja serwerowa . . . . .	12
2.2.2. Analizowane architektury rozproszonej bazy danych . . . . .	13
2.3. Wymagania niefunkcjonalne . . . . .	15
2.3.1. Wykorzystywane technologie . . . . .	15
2.3.2. Narzędzia wspomagające ocenę wydajności systemów bazodanowych . . . . .	15
<b>3. Projekt systemu</b> . . . . .	<b>18</b>
3.1. Przypadki użycia . . . . .	18
3.2. Projekt bazy danych . . . . .	19
3.3. Projekt mechanizmów replikacji . . . . .	20
3.3.1. Pgbpool-II . . . . .	20
3.3.2. Bucardo . . . . .	21
3.3.3. Replikacja logiczna . . . . .	22

3.3.4.	Replikacja strumieniowa . . . . .	22
3.4.	Projekt testów wydajnościowych . . . . .	23
3.4.1.	Wykorzystywane operacje bazodanowe . . . . .	23
3.4.2.	Sposób prowadzenia testów . . . . .	23
<b>4.</b>	<b>Implementacja systemu . . . . .</b>	<b>25</b>
4.1.	Aplikacja serwerowa . . . . .	25
4.1.1.	Struktura plików projektu . . . . .	25
4.1.2.	Szczegóły implementacji wybranych funkcjonalności . . . . .	26
4.2.	Baza danych . . . . .	28
4.2.1.	Konfiguracja replikacji . . . . .	29
4.2.2.	Konfiguracja Bucardo . . . . .	31
4.2.3.	Konfiguracja Pgpool-II . . . . .	32
4.2.4.	Konfiguracja replikacji logicznej . . . . .	33
4.2.5.	Konfiguracja replikacji strumieniowej . . . . .	33
<b>5.</b>	<b>Testowanie i ocena wydajności systemu . . . . .</b>	<b>35</b>
5.1.	Przygotowanie środowiska testowego . . . . .	35
5.1.1.	Aplikacja serwerowa . . . . .	35
5.1.2.	JMeter . . . . .	36
5.2.	Wyniki pomiarów czasu wykonania zapytań . . . . .	38
5.2.1.	Odczyt wszystkich rekordów . . . . .	39
5.2.2.	Odczyt jednego rekordu . . . . .	40
5.2.3.	Zapis rekordu . . . . .	46
5.2.4.	Aktualizacja rekordu . . . . .	46
5.2.5.	Opóźnienie replikacji danych . . . . .	49
5.3.	Wnioski z testów . . . . .	52
<b>6.</b>	<b>Podsumowanie . . . . .</b>	<b>54</b>
	<b>Literatura . . . . .</b>	<b>56</b>
<b>A.</b>	<b>Opis załączonej płyty CD/DVD . . . . .</b>	<b>59</b>

# Spis rysunków

2.1. Schemat replikacji master-slave . . . . .	13
2.2. Okno programu JMeter [1] . . . . .	16
2.3. Okno programu Spawner Data Generator [2], definicja tabel . . . . .	17
2.4. Okno programu Spawner Data Generator [2], konfiguracja pliku wyjściowego . . . . .	17
3.1. Przypadki użycia aplikacji serwerowej . . . . .	18
3.2. Przypadki użycia aplikacji testującej . . . . .	19
3.3. Przykładowa tabela . . . . .	19
3.4. Zmodyfikowana tabela A na serwerze slave . . . . .	20
3.5. Architektura systemu pgpool . . . . .	21
3.6. Architektura systemu bucardo . . . . .	21
3.7. Architektura systemu replikacji logicznej . . . . .	22
3.8. Architektura systemu replikacji strumieniowej . . . . .	23
4.1. Struktura plików projektu . . . . .	26
4.2. Lista instancji maszyn wirtualnych w AWS [3] . . . . .	29
4.3. Reguły dostępu sieciowego [3] . . . . .	29
5.1. Tworzenie tabel przy pomocy frameworka Hibernate [4] . . . . .	36
5.2. JMeter - połączenie z bazą danych [1] . . . . .	36
5.3. JMeter - konfiguracja liczby aktywnych użytkowników [1] . . . . .	37
5.4. JMeter - definicja zmiennych losowych [1] . . . . .	37
5.5. JMeter - definicje zapytań testowych [1] . . . . .	38
5.6. Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu wszystkich danych . . . . .	41
5.7. Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu wszystkich danych . . . . .	42

5.8. Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu jednego rekordu . . . . .	44
5.9. Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu jednego rekordu . . . . .	45
5.10. Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu zapisu . . . . .	47
5.11. Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu zapisu . . . . .	48
5.12. Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu aktualizacji rekordu . . . . .	50
5.13. Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu aktualizacji rekordu . . . . .	51
5.14. Wykres zależności opóźnienia replikacji danych od liczby rekordów zapisywanych w jednym zapytaniu . . . . .	52

# Spis tabel

3.1. Schemat zapytań testowych . . . . .	23
3.2. Liczność rekordów w tabelach . . . . .	24
3.3. Liczba powtórzeń testu w zależności od liczby aktywnych użytkowników . . . . .	24
3.4. Liczba rekordów wstawianych do bazy danych w testach opóźnienia replikacji . . . . .	24
5.1. Zakresy wykorzystanych zmiennych losowych . . . . .	37
5.2. Wyniki badań opóźnienia replikacji systemów bucardo i logicznego . . . . .	49

# Spis listingów

4.1. Konfiguracja połączenia z bazą danych . . . . .	27
4.2. Przykładowa klasa modelu danych . . . . .	28
4.3. Konfiguracja bucardo . . . . .	31
4.4. Konfiguracja pgpool-II . . . . .	32
4.5. Konfiguracja replikacji strumieniowej na serwerze master . . . . .	33

# Rozdział 1

## Wstęp

Niniejszy dokument opisuje przebieg realizacji projektu dyplomowego magisterskiego o temacie *Analiza i ocena wydajności mechanizmów replikacji w systemie bazodanowym wykorzystującym PostgreSQL* na kierunku Informatyka, na wydziale Elektroniki Politechniki Wrocławskiej.

### 1.1. Cel i zakres pracy

Celem pracy jest opracowanie niezależnych systemów wykorzystujących różne dostępne metody replikacji danych i przetestowanie wpływu wybranej metody replikacji danych, jak i jej parametrów konfiguracyjnych na wydajność bazy danych. Miarą wydajności jest czas odpowiedzi, a w przypadku testów replikacji wielkość opóźnienia.

Zakres pracy obejmuje:

- analizę problemu w oparciu o dostępną literaturę,
- projekt oraz wykonanie systemu bazowanego opartego na PostgreSQL [5],
- projekt oraz wykonanie webowego interfejsu dostępowego do bazy danych,
- konfigurację mechanizmów replikacji bazy danych,
- projekt testów benchmarkowych pozwalających określić wydajność systemu,
- analizę statystyczną i ocenę porównawczą wyników badań wydajnościowych zastosowanych mechanizmów replikacji danych.



## 1.2. Metody replikacji w systemach bazodanowych

Replikacja to proces polegający na kopiowaniu bazy danych pomiędzy różnymi serwerami, w celu utworzenia kopii zapasowej lub wykorzystania jako serwer zapasowy. Stosowanie replikacji pozwala na zwiększenie bezpieczeństwa danych w razie awarii, a także zmniejszenie czasu dostępu do danych poprzez rozłożenie obciążeń pomiędzy serwery lub w przypadku czasowej niedostępności serwerów [6].

Metody replikacji danych podzielić można na synchroniczne oraz asynchroniczne. W przypadku metod synchronicznych transakcja zatwierdzana jest dopiero, gdy wszystkie serwery potwierdzą jej poprawne wykonanie. Zapewnia to spójność danych we wszystkich serwerach bez występowania opóźnień, jednak może powodować większe opóźnienia lub zawieszenie transakcji w przypadku niedostępności któregoś z serwerów. Replikacja synchroniczna wymaga dużych przepustowości łącza, co przekłada się na duży koszt wytworzenia i utrzymania systemu.

W przypadku replikacji asynchronicznej, dane kopiowane są co określony czas lub po uaktywnieniu wyzwalacza. Konsekwencją takiego sposobu replikacji jest minimalizacja czasu dostępu w trakcie dodawania lub edycji rekordów, co przekłada się na dużą wydajność aplikacji, jednak może powodować powstawanie czasowych niespójności pomiędzy bazą główną a replikami. W zależności od przepustowości sieci, konfiguracji systemu i rozmiaru danych opóźnienie to może wynosić nawet do kilku godzin. Stosowanie replikacji asynchronicznej pozwala na znaczne zmniejszenie kosztu wytworzenia systemu ze względu na znacznie niższe wymagania.

Ze względu na sposób przesyłania danych metody replikacji można podzielić na *statement-based* i *row-based*. Replikacja *statement-based* polega na ponownym wykonaniu na każdym z serwerów zapytań, które zostały użyte do wprowadzenia zmian na serwerze głównym. Replikacja *row-based* tworzy kopie rekordów powielając rzeczywiste zmiany w tabelach [7].

W zależności od sposobu dostępu przez użytkownika wyróżnić można systemy master-slave lub multimaster. W systemach master-slave występuje główny węzeł replikacji, z którym komunikują się użytkownicy, a zmiany dokonywane na nim propagowane są do serwerów slave. W architekturze multimaster serwery są sobie równoważne, użytkownicy mogą dokonywać zmian na dowolnym serwerze.

## 1.3. Istniejące wyniki badań wydajności systemów baz danych

Wraz z powstawaniem nowych systemów zarządzania bazami danych pojawiają się liczne artykuły i prace zajmujące się ich porównywaniem. Większość z nich opisuje jednak różnice w dostarczanych funkcjonalnościach, a rzadziej w wydajności systemów. Zdecydowana większość testów wydajnościowych, których wyniki są publikowane przeprowadzana jest na bazach NoSQL. Duże trudności sprawia również znalezienie prac skupiających się na systemach replikacji.

W artykule *Benchmark: PostgreSQL, MongoDB, Neo4j, OrientDB and ArangoDB* [8] opisane są wyniki testów porównawczych wydajności wymienionych systemów. Do analizy czasu wykonania podstawowych operacji bazodanowych wykorzystana została baza danych portalu społecznościowego Pokec dostępna ze strony Stanford Network Analysis Project [9]. Wykonane zostały następujące testy:

- zapis jednego dokumentu (stworzenie profilu użytkownika),
- odczyt jednego dokumentu (odczyt profilu użytkownika),
- agregacja danych (uzyskanie statystyk o rozkładzie wieku użytkowników).

Badania wykazały, że bazy danych PostgreSQL mogą konkurować z bazami NoSQL pod względem szybkości dla operacji odczytu i zapisu, lecz są przeważnie wolniejsze w operacjach agregacji danych.

W prezentacji z pracy dyplomowej *Analiza porównawcza wybranych własności systemów zarządzania bazami danych* [10] Mirosława Lacha przedstawione zostały wyniki badań wydajności systemów PostgreSQL, Microsoft SQL Server, MySQL oraz Oracle. Testowany był wpływ zmieniającej się ilości danych oraz liczby użytkowników na czasy odpowiedzi baz danych. Wyniki badań pokazują, że PostgreSQL wypada od pozostałych systemów nieznacznie gorzej, jednak we wszystkich systemach występują jednakowe zależności czasu odpowiedzi od obciążenia.

Artykuł *Millions of Queries per Second: PostgreSQL and MySQL's Peaceful Battle at Today's Demanding Workloads* opublikowany na blogu Percona [11] opisuje porównanie wydajności baz MySQL oraz PostgreSQL. Wykorzystane zostały bardzo wydajne maszyny, złożone z czterech procesorów posiadających łącznie 72 rdzenie oraz 3 TB pamięci RAM. Do badań użyte zostały standardowe testy dostępne w narzędziach pgbench [12] dla PostgreSQL oraz SysBench [13] dla MySQL, jednak autor artykułu nie podaje szczegółowych informacji na temat ich implementacji.

Zbadany został wpływ liczby aktywnych użytkowników na liczbę zapytań przetwarzanych na sekundę. Badania pokazały, że PostgreSQL jest bardziej wydajny w zapytaniach odczytu danych oraz osiąga bardzo zbliżone wyniki w operacjach zapisu.

Przeprowadzając analizę dostępnych wyników badań wydajności nie znaleziono prac skupiających się bezpośrednio na tematyce replikacji danych, a w szczególności na wykorzystaniu mechanizmów replikacji badanych w tej pracy.

## Rozdział 2

# Analiza wymagań systemu

### 2.1. Architektura systemu

Po przeprowadzeniu analizy literaturowej problemu pracy opracowana została koncepcja rozwiązania. System złożony będzie z następujących części:

- aplikacja serwerowa udostępniająca API w stylu REST pozwalająca na wywoływanie zapytań na bazie danych,
- bazy danych wykorzystujące wybraną metodę replikacji danych,
- oprogramowanie testowe bazy danych.

Aplikacje obciążające bazy danych dla celów testowych pozwalają na wysyłanie zapytań zarówno bezpośrednio do bazy danych, jak i za pośrednictwem zapytań HTTP. Rolą aplikacji serwerowej będzie przekierowanie zapytania HTTP do odpowiedniej bazy danych i wywołanie na niej zapytania SQL. W aplikacji serwerowej zostaną skonfigurowane połączenia do wszystkich utworzonych baz danych, a wybór bazy, która ma zostać obciążona zapytaniami testowymi odbywać się będzie w momencie uruchomienia aplikacji.

### 2.2. Wymagania funkcjonalne

#### 2.2.1. Aplikacja serwerowa

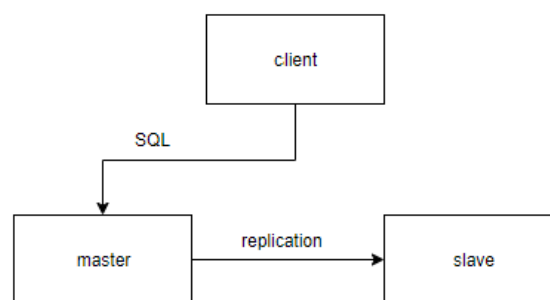
Wymagania aplikacji serwerowej skupiają się wokół zapewnienia komunikacji pomiędzy oprogramowaniem testującym a bazami danych.

- Dostęp do aplikacji serwerowej powinien być uzyskiwany poprzez zapytania HTTP.
- Dostęp nie powinien wymagać logowania ani autoryzacji.
- Aplikacja w momencie uruchomienia powinna zweryfikować schemat bazy danych, a w razie potrzeby utworzyć brakujące tabele lub dokonać niezbędnych modyfikacji na tabelach istniejących.
- Aplikacja powinna pozwalać na wykonanie zaprojektowanych zapytań testujących na analizowanej bazie danych.
- Aplikacja powinna pozwalać na wybór bazy danych, na której prowadzone będą testy. Wyboru dokonuje się poprzez edycję pliku konfiguracyjnego.

### 2.2.2. Analizowane architektury rozproszonej bazy danych

- W pracy badane będą cztery systemy replikacji danych:
  - Bucardo [14],
  - pgpool-II [15],
  - replikacja logiczna [16],
  - replikacja strumieniowa [17].

Systemy bucardo, replikacji logicznej i replikacji strumieniowej działają w trybie asynchronicznym i powinny zostać skonfigurowane według architektury master-slave. Replikacja danych odbywać się będzie w jednym kierunku, z serwera master na serwer slave. Wyjątkiem jest system pgpool-II, który działa w trybie synchronicznym.



Rys. 2.1: Schemat replikacji master-slave

#### Bucardo

Bucardo [14] to oprogramowanie dostarczane przez firmę End Point Corporation. Podstawą jego działania jest proces nasłuchujący zapytań NOTIFY generowanych w trakcie dokonywania

zmian na replikowanych bazach danych i reagowanie na nie. Informacje potrzebne do działania systemu przechowywane są w głównej bazie danych bucardo tworzonej na serwerze master obok bazy replikowanej. Baza bucardo zawiera listę wszystkich replikowanych baz oraz informacje o sposobie uzyskania połączenia z nimi. Do działania konieczne jest zdefiniowanie listy tabel podlegających replikacji. Po konfiguracji triggerów rozpoczynają zapisywanie informacji o tym, które rekordy w obserwowanych tabelach zostały zmienione, a następnie ich zawartość jest przesyłana do pozostałych serwerów. Replikacja działa w trybie asynchronicznym.

### **Pgpool-II**

Pgpool-II [15] jest to middleware, czyli oprogramowanie pośredniczące w wymianie danych pomiędzy klientem a serwerem bazodanowym. Udostępnia ono funkcjonalności replikacji oraz równoważenia obciążenia pomiędzy serwerami. Implementowanym typem replikacji jest *statement-based replication*, który opiera się na wykonywaniu zadanych zapytań SQL na każdym serwerze podlegającym kopiowaniu danych. Pgpool-II jest systemem replikacji synchronicznej. W celu zredukowania czasu potrzebnego na przetworzenie zapytania pgpool otwiera stałą liczbę połączeń z serwerami bazodanowymi i utrzymuje je pomiędzy kolejnymi zapytaniami.

### **Replikacja logiczna**

Dostępny w bazach danych PostgreSQL od wersji 10 system replikacji logicznej [16] wykorzystuje przesyłanie listy zmian pomiędzy serwerami. Na serwerze master polega to na tworzeniu publikacji będących listą rekordów, w których nastąpiła zmiana, zaś na serwerach slave tworzone są subskrypcje wykrywające zmiany danych w określonych publikacjach. System ten nie obsługuje zapytań typu DDL (z ang. Data Definition Language, zapytania modyfikujące strukturę bazy danych). Replikacja odbywa się poprzez kopiowanie zawartości rekordów o określonym kluczu głównym z serwera master na serwery slave. Działa w trybie asynchronicznym.

### **Replikacja strumieniowa**

Replikacja strumieniowa [17] jest dostępna w bazach danych PostgreSQL od wersji 9. Do tworzenia kopii danych wykorzystuje ona funkcje archiwizacyjne, które tworzą w systemie plików serwera dziennik zmian dokonywanych na bazie danych. Poprzez przesyłanie tego dziennika

do innych serwerów możliwe jest odtworzenie bazy danych na podstawie jego wpisów. Replikacja strumieniowa jest metodą asynchroniczną, jednak cechuje się wystarczającą szybkością, aby pełnić rolę replikacji synchronicznej dla niewielkich ilości danych. Utworzony serwer kopii zapasowej pracuje w trybie tylko do odczytu.

## **2.3. Wymagania niefunkcjonalne**

### **2.3.1. Wykorzystywane technologie**

Oprócz opisanych wyżej technologii replikacji oraz systemu bazodanowego PostgreSQL podczas realizacji projektu wykorzystane zostaną dodatkowe narzędzia:

- Aplikacja serwerowa zostanie zaimplementowana w języku Java z wykorzystaniem frameworka Spring Boot [18].
- Do utworzenia warstwy dostępu do danych użyty zostanie framework Hibernate [4] oraz standard JPA [19].
- Jako serwis hostingowy serwerów bazodanowych wykorzystany zostanie portal Amazon Web Services [3].

### **2.3.2. Narzędzia wspomagające ocenę wydajności systemów bazodanowych**

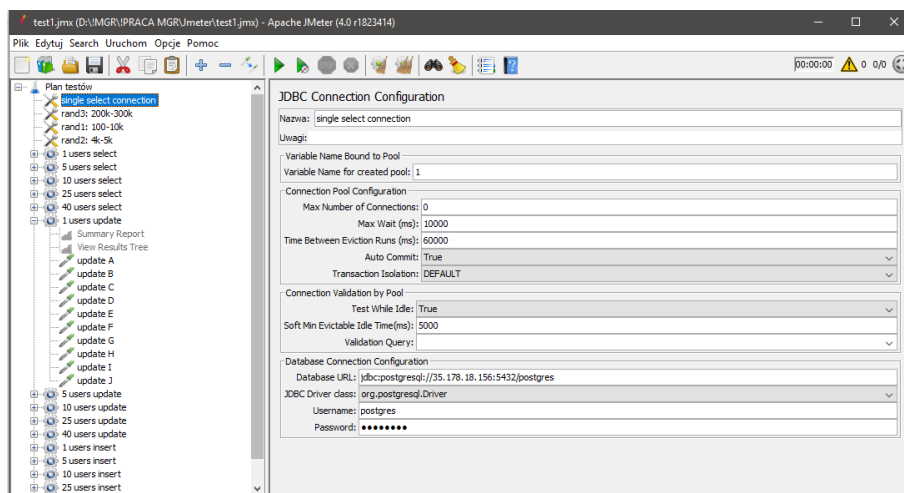
#### **Pgbench**

Pgbench [12] jest oprogramowaniem benchmarkowym dostarczającym razem z instalacją bazy danych PostgreSQL. Do narzędzia uzyskuje się dostęp z poziomu linii poleceń systemu. Pgbench pozwala na utworzenie tabel testowych w wyznaczonej bazie danych oraz wypełnienie ich przykładowymi danymi. Bazę danych inicjalizuje się komendą `pgbench` z flagą `-i`. Możliwe jest zdefiniowanie dodatkowych parametrów:

- `-h` - adres hosta,
- `-p` - port,
- `-s` - skala generowanych danych (domyślnie generowane jest 100 tys. wierszy),
- `-d` - baza danych,
- `-U` - nazwa użytkownika,
- `-P` - hasło użytkownika.

## JMeter

Aplikacja JMeter [1] tworzona przez organizację Apache Software Foundation to narzędzie zaprojektowane do projektowania testów obciążeniowych i mierzenia wydajności. W celu dokonania pomiaru konieczne jest utworzenie planu testowego, składającego się z konfiguracji połączeń, grup wątków imitujących użytkowników systemu oraz zapytań wykonywanych na bazie danych. Możliwe jest też użycie dodatkowych elementów, jak np. generatory zmiennych losowych oraz tworzenie raportów i wykresów.



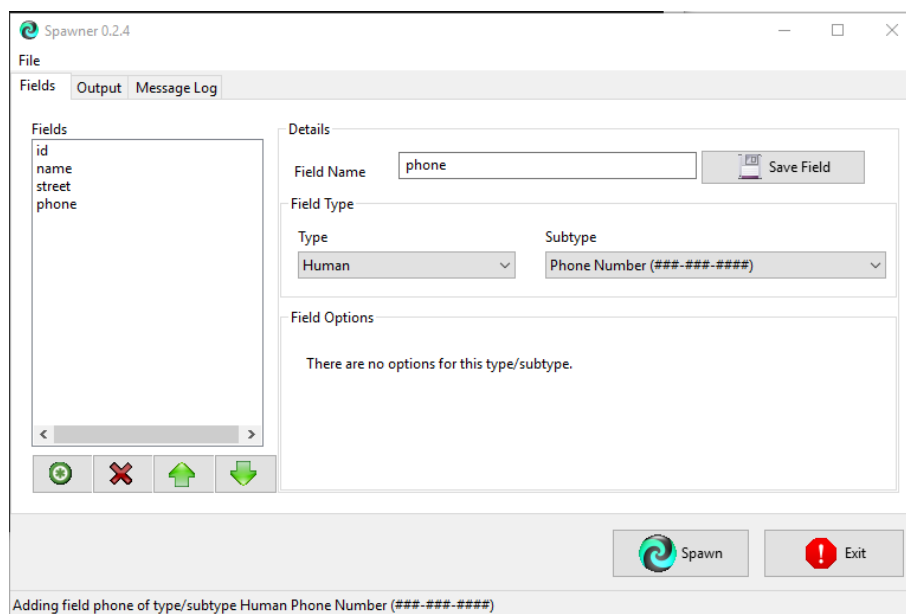
Rys. 2.2: Okno programu JMeter [1]

Program udostępnia możliwość uruchomienia w trybie konsolowym na czas prowadzenia testów wydajnościowych. Interfejs graficzny służy do projektowania testów i pozwala na ich próbne uruchomienie.

## Spawner Data Generator

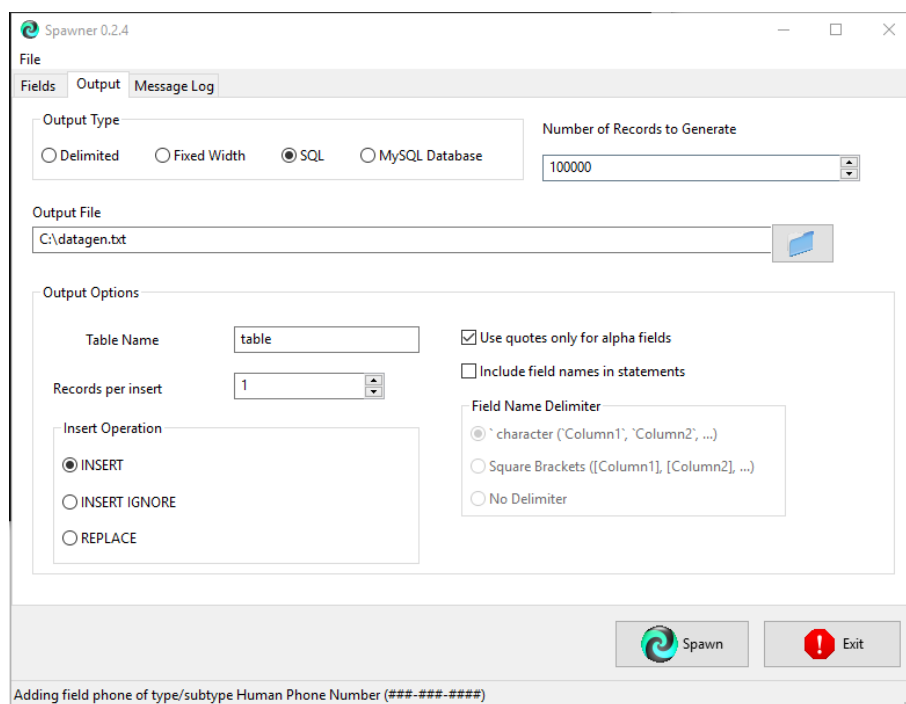
Program Spawner Data Generator [2] jest generatorem przykładowych danych na potrzeby testów systemów bazodanowych. Praca z programem polega na odtworzeniu schematu tabeli z bazy danych oraz zdefiniowanie dla każdego pola typu danych, jakie mają zostać wygenerowane. Możliwe jest generowanie realistycznych danych, takich jak np. numery telefonu, imiona, adresy e-mail, jak również danych przypadkowych, np. losowe ciągi znaków, liczby czy losowe ciągi słów z tekstu *Lorem ipsum*.





Rys. 2.3: Okno programu Spawner Data Generator [2], definicja tabel

Generowane dane można zapisać do pliku w formie z separatorem lub jako zapytania SQL. W przeciwieństwie do licznych rozwiązań dostępnych online, Spawner Data Generator nie posiada ograniczenia liczby generowanych rekordów.



Rys. 2.4: Okno programu Spawner Data Generator [2], konfiguracja pliku wyjściowego

## Rozdział 3

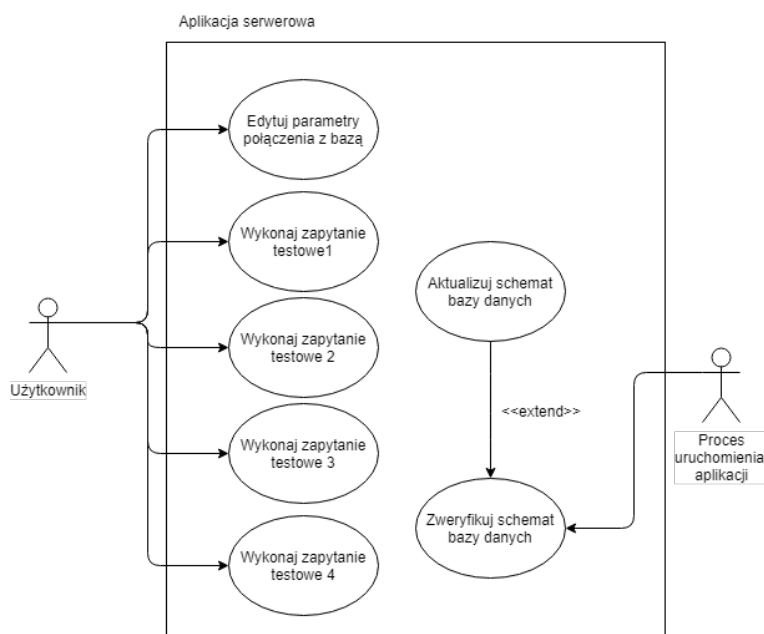
# Projekt systemu

Na podstawie określonych wymagań opracowany został projekt systemu. Na projekt składa się analiza przypadków użycia systemu, projekt schematu bazy danych, projekt mechanizmów replikacji oraz plan prowadzenia testów wydajnościowych.

### 3.1. Przypadki użycia

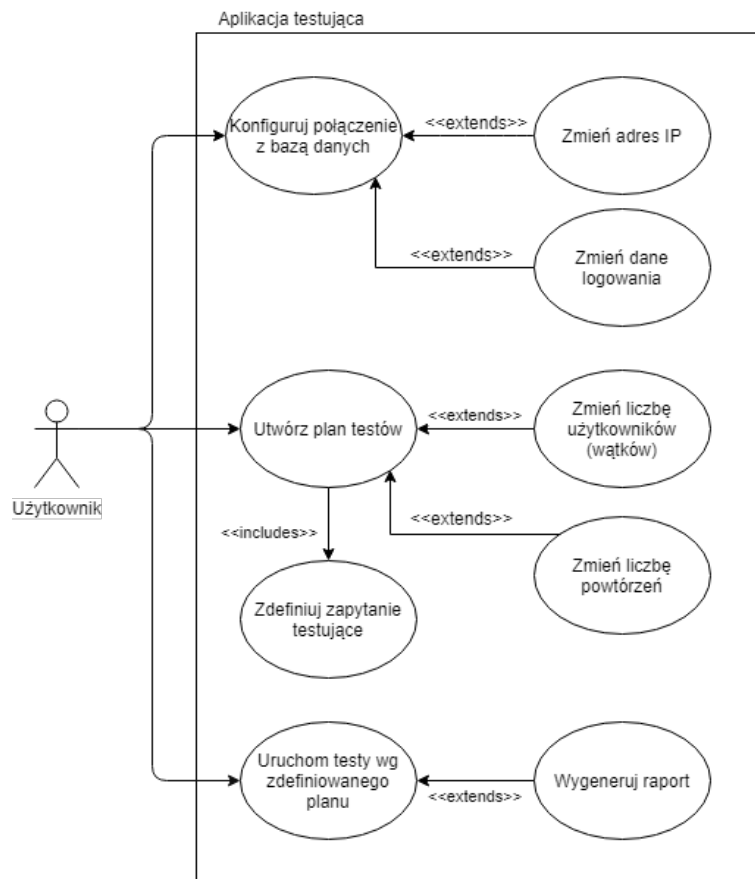
Na poniższych rysunkach przedstawiono przypadki użycia systemu.

#### Aplikacja serwerowa



Rys. 3.1: Przypadki użycia aplikacji serwerowej


## Aplikacja testująca



Rys. 3.2: Przypadki użycia aplikacji testującej


## 3.2. Projekt bazy danych

Na potrzeby testów utworzona zostanie baza danych składająca się z 10 jednakowych tabel, oznaczonych literami od *A* do *J*. Tabele składać się będą z 7 kolumn: numeru id, 3 kolumn numerycznych oraz 3 kolumn tekstowych, oznaczonych literami od *a* do *f*. Schemat tabeli przedstawia rysunek 3.3.

c	
 id	INTEGER
a	INTEGER
b	INTEGER
c	INTEGER
d	CHARACTER VARYING(255)
e	CHARACTER VARYING(255)
f	CHARACTER VARYING(255)

Rys. 3.3: Przykładowa tabela

W tabeli A dodana zostanie dodatkowa kolumna *timestamp*, z domyślną wartością *current\_timestamp*, zapisująca czas utworzenia każdego rekordu w bazie danych. Dodatkowo w tabeli A na serwerze pełniącym funkcję *slave* utworzona zostanie dodatkowa kolumna *replica\_timestamp* o takich samych właściwościach, jak kolumna *timestamp*. Kolumny te posłużą do ocenienia opóźnienia replikacji przy wstawianiu rekordów.

a	
 <b>id</b>	INTEGER
<b>a</b>	INTEGER
<b>b</b>	INTEGER
<b>c</b>	INTEGER
<b>d</b>	CHARACTER VARYING(255)
<b>e</b>	CHARACTER VARYING(255)
<b>f</b>	CHARACTER VARYING(255)
<b>timestamp</b>	TIMESTAMP(6) WITHOUT TIME ZONE
<b>replica_timestamp</b>	TIMESTAMP(6) WITHOUT TIME ZONE

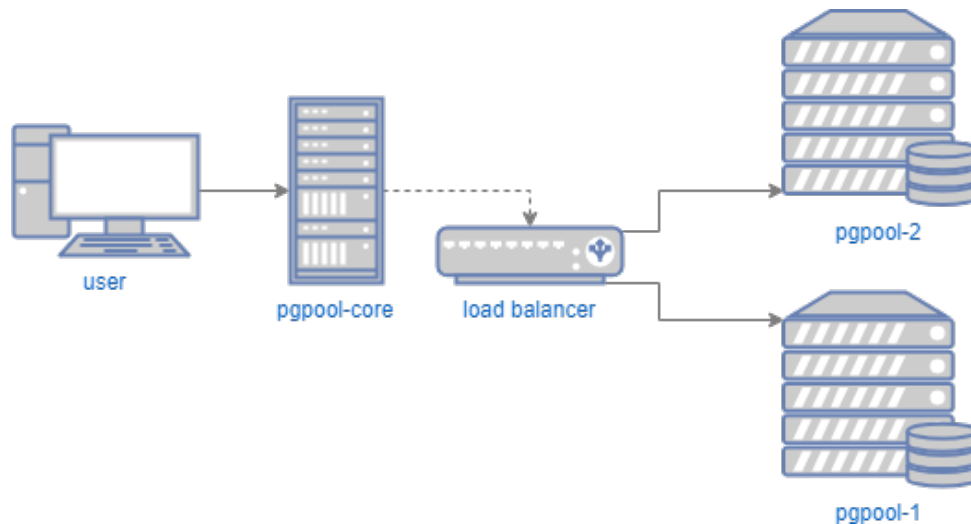
Rys. 3.4: Zmodyfikowana tabela A na serwerze slave

### 3.3. Projekt mechanizmów replikacji

#### 3.3.1. Pgpool-II

Pgpool-II jest oprogramowaniem działającym w warstwie middleware, co oznacza, że pośredniczy ono w komunikacji między użytkownikiem, a innymi warstwami systemu, w tym wypadku bazami danych. Oprogramowanie może działać jako proces na jednym z komputerów będących jednocześnie serwerem baz danych, jak i jako samodzielna maszyna [15].

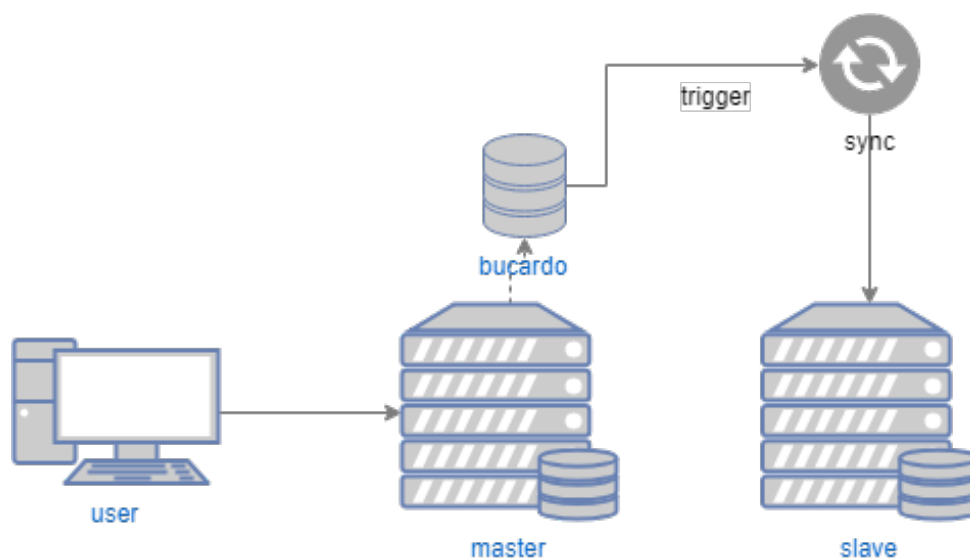
Na potrzeby testów utworzone zostaną 3 serwery: pgpool-core, na którym zostanie uruchomione oprogramowanie pgpool-II oraz serwery pgpool1 oraz pgpool2, będące serwerami baz danych PostgreSQL. Proces aplikacji pgpool pełni rolę tzw. *load balancera*, który automatycznie decyduje, na który z serwerów przekazać zapytanie zależnie od obciążenia sieci. Dostęp do baz danych odbywać się będzie przez domyślny port 5432. Domyślnym portem obsługi zapytań na serwerze pgpool-core jest 9999, jednak ze względu na to, że nie ma na nim instancji PostgreSQL wykorzystany zostanie port 5432. Konfiguracja na serwerze pgpool-core wymaga podania adresów IP wszystkich serwerów baz danych. Projektowany system wymagał będzie posiadania dwóch adresów statycznych.



Rys. 3.5: Architektura systemu pgpool

### 3.3.2. Bucardo

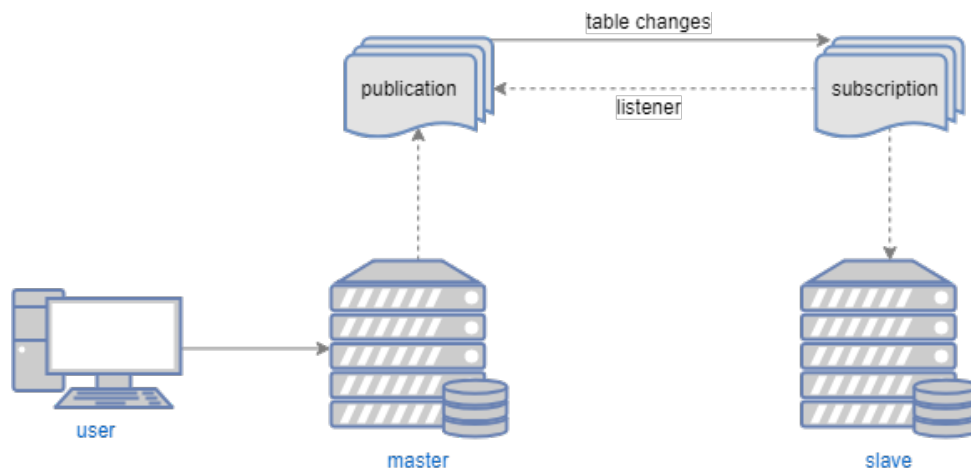
System replikacji bucardo wymaga do działania instancji bazy PostgreSQL. Informacje o zmianach zachodzących w tabelach, a także konfiguracja replikacji przechowywane są w lokalnej bazie danych o nazwie bucardo. Z tego względu zostanie on zainstalowany na serwerze master, a replikacja zostanie skonfigurowana z bazy lokalnej do bazy na serwerze zdalnym. Replikacja wymaga również utworzenia użytkownika bucardo z uprawnieniami SUPERUSER na wszystkich serwerach. Komunikacja użytkownika będzie odbywać się wyłącznie z serwerem master, na domyślnym porcie 5432, a zmiany replikowane asynchronicznie na serwer slave. Wymagane będzie posiadanie jednego statycznego adresu IP dla serwera slave [20].



Rys. 3.6: Architektura systemu bucardo

### 3.3.3. Replikacja logiczna

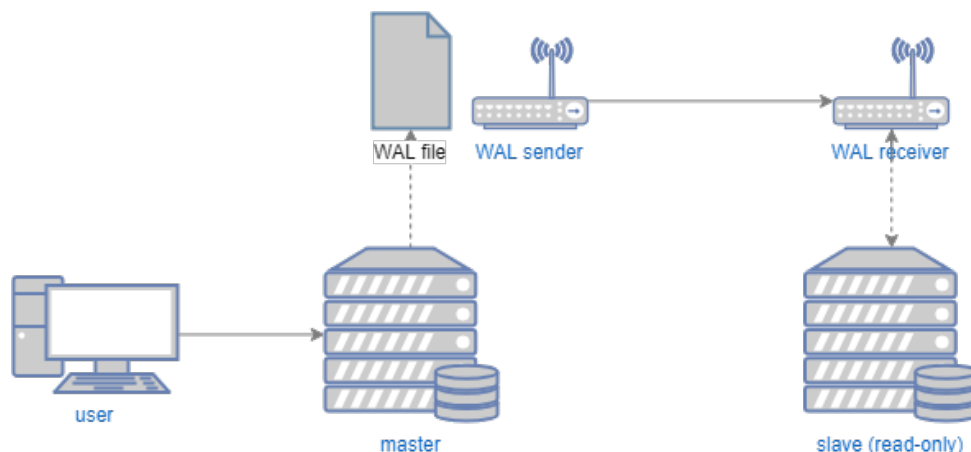
System replikacji logicznej składać się będzie z dwóch serwerów. Na serwerze master utworzona zostanie publikacja obejmująca wszystkie tabele bazy danych postgres. Publikacja rejestruje wszystkie operacje dokonywane na bazie danych i udostępnia je do odczytu przez subskrybentów. Na serwerze slave utworzona zostanie subskrypcja nasłuchująca zmian w publikacji serwera master i kopiująca te zmiany w sposób asynchroniczny do lokalnej bazy danych. Dla użytkownika udostępniony zostanie port 5432 serwera master. Wymagane będzie posiadanie jednego statycznego adresu IP dla serwera master [21].



Rys. 3.7: Architektura systemu replikacji logicznej

### 3.3.4. Replikacja strumieniowa

System replikacji strumieniowej wykorzystuje dwa serwery. Wykorzystując funkcje archiwizacji bazy danych na serwerze master utworzony zostanie plik WAL (Write Ahead Log) przechowujący zmiany dokonywane na lokalnej bazie danych. Przy pomocy procesów WAL sender oraz WAL receiver w trybie asynchronicznym zostanie utworzona kopia tego pliku na serwerze slave, który następnie wykorzysta go do odtworzenia zmian. Do zestawienia połączenia konieczne jest posiadanie jednego statycznego adresu IP dla serwera slave. Serwer slave działać będzie w trybie hot-standby, oznacza to, że będzie działać w trybie tylko do odczytu i obsługiwać wyłącznie zapytania SELECT. Użytkownik komunikować się będzie z serwerem master na porcie 5432 [22].



Rys. 3.8: Architektura systemu replikacji strumieniowej

## 3.4. Projekt testów wydajnościowych

### 3.4.1. Wykorzystywane operacje bazodanowe

Testy wydajnościowe systemu przeprowadzone zostaną przy pomocy zapytań CRUD. Wykorzystane zostaną operacje SELECT, INSERT oraz UPDATE. Schemat zapytań wykorzystywanych w testach przedstawia poniższa tabela.

Tab. 3.1: Schemat zapytań testowych

nazwa testu	schemat zapytania
All select	SELECT * FROM <table>
Single select	SELECT * FROM <table> where id = <value>
Insert	INSERT INTO <table> VALUES (<value1>,<value2>,...)
Update	UPDATE <table> SET <column> = <value1> WHERE id = <value2>

### 3.4.2. Sposób prowadzenia testów

Przed rozpoczęciem testów tabele w bazie danych zostaną wypełnione przykładowymi danymi w następującej liczbie:

Tab. 3.2: Liczność rekordów w tabelach

tabela	liczba rekordów
A	10 000
B	20 000
C	30 000
D	40 000
E	50 000
F	60 000
G	70 000
H	80 000
I	90 000
J	100 000

Dla każdej tabeli przeprowadzone zostaną testy dla pięciu różnych liczb aktywnych użytkowników: 1, 5, 10, 25, 40. Testy zostaną powtórzone tak, aby wykonane zostało co najmniej 50 zapytań dla każdej kombinacji parametrów wejściowych. Liczbę wymaganych powtórzeń przedstawia tabela 3.3.

Tab. 3.3: Liczba powtórzeń testu w zależności od liczby aktywnych użytkowników

liczba użytkowników	liczba powtórzeń testu
1	50
5	10
10	5
25	2
40	2

W celu analizy opóźnienia replikacji przeprowadzone zostaną dodatkowe testy z wykorzystaniem zapytania INSERT. Badany będzie wpływ liczby rekordów zapisywanych w jednym zapytaniu na czas replikacji.

Tab. 3.4: Liczba rekordów wstawianych do bazy danych w testach opóźnienia replikacji

Liczba rekordów w zapytaniu	zakres numerów id	Liczba zapytań	Liczba rekordów
1	0 - 9 999	10 000	10 000
20	20 000 - 29 999	500	10 000
40	40 000 - 49 999	250	10 000
60	60 000 - 69 999	166	9 960
80	80 000 - 89 999	125	10 000
100	100 000 - 109 999	100	10 000



## Rozdział 4

# Implementacja systemu

W tym rozdziale została opisana implementacja zaprojektowanego systemu. Projekt systemu zakładał stworzenie aplikacji serwerowej do komunikacji z bazami danych, czterech systemów replikacji składających się z dwóch instancji bazy danych, oraz wykorzystanie aplikacji benchmarkowej do testów wydajnościowych.

Po przeprowadzeniu wstępnych testów na częściowo ukończonym systemie konieczne było wprowadzenie pewnej modyfikacji. Ze względu na zastosowane w implementacji technologie testy wydajnościowe prowadzone będą bezpośrednio na bazach danych, z pominięciem aplikacji serwerowej. Jest to spowodowane użyciem frameworka Hibernate [4] oraz Spring JPA [19] do realizacji warstwy dostępu do bazy danych, co ma wpływ na otrzymywane wyniki pomiaru czasu dostępu i mogłoby prowadzić do niedokładnej analizy. Aplikacja serwerowa wykorzystana zostanie do utworzenia, weryfikacji i aktualizacji schematu bazy danych na wybranym serwerze przy pomocy mapowania obiektowo-relacyjnego frameworka Hibernate.

### 4.1. Aplikacja serwerowa

#### 4.1.1. Struktura plików projektu

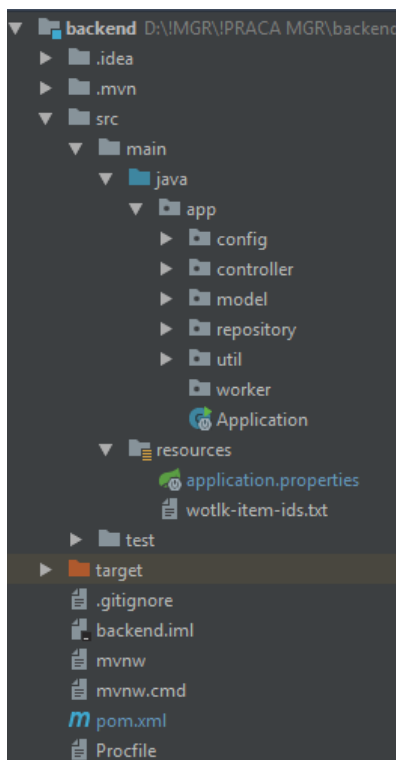
Projekt został wykonany w środowisku IntelliJ IDEA [23]. Pliki klas projektu zostały podzielone na paczki zgodnie z ich przeznaczeniem:

- config - klasy konfiguracyjne,
- model - model danych,
- controller - kontrolery zapytań HTTP,

- repository - klasy realizujące dostęp do bazy danych,
- util - dodatkowe funkcje użytkowe.

Ze względu na zmianę koncepcji realizacji projektu, przygotowana struktura plików uwzględnia elementy, których implementacja nie była wymagana do przeprowadzenia testów, np. paczka controller czy też repository.

Oprócz plików klas Javy istotnymi z punktu widzenia implementacji są pliki `application.properties` oraz `pom.xml`.



Rys. 4.1: Struktura plików projektu

### 4.1.2. Szczegóły implementacji wybranych funkcjonalności

#### Konfiguracja połączenia z bazą danych

Konfigurację warstwy połączenia z bazą danych realizuje klasa `DatabaseConfig`. Połączenie z bazą danych realizowane jest przy pomocy sterownika JDBC [24] za pośrednictwem frameworka Hibernate. Wykorzystuje ona plik `application.properties`, z którego pobiera dane do połączenia z bazą danych. W wyniku działania przedstawionych na poniższym listingu metod konfiguracyjnych tworzony jest obiekt `dataSource`, który zawiera niezbędne do ustanowienia połączenia dane, a także definiowane są dodatkowe ustawienia powiązane z frameworkiem Hibernate, między innymi sterownik bazy danych oraz ścieżka do plików modelu danych.

Listing 4.1: Konfiguracja połączenia z bazą danych

```

@Configuration
@EnableTransactionManagement
public class DatabaseConfig {

    @Autowired
    private Environment env;

    @Bean(name = "dataSource")
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(env.getProperty("db.driver"));
        dataSource.setUrl(env.getProperty("db.url"));
        dataSource.setUsername(env.getProperty("db.username"));
        dataSource.setPassword(env.getProperty("db.password"));
        return dataSource;
    }

    [...]
}

```

Parametry wykorzystywane w ramach konfiguracji określone są w pliku `application.properties`. Przez zmianę podanego adresu IP dokonuje się wyboru testowanej bazy danych. Pozostałe parametry określają dane logowania i sposób aktualizacji schematu bazy danych podczas uruchomienia aplikacji.

```

# Database
db.driver= org.postgresql.Driver
db.url= jdbc:postgresql://35.177.119.219:5432/postgres
db.username= postgres
db.password= postgres

# Hibernate
hibernate.dialect= org.hibernate.dialect.PostgreSQL94Dialect
hibernate.show_sql= true
hibernate.hbm2ddl.auto= update
entitymanager.packagesToScan= app.model

```

## Model danych

W projekcie zostało wykorzystane mapowanie obiektowo-relacyjne do utworzenia schematu bazy danych. Tabele w schemacie definiowane są przez klasy znajdujące się w pakiecie `model`.

Listing 4.2: Przykładowa klasa modelu danych

```
@Getter
@Setter
@Entity
public class A {

    @Id
    int id;

    int a,b,c;
    String d,e,f;

    @Column(columnDefinition = "timestamp default CURRENT_TIMESTAMP")
    Date timestamp;
}
```

Adnotacje `@Getter` oraz `@Setter` pochodzą z biblioteki Lombok [25] i służą do automatycznego generowania metod dostępu do zmiennych. Adnotacja `@Entity` deklaruje klasę jako tabelę modelu, a `@Id` określa klucz główny tabeli. Kolumna `timestamp` posiada dodatkowe parametry definiujące, które ustawiają jej domyślną wartość na czas utworzenia rekordu w bazie. Jest to wykorzystywane do pomiaru opóźnienia replikacji.

## 4.2. Baza danych

Jako serwis hostingowy serwerów bazodanowych wybrano Amazon Web Services [3]. Za pomocą tej platformy możliwe jest utworzenie wirtualnej maszyny o określonych zasobach na serwerach Amazon. Serwis udostępnia możliwość utworzenia maszyny posiadającej 1 rdzeń procesora, 1 GB pamięci RAM i 30 GB miejsca na dysku SSD bez dodatkowych opłat, oraz 750 godzin bezpłatnej pracy serwerów. Pula ta rozkłada się pomiędzy wszystkie aktywne serwery.

Na potrzeby projektu utworzone zostało 9 serwerów działających pod kontrolą systemu Ubuntu 16.04.

Launch Instance

Connect

Actions

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Name	Instanc	Inst	Avail	Instance State	Stat	Alarm	Public C	IPv4 Public IP
<input type="checkbox"/>	logical master	i-0e2...	t2....	eu-...	<span>●</span> running	<span>✓</span> ...	N...	ec2-35...	35.177.38.10
<input type="checkbox"/>	pgpool 2	i-009...	t2....	eu-...	<span>●</span> stopped		N...	ec2-35...	35.178.97.25
<input type="checkbox"/>	bucardo slave	i-00c...	t2....	eu-...	<span>●</span> stopped		N...	ec2-52...	52.56.122.85
<input type="checkbox"/>	pgpool 1	i-00d...	t2....	eu-...	<span>●</span> stopped		N...	ec2-35...	35.176.165.93
<input type="checkbox"/>	pgpool core	i-039...	t2....	eu-...	<span>●</span> stopped		N...	-	-
<input type="checkbox"/>	stream slave	i-080...	t2....	eu-...	<span>●</span> stopped		N...	-	-
<input type="checkbox"/>	stream master	i-0af0...	t2....	eu-...	<span>●</span> stopped		N...	ec2-35...	35.177.141.137
<input type="checkbox"/>	logical slave	i-0d8...	t2....	eu-...	<span>●</span> stopped		N...	-	-
<input type="checkbox"/>	bucardo master	i-0f2d...	t2....	eu-...	<span>●</span> stopped		N...	-	-

Rys. 4.2: Lista instancji maszyn wirtualnych w AWS [3]

Domyślnie, utworzone serwery posiadają dynamiczne adresy IP przypisywane w momencie uruchomienia maszyny. Serwis hostingowy AWS pozwala na zarezerwowanie 5 statycznych adresów na konto i przypisanie ich do wybranych maszyn. Sposób ich przypisania pokazano na rysunku 4.2.

W procesie tworzenia maszyny konieczne jest określenie reguł dostępu sieciowego do serwerów. Utworzone zostały następujące reguły:

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
Custom TCP Rule	TCP	5432	0.0.0.0/0
Custom TCP Rule	TCP	9999	0.0.0.0/0
Custom ICMP Rule - IPv4	Echo Reply	N/A	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0

Rys. 4.3: Reguły dostępu sieciowego [3]

- port 5432 - port komunikacyjny baz danych PostgreSQL,
- port 9999 - port komunikacyjny pgpool-II,
- Echo Reply - odpowiedź serwera na komendę ping,
- port 22 - połączenie SSH do konfiguracji serwera.

### 4.2.1. Konfiguracja replikacji

Konfigurację replikacji na wszystkich serwerach rozpoczęto od instalacji bazy danych PostgreSQL. System Ubuntu w wersji 16.04 posiada w repozytoriach źródła dla bazy PostgreSQL w

wersji 9.4, jednak w projekcie konieczne jest posiadanie wersji 10. Zainstalowanie tej wersji wymaga kilku dodatkowych kroków. Pierwszym krokiem jest instalacja pakietu `ssl-cert`.

W konfiguracji programu `apt` używanego do instalacji konieczne jest dodanie adresu do repozytorium zawierającego kod programu. Należy edytować plik `/etc/apt/sources.list` i dodać do niego wpis `deb http://cz.archive.ubuntu.com/ubuntu xenial main`. Następnie należy zaktualizować konfigurację źródeł programu komendą `sudo apt update` oraz dokonać instalacji komendą `sudo apt install ssl-cert`.

Kolejnym krokiem jest dodanie do źródeł programu `apt` adresu do wersji 10 bazy PostgreSQL. Służy do tego plik `/etc/apt/sources.list.d/pgdg.list`, w którym należy umieścić wpis `deb http://apt.postgresql.org/pub/repos/apt/ xenial-pgdg main`

Następnie konieczne jest zaimportowanie klucza do repozytorium: `wget -quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -` oraz dokonanie ponownej aktualizacji źródeł programu `apt`.

Po ukończeniu tych kroków możliwe jest zainstalowanie PostgreSQL w wersji 10 za pomocą komendy `sudo apt install postgresql -y`.

Ostatnim krokiem wspólnym dla wszystkich serwerów jest konfiguracja autoryzacji połączeń sieciowych. W tym celu należy dokonać edycji pliku `/etc/postgresql/10/main/postgresql.config` aby zezwolić na nasłuchiwanie połączeń ze wszystkich adresów:

```
listen_addresses = '*'
```

oraz pliku `/etc/postgresql/10/main/pg_hba.conf` określającego sposoby autoryzacji użytkowników. Ze względu na to, że konfigurowane serwery nie posiadają żadnych danych wymagających szczególnej ochrony konfiguracja zezwala na dowolne połączenie bez konieczności autoryzacji:

```
local    all             postgres                                trust

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local    all             all              trust
# IPv4 local connections:
host     all             all              127.0.0.1/32           trust
host     all             all              0.0.0.0/0               trust
```

```
# IPv6 local connections:
host      all             all             ::1/128         trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local     replication      all             trust
host      replication      all             127.0.0.1/32    trust
host      replication      all             ::1/128         trust
```

W serwerach slave systemów bucardo i replikacji logicznej konieczna jest modyfikacja jednej z tabel w celu umożliwienia pomiaru opóźnienia replikacji. Należy w tym celu wywołać następujące zapytanie:

```
ALTER TABLE a ADD replica_timestamp TIMESTAMP DEFAULT current_timestamp;
```

### 4.2.2. Konfiguracja Bucardo

Konfigurację replikacji bucardo [26], [20] rozpoczyna się od instalacji na serwerze master komendą `sudo apt install bucardo`. Przed użyciem programu zalecane jest stworzenie w bazie danych użytkownika bucardo z odpowiednim poziomem uprawnień. Bazę danych uruchamia się poprzez komendę `psql`. Do stworzenia użytkownika używane są 2 zapytania:

```
CREATE USER bucardo SUPERUSER PASSWORD 'a' LOGIN;
GRANT ALL ON SCHEMA public TO bucardo.
```

Kolejnym krokiem jest inicjalizacja programu bucardo. Po wywołaniu komendy `sudo bucardo install` uruchomiony zostaje konfigurator, w którym określić można użytkownika, bazę danych, adres i port oraz ścieżkę plików dla programu. Po zatwierdzeniu wyboru utworzona zostaje baza danych o nazwie bucardo przeznaczona na konfigurację replikacji.

Proces konfiguracji replikacji wymaga określenia połączeń do baz danych, utworzenia grupy replikacji, zdefiniowania replikowanych tabel i utworzenia triggera replikującego. Pełny skrypt konfiguracji przedstawia poniższy listing.

Listing 4.3: Konfiguracja bucardo

```
sudo bucardo add db master db=postgres user=bucardo pass=a host=localhost
sudo bucardo add db slave db=postgres user=bucardo pass=a host=52.56.122.85
sudo bucardo add all tables db=master relgroup=rel
sudo bucardo add dbgroup dbgroup master:source slave:target
sudo bucardo add sync bsync dbgroup=dbgroup relgroup=rel autokick=1
sudo bucardo validate all
```

```
sudo bucardo start
```

### 4.2.3. Konfiguracja Pgpool-II

Instalacja pgpool-II [27] odbywa się poprzez wywołanie komendy `sudo apt install pgpool2`. Po zakończeniu instalacji konieczna jest edycja pliku konfiguracyjnego `pgpool.conf` znajdującego się w `/etc/pgpool2/`. Domyślna konfiguracja dostarczana z programem jest wystarczająco dobra na potrzeby tego projektu, wymagane jest jednak zezwolenie na połączenie z serwerem. W tym celu należy edytować linię `listen_addresses` i ustawić wartość parametru na `'*'`. Według ustalonego projektu wymagana jest też zmiana portu z 9999 na 5432. Następnie należy podać dane do połączenia z bazami danych. Kolejne serwery definiować można przez dopisanie numeru do nazwy parametru.

Listing 4.4: Konfiguracja pgpool-II

```
listen_addresses = '*'

                                # Host name or IP address to listen on:
                                # '*' for all, '' for no TCP/IP conns
                                # (change requires restart)

port = 5432

                                # Port number
                                # (change requires restart)

# - Backend Connection Settings -

backend_hostname0 = '35.176.165.93'
                                # Host name or IP address

backend_port0 = 5432

                                # Port number for backend 0

backend_weight0 = 1

                                # Weight for backend 0

backend_data_directory0 = '/var/lib/postgresql/10/main'
                                # Data directory for backend 0

backend_flag0 = 'ALLOW_TO_FAILOVER'
                                # Controls various backend behavior
                                # ALLOW_TO_FAILOVER or DISALLOW_TO_FAILOVER

backend_hostname1 = '35.178.97.25'

backend_port1 = 5432
```



```
backend_weight1 = 1
backend_data_directory1 = '/var/lib/postgresql/10/main'
backend_flag1 = 'ALLOW_TO_FAILOVER'
```

Po zapisaniu zmian w pliku konieczne jest ponowne uruchomienie aplikacji.

#### 4.2.4. Konfiguracja replikacji logicznej

Replikacja logiczna jest mechanizmem wbudowanym w PostgreSQL i nie wymaga instalacji dodatkowych aplikacji. Konfiguracja została wykonana w oparciu o poradnik znaleziony w internecie [21].

**Serwer master.** Konfigurację rozpoczyna się od edycji pliku konfiguracyjnego `/etc/postgresql/10/main/postgresql.config`, w którym należy ustawić wartość parametru `wal_level` na `logical`. Następnie po zalogowaniu do bazy danych należy utworzyć publikację zawierającą tabele podlegające replikacji, w tym przypadku są to wszystkie tabele bazy *postgres*.

```
CREATE PUBLICATION pub FOR ALL TABLES;
```

**Serwer slave.** Konfiguracja replikacji po stronie serwera slave sprowadza się do utworzenia subskrypcji nasłuchującej zmian publikowanych z serwera master. W zapytaniu definiującym subskrypcję należy określić adres bazy danych, dane do logowania oraz nazwę publikacji:

```
CREATE SUBSCRIPTION sub CONNECTION 'dbname=postgres user=postgres
host=35.177.38.10 password=postgres' PUBLICATION pub;
```

Po zakończeniu konfiguracji warto zrestartować serwer bazy danych, aby upewnić się, że wprowadzone zmiany zostaną zastosowane. Można to zrobić komendą `sudo systemctl restart postgresql`.

#### 4.2.5. Konfiguracja replikacji strumieniowej

W celu konfiguracji replikacji strumieniowej [22] należy edytować plik `/etc/postgresql/10/main/postgresql.config` na serwerze master i ustawić w nim parametry przedstawione na poniższym listingu.

Listing 4.5: Konfiguracja replikacji strumieniowej na serwerze master

```
listen_addresses = '*'
```

```
wal_level = hot_standby  
max_wal_senders = 10  
wal_keep_segments = 32
```

```
archive_mode = on  
archive_command = 'cp %p /home/ubuntu/archive/%f'
```

Następnie w tym samym pliku na serwerze slave należy ustawić parametr `hot_standby=on` oraz utworzyć plik `recovery.conf` w ścieżce danych bazy, domyślnie `/var/lib/postgresql/10/main`. Zawartość pliku przedstawiono poniżej:

```
standby_mode          = 'on'  
primary_conninfo      = 'host=35.177.141.137 port=5432 user=postgres  
                        password=postgres'
```

## Rozdział 5

# Testowanie i ocena wydajności systemu

### 5.1. Przygotowanie środowiska testowego

W niniejszym punkcie opisano sposób przygotowania środowiska umożliwiającego testowanie i ocenę wydajności mechanizmów replikacji.

#### 5.1.1. Aplikacja serwerowa

W celu przygotowania środowiska do testów konieczne jest wygenerowanie schematu bazy danych. W przypadku systemów bucardo oraz replikacji logicznej krok ten należy wykonać przed konfiguracją replikacji.

Poprzez zmianę parametru adresu do bazy danych w pliku konfiguracyjnym należy wygenerować schemat na każdym z serwerów. W celu aktualizacji schematu przez Hibernate konieczne jest ustawienie parametru `hibernate.hbm2ddl.auto` w pliku `application.properties` na wartość `update`. Zapis z prawidłowego uruchomienia aplikacji wygląda następująco:

```

: HHH000412: Hibernate Core (5.0.12.Final)
: HHH000206: hibernate.properties not found
: HHH000021: Bytecode provider name : javassist
: HCANN000001: Hibernate Commons Annotations (5.0.1.Final)
: HHH000400: Using dialect: org.hibernate.dialect.PostgreSQL94Dialect
: HHH000424: Disabling contextual LOB creation as createClob() method threw error : java.lang.reflect.InvocationTargetException
: HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@2d6c22
: HHH000228: Running hbm2ddl schema update
: HHH000262: Table not found: A
: HHH000262: Table not found: A
: HHH000262: Table not found: B
: HHH000262: Table not found: B
: HHH000262: Table not found: C
: HHH000262: Table not found: C
: HHH000262: Table not found: D
: HHH000262: Table not found: D
: HHH000262: Table not found: E
: HHH000262: Table not found: E
: HHH000262: Table not found: F
: HHH000262: Table not found: F
: HHH000262: Table not found: G
: HHH000262: Table not found: G
: HHH000262: Table not found: H
: HHH000262: Table not found: H
: HHH000262: Table not found: I
: HHH000262: Table not found: I
: HHH000262: Table not found: J
: HHH000262: Table not found: J
: Initialized JPA EntityManagerFactory for persistence unit 'default'

```

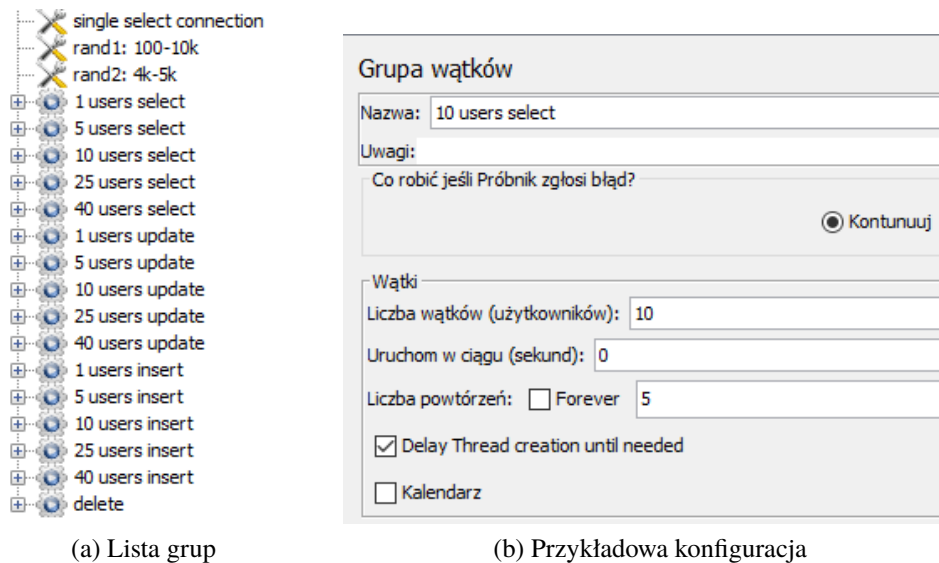
Rys. 5.1: Tworzenie tabel przy pomocy frameworka Hibernate [4]

### 5.1.2. JMeter

Przygotowanie planu testów w aplikacji JMeter rozpoczyna się od stworzenia konfiguracji połączenia z bazą danych. Oprócz danych niezbędnych do połączenia konieczne jest zdefiniowanie nazwy używanej w późniejszej konfiguracji do powiązania testów z konfiguracją, w tym wypadku zdefiniowaną nazwą jest '1'.

Rys. 5.2: JMeter - połączenie z bazą danych [1]

Kolejnym krokiem jest konfiguracja symulowanej liczby użytkowników, którzy wykonują zapytania w tym samym czasie. Zgodnie ze specyfikacją testów zdefiniowane zostały grupy zawierające 1, 5, 10, 25 i 40 użytkowników dla każdego z czterech badanych zapytań.

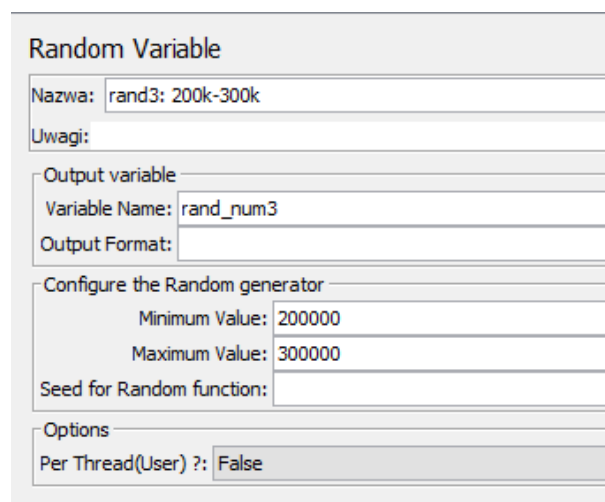


Rys. 5.3: JMeter - konfiguracja liczby aktywnych użytkowników [1]

Kolejnym krokiem jest utworzenie zmiennych losowych wykorzystywanych w zapytaniach. W tym przypadku utworzone zostały trzy zmienne losowe, których zakresy pokazano w tabeli 5.1.

Tab. 5.1: Zakresy wykorzystanych zmiennych losowych

nazwa zmiennej	wartość minimalna	wartość maksymalna
rand_num	100	10 000
rand_num2	4 000	5 000
rand_num3	200 000	300 000



Rys. 5.4: JMeter - definicja zmiennych losowych [1]

Ostatnim elementem konfiguracji planu testów jest utworzenie zapytań JDBC zawierających skrypty SQL, które mają zostać wykonane na bazie danych. W konfiguracji konieczne jest podanie zdefiniowanej wcześniej nazwy połączenia. Przykładowe zapytania przedstawiono na poniższych rysunkach.

The screenshot shows the 'JDBC Request' configuration in JMeter. The 'Nazwa' (Name) field is set to 'Single select A'. The 'Uwagi' (Comments) field is empty. The 'Variable Name Bound to Pool' field is empty. The 'Variable Name of Pool declared in JDBC Connection Configuration' field is set to '1'. The 'SQL Query' section shows 'Query Type' as 'Select Statement' and the query text: '1 select \* from a WHERE id = 456;'. The query text is highlighted in yellow.

(a) Single select

The screenshot shows the 'JDBC Request' configuration in JMeter. The 'Nazwa' (Name) field is set to 'update A'. The 'Uwagi' (Comments) field is empty. The 'Variable Name Bound to Pool' field is empty. The 'Variable Name of Pool declared in JDBC Connection Configuration' field is set to '1'. The 'SQL Query' section shows 'Query Type' as 'Update Statement' and the query text: '1 update a set b = \${rand\_num} where id = \${rand\_num2};'. The query text is highlighted in yellow.

(b) Update

The screenshot shows the 'JDBC Request' configuration in JMeter. The 'Nazwa' (Name) field is set to 'insert c'. The 'Uwagi' (Comments) field is empty. The 'Variable Name Bound to Pool' field is empty. The 'Variable Name of Pool declared in JDBC Connection Configuration' field is set to '1'. The 'SQL Query' section shows 'Query Type' as 'Update Statement' and the query text: '1 insert into c values (\${rand\_num3},\${rand\_num2},1000,200,'jmeter','a','a');'. The query text is highlighted in yellow.

(c) Insert

Rys. 5.5: JMeter - definicje zapytań testowych [1]

## 5.2. Wyniki pomiarów czasu wykonania zapytań

Zaprojektowany plan testowy został wykonany dla każdego z badanych mechanizmów replikacji. W przypadku aplikacji pgpool zapytania kierowane były na serwer pgpool-core, a w przypadku pozostałych metod serwery master. Zbadany został wpływ rozmiaru danych oraz liczby aktywnych użytkowników na czas odpowiedzi serwera. Czasy zostały uśrednione dla około 50 prób na zapytanie, a wyniki zostały opisane poniżej.

W wynikach przedstawiony został czas średni odpowiedzi na zapytanie oraz jego współczynnik zmienności. Współczynnik zmienności to miara zróżnicowania rozkładu cechy. Najczęściej definiowany jest wzorem [28]:

$$V = \frac{s}{\bar{x}}, \quad \bar{x} \neq 0$$

gdzie

$s$  to odchylenie standardowe z próby,

$\bar{x}$  to średnia arytmetyczna z próby.

Mierzy on, jaką część średniej stanowi odchylenie standardowe. W odróżnieniu od odchylenia standardowego, jest on miarą niemianowaną oraz względną zależną od średniej arytmetycznej, dzięki czemu umożliwia porównywanie stopnia zróżnicowania różnych cech statystycznych. Takie porównania nie są możliwe dla miar bezwzględnych. Dla celów praktycznych można podawać wartość współczynnika w procentach po przemnożeniu przez 100. Przyjmuje się, że cechy o wartości zróżnicowania poniżej 10% są nieistotne statystycznie [29].

Wykresy zawarte w kolejnych punktach przedstawiają zależność czasu odpowiedzi od rodzaju replikacji oraz liczby aktywnych użytkowników. Serie danych przedstawiają poszczególne tabele o różnym rozmiarze, zgodnie z licznosciami podanymi w tabeli 3.2. Wyniki uzyskane w badaniach dla tabeli A zostały pominięte, gdyż zawierały znaczne błędy pomiaru wynikające z ustanawiania połączeń przez aplikację testującą.

### 5.2.1. Odczyt wszystkich rekordów

#### Średni czas odpowiedzi

Średnie czasy odpowiedzi dla testu odczytu wszystkich rekordów w tabeli przedstawiono na wykresie 5.6. Badania wykazały, że niezależnie od wybranej metody replikacji, liczba aktywnych użytkowników ma znaczący wpływ na czasy odpowiedzi systemu.

Najlepszym systemem okazał się pgpool, który dla 40 użytkowników osiąga wyniki o około 25% lepsze od pozostałych systemów. Analizując poszczególne serie danych (na wykresie słupki o jednakowym kolorze) można zauważyć, że szybkość wzrostu w zależności od liczby użytkowników zmienia się w sposób nieliniowy. Dla tabel o dużym rozmiarze, np. H, I, J, zauważalne są znacznie większe różnice pomiędzy kolejnymi wartościami niż dla tabel małych.

Skupiając uwagę na poszczególnych kolumnach wykresu w obrębie jednej liczby użytkowników można dostrzec, że rozmiar danych powoduje liniowy wzrost średniego czasu odpowiedzi. Warto również zauważyć, że wyniki bucardo dla 1 oraz 5 użytkowników są widocznie gorsze od wyników uzyskanych w przypadku pozostałych mechanizmów.

#### Współczynnik zmienności

Na wykresie 5.7 pokazano wyniki obliczeń współczynnika zmienności dla testu odczytu wszystkich rekordów w tabeli. Jak widać, największe zróżnicowanie danych osiągające od około 50%

do ponad 80% uzyskano dla mechanizmu pgpool przy obciążeniu jednego użytkownika. Przy większej liczbie użytkowników wyniki tego mechanizmu są porównywalne do mechanizmów replikacji logicznej oraz strumieniowej i zawierają się w przedziale od 10 do 30 procent. W przypadku systemu bucardo liczba użytkowników ma znaczący wpływ na zróżnicowanie czasu odpowiedzi. Wartości uzyskane dla 1, 5 oraz 10 użytkowników są dwukrotnie wyższe od pozostałych.

Na podstawie wykresu nie da się jednoznacznie określić wpływu rozmiaru danych na zróżnicowanie czasu odpowiedzi.

### 5.2.2. Odczyt jednego rekordu

#### Średni czas odpowiedzi

Wyniki badań czasu odczytu pojedynczego rekordu przedstawiono na wykresie 5.8. Bez względu na wybraną metodę replikacji danych, czasy uzyskane przy obciążeniu jednego użytkownika są wyraźnie niższe od pozostałych.

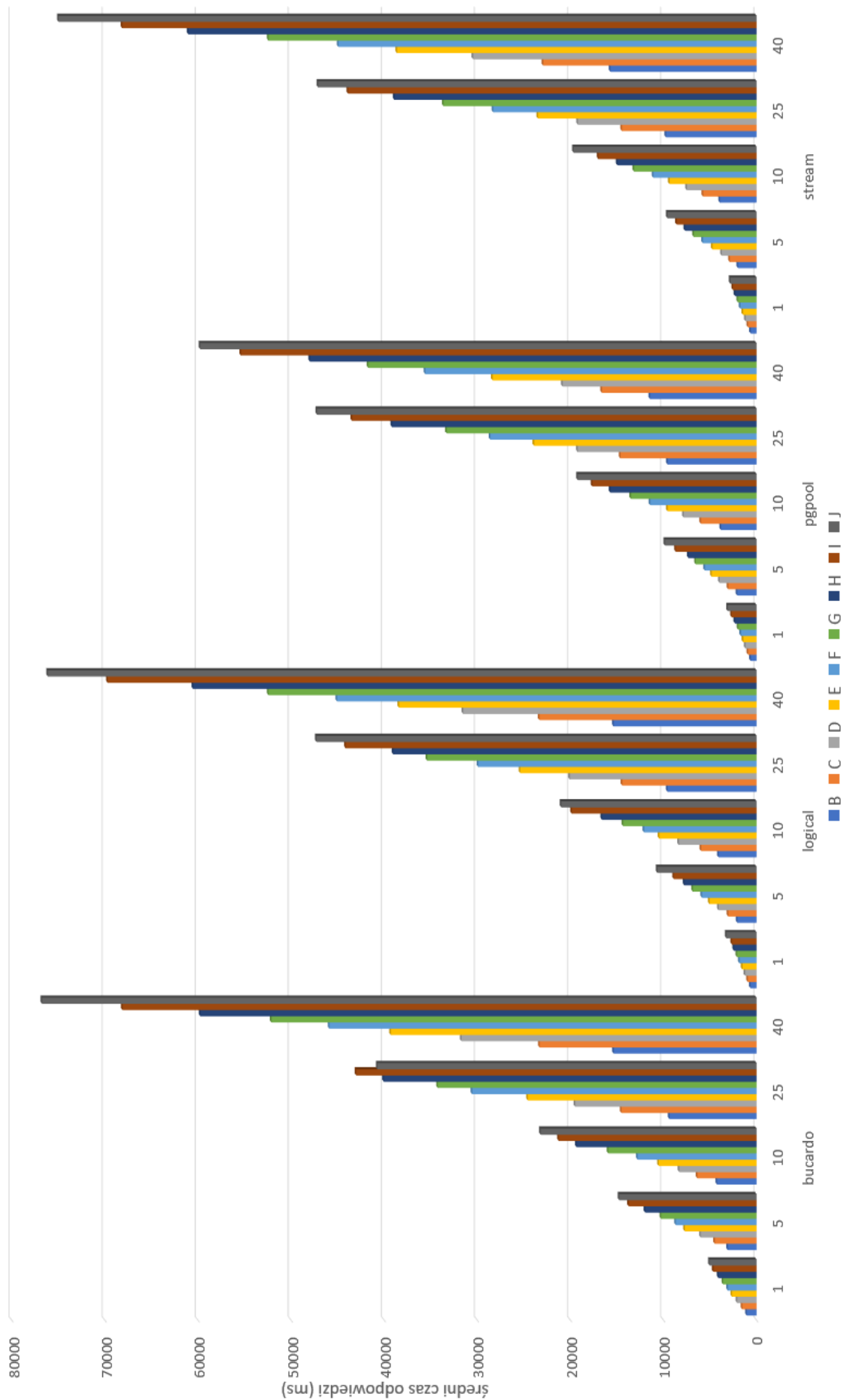
Systemy replikacji strumieniowej i logicznej uzyskały w tym teście nieco lepsze wyniki od systemów bucardo i pgpool. Czasy odpowiedzi w replikacji strumieniowej są najbardziej odporne na zmieniające się obciążenie i uzyskują zbliżone wyniki dla wszystkich badanych liczb aktywnych użytkowników, a także zachowują największą stabilność przy zmieniającym się rozmiarze danych.

W przypadku systemu pgpool uzyskane wyniki są zbliżone do tych z replikacji strumieniowej pod względem wielkości, jednak występują w nich znacznie większe wahania przy testach dla tabel o różnych rozmiarach.

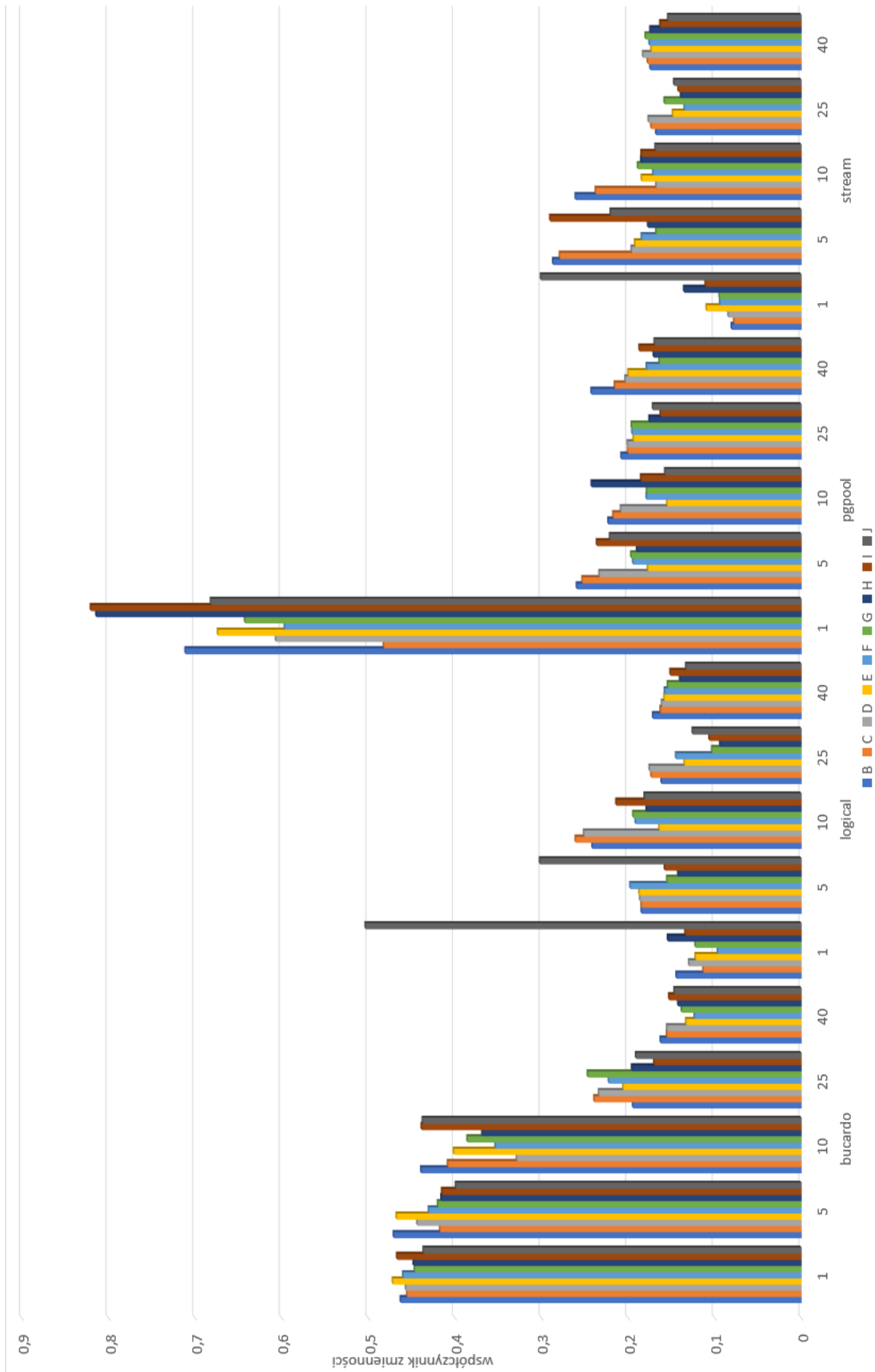
Najgorzej z testem poradził sobie system replikacji logicznej, którego wyniki dla dużych obciążeń są ponad dwukrotnie większe od uzyskanych przez inne systemy. Można również zauważyć bardzo duży rozrzut w zależności od badanej tabeli.

Najbardziej zaskakujące są wyniki systemu bucardo, w którego przypadku czasy odpowiedzi osiągają maksymalne wartości dla 5 użytkowników i spadają przy większych obciążeniach do poziomu porównywalnego z replikacją pgpool i strumieniową.





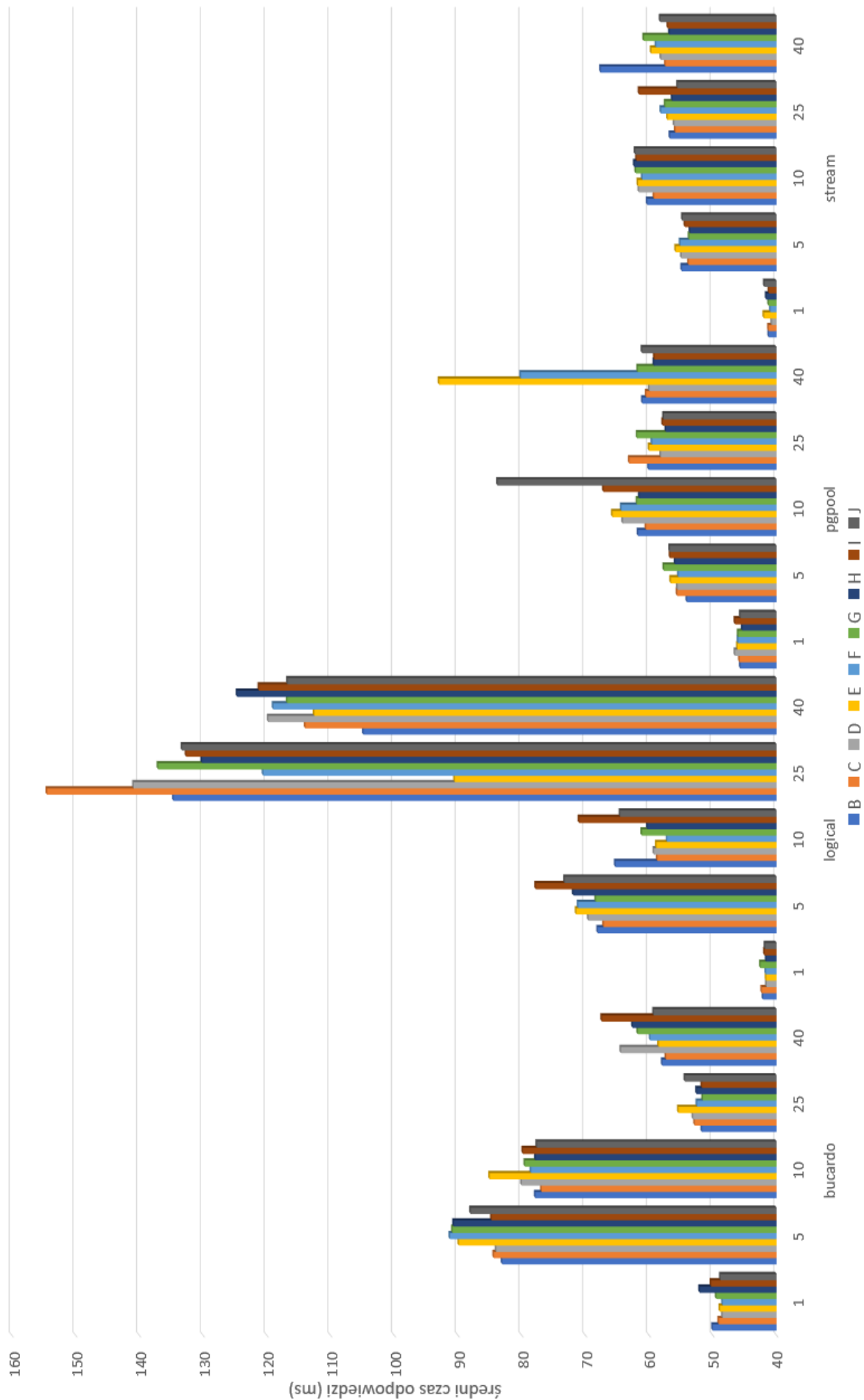
Rys. 5.6: Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu wszystkich danych



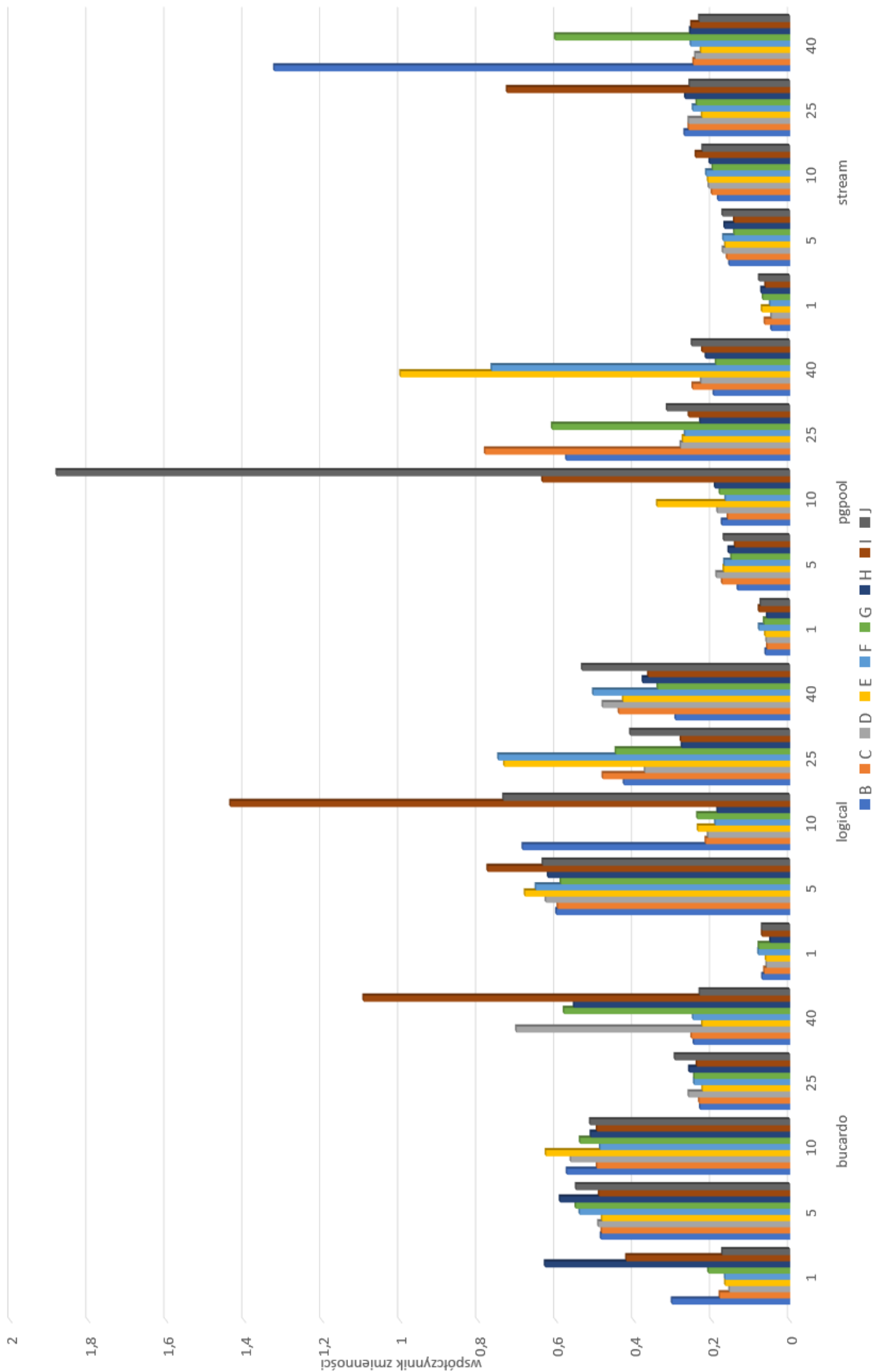
Rys. 5.7: Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu wszystkich danych

### **Współczynnik zmienności**

Zależność współczynnika zmienności od badanych parametrów pokazano na wykresie 5.9. Po porównaniu tego wykresu z wartościami dla poprzedniego testu widać, że zróżnicowanie wyników dla odczytu jednego rekordu jest znacznie większe. System replikacji strumieniowej uzyskuje najbardziej jednolite wyniki, jednak nie jest pozbawiony obserwacji znacznie odbiegających od średniej. Zróżnicowanie replikacji pgpool dla większości badanych przypadków dorównuje replikacji strumieniowej, jednak zawiera więcej odchyleń. Najbardziej zróżnicowane wyniki uzyskano z badań systemów bucardo i replikacji logicznej. Ogólny poziom wartości współczynnika zmienności jest w przybliżeniu dwukrotnie wyższy od pozostałych systemów. W każdym z analizowanych systemów występują znaczące wahania poziomu zróżnicowania czasu odpowiedzi.



Rys. 5.8: Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu jednego rekordu



Rys. 5.9: Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu odczytu jednego rekordu

### 5.2.3. Zapis rekordu

#### Średni czas odpowiedzi

Pierwszą obserwacją, jaką można odczytać z wykresu ilustrującego wyniki pomiarów średniego czasu odpowiedzi dla testu zapisu (rys. 5.10) jest to, że system replikacji logicznej jest zdecydowanie wolniejszy od pozostałych. Uzyskiwane przez niego wyniki oscylują od 120 do około 145ms, podczas gdy inne systemy w większości przypadków uzyskują wyniki poniżej 50ms. Występują też znaczne różnice w czasach dla poszczególnych tabel w obrębie jednej liczby połączeń oraz dla różnych liczb połączeń.

Można zauważyć, że rozmiar danych ma znikomy wpływ na czas odpowiedzi. Nieco bardziej istotna jest liczba użytkowników, jednak dla badanych poziomów obciążenia różnice są niewielkie. Widoczny jest wzrost w przypadku replikacji strumieniowej i bucardo dla 25 oraz 40 aktywnych połączeń.

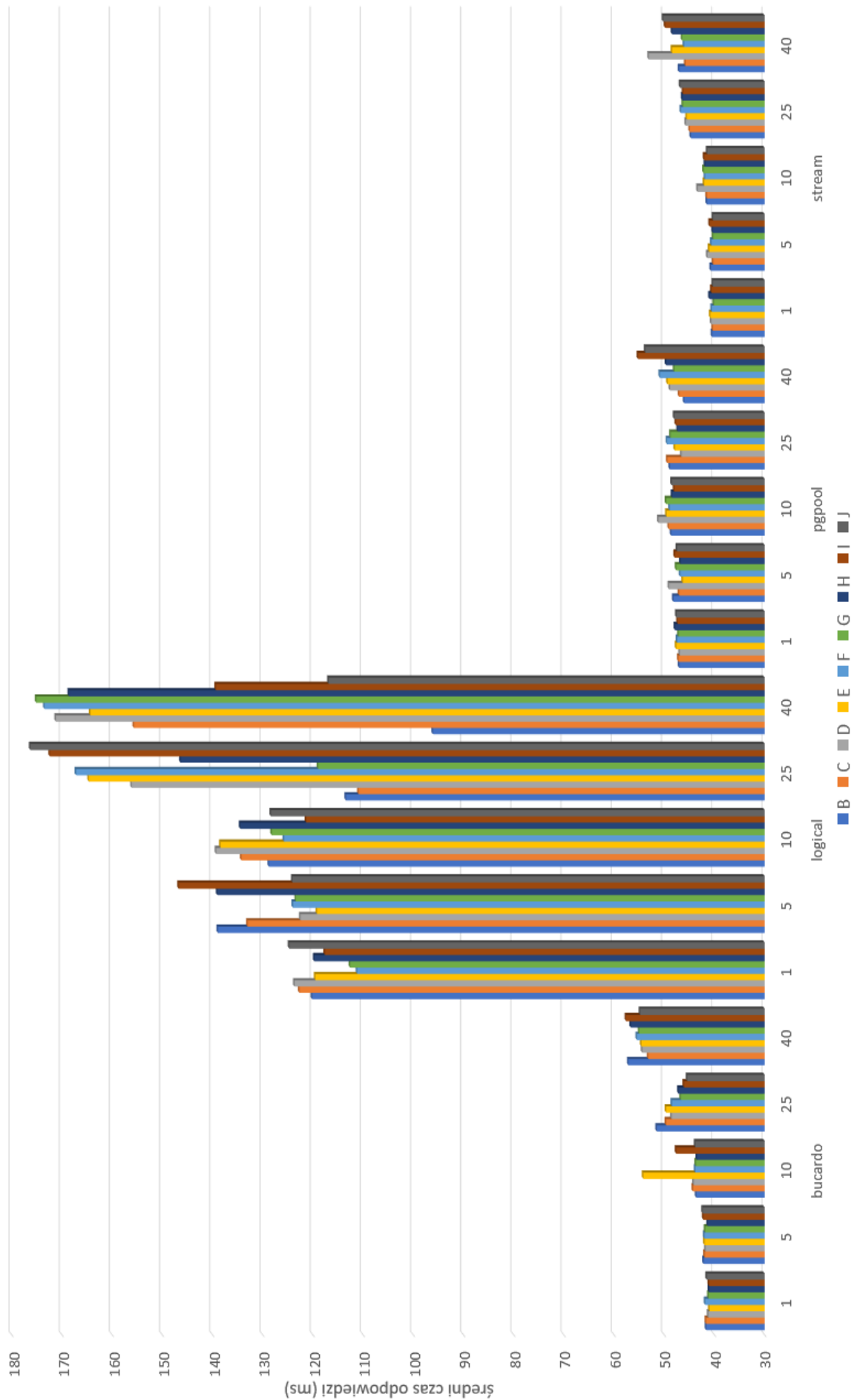
#### Współczynnik zmienności

Replikacja logiczna, która okazała się osiągać najgorsze czasy zapisu rekordu jest też najbardziej zróżnicowana. Wartości współczynnika zmienności zawierają się w przedziale od 10 do 60 procent. W przypadku pozostałych badanych systemów w zdecydowanej większości uzyskanych wyników wartość współczynnika wynosi poniżej 10%. W wynikach systemu bucardo występują dwa znaczne odchylenia od ogólnego poziomu zróżnicowania, jednak trudno określić, czy są one zależne od któregośkolwiek z badanych parametrów.

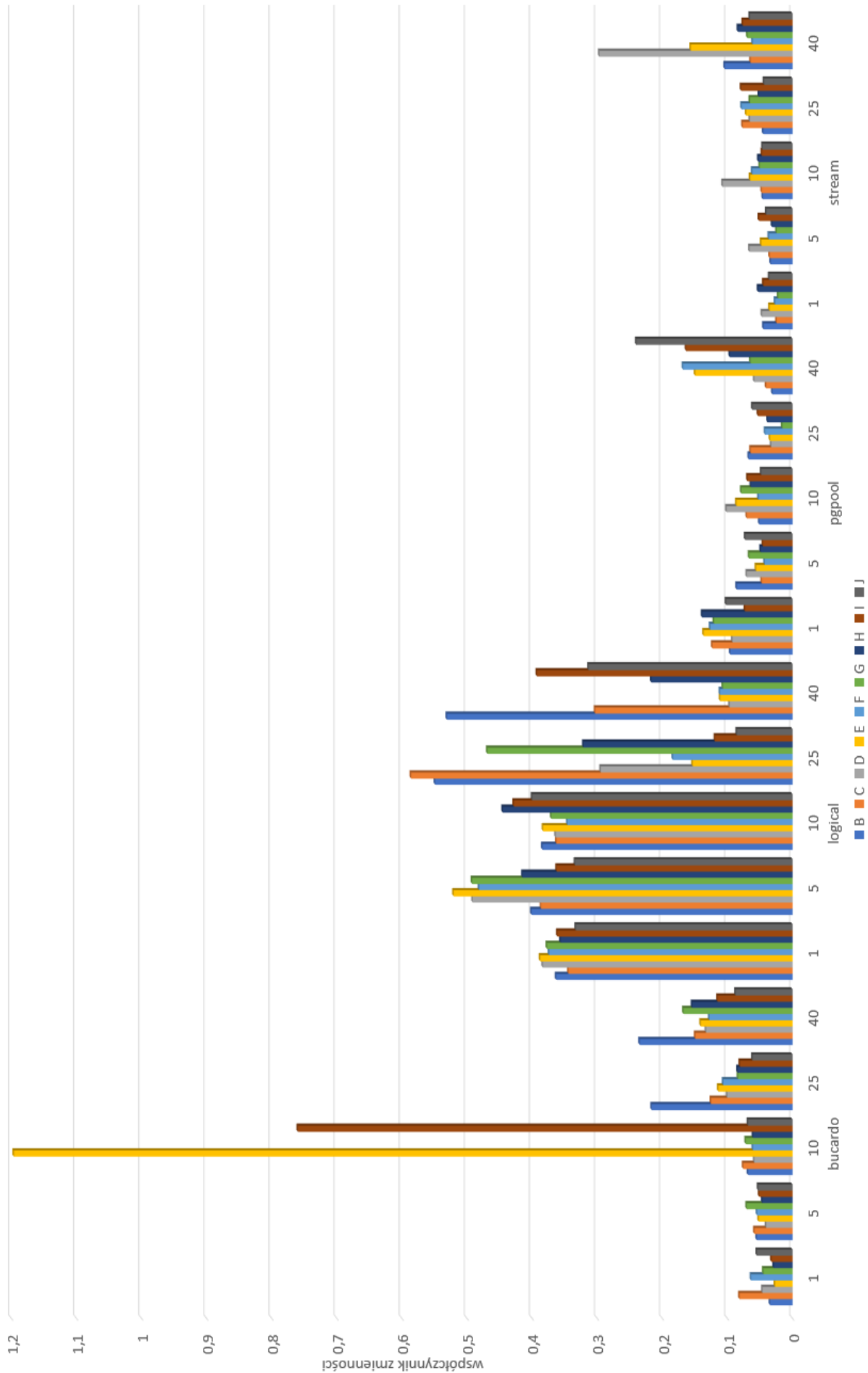
### 5.2.4. Aktualizacja rekordu

#### Średni czas odpowiedzi

Wyniki tego testu przedstawione zostały na rysunku 5.12. Podobnie jak w przypadku testu zapisu rekordu, czasy odpowiedzi replikacji logicznej są wyższe od pozostałych systemów. Występują różnice dla różnych rozmiarów tabel oraz liczb użytkowników, jednak nie jest możliwe określenie zależności. System bucardo oraz replikacja strumieniowa uzyskały najlepsze czasy dla małego obciążenia. W przypadku tych systemów widać również wpływ liczby użytkowników na czas. Dla 25 i 40 aktywnych połączeń zaobserwować można niewielki wzrost. Wpływ



Rys. 5.10: Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu zapisu



Rys. 5.11: Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu zapisu



rozmiaru danych na czas aktualizacji jest znikomy, jednak coraz większy wraz z rosnącym obciążeniem. Dla replikacji systemem pgpool wyniki są bardzo zbliżone dla wszystkich poziomów obciążenia z wyjątkiem ostatniego, gdzie zaobserwować można niewielki wzrost.

### Współczynnik zmienności

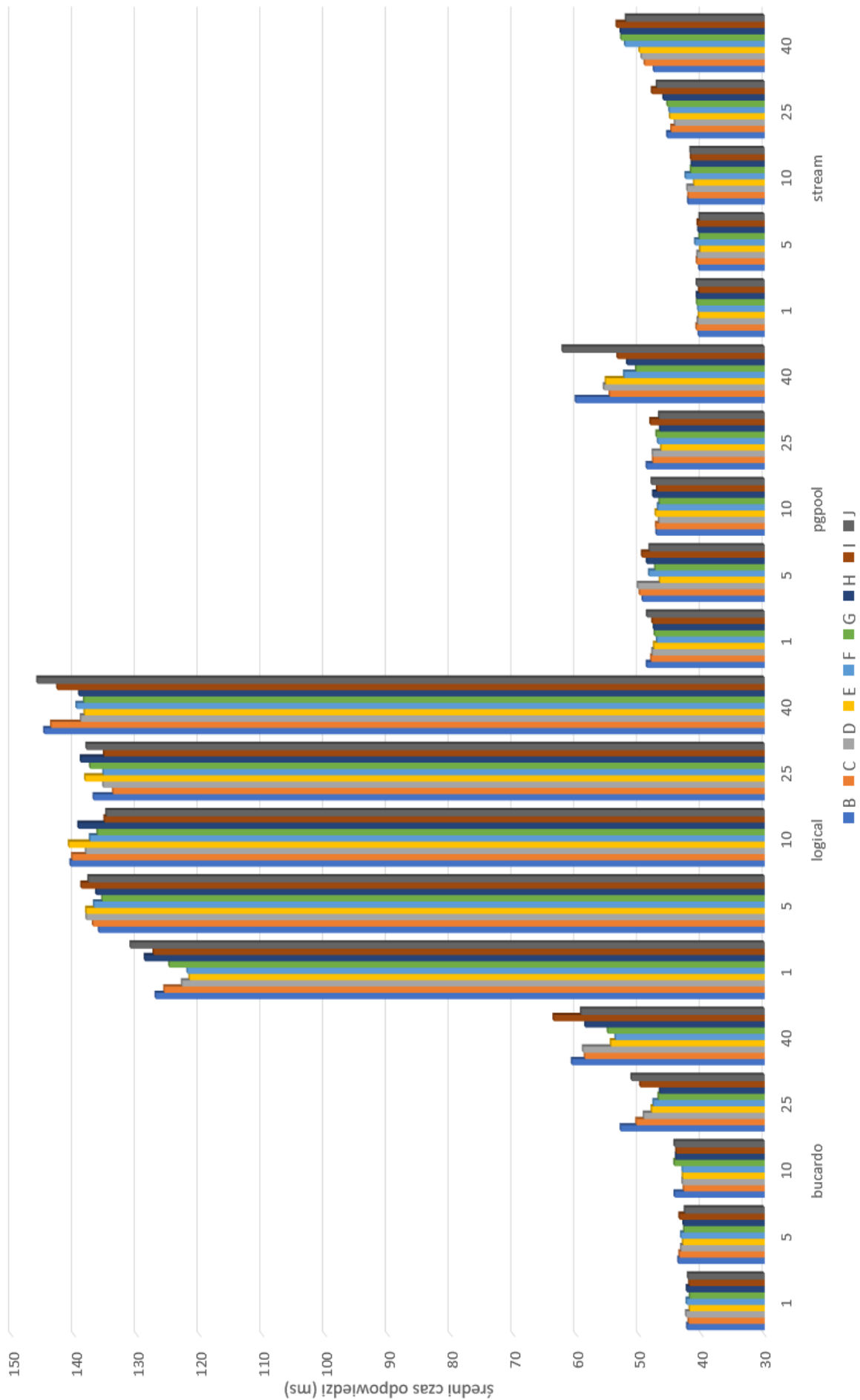
Współczynniki zmienności dla testu aktualizacji zaprezentowane zostały na wykresie 5.13. Łątwo zauważyć, że test ten jest najmniej zróżnicowany ze wszystkich. Wartość współczynnika wśród wszystkich kombinacji badanych parametrów osiąga tu maksymalnie 27%. Znacznie bardziej niż w przypadku zapisu widoczny jest wpływ liczby użytkowników na zróżnicowanie wyników. Replikacja bucardo oraz strumieniowa wykazują stały wzrost zróżnicowania wraz ze zwiększającym się obciążeniem. Replikacja logiczna, która wykazywała się najgorszymi czasami aktualizacji danych jest też najmniej zróżnicowana dla obciążenia powyżej jednego użytkownika. Przy jednym połączeniu współczynniki osiągają z kolei jedne z wyższych wartości na tle badanych systemów.

#### 5.2.5. Opóźnienie replikacji danych

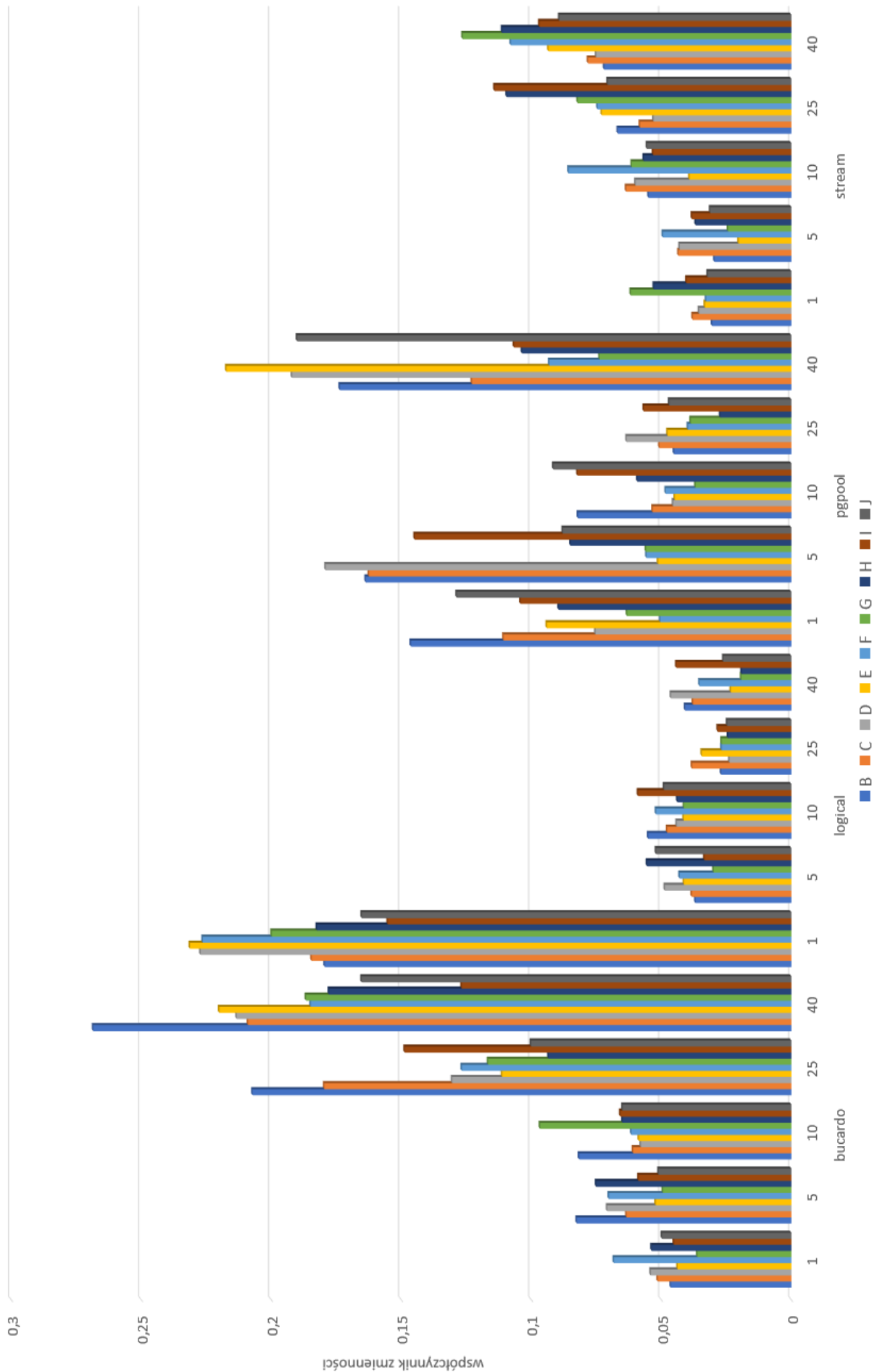
Przed przeprowadzeniem testu zegary na serwerach zostały zsynchronizowane. Wyniki badań opóźnienia replikacji przedstawiono w tabeli 5.2 oraz na wykresie 5.14. Test ten przeprowadzony został dla systemów bucardo oraz replikacji logicznej. Jak można zauważyć, wyniki uzyskiwane przez replikację logiczną są zdecydowanie lepsze. Opóźnienie wyniosło od 1 do 4 milisekund, podczas gdy replikacja bucardo potrzebowała ponad 200ms. Liczba wierszy zapisywanych w jednym zapytaniu powoduje liniowy wzrost czasu opóźnienia replikacji logicznej, czego nie można powiedzieć o systemie bucardo, w którego przypadku nie jest widoczna żadna zależność od liczby rekordów.

Tab. 5.2: Wyniki badań opóźnienia replikacji systemów bucardo i logicznego

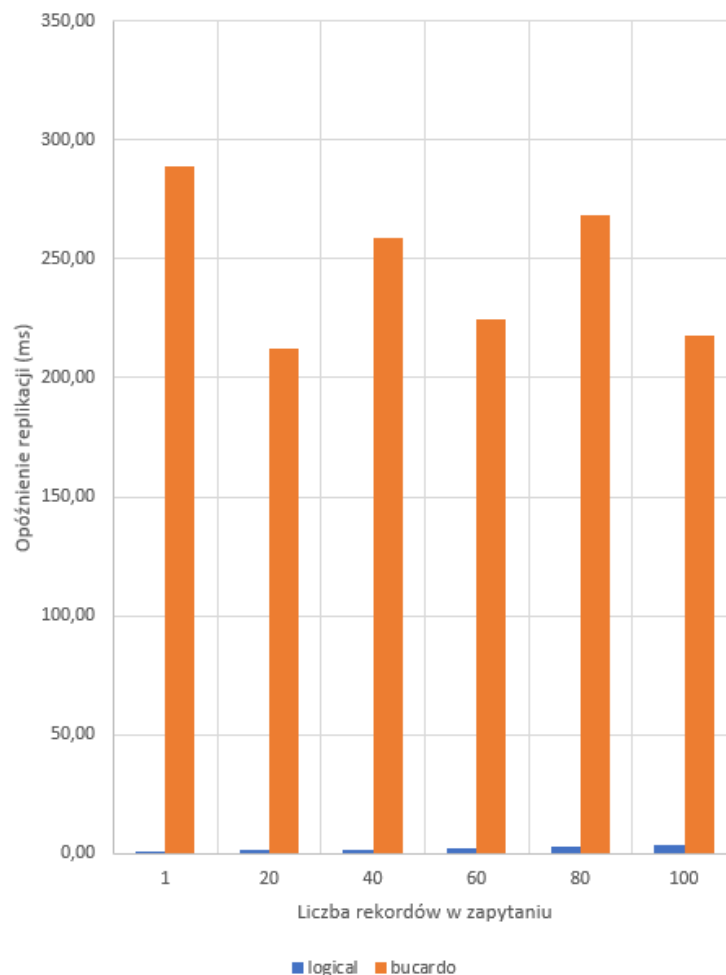
liczba rekordów w zapytaniu	logical	bucardo
1	0,98	289,05
20	1,36	212,38
40	1,83	259,05
60	2,42	224,36
80	2,93	268,48
100	3,65	217,96



Rys. 5.12: Wykres zależności średniego czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu aktualizacji rekordu



Rys. 5.13: Wykres zależności współczynnika zmienności czasu odpowiedzi od metody replikacji i liczby użytkowników dla testu aktualizacji rekordu



Rys. 5.14: Wykres zależności opóźnienia replikacji danych od liczby rekordów zapisywanych w jednym zapytaniu

## 5.3. Wnioski z testów

### Odczyt danych

Zaobserwowany został wzrost czasu odpowiedzi przy zwiększeniu rozmiaru danych, a także przy wzrastającej liczbie użytkowników. Uzyskane czasy odczytu wszystkich rekordów z tabeli dla systemu pgpool-II były znacznie niższe od pozostałych systemów, co jest zgodne z oczekiwaniami. System pgpool-II zawiera mechanizm równoważenia obciążeń, który rozkłada zapytania wysyłane do głównego serwera pomiędzy dostępne bazy danych. Wpływ działania tej funkcjonalności był najbardziej widoczny przy największym badanym obciążeniu, tj. 40 aktywnych połączeń. Warto jednak wspomnieć, że dla jednego użytkownika system ten wprowadził dodatkowe opóźnienia, które objawiały się zwiększonym zróżnicowaniem wyników.

W pozostałych przypadkach uzyskane wyniki były prawie że identyczne. Można zatem stwierdzić, że w przypadku replikacji asynchronicznej wybrana metoda nie wpływa na czas odpowiedzi systemu przy pobieraniu dużej liczby danych. Wpływ liczby użytkowników na czas odpowiedzi może zostać zminimalizowany poprzez zwiększenie wydajności serwerów bazodanowych.

Badania czasu odczytu jednego rekordu pokazały większe różnice pomiędzy metodami replikacji, widoczne w szczególności dla replikacji logicznej. Znikł jednak wpływ liczby rekordów oraz w większości liczby połączeń, co jest stosunkowo przewidywalnym rezultatem. Zapytania zwracające jeden rekord wykonywane są znacznie szybciej, więc sytuacje, w których użytkownicy czekają przez niewydajność systemu są rzadsze.

### **Zapis danych**

W badaniach czasu zapisu danych, zarówno w przypadku zapytań INSERT jak i UPDATE najwolniejszą okazała się replikacja logiczna. Mechanizm ten w trakcie wykonywania zapisu do bazy danych zmuszony jest na przeprowadzenie dodatkowych operacji związanych z utrzymywaniem aktualności publikacji zawierających dziennik zmian. Pomimo zastosowania podobnych mechanizmów w innych systemach, nie został zaobserwowany ich wpływ na czas zapisu. Replikacja pgpool osiągnęła nieznacznie wyższe czasy, co wynika z synchronicznego sposobu replikacji. Zapytania muszą zostać wykonane na wszystkich serwerach i konieczne jest otrzymanie potwierdzenia z każdego z nich przed przesłaniem odpowiedzi do użytkownika.

### **Replikacja**

Opóźnienie replikacji w systemie bucardo okazało się być znacznie wyższe niż w przypadku replikacji logicznej. Niemożliwe było także określenie dla tego systemu zależności pomiędzy kolejnymi badanymi wartościami liczby zapisywanych rekordów. System bucardo nie udostępnia zbyt wiele informacji na temat sposobu wykonywania replikacji, dlatego trudno jest jednoznacznie stwierdzić co może być powodem powstawania tak znacznych opóźnień.

# Rozdział 6

## Podsumowanie

Celem pracy było zaprojektowanie oraz zaimplementowanie systemu pozwalającego na przeprowadzenie badań wydajnościowych na wybranych mechanizmach replikacji systemu bazodanowego PostgreSQL. W ramach realizacji projektu wykonana została aplikacja serwerowa wykorzystująca framework Spring oraz konfiguracja czterech metod replikacji: bucardo, pgpool-II, replikacji logicznej oraz strumieniowej.

Projekt systemu zakładał prowadzenie badań za pośrednictwem tworzonej aplikacji serwerowej, jednak w trakcie wstępnych testów okazało się, że wprowadza ona opóźnienia, które wpłynęłyby na późniejszą analizę wyników. W związku z tym zdecydowano się ograniczyć działanie aplikacji do tworzenia schematu baz danych. Zaprojektowano i zaimplementowano cztery systemy replikacji składające się z dwóch serwerów bazodanowych każdy. Jako serwis hostingowy serwerów użyty został AWS. Serwery pracują pod kontrolą systemu Ubuntu 16.04, a działające na nich bazy danych są w wersji 10.4 systemu PostgreSQL.

Zaprojektowane zostały testy wydajnościowe oparte o operacje `SELECT`, `INSERT` oraz `UPDATE`. Testy przeprowadzone zostały dla zmieniającej się liczby aktywnych użytkowników z ze zbioru 1, 5, 10, 25, 40 oraz dla różnego rozmiaru danych w tabelach, od 20 000 do 100 000 rekordów. Jako program testujący wydajność wykorzystany został Apache JMeter. Z przeprowadzonych badań zgodnie z oczekiwaniami wynikało, że odczyt danych za pośrednictwem mechanizmów systemu pgpool-II jest szybszy od pozostałych systemów. Rozmiar danych oraz liczba aktywnych użytkowników miały wpływ na czas odczytu. W przypadku operacji zapisu, wyraźnie wolniejszy od pozostałych okazał się system replikacji logicznej. Rozmiar danych oraz liczba połączeń z bazą miały znacznie mniejszy wpływ na wyniki niż miało to miejsce dla operacji odczytu.

Przeprowadzone badania pokazują, że dostępne systemy replikacji mogą wpływać na wydajność osiąganą przez serwery bazodanowe. Projektując system konieczne jest przeprowadzenie odpowiednich testów i dobranie systemu, który najlepiej nadaje się do wybranego zastosowania. Systemy, w których konieczne jest częste pobieranie dużej ilości danych, a ich aktualizacje lub rozszerzanie są rzadsze mogą skłaniać się ku systemowi pgpool, podczas gdy do tworzenia kopii zapasowej bez konieczności edycji lepszy może być mechanizm replikacji strumieniowej. Z punktu widzenia konfiguracji jest on jednak bardziej skomplikowany. Najprostszym systemem, którego konfiguracja wymaga zaledwie kilku komend i zapytań SQL jest mechanizm replikacji logicznej, jednak uzyskiwane przez niego czasy zapisu są znacząco gorsze. Ewentualnym celem dalszych prac może być badanie wpływu poszczególnych parametrów konfiguracyjnych każdego z systemów, takich jak dozwolona liczba połączeń czy rozmiar bloków plików archiwizacyjnych na czasy odpowiedzi.

# Literatura

- [1] Apache Software Foundation. JMeter. 2018. <http://jmeter.apache.org/> [dostęp 8 lipca 2018].
- [2] Ryanjz, Tlacuache. Spawner Data Generator. 2018. <https://sourceforge.net/projects/spawner/> [dostęp 8 lipca 2018].
- [3] Amazon Web Services. 2018. <https://aws.amazon.com/> [dostęp 8 lipca 2018].
- [4] Red Hat. Hibernate ORM. In *Idiomatic persistence for Java and relational databases*. <http://hibernate.org/orm/> [dostęp 8 lipca 2018].
- [5] The PostgreSQL Global Development Group. PostgreSQL: . In *The world's most advanced open source database*, 1996-2018. <https://postgresql.org/> [dostęp 8 lipca 2018].
- [6] Hans-Jürgen Schönig. *PostgreSQL Replication. Second Edition*. Packt Publishing Ltd., 2015.
- [7] The PostgreSQL Global Development Group. Documentation, Chapter 25: High Availability, Load Balancing, and Replication, Section 25.1 Comparison of different solutions . 2018. <https://www.postgresql.org/docs/10/static/different-replication-solutions.html> [dostęp 8 lipca 2018].
- [8] Claudius Weinberger. Benchmark: PostgreSQL, MongoDB, Neo4j, OrientDB and ArangoDB. 2015. <https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/> [dostęp 8 lipca 2018].
- [9] Stanford University. Pokec social network. 2012. <https://snap.stanford.edu/data/soc-pokec.html> [dostęp 8 lipca 2018].
- [10] Mirosław Lach. Analiza porównawcza wybranych własności systemów zarządzania bazami danych. 2008. <http://docplayer.pl/3080991-Analiza-porownawcza->



- wybranych-wlasnosci-systemow-zarzadzania-bazami-danych.html [dostęp 8 lipca 2018].
- [11] Anastasia Raspopina, Sveta Smirnova. Millions of Queries per Second: PostgreSQL and MySQL's Peaceful Battle at Today's Demanding Workloads. 2017. <https://www.percona.com/blog/2017/01/06/millions-queries-per-second-postgresql-and-mysql-peaceful-battle-at-modern-demanding-workloads/> [dostęp 8 lipca 2018].
- [12] The PostgreSQL Global Development Group. PostgreSQL Client Applications. Pgbench. 2018. <https://www.postgresql.org/docs/10/static/pgbench.html> [dostęp 8 lipca 2018].
- [13] akopytov. Sysbench: Scriptable database and system performance benchmark. 2018. <https://github.com/akopytov/sysbench> [dostęp 8 lipca 2018].
- [14] End Point Corporation. Bucardo, Asynchronous PostgreSQL Replication System. 2018. <https://bucardo.org/Bucardo/> [dostęp 8 lipca 2018].
- [15] PgPool Global Development Group. Pgpool-II. 2003-2018. [http://www.pgpool.net/mediawiki/index.php/Main\\_Page](http://www.pgpool.net/mediawiki/index.php/Main_Page) [dostęp 8 lipca 2018].
- [16] The PostgreSQL Global Development Group. Documentation, Chapter 31: Logical Replication. 2018. <https://www.postgresql.org/docs/10/static/logical-replication.html> [dostęp 8 lipca 2018].
- [17] The PostgreSQL Global Development Group. Documentation, Chapter 26: Hot Standby. 2018. <https://www.postgresql.org/docs/10/static/hot-standby.html> [dostęp 8 lipca 2018].
- [18] Pivotal Software, Inc. Spring Boot. 2018. <https://spring.io/projects/spring-boot> [dostęp 8 lipca 2018].
- [19] Pivotal Software, Inc. Spring Data JPA - Reference Documentation. 2018. <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/> [dostęp 8 lipca 2018].
- [20] End Point Corporation. Bucardo, pgbench example. 2018. [https://bucardo.org/Bucardo/pgbench\\_example/](https://bucardo.org/Bucardo/pgbench_example/) [dostęp 8 lipca 2018].

- 
- [21] PostgresCourse.com. Logical Replication with PostgreSQL 10. 2017. <https://www.youtube.com/watch?v=HkgYdirSdi0> [dostęp 8 lipca 2018].
- [22] PostgreSQL Wiki. Streaming Replication. 2016. [https://wiki.postgresql.org/wiki/Streaming\\_Replication](https://wiki.postgresql.org/wiki/Streaming_Replication) [dostęp 8 lipca 2018].
- [23] JetBrains. IntelliJ IDEA the Java IDE. 2018. <https://www.jetbrains.com/idea/> [dostęp 8 lipca 2018].
- [24] The PostgreSQL Global Development Group. PostgreSQL JDBC Driver. 1996-2018. <https://jdbc.postgresql.org/> [dostęp 8 lipca 2018].
- [25] Reinier Zwitserloot, Roel Spilker. Project Lombok. 2009-2018. <https://projectlombok.org/> [dostęp 8 lipca 2018].
- [26] Srijan Choudhary. PostgreSQL replication using Bucardo. 2015. <https://www.srijn.net/2015/09/postgresql-replication-using-bucardo/> [dostęp 8 lipca 2018].
- [27] PgPool Global Development Group. Documentation, Chapter 7: Configuration Examples. <http://www.pgpool.net/docs/latest/en/html/example-basic.html> [dostęp 8 lipca 2018].
- [28] Redakcja Wikipedii. Wikipedia - Współczynnik zmienności. 2015. [https://pl.wikipedia.org/wiki/Wsp%C3%B3%C5%82czynnik\\_zmienno%C5%9Bci](https://pl.wikipedia.org/wiki/Wsp%C3%B3%C5%82czynnik_zmienno%C5%9Bci) [dostęp 8 lipca 2018].
- [29] Zygmunt Bobowski. *Wybrane metody statystyki opisowej i wnioskowania statystycznego*. WWSZiP, 2004.

## Dodatek A

# Opis załączonej płyty CD/DVD

Na załączonej płycie w głównym katalogu znajduje się plik `W04_209787_2018_praca_magisterska.pdf` będący elektroniczną wersją tego dokumentu oraz się dwa katalogi:

- **serwer** - zawiera kod aplikacji serwerowej stworzonej w technologii Spring Framework. Struktura plików w katalogu odpowiada projektom tworzonym w środowisku IntelliJ IDEA;
- **jmeter** - zawiera pliki definiujące plan testów użytych podczas wykonywania badań.