

Akademia Górniczo-Hutnicza

im. Stanisława Staszica w Krakowie

Wydział Inżynierii Metali i Informatyki Przemysłowej

Projekt końcowy

Replikacja danych

Zespół w składzie

1. *Sylwia Miedzińska*
2. *Piotr Michno*
3. *Michał Mielec*
4. *Konrad Łata*
5. *Paweł Kozub*

**Kraków 2016**

Spis treści

[**WSTĘP** 3](#_Toc453188810)

[a) Czym jest replikacja? 3](#_Toc453188811)

[b) Powody zwielokrotniania 3](#_Toc453188812)

[c) Podstawowy model replikacji danych. 4](#_Toc453188813)

[**Tabelaryczne zestawienie algorytmów** 5](#_Toc453188814)

[**Wybór algorytmu – sposób oceny** 9](#_Toc453188815)

[**Tabelaryczne zestawienie technologii rozproszonych** 10](#_Toc453188816)

[**Implementacja** 13](#_Toc453188817)

[**Analiza porównawcza implementacji (wykresy) czas/trudność** 15](#_Toc453188818)

[a) nakładu (czasu) waszej pracy wymaganego do zaimplementowania 15](#_Toc453188819)

[b) porównania szybkości działania tego samego algorytmu w różnych technologiach 16](#_Toc453188820)

[**Bibliografia** 19](#_Toc453188821)

[**Spis tabel** 20](#_Toc453188822)

[**Spis wykresów** 21](#_Toc453188823)

# **WSTĘP [3][4]**

## Czym jest replikacja?

Replikacja danych jest ważnym zagadnieniem w systemach rozproszonych. Replikacja danych polega na przechowywaniu kopii danych na wielu komputerach. Standardowa replikacja polega na utworzeniu kopii zdalnej tabeli z bazie lokalnej. Tabela będąca kopią nazywa się **repliką**, a tabela na podstawie, której utworzono replikę nazywa się **tabelą źródłową** (nazywaną również tabelą master lub tabelą bazową).

Replika może zawierać wszystkie atrybuty i rekordy tabeli źródłowej lub ich podzbiór.

W architekturze standardowej replikacji replika jest tylko do odczytu. Replika posiada cechę automatycznego odświeżania, tzn. zmiany zawartości tabeli źródłowej propagują się do repliki automatycznie.

## Powody zwielokrotniania

Replikacja pozwala nam na:

* Skalowalność – możliwe jest rozłożenie obciążenia pomiędzy wieloma serwerami. Operacje zapisu i aktualizacji rekordów odbywają się na jednym serwerze, a pobieranie i przeszukiwanie danych z drugiego.
* Bezpieczeństwo – Zwielokrotnianie w celach efektywnościowych jest ważne w sytuacjach, w których system rozproszony należy skalować w wymiarze liczbowym i geograficznym. Skalowanie w wymiarze liczbowym występuje na przykład wówczas, gdy wzrasta liczba procesów wymagających dostępu do danych zarządzanych przez jeden serwer. W tym wypadku efektywność możemy polepszyć przez zwielokrotnienie serwera i podział pracy

Istnieje wiele korzyści z zastosowania tej techniki:

1. **zwiększona wydajność** - więcej obciążenia (na przykład żądania klientów) mogą być tolerowane, ponieważ obciążenie jest dzielone między kilka procesów. Ponadto, zmniejszone może być opóźnienie replikacji danych bliżej użytkownika. Niestety, korzyści są mniejsze, jeżeli replikowane dane są do odczytu / zapisu.
2. **zwiększona dostępność** - replikacja pomaga w tolerowaniu poszczególnych awarii serwera. Jeśli połączenie z serwerem nie powiedzie się z prawdopodobieństwem p, liczba serwerów potrzebnych do zapewnienia określonego poziomu usług to - Dostępność = 1-PN.
3. **odporność na awarie** - Na przykład, jeśli serwer grupy serwerów n posiada złą informację inne mogą przegłosować nieprawidłowy serwer w celu zapewnienia poprawnych danych dla klienta.

## Podstawowy model replikacji danych.

Z punktu widzenia klienta widzenia, jest tylko jedna logiczna kopia danych. Jeśli klient dokonuje aktualizacji danych to zmiany powinny znaleźć odzwierciedlenie we wszystkich istniejących replikach.

**Podstawowy model** zarządzania replikowanymi danymi:

- Klienci wysyła żądanie do systemu(front-end).

- System(front-end) zapewnia przejrzystość, ukrywając fakt, że dane są replikowane.

- System(front-end) komunikuje się z jednym lub więcej menedżerów repliki do pobierania / zapisywania danych.

- Zarządcy repliką współdziałają w celu zapewnienia, że dane są spójne.

# **Tabelaryczne zestawienie algorytmów**

|  |  |
| --- | --- |
| Algorytmy pasywne | Algorytmy aktywne |
| Kopia podstawowa – pisanie zdalne | **Brak kopii podstawowej - z globalnym znacznikiem czasu Lamporta** |
| Kopia podstawowa – pisanie globalne | **Brak kopii podstawowej – z procesem porządkowym (koordynatorem nadającym unikalne id operacjom)** |
|  | **Brak kopii podstawowej – z głosowaniem kworum** |

Tabela . Podział wybranych algorytmów na pasywne i aktywne [10]

|  |  |  |
| --- | --- | --- |
| Nazwa algorytmu | Nazwisko Autora | Data powstania |
| Kopia podstawowa – pisanie zdalne | Budhijara | 1993 |
| Kopia podstawowa – pisanie lokalne | Li, Hudak | 1989 |
| Brak kopii podstawowej - z globalnym znacznikiem czasu Lamporta | Rodrigues | 1996 |
| Brak kopii podstawowej – z procesem porządkowym (koordynatorem nadającym unikalne id operacjom) | Fonseca H | 1996 |
| Brak kopii podstawowej – z głosowaniem kvorum | Thomas, Gifford | 1979 |

Tabela . Zestawienie twórców algorytmów i roku powstania.[1]

**Analiza porównawcza algorytmów**

|  |  |  |
| --- | --- | --- |
| Nazwa algorytmu | Opis algorytmu | Wady i problemy związane z użyciem algorytmu |
| Kopia podstawowa – pisanie/czytanie tylko z podstawowej [4] | Operacje pisania i czytania oparte są na jednym zdalnym serwerze  Rezultat : dane nie są zwielokrotniane lecz umieszczone na jednym zdalnym serwerze.  Proces, które chce wykonać operację zapisania jednostki danych x, przekazuje tę operację do serwera głównego x. Serwer ten wykonuje uaktualnienie na lokalnej kopii x, po czym przekazuje uaktualnienie do serwerów zapasowych. Każdy serwer zapasowy dokonuje również aktualizacji i wysyła potwierdzenie z powrotem do serwera podstawowego. Gdy wszystkie serwery zapasowe uaktualnią swoje kopie lokalne, wówczas serwer podstawowy wysyła potwierdzenie do procesu, który zapoczątkował te działania | - Potencjalnym problemem efektywności w tym schemacie może być dość długi czas, który mija, zanim procesowi inicjującemu aktualizację zezwoli się na dalszą pracę. Wskutek tego wszystkie aktualizacje realizujemy jako operacje blokowane. Możemy też zastosować metodę bez blokowania. Gdy tylko serwer główny uaktualni swoją lokalną kopię x, zwraca potwierdzenie. Dopiero potem powiadamia serwery zapasowe, aby też wykonały uaktualnienia.  - Główny problem w nieblokowanych protokołach podstawa-zapas dotyczy tolerowania awarii. W schemacie z blokowaniem procesu klient wie na pewno, że aktualizacja została wykonana na kilku innych serwerach zapasowych. Pewności tej nie ma w rozwiązaniu bez blokowania |
| Kopia podstawowa – zapasowa – czytanie z kopii zapasowej [5] | Istnieją dwa rodzaje protokołów pisania lokalnego, opartego na kopii podstawowej. W pierwszym rodzaju każda jednostka danych x ma tylko jedną kopię. Mówiąc inaczej - nie ma zwielokrotnieli. Ilekroć proces chce wykonać operację na jednostce danych, tylekroć jest do niego najpierw przesyłana ta jedyna kopia, po czym jest wykonywana operacja. Ten protokół tworzy w istocie w pełni rozproszoną, niezwielokrotnioną wersję pamięci danych. Spójność jest oczywista, gdyż zawsze istnieje tylko jedna kopia każdej jednostki danych. | - Jedną z głównych trudności w tej metodzie pełnej wędrówki jest śledzenie aktualnego miejsca pobytu każdej jednostki danych |
| Brak kopii podstawowej - z globalnym znacznikiem czasu Lamporta [6] | W protokołach zwielokrotnionych zapisów (ang. replicated-writes) operacje pisania możemy wykonywać na wielu kopiach, a nie tylko na jednej, jak w wypadku kopii podstawowych. Potencjalnym problemem aktywnego zwielokrotnienia jest konieczność wykonywania operacji wszędzie w tym samym porządku. Jest więc potrzebny mechanizm całkowicie uporządkowanego rozsyłania. Rozsyłanie takie możemy zrealizować przy użyciu znaczników czasu Lamporta. | - konieczność wykonywania operacji wszędzie w tym samym porządku  - znaczniki czasu Lamparta źle się skalują w wielkich systemach rozproszonych |
| Brak kopii podstawowej – z procesem porządkowym (koordynatorem nadającym unikalne id operacjom) [1] | Każdą operacje przekazuje się najpierw porządkowemu  Porządkowy przypisuje niepowtarzalny numer  Porządkowy przekazuje tą operacje do wszystkich kopii.  Uwaga! Operacie wykonywane są w kolejności numerów porządkowych. | - problem zwielokrotnionych wywołań |
| Brak kopii podstawowej – z głosowaniem kworum [1] | Rozważamy rozproszony system plików , zakładamy że plik jest zwielokrotniany na N serwerach  W celu zaktualizowania pliku klient musi się skontaktować przynajmniej z połową + 1 serwerów.  Po zgodzie serwerów plik zostaje podmieniony a nowa wersja zostaje zaopatrzona nowym numerem – numer wersji służy do identyfikowania wersji pliku i jest taki sam dla wszystkich nowo zaktualizowanych kopii pliku. Aby przeczytać zwielokrotniony plik klient również musi się skontaktować z ponad połową serwerów i prosić je o wysłanie numerów wersji. | - Uproszczony schemat Gifforda: Do czytania pliku mającego N zwielokrotnień wymaga się od klienta kworum czytania, Czyli dowolnego zbioru Nr  lub więcej serwerów, podobnie do zmodyfikowania pliku potrzeba kworum pisania, co najmniej Nw serwerów, Wartości te muszą spełniać następujące ograniczenia  Nr + Nw >N  Nw>N/2    Na rysunku a może wystąpić konflikt pisanie – pisanie ponieważ nie jest spełniony warunek 1. W szczególności gdy jeden klient odbierze swój zbiór do zapisu {A, B, C, E, F, G}, a drugi wybierze {D, H, I, J, K, L}  Na rysunku b jest szczególnie interesująca sytuacja ponieważ Nr wynosi tu 1 co oznacza możliwość czytana zwielokrotnionego pliku za pomocą dowolnej liczby kopii – co oznacza aktualizowanie wszystkich kopi przy zapisywaniu – nazwa schematu – czytaj jedno zapisuj wszystko. |

Tabela Opis algorytmów i problemy związane z ich użyciem.

## Zakres stosowalności

* Algorytmy, w których każda kopia umożliwia czytanie są wykorzystywane w bardzo rozproszonych serwerach, np. DNS lub CDN.
* Aktywa replikacja danych znajduje zastosowanie w rozproszonych systemach plików.(OpenAFS, Google Cloud Starage)
* Protokół Gossip, system BAYOU, *Coda* (Constant Data Availability)[2]

# **Wybór algorytmu – sposób oceny**

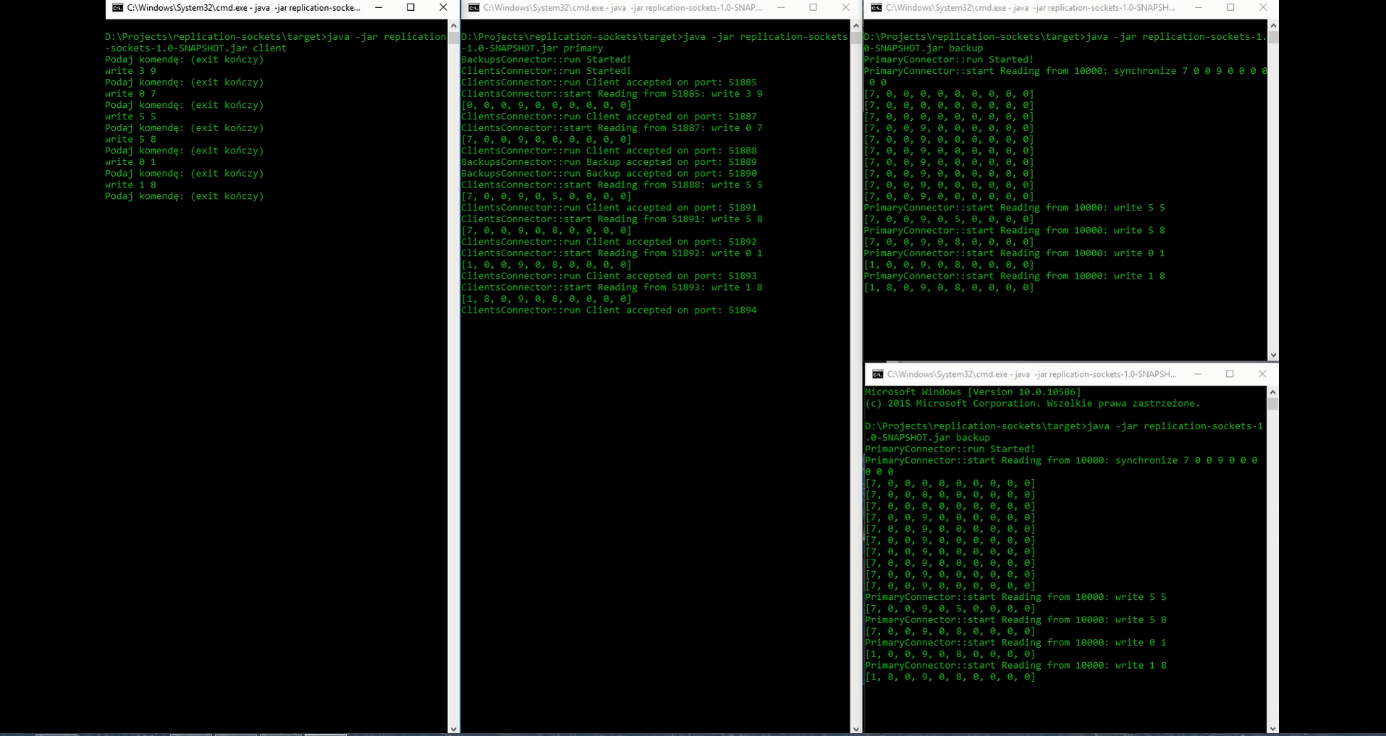
1. Wybrany algorytm jest najprostszy i najłatwiejszy w implementacji
2. Wybrany algorytm nie zawiera w sobie dodatkowych zagadnień z zakresu systemów rozproszonych.
3. Wybrany algorytm pozwala w najbardziej przejrzysty i czytelny sposób przestawić zasadę działania algorytmów replikacji.

# **Tabelaryczne zestawienie technologii rozproszonych**

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa | Opis | Zalety | Wady |
| SOAP [11] | * Simple Object Access Protocol, protokół komunikacyjny oparty o XML. Przekazuje wywołania komponentów Web Services. Działa miedzy innymi z: HTTP, HTTPS, SMTP, JMS oraz RMI. Znaczniki:   <Envelope> - cały komunikat,  <Header> - nagłówek,  <Body> - informacja o żądaniu i odpowiedzi,  <Fault> - opis błędów. | * Niezwykłą elastyczność protokołu, który pozwala przenosić właściwie dowolne informacje * Możliwość definiowania struktury i semantyki przenoszonych informacji * Możliwość łączenia z różnymi protokołami transportowymi (np. HTTP) * Możliwość realizacji różnych scenariuszy komunikacji * Akceptowalność protokołu przez właściwie wszystkie systemy komputerowe i środowiska systemowe * Niezawodność protokołu dzięki ścisłemu zdefiniowaniu sytuacji wystąpienia błędu oraz zachowania aplikacji w takich okolicznościach | * Duży narzut samego języka XML (rozmiar komunikatu jest znacząco większy niż sumaryczny rozmiar danych w nim zawartych) * Jest jeszcze dość młodym protokołem, podlega rozwojowi i modyfikacjom (chociaż jest już dość dobrze i ściśle zdefiniowany) * Trudność w utrzymaniu aplikacji klienta |
| RMI [12] | * Remote Method Invocation, umożliwia programowanie rozproszone w Javie. Mechanizm zdalnych wywołań umożliwia wywołanie metod z obiektów pod kontrolą innych maszyn wirtualnych języka Java. Mogą działać na różnych komputerach. | * Prostota jego użytkowania, w porównaniu np. z COBRA, RMI jest proste i mało skomplikowane * Troszczy się o szczegóły przesyłania obiektów ich serializację i deserializację * Zapewnia niezależność programów od procesora wystarczy działająca maszyna Javy | * Wsparcie tylko dla programów napisanych w javie * Zarówno klient jak i serwer musza być aplikacjami lub apletami javy i nie można komunikować się z programami napisanymi w żadnym innym języku programowaniu za pomocą tego mechanizmu   (o ile użycie dla klienta Javy jest ok o tyle dla serwera jest często nie możliwe). |
| Sockety [21] | * Narzędzie do komunikacji pomiędzy procesem działającym na tej samej maszynie bądź na innym.   Do stworzenia socketa potrzeba: protokołu, domeny oraz typu komunikacji.  Typowe metody:  Bind() – przypisanie adresu,  Listen() – nasłuchiwanie klientów (oznacza socket jako pasywny)  Accept() – akceptowanie i obsługa oczekujących klientów  Connect() – nawiązanie połączenia z serwerem  Read() – przesyła dane,  Shutdown() – niszczenie socketu. | * Wydajne, * Niski narzut na ruch sieciowy, * Wysłanie tylko zaktualizowanej informacji. | * Problemy z bezpieczeństwem, * Klient i serwer musza posiadać mechanizmy pozwalające zinterpretować dane. |
| REST [14] | * REpresentional State Transfer, zamiast XML używa prostego URL. Większość zadań można uzyskać poprzez żądania HTTP 1.1 takie jak GET, POST, PUT, DELETE. Dane można przesyłać przez JSON, RSS. | * Prostota, * Mniejszy narzut obliczeniowy, * Wykorzystanie znanej i przetestowanej infrastruktury: Web, * Możliwość stosowania serwerów pośredniczących, * Mniej problemów ze współoperacyjnością (jednolity interfejs) * Minimum narzędzi potrzebnych do implementacji | * Dodaje znikome opóźnienia, * Żądania nie są wystarczające dla długich ciągów danych, |
| MPI [18] | * Message Passing Interface, protokół przesyłania komunikatów pomiędzy procesami programów równoległych.   Komunikacja może być grupowa bądź punktowa.  MPI\_Init – inicjalizacja MPI,  MPI\_Send – wysyłanie blokujące,  MPI\_Recv – odbiór blokujący  MPI\_Finalize – zakończenie działania. | * Wysoka wydajność, * Efektywna obsługa dużej liczby procesów, * Dobra dokumentacja, * Bogata biblioteka funkcji, * Przenośność, | * Złożony sposób tworzenia programów równoległych, * Statyczna konfiguracja jednostek przetwarzających, |
| CORBA [15] | * Common Object Request Broker Architecture, przeznaczona przede wszystkim do wspomagania programowania pomiędzy systemami niekompatybilnymi. Określa metody dostępu do obiektów i komunikacji między obiektami. Tworzenie aplikacji w tym standardzie wymaga:  1. Zdefiniowania specyfikacji w języku IDL (Język definicji interfejsu) 2. Kompilację do języka docelowego 3. Implementację serwera na podstawie specyfikacji 4. Programowanie klienta 5. Uruchomienie ORB (Pośrednik zleceń obiektowych), serwera i klienta. | * Architektura CORBA jest otwartym rozwiązaniem opartym na opublikowanej specyfikacji * Jest niezależna od sprzętu i systemu operacyjnego. Współdziałające komponenty mogą działać na różnych architekturach sprzętowych i pod kontrolą różnych systemów operacyjnych. * Obiekt programowy zgodny z architekturą COBRA posiada ściśle zdefiniowany interfejs, poprzez który odbywa się komunikacja. Zmiany w implementacji obiektu nie mają wpływu na inne obiekty, o ile zostanie zmieniony interfejs. * Komunikacja pomiędzy obiektami programowymi zgodnymi z COBRA odbywa się przy wykorzystaniu IIOP. Obiekty programowe mogą ze sobą w pełni współpracować, nawet jeżeli działają na różnych systemach operacyjnych i zostały utworzone z wykorzystaniem różnych języków programowania. * Obiekty zbudowane na jednej platformie mogą być wykorzystane z każdej innej z obsługiwanych platform * Budowa aplikacji odbywa się zgodnie z zasadami techniki obiektowej. * Dostęp do obiektów bez konieczności określania ich położenia. | * Brak standardowego i szeroko zaimplementowanego mechanizmu bezpieczeństwa * Przywiązanie do szczegółów technicznych języków niskiego rzędu * Trudność we współdziałaniu i przenośności * Konieczna jest komunikacja makroskopowa * Problemy z bezpieczeństwem. |

Tabela Zestawienie wybranych technologii

# **Implementacja**



Screen z działania w technologii Socket

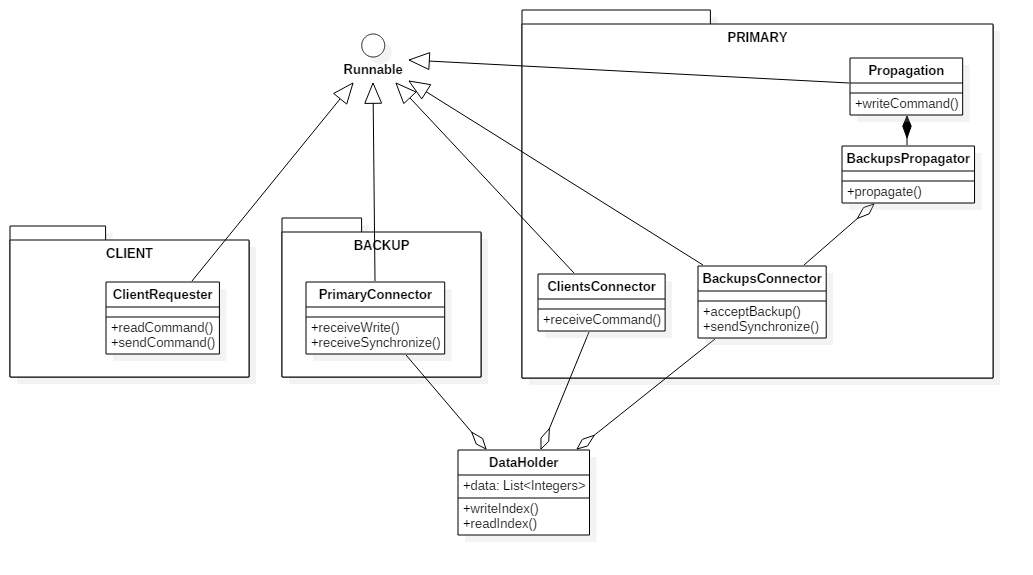
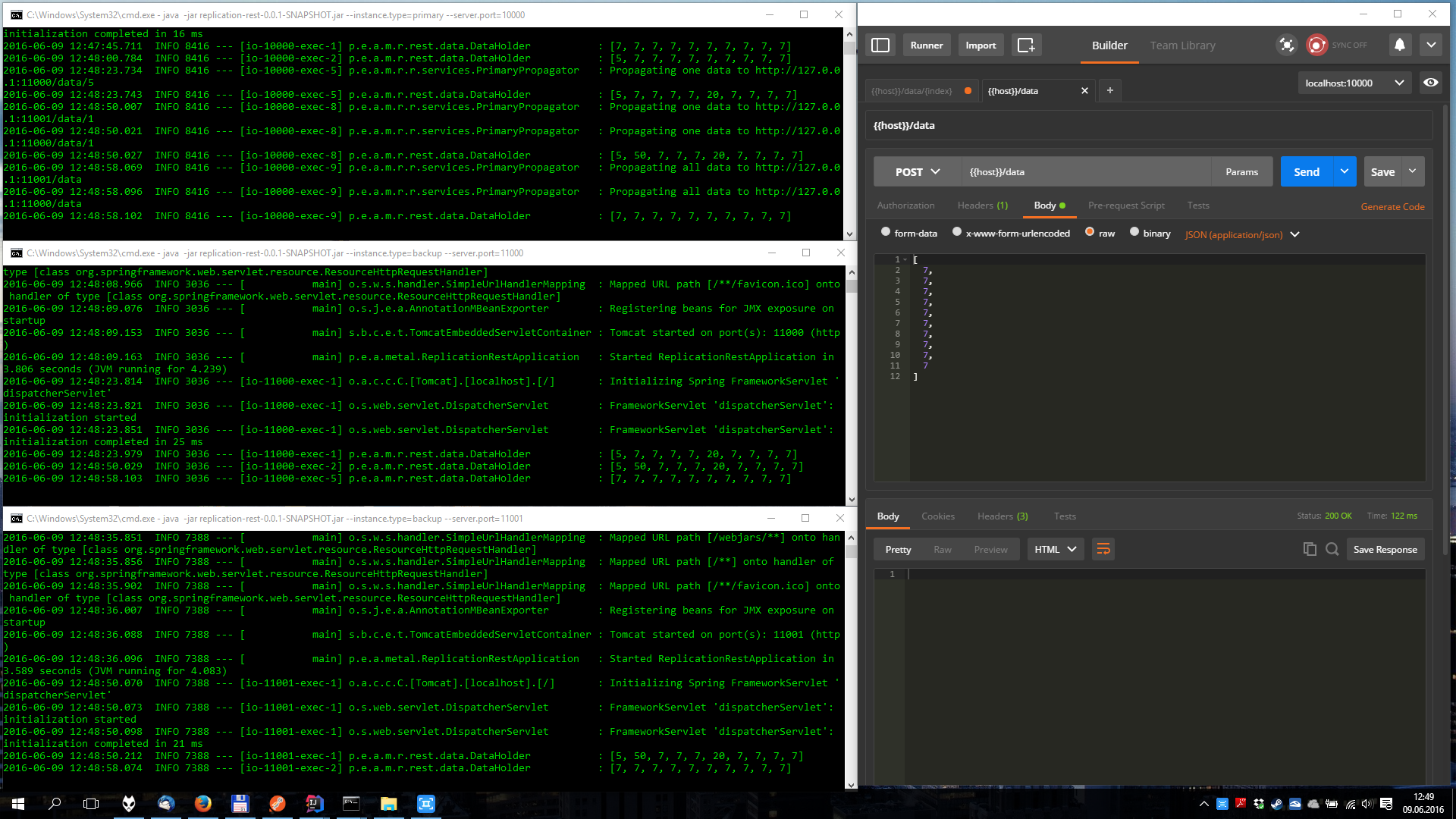


Diagram klas dla aplikacji w technologii Socket



Screen z działania aplikacji w technologii REST

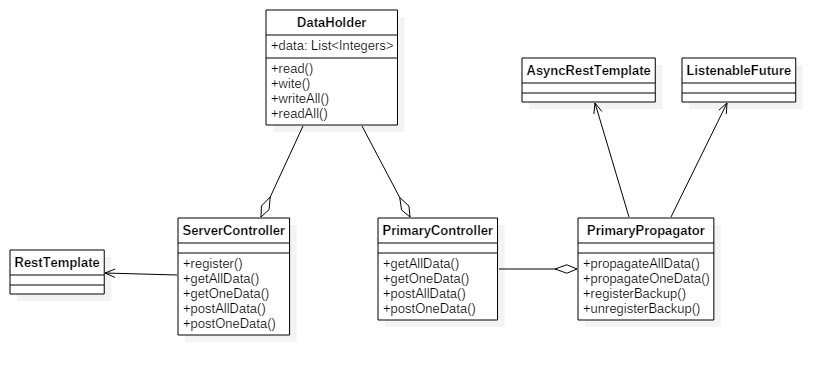


Diagram klas aplikacji w technologii REST

# **Analiza porównawcza implementacji (wykresy) czas/trudność [17]**

## nakładu (czasu) waszej pracy wymaganego do zaimplementowania

|  |  |
| --- | --- |
| Nakład pracy | Czas [h] |
| Sockety | 16 |
| REST | 26 |

Tabela Nakład czasu pracy w poszczególnych technologiach

Wykres Nakład czasu pracy w poszczególnych technologiach

Różnice w nakładzie pracy wynikały z nieznajomości zagadnienia, a także technologii. Nie znaliśmy technologii REST co pociągało za sobą przyswojenie podstaw, a dopiero w kolejnym kroku napisanie programu zgodnie z opisem algorytmu. Część czasu została poświęcona na testy działania programu i zebranie danych, przedstawianych w podpunkcie B.

Z drugiej strony usługi typu REST działają w środowisku serwera aplikacyjnego TOMCAT [19]. Serwer ten do obsługi każdego żądania od klienta tworzy osobny wątek, zwalnia to programistę z konieczności zarządzania tymi wątkami, w socketach sytuacja jest odwrotna. Aby możliwe były równoległe połączenia trzeba ręcznie tworzyć osobne wątki dla każdego socketu [21].

## porównania szybkości działania tego samego algorytmu w różnych technologiach

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technologia | Czas połączenia [s] | Czas przesyłania dużego łańcucha znaków [s] | Wielkość wiadomości dla przykładowego łańcucha znakowego [kB] | Wielkości wiadomości dla przykładowego ciągu liczb [kB] |
| Sockety | 0,00234 | 0,001777 | 2,479 | 80,875 |
| REST | 0,00483 | 0,184198 | 4,905 | 330,547 |

Tabela Zestawienie szybkości działania tego samego algorytmu w dwóch różnych technologiach

Wykres Wielkość wiadomości dla przykładowego łańcucha znakowego

Wykres Wielkości wiadomości dla przykładowego ciągu liczb

Wykres Czas połączenia

Wykres Czas przesyłania dużego łańcucha znaków

# Wielkość przesyłanych wiadomości dla ciągu znaków wypada na korzyść technologii Socketów. Jeszcze większą różnicę można zauważyć w przypadku przesyłania dużego ciągu liczb. Odbija się to również na czasie przesyłania odpowiednich danych.

Rest bazuje na protokole http, wiąże się to z koniecznością opakowania przesyłanych danych w żądanie HTTP. Gdy przesyłania jest mała ilość informacji to większość danych stanowią nagłówki HTTP [17].

Wykres Ilości kodu w poszczególnych technologiach.

Implementując usługi typu REST został wykorzystany framework Springboot i serwer aplikacyjny TOMCAT. Posiadają one wbudowane funkcje sieciowe, co zmniejsza nakład pracy koniecznej do uzyskania połączenia między komputerami. Springboot pozwala deklaratywnie definiować kształt usług za pomocą adnotacji języka Java. Implementując komunikacje za pomocą Socketów wystąpiła konieczność ręcznego zarządzania połączeniami i wątkami w aplikacji.

Wykres 7 Czas uruchamiania implementacji.

# **Bibliografia**

·

1. Andrew  S.Tanenbaum, Maarten Van Steen, „Distribiuted Systems – Principles and Paradigms” Second Edition, 2007 , ISBN 0-13-239227-5
2. Sukumar Ghosh, „Distributed Systems – An Algorithmic Approach”, 2007, ISBN 1-58488-564-5
3. George Coulouris, „Distributed Systems – Concepts and Design” Fifth Edition , 2012, ISBN 0-13-214301-1
4. Mullender Sape, „Dustributed Systems” Second Edition, 1993, ISBN 978-0201624275
5. Li K.,Hudak P., „Memory Coherence in Shared Virtual Memory Systems”, 1989
6. Fonseca H., Verissimo P., „Totally Ordered Multicast in Large-Scale Systems”, Opublikowano: „Proceedings of the 16th International conference on Distributed Computing Systems”, 1996, ISBN 0-8186-7399-0
7. Stanford University, „Chapter 14 – Replication”, http://www-cs-students.stanford.edu/~dbfaria/quals/summaries/Coulouris-chap14.txt (dostęp 6.05.2016)
8. Sami Rollins, „Replication”, 10.2008 http://www.cs.usfca.edu/~srollins/courses/cs682-s08/web/notes/replication.html (dostęp 10.05.2016)
9. K. Banas „Systemy Równoległe i Rozproszone - Wykład 13”, 03.2016 http://www.metal.agh.edu.pl/~banas/SRR/SRR\_W13\_Rozglaszanie\_Uzgadnianie.pdf (dostęp 12.05.2016)
10. Politechnika Warszawska, „Rozproszone systemy operacyjne”, 06.2007, http://www.ia.pw.edu.pl/~tkruk/edu/rsob2010/rso\_proj2007/rso2007 (dostęp 1.06.2016)
11. Oracle, „Simple Object Access Protocol Overview”, 2001, https://docs.oracle.com/cd/A97335\_01/integrate.102/a90297/overview.htm (dostęp 12.05.2016)
12. Polsko-Japońska Akademia Technik Komputerowych,  „RMI – programowanie rozproszone”, 2010, http://edu.pjwstk.edu.pl/wyklady/mpr/scb/W11/W11.html (dostęp 20.05.2016)
13. · M.Zakrzewicz, „Wprowadzenie do technologii Web Services: SOAP, WSDL i UDDI”, 05.2006, ·

http://www.cs.put.poznan.pl/mzakrzewicz/pubs/ploug06ws.pdf (dostęp 20.05.2016)

1. J.Brzeziński, C.Sobaniec – Politechnika Poznańska „Usługi sieciowe REST”, 2013 - https://www.soa.edu.pl/c/document\_library/get\_file?uuid=46b0faf6-6743-4184-ab16-dbddfd413685&groupId=10122 (dostęp 20.05.2016)
2. T.Olas – Politechnika Częstochowska, 2011 „Oprogramowanie systemów równoległych i rozproszonych”, http://icis.pcz.pl/~olas/srr/wyklad8.4.pdf (dostęp 2.06.2016)
3. Robert Werembel – „Rozproszeone bazy danych – replikacja danych (Wykład 1) http://wazniak.mimuw.edu.pl/images/5/55/ZSBD-2st-1.2-lab1.tresc-1.1.ppt(dostęp 8.06.2016)
4. Ph. D. Simon Tuffs, „How fast is your network today?”, 2004

http://soap-stone.sourceforge.net/ (dostęp 08.06.2016)

1. K. Banaś, „Programowanie równoległe – wykład 10”, 2015

http://www.metal.agh.edu.pl/~banas/PR/PR\_W10\_MPI\_wstep.pdf (dostęp 08.06.2016)

1. University of California, „Architectural Styles and the Design of Network-based Software Architectures”, 2000, https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (dostęp 03.06.2016)
2. Ph. D. Martin Fowler, „Rishardson Maturity Model”, 2010, http://martinfowler.com/articles/richardsonMaturityModel.html (dostęp 02.06.2016)
3. Oracle Java Documentation „All About Sockets”, 2015, https://docs.oracle.com/javase/tutorial/networking/sockets/ (dostęp 04.06.2016)

# **Spis tabel**

[Tabela 1. Podział wybranych algorytmów na pasywne i aktywne 5](#_Toc453188649)

[Tabela 2. Zestawienie twórców algorytmów i roku powstania. 5](#_Toc453188650)

[Tabela 3 Opis algorytmów i problemy związane z ich użyciem. 8](#_Toc453188651)

[Tabela 4 Zestawienie wybranych technologii 11](#_Toc453188652)

[Tabela 5 Nakład czasu pracy w poszczególnych technologiach 12](#_Toc453188653)

[Tabela 6 Zestawienie szybkości działania tego samego algorytmu w dwóch różnych technologiach 13](#_Toc453188654)

# **Spis wykresów**

[Wykres 1 Nakład czasu pracy w poszczególnych technologiach 12](#_Toc453188748)

[Wykres 2 Wielkość wiadomości dla przykładowego łańcucha znakowego 13](#_Toc453188749)

[Wykres 3 Wielkości wiadomości dla przykładowego ciągu liczb 14](#_Toc453188750)

[Wykres 4 Czas połączenia 14](#_Toc453188751)

[Wykres 5 Czas przesyłania dużego łańcucha znaków 15](#_Toc453188752)