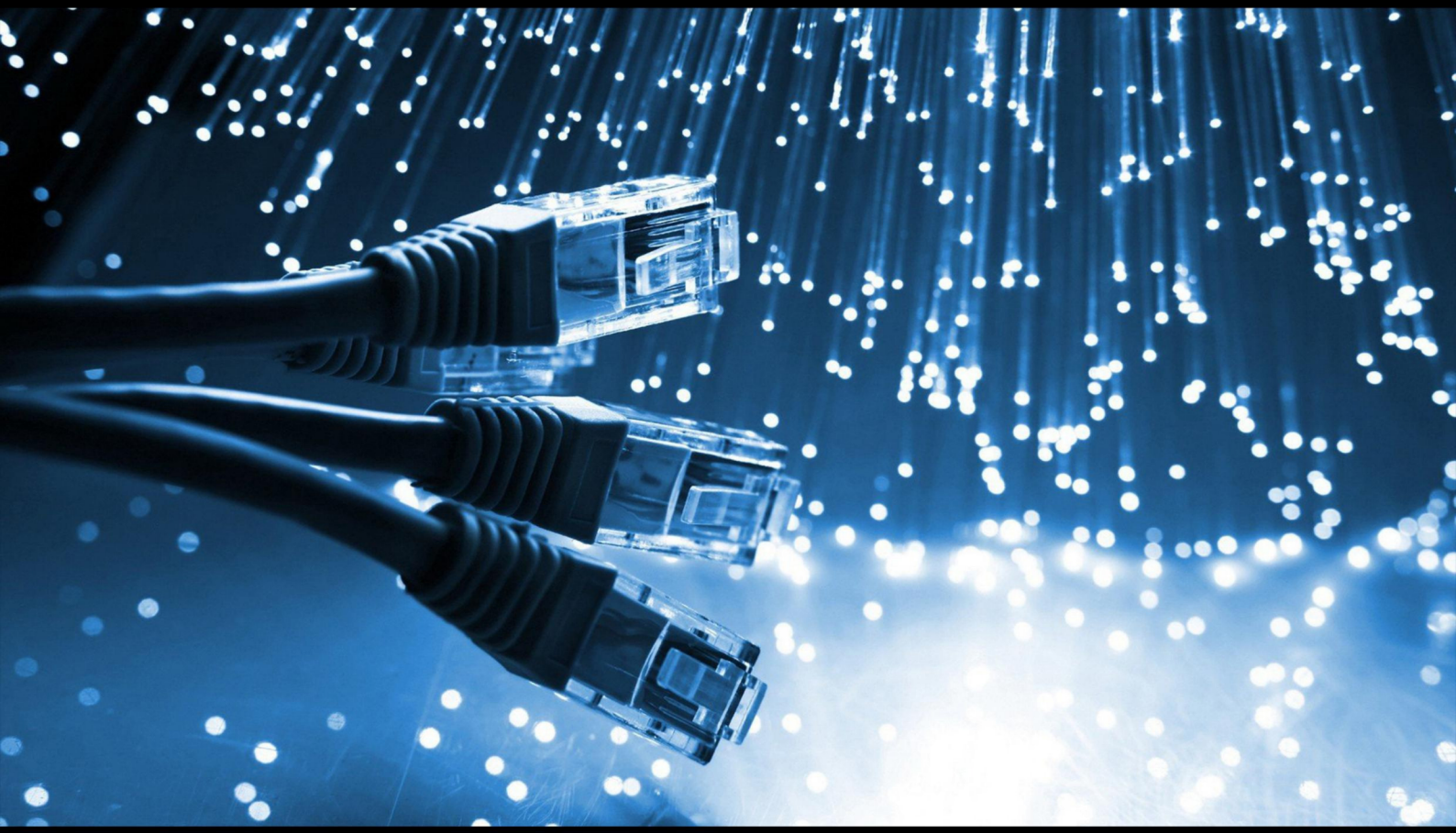
JCSI

Journal of Computer Sciences Institute

Volume 23/2022



Department of Computer Science
Lublin University of Technology

jcsi.pollub.pl

ISSN: 2544-0764

Redakcja JCSI
e-mail: jcsi@pollub.pl
www: jcsi.pollub.pl
Katedra Informatyki
Wydział Elektrotechniki i Informatyki

Politechnika Lubelska
ul. Nadbystrzycka 36 b
20-618 Lublin

Redaktor naczelny:
Tomasz Zientarski
e-mail: t.zientarski@pollub.pl

Redaktor techniczny:
Beata Pańczyk,
e-mail: b.panczyk@pollub.pl

Recenzenci numeru:

dr Edyta Łukasik
dr Adam Kiersztyn
dr Rafał Stęgierski
dr inż. Maciej Pańczyk
dr Mariusz Dzieńkowski
dr inż. Marek Miłosz, prof. uczelni
dr Paweł Powroźnik
dr Waldemar Suszyński
dr Michał Dolecki
dr inż. Sławomir Przyłucki
dr inż. Dariusz Gutek
dr inż. Tomasz Szymczyk
dr inż. Grzegorz Kozieł

Skład komputerowy:
Anna Sałamacha
e-mail: a.salamacha@pollub.pl

Projekt okładki:
Marta Zbańska

JCSI Editorial
e-mail: jcsi@pollub.pl
www: jcsi.pollub.pl
Department of Computer Science
Faculty of Electrical Engineering and Computer Science
Lublin University of Technology
ul. Nadbystrzycka 36 b
20-618 Lublin, Poland

Editor in Chief:
Tomasz Zientarski
e-mail: t.zientarski@pollub.pl

Assistant editor:
Beata Pańczyk,
e-mail: b.panczyk@pollub.pl

Reviewers:

Edyta Łukasik
Adam Kiersztyn
Rafał Stęgierski
Maciej Pańczyk
Mariusz Dzieńkowski
Marek Miłosz
Paweł Powroźnik
Waldemar Suszyński
Michał Dolecki
Sławomir Przyłucki
Dariusz Gutek
Tomasz Szymczyk
Grzegorz Kozieł

Computer typesetting:
Anna Sałamacha
e-mail: a.salamacha@pollub.pl

Cover design:
Marta Zbańska

ISSN 2544-0764

Spis treści

Contents

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 66-70
Received: 19 October 2021
Accepted: 5 November 2021

Comparative analysis of selected programming frameworks of Java-based web applications

Analiza porównawcza wybranych szkieletów programistycznych aplikacji webowych opartych na języku Java

Radosław Książek*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article presents the results of a comparative analysis of Spring (with Spring Boot configuration), Micronaut and Quarkus programming frameworks. The recently observed increase in popularity of these solutions made it necessary to determine their application profile. In order to determine the characteristics of the researched technologies, a series of performance and optimization tests of applications built on the basis of the above-mentioned programming frameworks were carried out. The results of the analyzes showed that thanks to the high degree of optimization, the Micronaut and Quarkus frameworks are perfectly adapted to work in cloud environments, while Spring (Boot) framework, despite its lower efficiency, is an irreplacible solution in complex projects.

Keywords: comparative study; Spring Boot; Micronaut; Quarkus

Streszczenie

Tematem artykułu jest analiza porównawcza szkieletów programistycznych Spring (z konfiguracją Spring Boot), Micronaut oraz Quarkus. Obserwowany w ostatnim czasie wzrost popularności tych rozwiązań uwarunkował konieczność wyznaczenia ich profilu zastosowań. W celu wyznaczenia charakterystyk badanych technologii, przeprowadzono szereg testów wydajnościowych oraz optymalizacyjnych aplikacji zbudowanych w oparciu o wymienione szkielety programistyczne. Rezultaty analiz wykazały, iż dzięki wysokiemu stopniu optymalizacji, szkielety Micronaut oraz Quarkus są doskonale przystosowane do pracy w środowiskach chmurowych, natomiast rozbudowany szkielet Spring (Boot), pomimo mniejszej wydajności, jest pozycją niezastąpioną w bardziej złożonych projektach.

Słowa kluczowe: analiza porównawcza; Spring Boot; Micronaut; Quarkus

*Corresponding author

Email address: radoslaw.ksiazek@pollub.edu.pl (R. Książek)

1. Wstęp

Rozwój rozbudowanych aplikacji internetowych spełniających szereg złożonych reguł biznesowych jest zagadnieniem pochłaniającym dużą ilość zasobów czasowych oraz ludzkich. Pożądany w dzisiejszych czasach wysoki stopień optymalizacji procesu planowania i implementacji takich serwisów istotnie redukuje koszty związane z wytwarzaniem, bądź utrzymywaniem oprogramowania. Dokonane przez autora pracy poszukiwania dostępnych płaszczyzn optymalizacji wspomnianych procesów wykazały, iż znaczący wpływ na łączny czas projektowania i opracowywania serwisów ma szkielet programistyczny wybrany do realizacji danego projektu.

2. Cel i zakres badań

Celem badań jest porównanie następujących szkieletów programistycznych:

- Spring MVC (z konfiguracją Spring Boot),
- Micronaut,
- Quarkus

w kontekście implementacji przykładowej aplikacji internetowej typu CRUD w technologii REST, w języku Java (wersja 15).

Zakres badań obejmuje dokonanie analizy porównywanych technologii z uwzględnieniem różnic strukturalnych i funkcjonalnych. Każdy szkielet programistyczny poddany został testom wydajnościowo-obciążeniowym, których rezultaty poddano analizie w sekcjach końcowych artykułu.

3. Przedmioty badań

Jako obiekty badań wybrano trzy z najpopularniejszych szkieletów programistycznych stanowiących podstawę rozwoju aplikacji napisanych w języku Java [1].

Pierwszą analizowaną pozycją jest Spring MVC (wydany w 2002r.) z bibliotekami konfiguracyjnymi Spring Boot (wydanymi w roku 2014.). Głównym celem tego szkieletu było uproszczenie procesu rozwoju aplikacji internetowych poprzez natywne wspieranie paradygmatu odwróconego sterowania (ang. IoC, Inversion of Control) oraz stosowania mechanizmów wstrzykiwania zależności [2]. Szkielet ten był przez długi czas najpopularniejszym następnikiem domeny JavaEE (ang. Java Enterprise Edition) i technologii EJB (ang. Enterprise JavaBeans).

W czasach wzrostu popularności architektury mikroserwisowej, zaszła potrzeba optymalizacji istniejących rozwiązań. Organizacja Micronaut Foundation, dostrze-

gając kluczowe wady w istniejących szkieletach programistycznych, stworzyła projekt Micronaut. Główną innowacją, którą wprowadzał ten szkielet było natywne wsparcie kompilacji projektu do obrazu kontenera platformy GraalVM [3]. Mechanizm ten znacznie zmniejsza zapotrzebowanie aplikacji na zasoby pamięciowe oraz przyczynił się do istotnej redukcji czasu uruchamiania serwisów. W celu zwiększenia stopnia optymalizacji szkieletu wdrożono koncepcję rozszerzonego przetwarzania kodu na etapie kompilacji z jednoczesnym odstąpieniem od stosowania mechanizmów refleksji [4].

Ostatnią z uwzględnionych w badaniach pozycji jest Quarkus. Szkielet ten został stworzony przez organizację RedHat w 2019r. jako odpowiedź na dynamicznie rosnące zainteresowanie technologiami chmurowymi. Głównym założeniem twórców szkieletu Quarkus była minimalizacja zasobów zużywanych przez opracowywane z jego wykorzystaniem aplikacje [5] (m.in. poprzez kompilację do natywnych obrazów kontenerów technologii GraalVM [6]) oraz natywne wsparcie dla stosu Kubernetess [7].

4. Analiza literatury

Z uwagi na długi okres aktywności projektu Spring na rynku deweloperskim aplikacji internetowych dostępne są liczne publikacje odnoszące się do zagadnień wydajności i funkcjonalności tego szkieletu programistycznego. W przypadku szkieletów Quarkus oraz Micronaut, dostępna baza literatury jest niewielka.

Dostępne porównania możliwości implementacji architektury REST w aplikacjach opartych m.in. o szkielet Spring Boot [8] wskazują na rozbudowaną pulę modułów wspierających obsługę zapytań w omawianej konwencji wymiany informacji.

W pracy Sayagh M. et al. [9] autorzy wskazują na wysoki stopień wsparcia konfiguracji wielu aspektów projektu (m.in. ustawienia warstwy repozytoriów oraz kontekstu aplikacji) przez szkielet konfiguracyjny Spring Boot.

Kwestie optymalizacji środowisk uruchomieniowych aplikacji poruszane są stosunkowo rzadko. Jedna z pozycji nawiązujących do tego zagadnienia ("Enhancing Performance of Cloud-based Software Applications with GraalVM and Quarkus") [10] odnosi się do aspektu wsparcia implementacji rozwiązań chmurowych w technologii GraalVM oraz powiązanym z nią szkieletem Quarkus. Rezultaty badań tej pracy podkreślają zwiększoną wydajność oprogramowania zbudowanego w oparciu o szkielety programistyczne wspierające technologię tworzenia obrazów natywnych aplikacji.

Przeprowadzone studia literaturowe wskazują na brak kompleksowych analiz szkieletów Micronaut oraz Quarkus. Niniejsza praca ma na celu uzupełnienie tego obszaru badań o szczegółowe zestawienia porównywanych technologii z uwzględnieniem analiz funkcjonalności poszczególnych szkieletów programistycznych.

5. Metoda badań

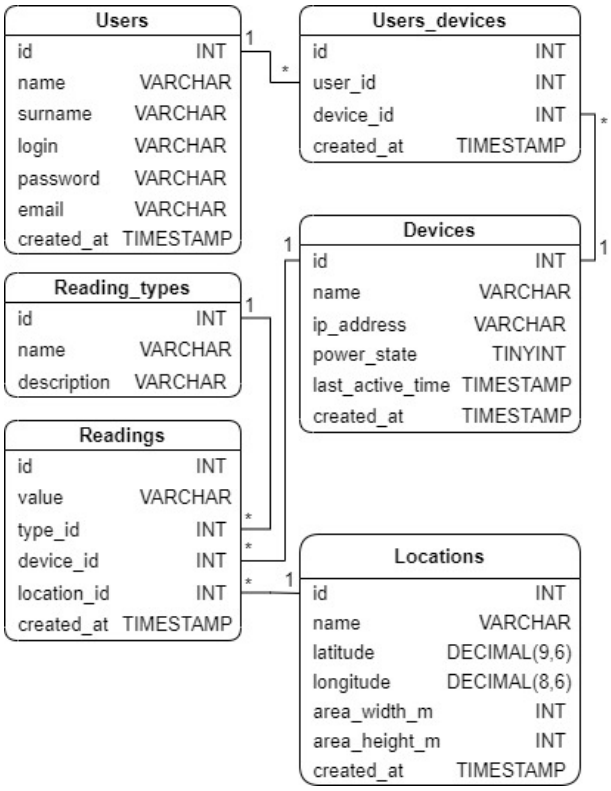
Eksperyment polegał na zaprojektowaniu oraz implementacji referencyjnej aplikacji internetowej typu REST API, umożliwiającej zarządzanie systemem pomiarów opartym o rozproszoną sieć czujników i powiązanych z nimi urządzeń.

W celu wyznaczenia wartości kryteriów przyrównania szkieletów programistycznych, dla każdej z porównywanych technologii przeprowadzone zostały testy wydajnościowe oraz ogólna analiza struktury kodu opracowanych aplikacji i użytych w nich funkcjonalności. Eksperyment został wykonany według scenariuszy testowych przedstawionych w Tabeli 1.

W celu zapewnienia odpowiedniej jakości rezultatów uzyskanych w toku dokonanej analizy, porównywane projekty aplikacji zostały zaimplementowane według opracowanej specyfikacji aplikacji referencyjnej. Zdefiniowana w niej prototypowa struktura projektu, encje oraz zachodzące między nimi relacje (Rys. 1) pozwoliły na zachowanie spójności porównywanych aplikacji w kontekście ich wymagań funkcjonalnych.

Tabela 1: Lista scenariuszy testowych realizowanych w ramach przeprowadzanej analizy szkieletów programistycznych

Lp.	Nazwa	Jednostka miary rezultatów	Opis
1.	Rozmiar pliku wykonywalnego	liczba megabajtów [MB]	Badanie rozmiaru pliku wykonywalnego będącego artefaktem końcowym procesu budowy projektu
2.	Czas budowania projektu	czas [s]	Badanie czasu kompilowania i pakowania kodu źródłowego projektu
3.	Czas uruchamiania	czas [s]	Badanie czasu uruchomienia aplikacji (do momentu obsługi pierwszych żądań)
4.	Zużywana pamięć operacyjna	liczba megabajtów [MB]	Badanie zużytej przez proces aplikacji, systemowej pamięci RAM
5.	Wydajność operacji odczytu z bazy danych	czas [s] wykonania 1 000 odczytów	Test polegający na odczycie 1 000 wpisów z encji ”readings”
6.	Wydajność operacji zapisu do bazy danych	czas [s] wykonania 1 000 zapisów	Test polegający na zapisie 1 000 wpisów do encji ”readings”
7.	Czas pierwszej odpowiedzi (treść statyczna/dynamiczna)	czas [s]	Badanie czasu wymaganego do przetworzenia pierwszej odpowiedzi aplikacji na żądanie od momentu jej uruchomienia (dla odpowiedzi o treści statycznej oraz dynamicznej)
8.	Liczba zapytań / sekundę	liczba	Badanie liczby zapytań możliwych do obsłużenia przez aplikację w ciągu jednej sekundy



Rysunek 1: Schemat przedstawiający zaprojektowaną strukturę bazy danych.

6. Eksperyment

Aplikacje opracowane w oparciu o analizowane szkielety programistyczne poddane zostały testom wydajnościowym oraz optymalizacyjnym zgodnie z wyznaczonymi scenariuszami testowymi. W Tabeli 2. przedstawiono konfigurację środowiska testowego, na którym dokonywane były pomiary.

Tabela 2: Specyfikacja środowiska testowego

Wersja JDK	15
Procesor	AMD Ryzen 4600H (3.0GHz, 6 rdzeni/12 wątków)
Pamięć operacyjna	16 GB (DDR4, 3200MHz)
Dysk	SSD 250GB (PCIe NVMe, PCIe 3.0 x4)
System operacyjny	Windows 10

7. Rezultaty

Rezultaty badania rozmiaru plików wykonywalnych aplikacji (Tabela 3.) wskazały, iż największy rozmiar pliku osiągnęła aplikacja oparta o szkielet Spring (Boot). Uzasadnieniem tego wyniku jest duża liczba dołączonych do projektu aplikacji bibliotek. Rozmiar plików wykonywalnych projektów opartych o szkielety Micronaut oraz Quarkus był zbliżony i wynosił niecałe 30MB.

Tabela 3: Rozmiary plików wykonywalnych aplikacji opracowanych w poszczególnych szkieletach programistycznych

Szkielet programistyczny	Spring (Boot)	Micronaut	Quarkus
Rozmiar pliku wykonywalnego [MB]	38,03	29,91	29,59

Eksperyment weryfikujący średni czas budowania projektów wykazał, iż proces ten trwał najkrócej w przypadku aplikacji opartej o szkielet Quarkus (11,43s). Najdłuższym czasem budowania projektu (14,53s) cechował się projekt wykorzystujący szkielet Spring (Boot). Otrzymane rezultaty przedstawiono w Tabeli 4.

Tabela 4: Średnie czasy budowania projektów aplikacji opracowanych w poszczególnych szkieletach programistycznych

Szkielet	Spring (Boot)	Micronaut	Quarkus
Czas budowania projektu [s]	14,53	12,02	11,43

Testy wykonane w oparciu o scenariusz rozpatrujący czasy uruchamiania aplikacji (Tabela 5.) wykazały, iż najszybciej uruchamiała się aplikacja oparta o szkielet Quarkus (3,24s). Jest to wynik o 1,71s mniejszy od rezultatu najwyższego, uzyskanego dla aplikacji wykorzystującej szkielet Spring (Boot) – 4,95s.

Tabela 5: Czasy uruchamiania aplikacji opracowanych w poszczególnych szkieletach programistycznych

Szkielet	Spring (Boot)	Micronaut	Quarkus
Czas uruchamiania aplikacji [s]	4,95	3,77	3,24

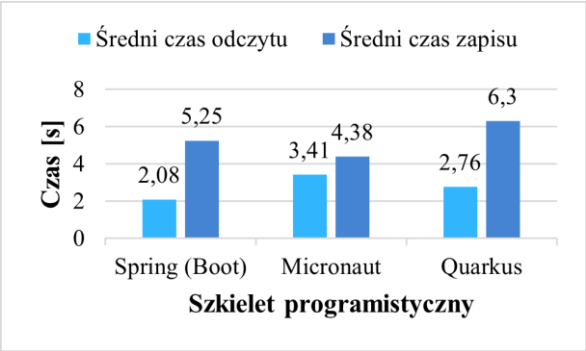
Badanie użycia pamięci operacyjnej przez aplikacje wykazało, iż największe zapotrzebowanie na pamięć RAM (ang. Random Access Memory) wykazała aplikacja oparta o szkielet Spring (Boot) (36,6MB). Najmniejszym użyciem pamięci charakteryzowała się aplikacja wykorzystująca technologię Micronaut (23,5MB). Wyniki badania przedstawione zostały w Tabeli 6.

Tabela 6: Zużywana pamięć operacyjna przez aplikacje opracowane w poszczególnych szkieletach programistycznych

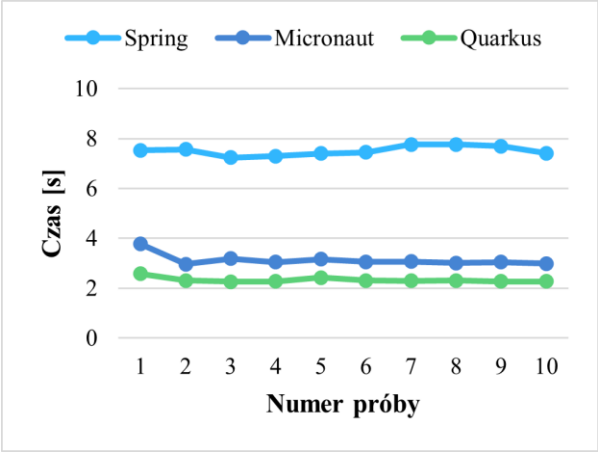
Szkielet	Spring (Boot)	Micronaut	Quarkus
Użyta pamięć operacyjna [MB]	36,6	23,5	24,8

Średni czas odczytu danych z bazy był najmniejszy dla aplikacji opartej o szkielet Spring (Boot) i wynosił 2.08s, zaś najmniejszy średni czas zapisu danych został odnotowany dla aplikacji zbudowanej na podstawie szkieletu Micronaut - 4,38s (Rys 2.).

Badania czasu pierwszej odpowiedzi przedstawiono na Rys 3. Aplikacja oparta o szkielet Quarkus wykazała najkrótszy czas pierwszej odpowiedzi (średnio około 2s), podczas gdy najdłuższe czasy zwłoki były generowane przez aplikację wykorzystującą technologię Spring (Boot) (niecałe 8s).



Rysunek 2: Rezultaty badania czasu trwania wymiany informacji z bazą danych.



Rysunek 3: Rezultaty badania czasu pierwszej odpowiedzi aplikacji.

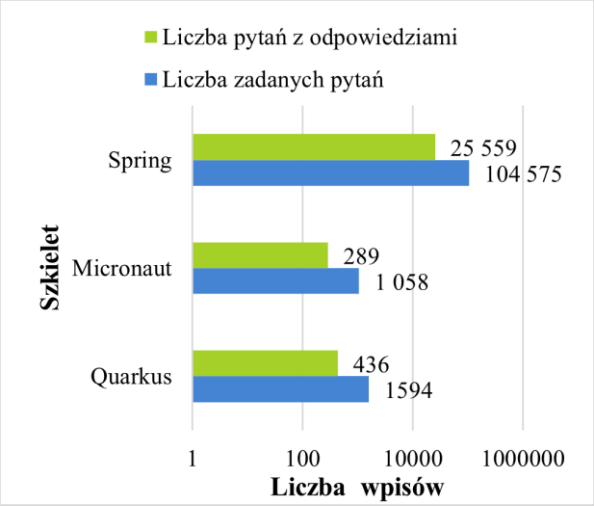
Badanie liczby zapytań obsłużonych przez aplikacje w ciągu sekundy (Tabela 7.) wykazało, iż najwięcej zapytań zostało obsłużonych przez serwis zbudowany w oparciu o szkielet Micronaut (8 962 zapytania). Aplikacja opracowana na podstawie szkieletu Spring (Boot) obsłużyła najmniej zapytań (6 558) spośród analizowanych pozycji.

Tabela 7: Wyniki badania liczby zapytań obsługiwanych przez aplikacje w ciągu jednej sekundy

Szkielet programistyczny	Spring (Boot)	Micronaut	Quarkus
Liczba zapytań na sekundę	6 558	8 962	8 560

W ramach weryfikacji dodatkowego kryterium porównawczego analizowanych szkieletów programistycznych przeprowadzono analizę stopnia wsparcia porównywanych technologii przez społeczność deweloperów. W tym celu przeprowadzono badanie liczby pytań oraz odpowiedzi związanych z konkretnymi tagami (w tym przypadku nazwami szkieletów programistycznych) w serwisie „stackoverflow.com". Rezultaty analiz przedstawiono na Rys 4.

Statystyki odnotowane dla szkieletu Spring (Boot) znacznie przewyższały wyniki pozostałych pozycji w zestawieniu. Jest to spowodowane istotnie dłuższym okresem egzystencji tego szkieletu na rynku deweloperskim aplikacji internetowych. Faktem wartym zaznaczenia w tym kontekście jest większe zainteresowanie zagadnieniami dotyczącymi szkieletu Quarkus (wydanym w roku 2019.) niż Micronaut (pierwsze wydanie w roku 2018.).



Rysunek 4: Statystyki odpowiedzi na pytania odnoszące się do zagadnień związanych z poszczególnymi szkieletami programistycznymi na stronie stackoverflow.com.

8. Wnioski

Przeprowadzone badania wykazały, iż przewodnim zastosowaniem technologii Spring MVC z konfiguracją Spring Boot jest implementacja aplikacji o wysokim poziomie złożoności. Wysoka dostępność kompatybilnych, zewnętrznych bibliotek zwiększa możliwości integracji dodatkowych funkcjonalności w aplikacji [11], a wysokie wsparcie społeczności ułatwia proces tworzenia oprogramowania mniej doświadczonym deweloperom. Rezultaty analiz wydajnościowych i optymalizacyjnych wskazały, iż szkielet Spring (Boot) w wielu scenariuszach był najmniej wydajnym rozwiązaniem spośród innych porównywanych technologii.

Szkielet Micronaut jest wysoce zoptymalizowanym rozwiązaniem, doskonale wpisującym się w infrastruktury systemów informatycznych opartych o architekturę mikroserwisową. Potwierdzają to przeprowadzone testy wydajnościowe, w których technologia ta zajmowała czołowe pozycje. Jedną z głównych cech specyficznych tego szkieletu programistycznego na tle porównywanych pozycji jest unikanie wykorzystywania refleksji i rozszerzone przetwarzanie kodu podczas kompilacji projektu [12]. Mechanizm ten znacznie optymalizuje zużycie zasobów przez działającą aplikację.

Jednym z głównych profili zastosowań szkieletu Quarkus jest tworzenie aplikacji wdrażanych w ekosystem technologii chmurowych. Dokonane analizy wydajnościowo-optymalizacyjne, w większości scenariuszy testowych, wykazały najwyższy stopień optymalizacji zasobów zużywanych przez działającą aplikację w porównaniu do pozostałych, badanych szkieletów programistycznych. Cechy te sprawiają, iż technologia Quarkus może w istotnej mierze przyczyniać się do redukcji kosztów związanych z rozwojem oraz utrzymaniem aplikacji internetowych działających w chmurze.

Literatura

[1] Analiza ankiety dotyczącej dziesięciu najpopularniejszych szkieletów programistycznych języka Java w roku 2020. https://jaxenter.com/java-trends-top-10-frameworks-2020-168867.html , [19.10.2021].

[2] M. Moises, Learn Microservices with Spring Boot: A Practical Approach to Restful Services Using RABBITMQ, Eureka, Ribbon and Cucumber, APRESS, 2018.

[3] T. W. Pusztai, T. Christos, D. Schahram, Engineering Heterogeneous Internet of Things Applications: From Models to Code. 2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC) (2019) 222-231, https://doi.org/10.1109/cic48465.2019.00036.

[4] N. Singh, Z. Dawood, Building Microservices with Micronaut: A Quick-Start Guide to Building High-Performance Reactive Microservices for Java Developers, Birmingham: Packt Publishing, 2021.

[5] F. Marchioni. Hands-on Cloud-Native Applications with Java and QUARKUS: Build High Performance Java Microservices on Kubernetes. PACKT Publishing Limited, 2019.

[6] S. A. Bueno, J. Porter, Quarkus Cookbook: Kubernetes-Optimized Java Solutions. Beijing: O'Reilly, 2020.

[7] T. Koleoso, Beginning Quarkus Framework Build Cloud-Native Enterprise Java Applications and Microservices, Berkeley, CA: Apress, 2020.

[8] R. Kwiatkowski, P. Kopniak, Comparison of Capabilities to Implement REST Services in Java Language Using the Popular Web Application Frameworks, Journal of Computer Sciences Institute 6 (2018) 92-96, https://doi.org/10.35784/jcsi.648.

[9] M. Sayagh, Z. Dong, A. Andrzejak, B. Adams, Does the Choice of Configuration Framework Matter for Developers? Empirical Study on 11 Java Configuration Frameworks, In 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM) (2017) 41-50.

[10] M. Šipek, D. Muharemagić, B. Mihaljević, A. Radovan, Enhancing Performance of Cloud-based Software Applications with GraalVM and Quarkus, In 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO) (2020) 1746-1751.

[11] Lista oficjalnych projektów-bibliotek Spring. https://spring.io/projects , [17.10.2021].

[12] A. B. Kumar, Vijay, Supercharge Your Applications with Graalvm: Hands-on Examples to Optimize and Extend Your Code Using GRAALVM's High Performance and Polyglot Capabilities, Birmingham: Packt Publishing, 2021.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 71–76
Received: 15 November 2021
Accepted: 13 January 2022

Preferences of modern mobile app users

Preferencje współczesnych użytkowników aplikacji mobilnych

Kamil Kasztelan*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The aim of the article is to assess the usefulness of mobile application functions according to the preferences of modern users. Each group of recipients has their own preferences regarding mobile applications. A survey was conducted in the Włodawa district in the first quarter of 2020. 150 random people took part in the survey. It has been noticed that life with a mobile device in hand has become a habit. The most popular group of applications chosen by the respondents are social applications. Users more willingly and more often use the help of mobile devices while shopping, looking for information about the product and promotions. It has been observed that the respondents pay special attention to application security, so they want to be sure that their data is safe. A small group of people are ready to give up a mobile device and start using traditional methods.

Keywords: mobile application; user satisfaction; preferences

Streszczenie

Celem artykułu jest ocena przydatności funkcji aplikacji mobilnych według preferencji współczesnych użytkowników. Każda grupa odbiorców posiada swoje preferencje, co do aplikacji mobilnych. Przeprowadzono ankietę w powiecie włodawskim w pierwszym kwartale 2020 roku. Wzięło w niej udział 150 przypadkowych osób. Zauważono że życie z urządzeniem mobilnym w ręku stało się już przyzwyczajeniem. Najbardziej popularną grupą aplikacji, jaką wybierają badani są aplikacje społecznościowe. Użytkownicy chętniej i częściej korzystają z pomocy urządzeń mobilnych podczas zakupów, szukając informacji o produkcie i promocjach. Zaobserwowano, że badani zwracają szczególną uwagę na zabezpieczenia aplikacji, chcą więc mieć pewność, że ich dane są bezpieczne. Mała grupa osób jest gotowa zrezygnować z urządzenia mobilnego i zacząć korzystać z tradycyjnych metod.

Słowa kluczowe: aplikacja mobilna; satysfakcja użytkownika; preferencje

*Corresponding author

Email address: kasztelan.kamil92@gmail.com (K. Kasztelan)

1. Wstęp

W artykule zostały przedstawione preferencje współczesnego użytkownika aplikacji mobilnych. Temat ten jest istotny, ponieważ, informacje te można wykorzystać do udoskonalania funkcji aplikacji. Wyższa jakość aplikacji zachęci nowych użytkowników do jej pobrania oraz zwiększy satysfakcję obecnych użytkowników.

2. Przegląd literatury

W artykule Andrzeja Sznajdera pt. „Wpływ mobilnej technologii informacyjnej na działalność marketingową przedsiębiorstw" [1] wykazano że rozwój technologii sprzyja rozwojowi biznesu mobilnego. Metoda badań polegała na analizie informacji z źródeł takich jak monografie, raporty z międzynarodowych firm oraz dane statystyczne. Udowodniono, że takie same trendy czyli Social-Local-Mobile występują po stronie podażowej jak i popytowej. Potwierdzono również tezę, że występuje szybki wzrost zainteresowania technologiami mobilnymi.

Według badań przeprowadzonych przez Krzysztofa Kubiaka w artykule pt. „Ocena wybranych aplikacji mobilnych w opinii użytkowników" [2] wynika, że najczęściej pobierane są aplikacje z grupy społecznościowych. Badania przeprowadzone zostały na 103 osobach, które wypełniały ankietę. Wśród ankietowanych było 66% kobiet i 34% mężczyzn. Głównymi cechami, które powinna zawierać aplikacją użytkownicy w ankiecie wskazali na: przydatność (34%), treść strony (21%), layout i grafika (14%), nawigacja i struktura (13%), interakcja (9%), możliwość wprowadzania danych (6%), intuicyjność (3%). W artykule stwierdzono również, że specjalnie zaprojektowane aplikacje zachęcają do korzystania z urządzenia mobilnego.

Magdalena Wójcik w swoim artykule pt. „Nowoczesne trendy w IT – potencjał dla bibliotek" [3] przeprowadziła badania, które wykazały, że niewiele instytucji kultury korzysta z technologii rozszerzonej rzeczywistości. Zastosowanie tej technologii np. w bibliotekach pozwoliło by im na ciągły rozwój, tak aby w lepszy sposób służyć obecnym jak i przyszłym potrzebom użytkowników. Analiza polegała na przeglądzie stron internetowych 25 największych bibliotek publicznych według rankingu American Library Association. Zaledwie dwie biblioteki posiadały informacje o używaniu technologii rozszerzonej rzeczywistości. Omawiana w artykule technologia hologramów jest już dostępna choć wciąż rzadko wykorzystywana w działalności instytucji kultury, szczególnie muzeów.

Piotr Kopyść w swoim artykule pt. „Aplikacja mobilna rzeczywistości poszerzonej jako nowa możliwość promocji turystyki na przykładzie województwa świętokrzyskiego" [4] wykazał, że aplikacje wykorzystujące

rozszerzoną rzeczywistość w świece turystyki to nowość. Użycie tego typu aplikacji stanie się sposobem do wyróżnienia się, a co za tym idzie do zwiększenia doznań turysty. Badania przeprowadzone zostały w województwie świętokrzyskim. Zastosowanie aplikacji tego typu w województwie świętokrzyskim zwiększyłoby zainteresowanie regionem oraz może mieć pozytywny wpływ na funkcjonowanie środowiska przyrodniczego, które jest jednym z głównych walorów turystycznych województwa świętokrzyskiego.

Podczas przeglądu literatury nie znaleziono prac, które badały preferencje współczesnych użytkowników aplikacji mobilnych w powiecie włodawskim.

3. Aplikacje mobline

Kilkadziesiąt lat temu trudno było przewidzieć, że większość ludzi będzie trzymać w swojej kieszeni komputer, który przerasta swoją mocą obliczeniową komputery używane w misjach Apollo. Rozwój technologii w ostatnich latach przyniósł wiele nowości, takich jak: dostęp do Internetu w dowolnym miejscu czy też to, że telefon służy jako wielofunkcyjne urządzenie. Telefon stał się smartfonem, z którego coraz więcej ludzi korzysta w każdej wolnej chwili i w każdym miejscu. Te wnioski dostrzegł również Andrzej Jakubik w swoim artykule [5] pt. „Zespół uzależnienia od Internetu". Na przykład w komunikacji miejskiej można zaobserwować, że ludzie korzystają ze smartfonów w różny sposób, m.in. przeglądając zdjęcia, grając w gry lub też czytając wiadomości [6-10]. Wszystkie te funkcje są zapewniane przez aplikacje mobilne.

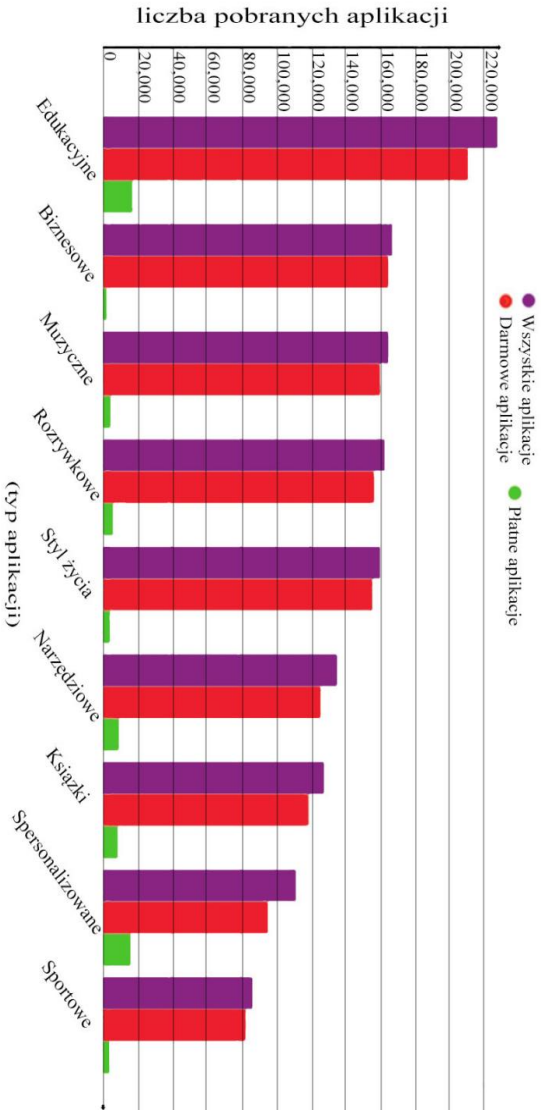
4. Typy aplikacji mobilnych

Aplikacje mobilne dzieli się ze względu na ich przeznaczenie. Wyróżnia się następujące aplikacje:

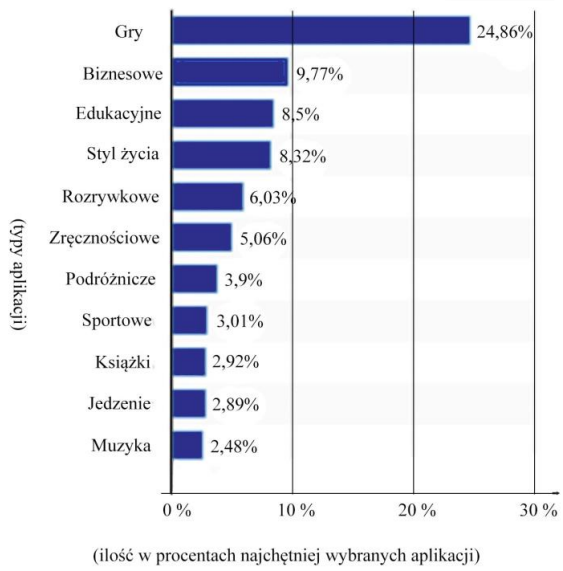
Aplikacje społecznościowe są jednym z podstawowych typów aplikacji, które użytkownik posiada na swoim urządzeniu mobilnym. Trudno wyobrazić sobie w dzisiejszych czasach kontakt ze światem bez takiej aplikacji. Najpopularniejszymi aplikacjami tego typu są: Facebook, Messenger, WhatsApp, Snapchat, Instagram, Twitter.
Aplikacje finansowe do tej grupy zalicza się głównie oprogramowanie bankowe, które pozwala na zarządzanie swoim kontem bankowym, ułatwia kontakt z obsługą klienta itp. Nie tylko aplikacje bankowe są wliczane do tej grupy, ale także aplikacje służące do zarządzania domowym budżetem czy też gromadzenia paragonów i faktur w sposób wirtualny, np. PanParagon, Kwitki, Kontomierz.
Aplikacje sportowe zyskują coraz większą popularność ze względu na panującą modę na zdrowe odżywianie i ekologiczny styl życia. Głównym zadaniem tych aplikacji jest pomoc w planowaniu treningu, liczeniu kalorii czy wręcz zastąpienie trenera personalnego. Do najpopularniejszych aplikacji tego typu należą Endomondo lub Runtastic oraz Fitatu.
Aplikacje nawigacyjne służą do wyznaczania trasy przez kierowców, ale także aplikacje do rozkładów jazdy komunikacji miejskiej. Dzięki tym aplikacjom w bardzo prosty sposób użytkownik może poruszać się bez obaw po dużych–miastach. Najpopularniejszymi aplikacjami w tej kategorii są: Google Maps, JakDojadę, Janosik, Here WeGo.
Aplikacje informacyjne - ich zadaniem jest zebranie i dostęp do najważniejszych wiadomości ze świata. Aplikacje tego typu posiadają czytelny interfejs. Informacje są pogrupowane według kategorii. Użytkownik może sam zdecydować, które informacje go interesują, a których nie chce wyświetlać. Jedną z najbardziej popularnych i cenionych aplikacji są Wiadomości Google, która zbiera i gromadzi informacje z większości serwisów. Do tej grupy aplikacji zalicza się również: Feedly.
Za pomocą aplikacji zakupowych użytkownik może skorzystać z programów lojalnościowych, wykorzystując kupony oraz rabaty i być na bieżąco z aktualnymi promocjami. Tego typu aplikacje pozwalają na dokonanie bezpośrednio zakupów. Najpopularniejszymi aplikacjami są Allegro oraz OLX.
Jedną z najważniejszych grup stanowią aplikacje multimedialne. Za pomocą aplikacji Tidal czy Spotify użytkownik może słuchać legalnie muzyki bez ograniczeń. Natomiast aplikacje, takie jak: YouTube lub Netflix to najlepsze portale z filmami. Dzięki tym aplikacjom dzień jest pełen rozrywki i atrakcji [8].

5. Popularne aplikacje w Google Play i Apple App Store

Za pomocą aplikacji mobilnych można rozszerzyć podstawowe funkcje urządzenia mobilnego. Dostępne są aplikacje darmowe lub płatne, ale także inne, które oferują usługi Premium czy mikropłatności. Darmowe aplikacje często utrzymują się dzięki wyświetlonym reklamom. W niektórych aplikacjach wyświetlanie reklam jest mało zauważalne i odczuwalne.
Aplikacje obecnie tworzone są przeważnie dla dwóch rodzin systemów: Android/WearOS i iOS/WatchOS. Wymienione systemy są głównie dostępne na smartfonach, smartwatchach czy tabletach. Łatwa instalacja i przejrzystość aplikacji pozwalają na docieranie do coraz większej grupy odbiorców. Aplikacje mobilne na system Android można pobrać ze Sklepu Google Play, natomiast na urządzenia z oprogramowaniem iOS ze sklepu AppStore.
Na rysunku nr 1 przedstawiono najpopularniejsze kategorie aplikacji w Google Play. Nie wliczając aplikacji z grupy rozrywki czyli gier. Na drugim miejscu popularności najczęściej pobieranych aplikacji jest kategoria związaną z edukacją. Zaskakujący jest fakt, że dopiero na trzecim miejscu znajdują się aplikacje odpowiedzialne za muzykę. Natomiast na następnych miejscach rankingu znajdują się aplikacje z grup takich jak rozrywka oraz styl życia [11].
Od marca 2017 roku na platformie Apple App Store najpopularniejszą pobieraną kategorią aplikacji były gry. Natomiast drugą najpopularniejszą kategorią były aplikacje biznesowe, a następnie aplikacje odpowiedzialne za edukację i związane ze stylem życia, co przedstawia rysunek nr 2.



Rysunek 1: Najpopularniejsze kategorie aplikacji w Google Play, nie wliczając gier [11].



Rysunek 2: Najpopularniejsze aplikacje w Apple App Store [11].

6. Wady i zalety aplikacji moblilnych

Aplikacje mobilne posiadają wiele zalet i wad. Jedną z większych zalet jest wygoda. Głównym powodem tego jest to, że są wygodniejsze w użytkowaniu na smartfonach niż ich odpowiedniki internetowe. Dodatkowo aplikacje mobilne szybciej ładują widok oraz są najczęściej bardziej intuicyjne w użyciu, ponieważ są optymalizowane do wielkości ekranu urządzenia. Kolejną zaletą aplikacji mobilnych jest ich sprawne działanie w przypadku braku połączenia z Internetem lub jego powolnym działaniem. Następną zaletą aplikacji mobilnych jest możliwość personalizacji i zmian ustawień. W aplikacjach mobilnych dużo łatwiej jest udostępnić użytkownikom możliwość zmian ustawień i preferencji niż na stronie internetowej. Najważniejszą zaletą aplikacji mobilnych jest większe bezpieczeństwo, ponieważ umieszczenie ich w Google Play lub App Store wiąże się z tym, że aplikacje muszą spełniać dodatkowe zasady bezpieczeństwa, do czego nie muszą stosować się strony internetowe. Zastosowanie dodatkowych zasad bezpieczeństwa może uchronić użytkownika przed większą liczbą zagrożeń, np. niechcianych reklam czy wirusów. Jedną z wad, jakie posiadają aplikacje mobilne jest ich mniejsza kompatybilność w porównaniu do technologii webowych. Tworzący aplikacje musi zdecydować się, na jakiej platformie aplikacja będzie działać. Stworzenie aplikacji na system operacyjny Android nie przyniesie korzyści w przypadku, gdy użytkownicy korzystają z urządzeń firmy Apple i odwrotnie. W odróżnieniu do stron internetowych aplikacje trzeba zainstalować. Instalacja wymaga od użytkownika poświęcenia czasu. Użytkownik musi zainstalować aplikację, ale również ją aktualizować. Aby można było korzystać z najnowszych funkcji aplikacji, trzeba ją aktualizować. Częste aktualizacje mogą zniechęcić użytkowników do korzystania z tej aplikacji [12-14].

7. Wzorce niezbędne podczas używania aplikacji przez użytkownika

Użytkownik, korzystając z aplikacji mobilnej powinien zwrócić uwagę na co najmniej jeden z wzorców podanych poniżej. Jest to trudny wybór, gdyż użytkownik oczekuje od aplikacji samych zalet. Każdy wzorzec został opisany poniżej [15]:

bezpieczna eksploracja - użytkownik czujący, że bez żadnych obaw może zapoznać się z interfejsem aplikacji;

pragnienie natychmiastowej satysfakcji - użytkownik odnoszący sukces po pierwszych minutach używania aplikacji;

satisficing - użytkownik przegląda interfejs aplikacji powierzchownie i wybiera pierwszy element, który testuje;

zmiany na bieżąco - uwaga użytkownika przypadkowo lub celowo jest odwracana, aby zmienić jego cel używania aplikacji;

odwlekanie decyzji - użytkownikowi pozwala się pominąć pewne czynności, które zajmują więcej czasu, aby

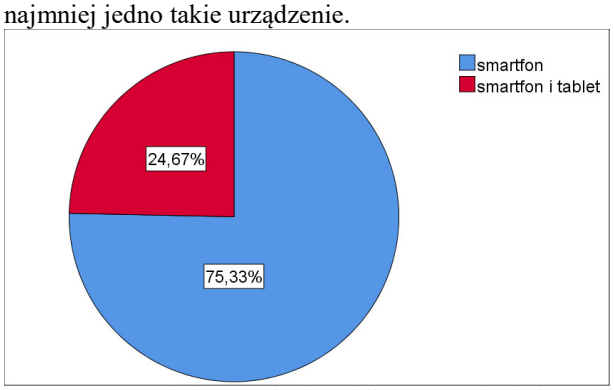
wrócić do nich później, np. rejestracja czy uzupełnienie profilu;
stopniowa konstrukcja - użytkownik może swobodnie podążać do przodu, jak i cofać się;
przyzwyczajenie - użytkownik intuicyjnie porusza się po interfejsie danej aplikacji;
mikroprzerwy - użytkownik może w każdej chwili przestać korzystać z aplikacji i wrócić dalej do niej po przerwie od stanu jej zamknięcia;
pamięć przestrzenna - użytkownik wymyśla własny system klasyfikacji, który ułatwia mu dostęp do elementów aplikacji;
pamięć prospektywna - użytkownik może zostawić przypomnienie, aby wrócić do załatwienia sprawy później;
wspomagane powtarzanie - użytkownik w różnych aplikacjach często musi powtarzać wielokrotnie te same czynności;
miłość do klawiatury - użytkownik nie potrafi poradzić sobie bez użycia klawiatury;
porady innych i osobiste rekomendacje - użytkownik często sugeruje się opinią innych odbiorców;

8. Badania ankietowe

Ankietę przeprowadzono w powiecie włodawskim w pierwszym kwartale roku 2020. Brało w niej udział 150 przypadkowo wybranych osób, z czego 49 osób to kobiety, które stanowiły 33% wszystkich badanych. Wszystkie osoby badane wypełniły ankietę poprawnie bez żadnych braków oraz błędów. Celem ankiety było zbadanie preferencji współczesnego użytkownika aplikacji mobilnej. Został przybliżony schemat statystycznej analizy danych.

9. Analiza wyników badań ankietowych

Ankieta pokazała, że we współczesnym świecie urządzenie mobilne jest niezbędnym elementem życia. Jak można zaobserwować na rysunku nr 3 że 100% badanych osób posiada smartfona, ale tylko 24.67% z nich posiada zarówno smartfon, jak i tablet. Z badań wynika, że nie ma osób, które nie posiadają ani jednego urządzenia mobilnego lub samego tabletu. Obecnie smartfon to urządzenie służące do wszystkiego, każdy posiada co najmniej jedno takie urządzenie.



Rysunek 3: Posiadane urządzenia mobilne przez osoby badane.

Aplikacje społecznościowe są najczęściej wybieranymi programami na urządzenia mobilne w grupie badanych osób, co przedstawia tabela 1. Tylko 13 na 150 badanych osób nie posiada tego typu aplikacji. Jak można było przypuszczać, rozrywka jest nieodzownym elementem życia codziennego, co potwierdzają wyniki ankiety, gdyż 90% badanych posiadają aplikacje, np. Spotify, YouTube czy też gry. Więcej niż połowa osób (55,33%) używa na swoim urządzeniu aplikacji użytkowych.
W grupie badanych osób 68% zadeklarowało, że nie posiada aplikacji z grupy informacyjnej. Użytkownicy prawdopodobnie dowiadują się o aktualnościach z portali społecznościowych, dlatego niewiele osób posiada tego typu aplikacje. Dynamicznie rozwijający się serwis społecznościowy Faceboook wypiera z rynku, takie aplikacje jak: Instagram czy Snapchat. Pokazują to wyniki ankiety, w których 58,67% badanych osób stwierdziło, że nie posiada aplikacji typu Instagram i pokrewnych. Pomimo mody na zdrowy tryb życia, aplikacje z grupy lifestylowe/sportowe są pobierane niezbyt chętnie, bo zaledwie 27,33% osób badanych zadeklarowało, że posiada te aplikacje. Aplikacje mobilne z dziedziny komunikatorów posiada ponad 77,33% badanych. Nieznaczna większość zaledwie 52% badanych osób zadeklarowało, że nie posiada aplikacji związanych z zakupami. Przyczyną tego może być fakt, że jeszcze stosunkowo często można spotkać osoby posiadające zwykłe karty promocyjne, a nie aplikacje do ich przechowywania. Duża grupa osób, bo aż 90,67% osób odpowiedziała, że nie posiada aplikacji randkowych.

Tabela 1: Posiadane aplikacje według wymienionych kategorii

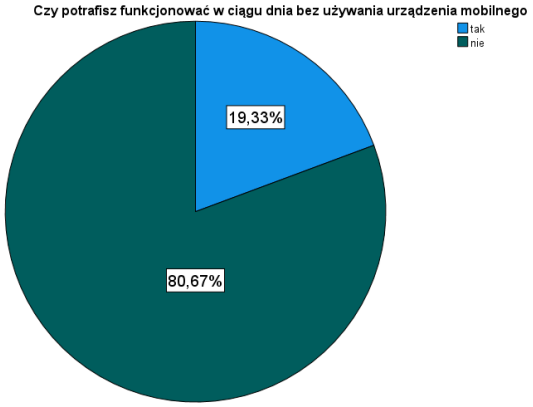
Kategorie	Wybór	Liczba	wartość (%)
społecznościowe (Facebook, Twitter, Google Plus)	nie wybrał	13	8,67%
	wybrał	137	91,33%
rozrywkowe (Spotify, YouTube, gry)	nie wybrał	15	10,00%
	wybrał	135	90,00%
narzędzia użytkowe (kalkulator, notatnik, mapa)	nie wybrał	67	44,67%
	wybrał	83	55,33%
informacyjne (przepisy kulinarne, informacje bieżące)	nie wybrał	102	68,00%
	wybrał	48	32,00%
fotograficzne (Instagram, Snapchat, itp.)	nie wybrał	88	58,67%
	wybrał	62	41,33%
lifestylowe/sportowe (Endomondo itp.)	nie wybrał	109	72,67%
	wybrał	41	27,33%
komunikatory (Messenger, Viber, WhatsApp)	nie wybrał	34	22,67%
	wybrał	116	77,33%
handlowe i zakupowe (Rossmann itp.)	nie wybrał	78	52,00%
	wybrał	72	48,00%
randkowe (Tinder, itp.)	nie wybrał	136	90,67%
	wybrał	14	9,33%

Tabela nr 2 przedstawia odpowiedzi badanych osób na pytanie „Które wzorce według Ciebie są niezbędne podczas używania aplikacji?”. Każda badana osoba mogła wybrać kilka odpowiedzi. Według badanych osób najważniejszym wzorcem jest to, że użytkownik w bezpieczny sposób i bez żadnych obaw może zapoznać się z interfejsem aplikacji (bezpieczna eksploracja - 64,7%). Na drugim miejscu według badanych osób pożądanym wzorcem są mikroprzerwy (36%), dzięki nim użytkownik może w każdej chwili przestać korzystać z aplikacji i wrócić dalej do niej po przerwie od stanu jej zamknięcia. Na kolejnym miejscu z 24,7% poparciem znajduje się taka cecha, jak przyzwyczajenie. Dzięki niej użytkownik aplikacji porusza się intuicyjnie po interfejsie aplikacji. Na dwóch ostatnich miejscach z bardzo małą liczbą odpowiedzi znajdują się takie odpowiedzi, jak: wspomagane powtarzanie - użytkownik często musi powtarzać wielokrotnie te same czynności (1,3%) oraz pragnienie natychmiastowej satysfakcji (6%) - użytkownik odnosi sukces po pierwszych minutach używania z aplikacji.
Jak można było przypuszczać użytkownik aplikacji stawia na bezpieczeństwo. Oczekuje od aplikacji tego, że w czasie gdy z niej korzystanic mu nie grozi. Natomiast badane osoby zwracają małą uwagę na satysfakcję z korzystania z aplikacji.

Tabela 2: Wzorce niezbędne podczas używania aplikacji według badanych osób

Kategorie	Wybór	Liczba	Wartość (%)
bezpieczna eksploracja	nie wybrał	53	35,30%
	wybrał	97	64,70%
pragnienie natychmiastowej satysfakcji	nie wybrał	141	94,00%
	wybrał	9	6,00%
satysfakcjonujący	nie wybrał	132	88,00%
	wybrał	18	12,00%
zmiany na bieżąco	nie wybrał	140	93,30%
	wybrał	10	6,70%
odwlekanie decyzji	nie wybrał	121	80,70%
	wybrał	29	19,30%
stopniowa konstrukcja	nie wybrał	136	90,70%
	wybrał	14	9,30%
przyzwyczajenie	nie wybrał	113	75,30%
	wybrał	37	24,70%
mikroprzerwy	nie wybrał	96	64,00%
	wybrał	54	36,00%
pamięć przestrzenna	nie wybrał	128	85,30%
	wybrał	22	14,70%
pamięć prospektywna	nie wybrał	140	93,33%
	wybrał	10	6,67%
wspomagane powtarzanie	nie wybrał	148	98,70%
	wybrał	2	1,30%
miłość do klawiatury	nie wybrał	130	86,70%
	wybrał	20	13,30%
porady innych i osobiste rekomendacje	nie wybrał	140	93,30%
	wybrał	10	6,70%

Jak można wywnioskować z rysunku nr 4, że 80,67% badanych nie potrafi wytrzymać bez swojego urządzenia nawet jednego dnia. Jest to niepokojące, ponieważ to staje się uzależnieniem. Sięganie po urządzenie mobilne staje się odruchem bezwarunkowym, gdzie można skłonić się do powiedzenia, że jest to równe uzależnieniu, z którym coraz częściej można się spotkać.



Rysunek 4: Podział osób badanych ze względu na umiejętność funkcjonowania bez urządzenia w ciągu dnia.

Na rysunku 5 można zaobserwować że większość osób badanych nie korzysta z rozszerzonej rzeczywistości.

Rysunek 5: Osoby korzystające z rozszerzonej rzeczywistości.

10. Wnioski

Użytkownicy chętniej i częściej korzystają z pomocy urządzeń mobilnych podczas zakupów: szukają informacji o produkcie, porównują ceny oraz opinie innych użytkowników.
Użytkownicy oczekują więcej od personalizacji treści i chcą bardziej angażować się w nią. Są lojalni względem aplikacji, która dostarcza im treści dostosowane do ich upodobań i zwyczajów.
Użytkownicy przenieśli się do świata wirtualnego i oczekują od aplikacji skutecznych i prostych metod płatności urządzeniem mobilnym, które są bezpieczne.
Jedną z najważniejszych rzeczy jest bezpieczeństwo użytkownika. Rosnąca liczba cyberataków sprawia, że

zwraca się więcej uwagi na zabezpieczenia aplikacji. Klienci chcą mieć pewność, że ich dane są chronione.
Życie z urządzeniem mobilnym w ręku stało się już przyzwyczajeniem. Przyzwyczajenia mają to do siebie, że trudno się ich pozbyć. Raczej mała grupa osób postanowi odstawić swojego smartfona i zacząć korzystać z tradycyjnej poczty.
Badania udowodniły, że użytkownicy najchętniej pobierają aplikacje społecznościowe. Natomiast obecnie użytkownicy stawiają bardziej na bezpieczeństwo niż na wygląd aplikacji.
Aplikacje wykorzystujące rozszerzoną rzeczywistość są mało popularne, gdyż użytkownik nie ma okazji do skorzystania z tych technologii po za domem. Instytucje kultury czy też branża turystyczna używając tego typu aplikacje wzbudziła by zainteresowanie wielu osób.

Literatura

[1] A. Sznajder, Wpływ mobilnej technologii informacyjnej na działalność marketingową przedsiębiorstw, Gospodarka Narodowa The Polish Journal of Economics 7-8 (2013) 37-61.

[2] K. Kubiak, Ocena wybranych aplikacji mobilnych w opinii użytkowników, Zeszyty Naukowe Uniwersytetu Szczecińskiego 41 (2015) 83-93.

[3] M. Wójcik, Nowoczesne trendy w IT – potencjał dla bibliotek, Przegląd biblioteczny 84 (2016) 575-589.

[4] P. Kopyść, Aplikacja mobilna rzeczywistości poszerzonej jako nowa możliwość promocji turystyki na przykładzie województwa świętokrzyskiego, Zeszyty Studenckiego Ruchu Naukowego Uniwersytetu Jana Kochanowskiego w Kielcach 3 (2019) 45-53.

[5] A. Jakubik, Zespół uzależnienia od Internetu, Colloquia Litteraria 3 (2002) 133-142.

[6] S. Conder, L. Darcey, Android: programowanie aplikacji na urządzenia przenośne, Helion, Gliwice, (2011).

[7] I. F. Darwin, Android. Receptury, Helion, Gliwice, (2013).

[8] Podział aplikacji i ich wykorzystanie, https://pl.wikipedia.org/wiki/Aplikacja_mobilna, [13.10.2020].

[9] J. Łosiak-Pilch, Aplikacje mobilne w promocji i edukacji zdrowotnej, Edukacja Technika Informatyka 8 (2017) 237-237.

[10] M. Ratalewska, Rozwój rynku aplikacji mobilnych, Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu 487 (2017) 262-272.

[11] Najbardziej popularne aplikacje mobilne, www.statista.com/statistics/279286/google-play-android-app-categories/, [12.12.2020].

[12] J. Nielsen, R. Budiu, Funkcjonalność aplikacji mobilnych, Helion, Gliwice, (2013).

[13] M. Szymańska, Rola Internetu w procesie edukacji dzieci i młodzieży, Problemy Opiekuńczo – Wychowawcze, Publiczna Biblioteka Pedagogiczna w Koninie 6 (2008) 22-24.

[14] J. Trybuchowska, Dostęp do internetu w Polsce i na świecie, Raport strategiczny IAB Polska Internet 4 (2012) 36-80.

[15] J. Tidwell, Projektowanie interfejsów, Helion, Gliwice, (2012).

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 77–83
Received: 24 November 2021
Accepted: 2 December 2021

Web application performance analysis using Angular, React and Vue frameworks

Analiza obciążeniowa aplikacji internetowych z użyciem szkieletów Angular, React i Vue

Konrad Bielak*, Bartłomiej Borek*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The aim of the work is to conduct a performance analysis and, on its basis, to indicate the most user-friendly and fastest operating framework. Three Internet applications have been written to test the speed of operation of selected technologies. Some of the most popular frameworks were analyzed: Angular, React and Vue. Static comparative criteria used in the work are: favorable and generally available documentation, application development, speed of application development, development and support by creators. The practical criteria are the times measured during the execution of the tests. The tests were performed with simple operations using the CRUD (Create Read Update Delete) function. The performance analysis carried out in this way shows the differences between the frameworks. The following developer-Persian tools were used for the comparative analysis: Google Analytics, Google Chrome, Mozilla Firefox, Chrome DevTools.

Keywords: Angular; React; Vue; Performance; JavaScript

Streszczenie

Celem pracy jest przeprowadzenie analizy wydajnościowej oraz wskazanie na jej podstawie najbardziej przyjaznego dla użytkownika i najszybciej działającego szkieletu. Do sprawdzenia szybkości działania wybranych technologii napisane zostały trzy aplikacje internetowe. Analizie poddano jedne z najpopularniejszych szkieletów, mianowicie: Angular, React i Vue. Statyczne kryteria porównawcze wykorzystane w pracy to: rozbudowana i ogólnodostępna dokumentacja, proces wytwarzania aplikacji, szybkość budowy aplikacji, rozwój oraz wsparcie przez twórców. Kryteriami praktycznymi są czasy mierzone podczas wykonywania testów. Testy wykonane zostały za pomocą prostych operacji korzystających z funkcji CRUD (ang. Create, Read, Update, Delete). Dokonano zestawienia wyników szybkości, w jakiej zrealizowane zostały poszczególne operacje oraz opis analizy technicznej danych szkieletów. Do analizy porównawczej wykorzystano narzędzia deweloperskie takie jak: Google Analytics, Google Chrome, Mozilla Firefox, Chrome DevTools.

Słowa kluczowe: Angular; React; Vue; analiza obciążeniowa; JavaScript

*Corresponding author

Email address: konrad.bielak@pollub.edu.pl (K. Bielak), bartlomiej.borek@pollub.edu.pl (B. Borek)

1. Wstęp

JavaScript to język skryptowy opracowany w latach 90-tych. Pierwotnie opracowany przez NetScape dla przeglądarki NetScape Navigator 2.0, aby uczynić strony internetowe bardziej przykuwającymi uwagę użytkowników. Przez lata JavaScript bardzo ewoluował, od języka skryptów witryny, do kodu po stronie serwera, aż do środowiska uruchomieniowego.

Zgodnie z rankingiem hackr.io, Angular, React i Vue to najpopularniejsze szkielety JavaScript [1]. React oraz Vue tworzą interfejsy graficzne za pomocą SPA (Single Page Application). Komponenty wielokrotnego użytku opisywane za pomocą JSX są wykorzystywane w interfejsie Reacta. Jest to zestawienie kodu JavaScript oraz HTML. Natomiast w Vue interfejs zbudowany jest z komponentów, które wykorzystują w swojej implementacji szablony HTML. Angular to następca szkieletu Google Angular JS. Jest on rozwijany przez ten sam zespół. Szkielet Angular narzuca podział poszczególnych komponentów na trzy elementy (logika, interfejs użytkownika i style), podczas gdy Vue zachowuje wszystkie komponenty w jednym pliku.

Większość nowoczesnych witryn internetowych używa JavaScript w pewnym kształcie lub formie, aby zwiększyć szybkość działania i obsługę stron interaktywnych. Tradycyjne strony internetowe to aplikacje wielostronicowe, gdzie nowy dokument HTML jest ładowany za każdym razem, gdy użytkownik zmienia stronę lub jej zawartość. Jest to stosunkowo powolna opcja w porównaniu z najnowszą wersją modelu rozwoju aplikacji jednostronicowej (ang. Single Page Application, SPA). W przypadku tego podejścia jedynie używane elementy, które uległy zmianie są pobierane z serwera i zaktualizowane.

Celem pracy jest przeprowadzenie analizy wydajności trzech wybranych szkieletów oraz wskazanie na jej podstawie szkieletu najbardziej przyjaznego i najszybszego dla użytkownika. Na potrzeby analizy zostały napisane trzy aplikacje, zaimplementowane w różnych szkieletach: Angular, React i Vue. Wszystkie programy

posiadają te same funkcjonalności i napisane w taki sposób, aby zminimalizować ilość kodu do niezbędnego działania. W artykule postawiono tezę badawczą: „Vue jest wydajniejszym szkieletem w mniejszych aplikacjach niż React i Angular". Niniejsze badanie ma na celu ukazanie ich zalet oraz wad, umożliwiając czytelnikowi podjęcie świadomej decyzji przy wyborze między nimi.

2. Przegląd literatury

W pracy przeanalizowano publikacje pod kątem wydajności tytułowych szkieletów. Okazało się to pomocne w doborze odpowiednich metod badawczych.

Pierwszy analizowany artykuł pod tytułem "Porównanie wydajności narzędzi do tworzenia interfejsu aplikacji typu SPA na przykładzie React i Vue.js" [2] przedstawia porównanie dwóch aplikacji posiadających ten sam interfejs użytkownika i oferujących taką samą funkcjonalność. Wykorzystano do tego wymienione w tytule narzędzia JavaScript. Zaimplementowano tabelę z możliwością zmiany liczby wyświetlanych wierszy. Dane zostały wygenerowane za pomocą biblioteki faker.js w postaci pliku JSON. Następnie aplikacje zostały uruchomione w dwóch przeglądarkach internetowych. Każda z nich posiadała własny silnik renderowania, co miało wpływ na różnice w wydajności. Zaimplementowane testy umożliwiały obliczenie czasu pomiędzy ich wywołaniami. Do argumentu funkcji wprowadza się identyfikator, służący do wyznaczenia bloku kodu jaki będzie poddany pomiarom. Rezultatem badania są różnice w wydajności wybranych narzędzi. Ostateczny wynik analizy wskazał, że React przewyższa nieznacznie wydajnością Vue.

W publikacji "Porównanie narzędzie do tworzenia aplikacji typu SPA na przykładzie Angular2 i React" [3] zostały porównane dwa najczęściej stosowane narzędzia do tworzenia aplikacji internetowych typu SPA. W badaniu wykorzystano aplikacje testowe, które posiadały takie same funkcjonalności. Skorzystano z narzędzi Google Chrome DevTools aby testować i porównać możliwości wybranych szkieletów. Pomaga to przeglądać i debugować pliki HTML, CSS oraz witryny JavaScript i sprawdza ruch sieciowy pobierany przez witrynę, sprawdza szybkość witryny i analizę wydajności środowiska wykonawczego. Szkielety porównywano pod kątek struktury aplikacji, dokumentacji, wsparcia społecznościowego oraz testów wydajnościowych. W rezultacie autor podał Angular2 jako lepsze narzędzie dla doświadczonego programisty, a React bardziej przystosowany do potrzeb początkującego użytkownika.

Kolejną analizowaną publikacją jest "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue.js, and Svelte" [4]. Przy wykorzystaniu różnych wzorców DOM wykonano podstawową ocenę każdego szkieletu. Aplikacje testowe zostały skonfigurowane z minimalną liczbą wymaganych elementów zgodnie z odpowiednią dokumentacją każdego szkieletu. Wszystkie cztery aplikacje zawierają tę samą funkcjonalność i podobne pliki: główny plik HTML, zatytułowany index.html i folder src, w którym zawarto odpowiedni kod testu. Test DOM jest wykonywany przy użyciu bezpośrednich aktualizacji DOM i wstawiania różnych elementów HTML, takich jak elementy i <p>. W aplikacji porównawczej każdy rodzaj trwałego lub złożonego przechowywania danych poza stanem lokalnym wykluczono, ponieważ rzutowałoby to na wyniki badania. We wnioskach wskazano React jako najbardziej wydajny szkielet spośród badanych.

W publikacji „Supporting Web Development Decisions by Comparing Three Major JavaScript Framework: Angular, React and Vue.js" [5] szkielety zostały poddane analizie według określonych kryteriów dobranych na potrzeby firm oraz osób, które chcą poznać daną technologię. Wzięto pod uwagę m.in. wydajność każdego szkieletu, szybkość pisania kodu oraz jakość dokumentacji. W rezultacie autor określił, kiedy stosować badane szkielety które najlepiej spełnią postawione w aplikacji wymagania.

W publikacji „Transitioning Angular 2 User Interface (UI) into React" [6] został opracowany interfejs użytkownika w Angularze 2. Następnie został on przeniesiony do React, co w rezultacie pokazało różnice między nimi. Badanie wykazało, że React dominuje w aspektach szablonowości, szybkości w przyswajaniu wiedzy i zarządzaniu pakietami.

W artykule „Analiza porównawcza wydajności frameworków Angular oraz oraz Vue.js" [7] zestawiono dwa tytułowe szkielety i porównano ich wydajność w kontekście tworzenia gier. Wzięto pod uwagę kryteria takie jak: czas wymiany danych z serwerem, renderowanie różnych komponentów aplikacji i ilość zajmowanej pamięci. W badaniu najwydajniejszy okazał się szkielet Vue.

W książce „Mastering Web Application Development with AngularJS" [8] pokazano jak AngularJS wpływa na tworzenie aplikacji a także jak wspomaga tworzenie czystego i łatwego kodu. Pozycja opisuje także jak wykorzystać szablony oparte na szkielecie Angular i czym różni się on od innych szkieletów.

Do napisania aplikacji wykorzystanych w niniejszym badaniu niezbędne były: dokumentacja Vue [9], dokumentacja React [10] oraz dokumentacja Angular [11]. Informacje zawarte w nich wyjaśniły ważne aspekty każdej badanej technologii. Pomogły odpowiednio wykorzystać badane szkielety w stworzeniu aplikacji testowej.

Artykuły „Biblioteki AngularJS i ReactJS – analiza wydajnościowa" [12] oraz „Analiza porównawcza narzędzi do budowania aplikacji SPA AngularJS, ReactJS, Ember.js" [13] zapewniły autorom wstępną wiedzę niezbędną do przeprowadzeniu badań. W pierwszym artykule analizowano szkielety pod względem trudności nauki, dodatkowych narzędzi charakterystycznych dla danego szkieletu, wydajności na urządzeniach stacjonarnych oraz mobilnych. W wyniku analizy została wybrana ta biblioteka, która okazała się najlepsza w kontekście wybranych kryteriów. W drugim artykule badano wydajność AngularJS i ReactJS na różnych

przeglądarkach. Wpłynęło to na wybór Google Chrome do przeprowadzenia badania. Autor pierwszego artykułu podał w jakich sytuacjach lepiej sprawdzą się badane szkielety. Natomiast wynikiem drugiego artykułu było wybranie Reacta jako najbardziej wydajnego oraz posiadającego najwięcej dodatkowych narzędzi. Autorzy zaznaczyli jednak, że warto wziąć pod uwagę fakt, iż jest on trudniejszy dla programistów rozpoczynających naukę.

3. Opis i uzasadnienie badanej technologii

Wybrane technologie są obecnie najbardziej funkcjonalnymi narzędziami do tworzenia aplikacji internetowych. Wszystkie oparte są na JavaScript. Budowa aplikacji na poszczególnych szkieletach, mimo podobieństw, wygląda jednak inaczej.

3.1 Angular

Angular jest szkieletem, który istnieje w dwóch wersjach, powszechnie określanych jako AngularJS i Angular 2+. AngularJS jest starszą wersją opartą na JavaScript która nie jest już rozwijana [10]. Natomiast Angular 2+ jest nowszy i oparty o TypeScript oraz HTML. TypeScript to język programowania typu open source, który został opracowany przez firmę Microsoft. Wykorzystanie go pozwala na statycznie pisanie i kompilowanie do JavaScript. Szkielet Angular 2+ służy do budowania aplikacji klienckich w HTML, CSS i JavaScript lub Typescript. Zapewnia aplikacjom czystą i luźno powiązaną strukturę, która jest łatwa do zrozumienia i prosta w utrzymaniu. Dzięki tej strukturze można między innymi wykorzystywać kod ponownie w różnych aplikacjach.

Aplikacje zbudowane przy użyciu szkieletu Angular składają się z modułów NgModule. Moduły te są kontenerami dla bloku kodu dedykowanego domenie aplikacji. Prosty przepływ pracy i ściśle powiązane zestawy możliwości generują łatwy i zrozumiały kod programistyczny. NgModules zapewniają kontekst kompilacji dla swojej zawartości. Zawierają składniki dostawców usług i inne pliki, co zwiększa jednak redundancję kodu. Funkcjonalności aplikacji bez problemu są eksportowane i importowane do innych modułów.

Jak podaje dokumentacja [7], każda aplikacja Angular ma moduł główny, według konwencji nazwany AppModule. Zapewnia on mechanizm ładowania początkowego, który uruchamia aplikację. Moduł główny może zawierać nieograniczoną hierarchę modułów podrzędnych. Szkielet ten częściowo oparty jest na wzorcu MVC (Model View Controller), mając komponent jako kontroler i szablon jako widok. Komponent kontroluje widok, który reprezentuje obszar ekranu jak również hierarchię widoków złożoną z poszczególnych modułów.

Aby zapewnić jak największą szybkość oraz wydajność aplikacji internetowej, Angular daje kontrolę nad skalowalnością. Pozwala to na spełnienie wymagań dotyczących danych, budując modele danych w RxJS, Imutable.js albo innym modelu push.

3.2 React

React jest biblioteką początkowo rozwijaną przez firmę Facebook jako port JavaScript XHP biblioteki PHP. XHP jest modyfikacją PHP, która pozwalała na tworzenie własnych komponentów. Dokumentacja React [9] podaje, że pierwsza wersja została wydana w maju 2013 roku. Zakres jest mniejszy niż w innych badanych technologiach, ponieważ renderowany jest tylko interfejs użytkownika aplikacji. Korzyścią używania tej technologii jest lekka konstrukcja, mniej kosztowna w nauce oraz użytkowaniu. Rozwój ten może być postrzegany jako bardzo ważny krok w ogólnej zmianie tworzenia aplikacji internetowych.

Największym sukcesem biblioteki React jest to, że zoptymalizowane są funkcjonalności modelu DOM. Model ten jest sposobem reprezentacji dokumentów XML i HTML w postaci modelu obiektowego. Manipulacja DOM jest stosunkowo kosztowna pod względem wykorzystywanych zasobów obliczeniowych. React został zaprojektowany, aby wykonywać jak najmniej manipulacji przy użyciu tego modelu. Zmniejszenie wymagań zużycia w tym szkielecie spowodowane jest zarządzaniem wirtualnym DOM. Podstawy JavaScript zawarte w tym szkielecie przyspieszają cały mechanizm działania aplikacji.

3.3 Vue

Vue można uznać za najnowszy szkielet JavaScript spośród wszystkich analizowanych w tym badaniu. Został stworzony przez firmę Google. W porównaniu do szkieletu Angular, jest bardziej uproszczony i skupiony tylko na widoku. Vue obsługuje DOM podobnie jak React, budując i używając oddzielnego wirtualnego DOM do zarządzania rzeczywistym DOM. Jedną z różnic jest to, że omawiany szkielet nie używa w tym celu oddzielnej modułowej biblioteki. Vue jest często opisywane jako postępowy szkielet, który jest używany do tworzenia interfejsu użytkownika na stronie internetowej. Nie jest ściśle związany z wzorcem MVVM (Model View ViewModel).

Vue może być wykorzystywany do małych projektów, gdzie podstawowa biblioteka używana jest w połączeniu z innymi technologiami jak i dla pełnowymiarowych SPA, zatem główną zaletą Vue jest skalowalność. Nie jest rozwijany przez duże przedsiębiorstwo, lecz przez aktywnych programistów. Kompilacja środowiska wykonawczego Vue jest odpowiedzialna za tworzenie instancji wraz z renderowaniem i zarządzaniem wirtualnym DOM. Istnieje również alternatywna pełna wersja, która zawiera środowisko wykonawcze i kompilator.

Informacje zawarte w dokumentacji [8] podają, że zależności pojedynczych komponentów we Vue są automatycznie śledzone podczas renderowania. System jest w stanie zweryfikować, które komponenty muszą się ponownie renderować, podczas zmiany stanu. To optymalizuje czas i pracę programisty, dzięki czemu może on skupić się na istotnych funkcjonalnościach w tworzeniu aplikacji. Podobnie jak React, jest to szkielet optymalny i szybki w swoim działaniu.

4. Metoda badań

Analiza porównawcza została przeprowadzona z wykorzystaniem kryteriów statycznych i praktycznych. Analiza porównawcza polegała na zestawieniu wyników uzyskanych przez poszczególne szkielety dla określonych kryteriów badawczych. Aby zapewnić wiarygodność wyników wszystkie pomiary zostały powtórzone 100 razy. Parametry sprzętu, na którym utworzono środowisko testowe przedstawiono w Tabeli 1.

Tabela 1: Parametry sprzętu

Parametr	Wartość
Procesor	Intel Core i3-1005G1
Ram	16GB
Dysk	256 GB SSD
Karta graficzna	Intel UHD Graphics
System operacyjny	Windows 10 Home

Statyczne kryteria porównawcze posłużyły do technicznej analizy wybranych technologii. Do kryteriów tych należą:

- wytwarzanie aplikacji testowej,
- szybkość budowy aplikacji lokalnie,
- szybkość uruchomienia aplikacji lokalnie,
- rozwój danego szkieletu.

Testy praktyczne wykonano z zastosowaniem aplikacji testowych uruchomionych lokalnie. Testy wykonano dla siedmiu przypadków:

- wczytywanie 1 000 wierszy,
- odświeżanie 1 000 wierszy,
- wybór pojedynczego wiersza,
- usunięcie 50 wierszy,
- edycja 3 losowych wierszy,
- stworzenie 10 000 wierszy,
- usunięcie 1 000 wierszy.

Dodatkowym kryterium była ilość kodu programistycznego niezbędnego do rozwinięcia aplikacji.

Wszystkie testy rozpoczynają się z 1 000 załadowanych na stronie wierszy, które ładowane są przed każdym pomiarem testowym. Kryteria porównawcze, za każdym razem są odświeżane i wykonywane z tego samego punktu. Dane w aplikacjach uzupełniane są automatycznie, a więc zniwelowany do zera jest błąd ludzki.

Celem aplikacji jest sprawdzenie szybkości działania wybranych szkieletów. Każda z aplikacji została napisana w możliwie najprostszy sposób, jaki został opisany w dokumentacji. Można zatem przybliżyć szybkość działania wszystkich testowych aplikacji. Aplikacje włączane były lokalnie na uprzednio zrestartowanych maszynach, które nie miały połączenia z Internetem. Wszystkie dane wynikowe zostały zapisane do osobnych plików a następnie zaimportowane na potrzeby analizy do formatu .xlsx.

Do przeprowadzenia badań użyto zestawu narzędzi Chrome DevTools. Jest to zestaw narzędzi wbudowanych bezpośrednio w przeglądarkę Google Chrome. Narzędzia te umożliwiają wykonanie pomiarów podczas wczytywania strony internetowej. Google Analytics użyto w celu monitorowania wydajności aplikacji.

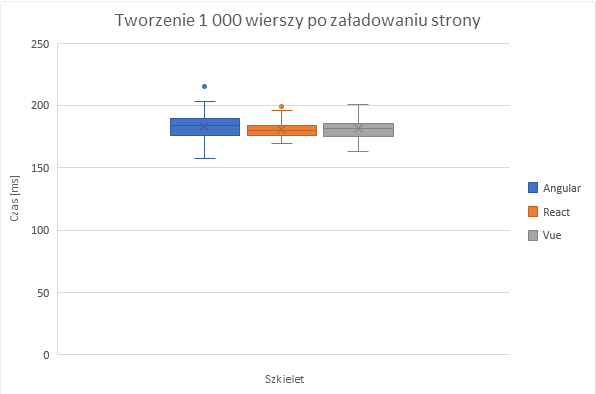
5. Statyczna analiza wyników

Oceną wydajności technologii poza wykonywaniem testów jest opis wsparcia technicznego, które dostarcza zespół twórców danego szkieletu. Najszybciej budującym i uruchamiającym się szkieletem jest Angular. Poprzez najszybsze przetwarzanie procesów związanych ze środowiskiem uruchomieniowym najlepiej radzi również sobie z przeglądarkami internetowymi. Następnym szkieletem jest Vue. Wykorzystywany jest on do tworzenia małych i średnich aplikacji, dlatego korzystanie z niego jest wygodne. Budowa i uruchomienie aplikacji w tym szkielecie jest bardzo zbliżone do Reacta, dlatego nie można jednoznacznie stwierdzić który z tych szkieletów jest szybciej działającym.

Angular jest szkieletem, w którym często tworzone są aktualizacje. Wspierany jest on przez Google, a cykl wydawania nowych wersji trwa 6 miesięcy. Twórcy przykładają bardzo dużą uwagę do aktualizacji tego szkieletu. React rozwija się poprzez komunikację ze społecznością programistów. Twórcy jak i zewnętrzni współtwórcy przesyłają swoje zmiany poprzez platformę GitHub, następnie wszystkie informacje przechodzą przez proces recenzji. Każdy programista może wprowadzić zmianę lub informację o błędzie, które zostaną zaakceptowane przez twórców. Zespół bardzo szybko poprawia krytyczne błędy znalezione w starszych aktualizacjach. Najmłodszym szkieletem spośród wybranych jest Vue. Autorski projekt byłego twórcy Angulara nie posiada dużej społeczności profesjonalistów, którzy wspierają tą technologie. Licencja open source w tym szkielecie ciągle powiększa swoją społeczność.

6. Praktyczna analiza wyników

Wyniki ze wszystkich testów zostały przedstawione na Rysunkach 1-6. Wyniki otrzymane zostały podzielone na 7 kategorii bazując na operacjach CRUD. Do każdego pomiaru stworzone zostały wykresy „skrzynka i wąsy”. Ten typ wykresu zawiera informacje odnośnie do położenia, rozproszenia i kształtu rozkładu danych. Dolna linia łącząca się z prostokątem oznacza minimalną wartość w zbiorze danych, a górna wyznacza maksymalną. Dolny bok oznacza pierwszy kwartyl, czyli środkową wartość między minimum, a medianą, a górny bok oznacza trzeci kwartyl, czyli wartość środkową między medianą, a maksymalną. Linia w środku prostokąta to mediana, a punkt „X” wyznacza średnią zbioru wartości. Pojedyncze punkty wyznaczają wartości odstające.

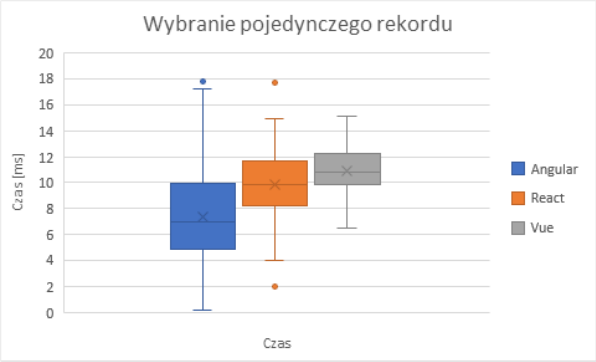


Rysunek 1: Porównanie czasów dla renderowania wierszy.

Czas tworzenia wierszy podczas ładowania strony jest przedstawiony na Rysunku 1. Najszybciej z tym zadaniem radzi sobie React. Stosowane w tej technologii mikroserwisy, najkrócej komunikują się z interfejsem użytkownika.

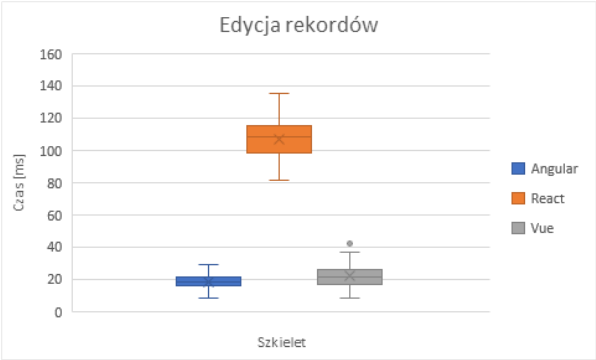
Rysunek 2: Porównanie czasów odświeżania wierszy.

Test odświeżania strony wypadł podobnie dla wszystkich trzech szkieletów. Technologie posiadają niemal identyczne przetwarzanie procesów, dotyczących podglądu załadowanych informacji.



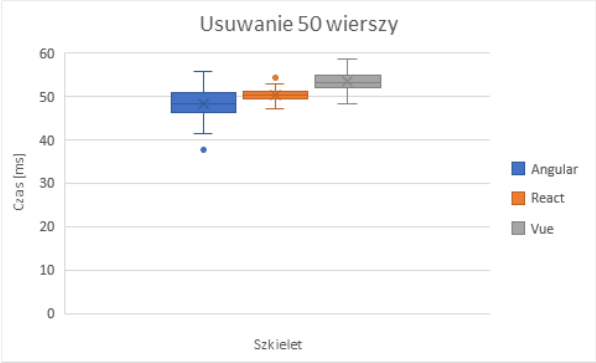
Rysunek 3: Porównanie czasów wyboru pojedynczego wiersza.

Angular najszybciej wykonał test mający na celu wybór wiersza poprzez jego zaznaczenie. Średni wynik tego szkieletu jest aż 30% krótszy od dwóch pozostałych. Komponenty w tej technologii bardzo dobrze się ze sobą komunikują, dzięki temu zaznaczenie danych w bardzo szybkim tempie dociera do interfejsu użytkownika.



Rysunek 4: Porównanie czasów edycji losowego wiersza.

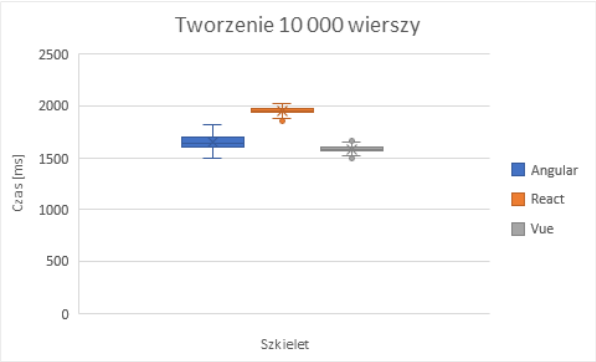
Edycja poszczególnych losowych wierszy jest testem, w którym React okazał się najsłabszy. Dwa pozostałe szkielety mają wynik na podobnym poziomie. Czas wykonania tego testu przez Reacta jest około pięciokrotnie większy niż Angulara i Vue. Zewnętrzne funkcje służące do edycji danych wykonują się zbyt długo, dlatego React nie jest wydajny podczas zmiany danych.



Rysunek 5: Porównanie czasów usunięcia 50 wierszy.

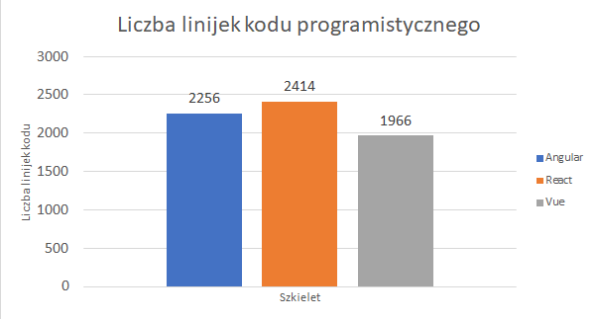
Usuwanie wierszy było przeprowadzone dla 50 i 1000 rekordów. Angular bardzo szybko usuwa małe ilości danych. Częste oczyszczanie danych z małych pakietów jest najlepszym rozwiązaniem w tej technologii. React i Vue są szkieletami, które szybciej usuwają większe pakiety danych. W tych technologiach czas usuwania danych ma rozkład logarytmiczny: im więcej danych jest do usunięcia, tym mniejszy jest czas usuwania jednej jednostki danych.

Rysunek 6: Porównanie czasów usunięcia 1 000 wierszy.



Rysunek 7: Porównanie czasów tworzenia 10 000 wierszy.

Ostatnim wykonywanym testem było utworzenie 10 000 nowych rekordów. W tym zadaniu Vue okazał się najbardziej wydajny, prawdopodobnie dzięki sposobie działania procesów. React jest szkieletem najwolniej uzupełniającym dane, ze względu na korzystanie z zewnętrznej funkcji. Angular zbliżony jest czasem do Vue, dobra komunikacja komponentów sprawia, iż ta technologia radzi sobie z dodawaniem danych.



Rysunek 8: Liczba linijek kodu programistycznego.

Ostatni wykres (Rysunek 8) przedstawia ilość kodu programistycznego niezbędnego do utworzenia aplikacji testowych. Używanie szkieletów JavaScript wpływa na lepszą organizację kodu, co ułatwia programiście zarządzanie aplikacją. Aplikacja wykorzystująca szkielet Vue zawiera najmniejszą liczbę linii kodu programu. Mimo to, różnice między konkurencyjnymi technologiami są niewielkie.

7. Wnioski

Teza badania została potwierdzona – Vue jest wydajniejszym szkieletem niż React i Angular. Analiza wyników wykazała, że najszybszym szkieletem jest Vue. Szybkość działania tej technologii opiera się na małych mikroserwisach, które bezproblemowo komunikują się między sobą. Najlepiej radzi on sobie z dużą ilością danych. Operacje usuwania i edycji danych również wykonują się tu szybko.

Test związany z edycją danych wykazał, że Angular jest drugim najszybciej działającym szkieletem. Edycja i usuwanie małej liczby pakietów danych bardzo dobrze przebiega w tej technologii. Badania pokazują, że dzięki szybkości usuwania małej ilości danych, ciągle można utrzymać aplikacje w czystości i pełnej sprawności.

Najmniej wydajnym szkieletem jest React. Uzyskany czas edycji danych wpływa na to, iż jest on najwolniejszą technologią. Renderowanie strony i usuwanie dużej ilości danych pokazuje jednak, że wewnętrzne funkcje tego szkieletu działają bardzo szybko. Jednak korzystanie z zewnętrznych funkcjonalności spowalnia działanie całej aplikacji.

Podczas przeprowadzonej analizy wykonanej na niewielkich aplikacjach, Vue okazał się być najbardziej wydajnym szkieletem. Jednak nie można uogólnić tego stwierdzenia bez dodatkowych badań. Każdy program stawia inne wymagania, przez co badanie innej aplikacji może przynieść inny rezultat. Aby wyniki badań były bardziej precyzyjne, warto powtórzyć je na większej liczbie próbek. Ponadto warto zaznaczyć, iż do badań została użyta aplikacja wykorzystująca podstawowe funkcjonalności, dlatego rozszerzenie jej o nowe możliwości dałoby więcej wniosków w badaniu.

Podsumowując, wszystkie badane szkielety mają wysoką wydajność. Mimo że Vue osiągnął najlepsze wyniki w badaniu, warto też wziąć pod uwagę inne biblioteki. Warto też być na bieżąco z nowymi publikacjami jakie powstają i śledzić dokumentacje.

Literatura

[1] S. K. Arora, 10 Best JavaScript Frameworks to Use in 2021, https://hackr.io/blog/best-javascript-frameworks, [23.01.2021].

[2] K. Boczkowski, B. Pańczyk, Comparison of the performance of tools for creating a SPA application interface - React and Vue.js, Journal of Computer Science Institute 14 (2020), 73-77, https://doi.org/10.35784/jcsi.1579.

[3] J. Kalinowska, B. Pańczyk, Comparison of tools for creating SPA applications using the examples of Angular 2 and React, Journal of Computer Sciences Institute 10 (2019) 1-4, https://doi.org/10.35784/jcsi.183.

[4] M. Levlin, DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte, Åbo Akademi, 2020.

[5] E. Wohlgethan, Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js. Diss. Hochschule für Angewandte Wissenschaften Hamburg, 2018.

[6] K. Simkhada, Transitioning Angular 2 User Interface (UI) into React, Helsinki Metropolia University of Applied Sciences, [12.04.2017].

[7] R. Baida, M. Andriienko M. Plechawska-Wójcik, Perfomance analysis of frameworks Angular and Vue.js, Journal of Computer Sciences Institute 14 (2020) 59-64, https://doi.org/10.35784/jcsi.1577.

[8] P. Kozlowki, Mastering Web Application Development with AngularJS, [23.08.2013].

[9] Dokumentacja Vue.js https://vuejs.org/v2/guide/, [30.11.2020].

[10] Dokumentacja React https://pl.reactjs.org/docs/getting-started.html, [22.03.2021].

[11] Dokumentacja Angular https://angular.io/docs, [30.11.2020].

[12] R. Nowacki, M. Plechawska-Wójcik, Comparative analysis of tools dedicated to building Single Page Applications – Angular Js, ReactJS, Ember.js, Journal of Computer Sciences Institute 2 (2016) 98-103, https://doi.org/10.35784/jcsi.122.

[13] K. Kowalczyk, M. Plechawska-Wójcik, Angular JS and ReactJS libraries – performance analysis, Journal of Computer Sciences Institute 2 (2016) 114-119, https://doi.org/10.35784/jcsi.126.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 84–88
Received: 8 February 2022
Accepted: 24 March 2022

Comparative analysis of software for smart homes

Analiza porównawcza oprogramowania dla inteligentnych domów

Mateusz Woliński*, Tomasz Szymczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article presents the results of a comparative analysis of smart home software. For this purpose, the domestic market of producers of such solutions was analyzed and selected 17 systems that are among the most popular. A comparative methodology was developed to determine which of these solutions turned out to be the best. The results are presented in tables and graphs.

Keywords: intelligent building; intelligent home systems; communication methods; home automation

Streszczenie

W artykule zaprezentowano wyniki analizy porównawczej oprogramowania inteligentnych domów. W tym celu przeanalizowano krajowy rynek producentów takich rozwiązań i wybrano 17 systemów, które należą do najpopularniejszych. Do określenia które z tych rozwiązań okazało się najlepsze stworzono metodologię porównawczą. Wyniki przedstawiono w tabelach i na wykresach.

Słowa kluczowe: inteligentny budynek; systemy inteligentnego domu; sposoby komunikacji; automatyka domowa

*Corresponding author

Email address: mateusz.wolinski@pollub.edu.pl (M. Woliński)

1. Wstęp

Interakcja między ludźmi a ich otoczeniem może zachodzić na wiele różnych sposobów. Ludzie zazwyczaj większość czynności wykonują w domu i możliwe jest osiągnięcie wielu korzyści, jeżeli to środowisko potrafi interaktywnie reagować na ich zachowania i gesty. Inteligentny dom to inteligentne otoczenie, które jest w stanie działać odpowiednio do zachowania jego mieszkańców.

Koncepcja inteligentnych domów jest rozwijana od lat 90 [1]. Według badaczy tematu, dom inteligentny to dom, który jest w stanie pomagać mieszkańcom żyć niezależnie i komfortowo za pomocą nowych technologii [2]. W domu inteligentnym wszystkie urządzenia mechaniczne i cyfrowe są ze sobą połączone tak, by stworzyć sieć, gdzie mogą się między sobą komunikować oraz stworzyć z użytkownikiem interaktywną przestrzeń.

Inteligentne domy to coś więcej niż tylko innowacja techniczna. Inteligentne domy to pierwsza konieczność dla nowoczesnego człowieka. Jako że w mieszkaniach spędzamy znaczną część naszego życia, powinny one jak najbardziej przyczyniać się do utrzymania naszego zdrowia i dobrego stanu. W szczególności powinno to obejmować zachowanie zdrowia psychicznego, bowiem w obecnej epoce cyfrowej każdy odczuwa presję przepływu informacji, która może prowadzić do dyskomfortu, stresu, a nawet chorób. Przetworzenie tych informacji i wybór tych najbardziej potrzebnych i interesujących - to zadanie systemów inteligentnego domu.

2. Badane systemy

Spośród szerokiej gamy systemów automatyki budynkowej, oferowanych przez różnych producentów na polskim rynku wybrano 17, które zostały pokrótce scharakteryzowane i poddane dalszej analizie. Badane systemy to:

1. Fibaro - bezprzewodowy system automatyki domowej oparty na technologii radiowej. Stwarza to możliwość uinteligentniania istniejących już tradycyjnych instalacji elektrycznych bez działań inwazyjnych takich jak remont. System Fibaro współpracuje również z systemami sterowania głosowego Google Assistant, Amazon Alexa, Apple Siri. System został wykonany w technologii Z-Wave, która stanowi wiodącą metodę w zakresie bezprzewodowej automatyki domowej. Najnowsza centrala Home Center 3 jest w stanie także obsługiwać łączność WiFi. Na rynku dostępny jest szeroki zakres urządzeń od różnych producentów, które można ze sobą łączyć w jeden system, gdyż są wzajemnie kompatybilne. System może funkcjonować bez dostępu do internetu, jednak taki dostęp jest potrzebny, gdy użytkownik jest poza zasięgiem własnej sieci Z-Wave [3].
2. Grenton - to polski system automatyki budynkowej zaprojektowany dla wszelkiego rodzaju budynków. Jest to system hybrydowy, obsługujący zarówno łączność bezprzewodową jak i kablową, więc ma zastosowanie w nowo budowanych jak i w istniejących budynkach. Komunikacja między CLU odbywa się przez magistralę systemową opartą na standardzie Ethernet. Zapewniają także łączność

z modułami IOM (Input/Output Module), które realizują funkcje wejść i wyjść. Komunikacja za pomocą magistrali lokalnej może odbywać się przewodowo (TF-Bus) lub bezprzewodowo, wykorzystując standard Z-Wave [4].

3. F&Home - kablowy system inteligentnego domu przeznaczony głównie do obiektów dopiero się budujących lub w trakcie generalnego remontu. Struktura instalacji w systemie F&Home ma schemat gwiazdy, co oznacza, że wszystkie instalacje kablowe schodzą się w jednym, centralnym punkcie [5].
4. F-Home Radio - F&Home RADIO to mieszany system automatyki budynkowej przeznaczony głównie dla domów jednorodzinnych i mieszkań. Rozwiązania F&Home należą do jednych z najtańszych na rynku, a modułowość systemu daje możliwość rozłożenia rozbudowy instalacji w czasie. Największą zaletą systemu F&Home RADIO jest dowolne programowanie. Narzędzie to pozwala na swobodną konfigurację oraz tworzenie funkcji logicznych by jak najlepiej dopasować się do personalnych potrzeb użytkujących. Jest to system otwarty [6].
5. DEIMIC - polski system inteligentnej automatyki domowej. Deimic One jest systemem uniwersalnym i może sterować automatyką budynkową, elementami wyposażenia oraz zarządzać instalacjami w praktycznie każdym obiekcie czy przestrzeni. Jednostka sterująca to Deimic One Master, opierająca się na systemie Linux i DEIMIC Core. Deimic to system kablowy działający w topologii gwiazdy z jednostką centralną, do której podłączone są wszystkie peryferia. Systemem można zarządzać za pomocą urządzeń mobilnych z zainstalowaną dedykowaną aplikacją, przez przeglądarkę lub wykorzystując standardowe przełączniki [7].
6. Teletask - został stworzony w latach 70. w Belgii i ciągle jest rozwijany. Stosowany jest obecnie w ponad 30 krajach. Jest to przewodowy system posiadający własną, modułową architekturę, polegającą na tworzeniu sieci interfejsów z różnych czujników, przekaźników, silników itp. Połączone są one za pomocą magistrali kablowej AUTOBUS, wykorzystującą do transmisji danych skrętkę. Łączy ona wszystkie elementy systemu z głównym modułem, którym jest jednostka centralna. Teletask ma możliwość kooperacji także z innymi producentami [8].
7. Exta Free - system bezprzewodowego sterowania to proste i elastyczne rozwiązanie sterowania radiowego. Exta Free znajduje zastosowanie w sterowaniu oświetleniem, roletami, napędami bram. Odbiorniki systemu pracują w trybie: monostabilnym, bistabilnym oraz czasowym. Użytkownik sam wybiera w jaki sposób ma działać urządzenie. System jest więc w pełni uniwersalny. Komunikacja odbywać się może nie tylko za pomocą nadajników radiowych, ale także za pomocą smartfonów czy tabletów [9].
8. MyHome - umożliwia realizację projektów o różnej wielkości – od automatyki pojedynczych funkcji w domu takich jak oświetlenie czy rolety, po wielofunkcyjne i zintegrowane systemy. Do zasilania urządzeń i przesyłania danych między nimi wykorzystywana jest dwuprzewodowa magistrala SCS BUS. Uzupełnieniem systemu magistralnego są rozwiązania radiowe, wykorzystujące w komunikacji protokół ZigBee. Elementy wykonawcze dostępne są w dwóch wersjach – modułowej, która służy do montażu w rozdzielnicach oraz jako elementy osprzętu, które montowane są podtynkowo. Takie rozwiązanie umożliwia funkcjonowanie systemu o budowie scentralizowanej lub peryferyjnej, czyli rozproszonej, zależnie od rodzaju obiektu oraz indywidualnych potrzeb [10].
9. KNX - opiera się na różnych mediach transmisyjnych. Najczęściej stosowanym jest jednoparowa skrętka (twisted pair), rzadziej przewody zasilające (power line) i fale radiowe. Czasami wykorzystywany jest także protokół internetowy. System KNX posiada najbardziej szerokie spektrum możliwości sterowania domem. Instalacja magistralna KNX jest trasowana dedykowanymi przewodami telekomunikacyjnymi. Jej podstawowym elementem jest linia łącząca wszystkie magistralne urządzenia. Każda z linii ma własne zasilanie. Projekt instalacji może mieć topologię dowolnego drzewa, łańcucha czy gwiazdy. Możliwe jest także łączenie i dowolne rozgałęzianie sieci. Jedynym warunkiem jest nietworzenie pętli. System KNX ma strukturę rozproszoną, więc nie wymaga jednostki centralnej. Każde urządzenie magistralne w instalacji posiada własny procesor niezbędny do niezależnej pracy. Dzięki temu w przypadku uszkodzenia jednego z urządzeń, wszystkie pozostałe działają normalnie. System KNX jest standardem wspieranym przez ponad 400 niezależnych producentów [11].
10. LCN - niemiecki system sterowania domem, powstały w roku 1992. System LCN jest systemem magistralnym, gdzie magistralą jest dodatkowy przewód w instalacji elektrycznej. W instalacji jednofazowej będzie to czwarty przewód, a w trójfazowej szósty. Przekrój tego przewodu jest taki sam jak pozostałych w instalacji. Wraz z przewodem neutralnym pełniącym rolę żyły powrotnej, tworzą obwód magistralny. W systemie nie występuje typowy podział urządzeń magistralnych na aktory i sensory. Urządzenia mają znacznie większą autonomię i mogą realizować jednakowo funkcje wejścia i wyjścia. Urządzenie magistralne posiada własny mikroprocesor i pamięć, w której przechowywana jest jego konfiguracja [12].
11. Loxone - jest systemem automatyki budynkowej powstałym w Austrii. Jest to system o architekturze scentralizowanej, gdzie jednostka centralna – Loxone Miniserver jest jednocześnie serwerem wizualizacji. Przetwarza on wszystkie sygnały przychodzące i wychodzące z systemu. System Loxone jest systemem hybrydowym. Łączność może być zarówno

przewodowa, oparta na dedykowanym protokole Loxone Tree, jak również bezprzewodowa, wykorzystująca także autorski system Loxone Air. Dlatego system może być wdrażany zarówno w nowobudowanych budynkach, jak i już istniejących. Loxone jest systemem zamkniętym, jednak dzięki dedykowanym modułom istnieje możliwość integracji z urządzeniami w standardzie KNX lub np. urządzeniami alarmowymi [13].

12. Nexo - polski system automatyki budynkowej produkowany przez firmę Nexwell Engineering. Jest to system scentralizowany logicznie i fizycznie. Komunikacja między składowymi systemu zachodzi przewodowo, poprzez magistralę Tukan. W przypadku uszkodzenia jednostki centralnej, urządzenia funkcjonują w zakresie podstawowym bez wsparcia jednostki głównej. Możliwe jest wtedy sterowanie oświetleniem i roletami z poziomu tradycyjnych przełączników. System dzieli się na trzy warianty: podstawowy, rozszerzony i ekskluzywny. Nexo posiada swój własny system alarmowy, w pełni połączony z automatyką budynku, dostępny już w podstawowym wariancie [14].
13. Ampio - jest polskim systemem automatyki budynkowej. Jest to system o rozproszonej logice, ale fizycznie hybrydowy. Elementy mogą być mocowane na szynie DIN w rozdzielnicy lub w puszkach elektrycznych. Komunikacja odbywa się przewodowo, wykorzystując magistralę CAN firmy Bosch, jednak ma możliwość pracy bezprzewodowej. System posiada moduł redundancji zasilania magistrali, który praktycznie eliminuje awarie jednostki centralnej. Urządzenia mają własne mikroprocesory z zapisaną konfiguracją. Uszkodzenie jednego urządzenia wyklucza tylko odbiory/sensory podłączone do niego. System Ampio znajduje zastosowanie we wszystkich rodzajach budownictwa [15].
14. Domatiq - jest rozwiązaniem systemowym, co oznacza, że jest zintegrowany z instalacją elektryczną. Działanie systemu realizowane jest w topologii multi-master, w której komunikacja pomiędzy poszczególnymi modułami zachodzi za pomocą magistrali CAN. Wdrażanie systemu odbywa się poprzez rozprowadzenie dedykowanej instalacji elektrycznej i zintegrowanie jej w tablicy rozdzielczej obiektu. Takie rozwiązanie jest gwarantem stabilności oraz znacznie zmniejsza awaryjność i konieczność częstego serwisu elementów [16].
15. Satel - Centrale alarmowe Integra firmy Satel są bardzo dobrym przykładem rozwiązania dającego możliwość połączenia systemu alarmowego oraz systemów automatyki budynkowej. Oprócz korzystania z klasycznych funkcji centrali alarmowej, możliwe jest stworzenie rozbudowanego systemu inteligentnego budynku. Producent daje do dyspozycji moduł INT-KNX-2, co pozwala rozbudowywać system o elementy działające w europejskim standardzie KNX. Efektem tego może być stworzenie systemu o bardzo dużej funkcjonalności i wygodzie użytkowania oraz pełnej kompatybilności z innymi podzespołami pracującymi w tym standardzie. Dodatkowe moduły rozszerzeń oraz komunikacji umożliwiają współpracę centrali z innymi systemami automatyki [17].
16. xComfort - system bezprzewodowy, komunikujący się w częstotliwości 868,3 MHz. Transmisja odbywa się dwukierunkowo, czyli urządzenie, które otrzymało polecenie potwierdza jego odebranie i wysyła o tym sygnał do urządzenia nadawczego. System działa w oparciu o architekturę rozproszoną, więc uszkodzenie centralnego sterownika nie ma wpływu na pracę systemu. Centrala systemu xComfort pozwala na sterowanie maksymalnie 99 – cioma urządzeniami. Umożliwia ona zdalne sterowanie oraz wizualizację działania systemu z poziomu aplikacji smartfona, tabletu oraz przeglądarki www [18].
17. Ferguson - kolejny polski system automatyki domowej. Cechami systemu są wszechstronność i modułowa budowa umożliwiająca dowolne konfigurowanie elementów, zależnie od potrzeb użytkowników. Najmocniejszą cechą systemu jest jego podsystem alarmowy. Komunikacja między urządzeniami odbywa się za pomocą protokołu ZigBee lub WiFi, natomiast całością zarządzać można z poziomu aplikacji na urządzenia mobilne lub przeglądarki [19].

3. Analiza porównawcza

Celem analizy porównawczej jest zdefiniowanie, który system posiada najlepsze parametry techniczne oraz największą funkcjonalność.

3.1. Kryteria analizy

Na potrzeby pracy została opracowana metodyka porównawcza. Zaproponowano przydział punktowy w trzech wariantach, ze względu na istotność danej właściwości: 1 – 3, 1 – 5, 1 – 10 oraz wyjątkowo 1 – 20 w ocenie kosztów systemu. System z największą sumą punktów został uznany za najlepszy. Wszystkie 68 cech branych pod uwagę zostało podzielonych i pogrupowane w 9 poszczególnych kategorii:

1. Funkcje i cechy ogólne,
2. Systemy HVAC (*heating, ventilation, air conditioning),*
3. Bezpieczeństwo,
4. Zarządzanie multimediami,
5. Kontrola urządzeń,
6. Sterowanie roletami,
7. Monitoring mediów,
8. Pozostałe cechy,
9. Uśredniony koszt systemu.

Dla przykładu do kategorii „Bezpieczeństwo” zostało przypisanych 16 funkcji systemów inteligentnych związanych z ochroną zdrowia i mienia domowników:

1. powiadamianie o sytuacjach alarmowych wybranych osób i służb,

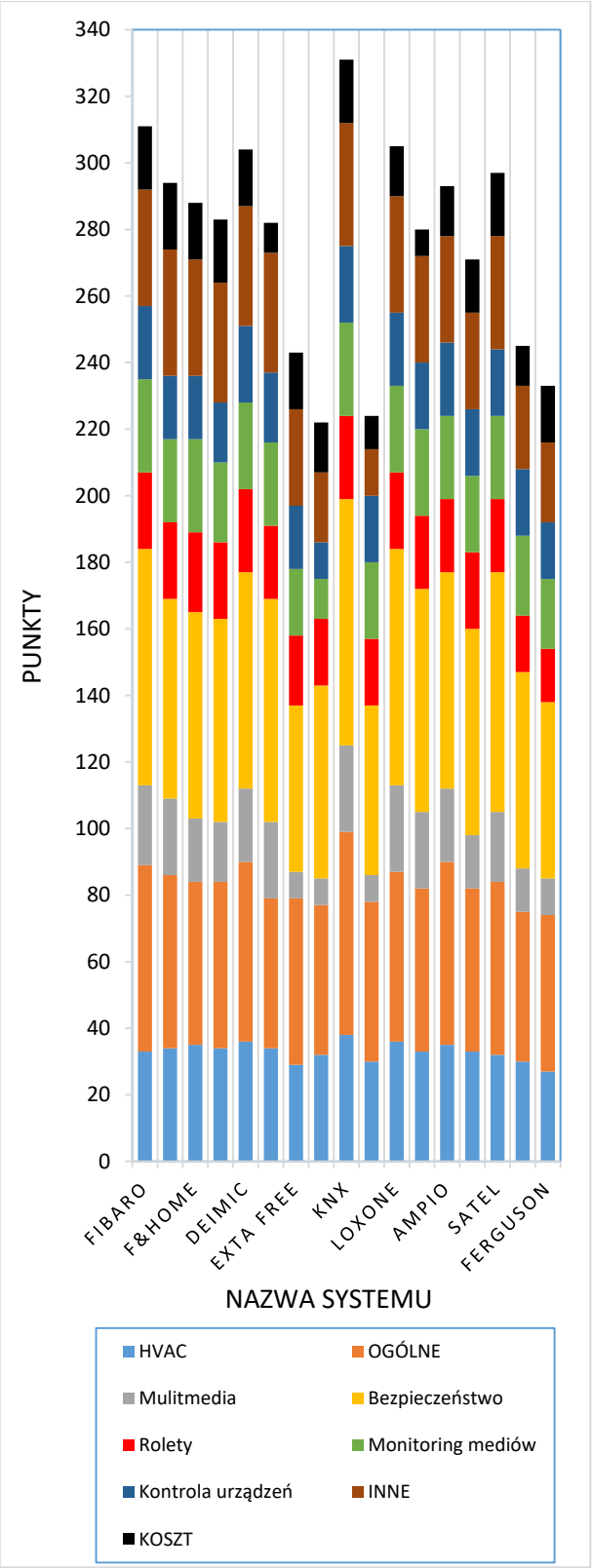
2. automatyczne odcinanie mediów w wypadku wykrycia zalania lub wycieku gazu,
3. symulacja obecności podczas pobytu poza domem,
4. automatyczne otwieranie rolet i bram w przypadku alarmu przeciwpożarowego,
5. informowanie lub automatyczne zamykanie otwartych okien,
6. przełączanie systemu wentylacji w wypadku alarmu pożarowego,
7. automatyczne zadziałanie systemu wentylacji i otwierania okien w przypadku wykrycia wycieku gazu lub tlenku węgla,
8. w przypadku alarmu miganie oświetlenia celem zwrócenia uwagi,
9. automatyczne rozbrajanie alarmu dla domowników nadchodzących od nieuzbrojonej części z wewnątrz domu,
10. własny podsystem alarmowy,
11. integracja z niezależnym alarmem,
12. możliwość sterowania kamerami obrotowymi poprzez smartfony i tablety z systemem Android, iOS lub przeglądarkę,
13. podgląd aktualnego obrazu z kamer,
14. programowanie czasowe (praca monitoringu w konkretnych godzinach, np. podczas pobytu w pracy),
15. tworzenie wirtualnych kluczy do domu,
16. powiadomienia i podgląd przez domofon.

W ocenie poszczególnych funkcji systemów inteligentnych domów w większości przyjęto zakres punktowy od 1 do 5, gdzie:

1 – oznacza brak danej cechy,
2 – dana zdolność występuje, jednak jest na podstawowym poziomie,
3 – zdolność jest na przeciętnym poziomie i pozwala użytkownikowi na więcej niż podstawowe możliwości,
4 – dana cecha jest bardzo dobrze rozwinięta, jednak do maksymalnej oceny brakuje szczegółów, które wyróżniają inne systemy,
5 – zdolność jest na bardzo wysokim poziomie i pozwala użytkownikom w pełni wykorzystywać daną funkcję systemu.

Analogicznie podział wygląda w pozostałych punktacjach (1 – 3, 1 – 10), tj. im więcej punktów, tym dany system lepiej wypada w danej kategorii.

Tabela 1 przedstawia sumaryczne zestawienie punktacji poszczególnych cech systemów smart home. Kolorem zielonym zaznaczono systemy z największą ilością punktów w danej kategorii. Całość zwizualizowano na wykresie: Rysunek 1.



Rysunek 1: Graficzne podsumowanie punktacji poszczególnych systemów.

Tabela 1: Zestawienie sum punktacji systemów smart home w poszczególnych kategoriach

	Ogólne	HV	Bezpi	Multi	Komf	Rolet	Mon	Inne	Koszt
Fibaro	56	33	71	24	22	23	28	35	19
Grenton	52	34	60	23	19	23	25	38	20
F&Home	49	35	62	19	19	24	28	35	17
F&Home Radio	50	34	61	18	18	23	24	36	19
DEIMIC	54	36	65	22	23	25	26	36	17
Teletask	45	34	67	23	21	22	25	36	9
Exta Free	50	29	50	8	19	21	20	29	17
MyHome	45	32	58	8	11	20	12	21	15
KNX	61	38	74	26	23	25	28	37	19
LCN	48	30	51	8	20	20	23	14	10
Loxone	51	36	71	26	22	23	26	35	15
Nexo	49	33	67	23	20	22	26	32	8
Ampio	55	35	65	22	22	22	25	32	15
Domatiq	49	33	62	16	20	23	23	29	16
Satel	52	32	72	21	20	22	25	34	19
xComfort	45	30	59	13	20	17	24	25	12
Ferguson	47	27	53	11	17	16	21	24	17

4. Wnioski

Automatyka domowa i systemy inteligentnego domu to wciąż rozwijające się dziedziny. Choć idea Smart Home ma już ponad pół wieku, to współczesne systemy ciężko nazwać inteligentnymi. Znaczna większość z nich działa na zasadzie akcji i reakcji. Wszelkiego rodzaju sensory zbierają informacje aktywując elementy wykonawcze według zapisanych wcześniej instrukcji i harmonogramów. Porównanie systemów inteligentnego domu dostępnych na rodzimym rynku pokazało, że obecnie nie ma idealnych rozwiązań. Producenci oferują bardzo konkurencyjną gamę rozwiązań, w stosunku jakość – cena. Głównymi kryteriami przy wyborze systemu automatyki staje się przede wszystkim sposób komunikacji elementów systemu. Zależne jest to od rodzaju budynku i jego instalacji oraz konieczności poważniejszych remontów. Według powyższego porównania, stabilność i pewność działania oraz bezpieczeństwo przesyłanych informacji gwarantuje przewodowy system KNX. Jest on także jednym z najbardziej rozbudowanych systemów dzięki swojej otwartości. W przypadku systemów bezprzewodowych, które dają możliwość nieinwazyjnej i łatwej instalacji oraz możliwości wdrożenia do dowolnego rodzaju obiektu, najlepiej wypadają firmy Fibaro, DEIMIC, Loxone i Ampio. Wszystko jednak zależy od klienta i opiera się na jego potrzebach, wizji oraz budżecie.

Literatura

[1] M. Chan, D. Estève, C. Escriba, E. Campo, A review of smart homes-present state and future challenges, Comput. Methods Progr. Biomed. 91 (2008) 55–59.

[2] J. Damico, Smart Home: The Ultimate Guide For Beginners, Independently Published, 2020.

[3] Opis system Fibaro, https://www.fibaro.com/pl/, [12.06.2020].

[4] Opis systemu Grenton, https://www.grenton.pl/funkcje-inteligentny-dom.html , [12.06.2020].

[5] Opis systemu F&Home, https://www.fhome.pl/pl/system-przewodowy.html , [13.06.2020].

[6] Opis systemu F&Home Radio, https://www.fhome.pl/pl/system-przewodowy/katalog-przewodowy/mh-khmh-kf.html , [13.06.2020].

[7] Opis systemu DEIMIC, https://ep.com.pl/rynek/prezentacje/11231-system-automatyki-budynkowejdeimic-one , [01.05.2020].

[8] Opis systemu Teletask, https://teletask.be/en/solutions/end-user/ , [16.06.2020].

[9] Opis systemu Exta Free, https://www.zamel.com/pl-PL/produkty/exta-free-sterowanie-bezprzewodowe, [16.06.2020].

[10] Opis systemu MyHome, http://www.automatykadomowa.legrand.pl/pl/opis_myhome.html , [17.06.2020].

[11] Opis systemu KNX, http://knxtoday.com/2019/07/13941/the-new-knx-tp1-256-topology-more-devices-and-fewer-line-repeaters.html , [25.06.2020].

[12] Opis systemu LCN, https://lcnpolska.pl/system/, [25.06.2020].

[13] Opis systemu Loxone, https://www.loxone.com/plpl/smart-home/ , [25.06.2020].

[14] Opis systemu Nexo, http://nexwell.eu/smart-home/, [26.06.2020].

[15] Opis systemu Ampio, https://ampio.pl/dokumenty-strefy/dokumentacje/ampio_folder_2020.pdf, [19.06.2020].

[16] Opis systemy Domatiq, https://domatiq.pl/pl/smarthome , [10.05.2020].

[17] Opis systemu Satel, https://www.dobrzemieszkaj.pl/najlepsze_domy/technologie/334/inteligentny_dom_system_otwarty_czy_zamkniety,226695.html , [03.07.2020].

[18] Opis systemu xComfort, https://www.moeller.pl/Documentation/Katalogi/Katalog_xComfort_2011.pdf , [21.06.2020].

[19] Opis systemu Ferguson, https://ferguson-digital.eu/download/smarthome/FS1SH_Smart_Hub/manuals/Smart_Hub_user_manual_PL_v1.3.pdf , [22.06.2020].

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 89–96
Received: 8 February 2022
Accepted: 5 March 2022

Comparative analysis of message brokers

Analiza porównawcza usług strumieniowego przesyłania danych

Mateusz Kaczor*, Paweł Powroźnik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article presents a comparative analysis of the two most popular message brokers: Apache Kafka and RabbitMQ. The purpose of this paper was to perform a comparative analysis of selected technologies and to determine their time efficiency. For the needs of the research four applications were prepared (two for each tested technology) that were sending and receiving messages. The research was supplemented with tests with the use of auxiliary tools and theoretical comparison. The comparative analysis of gathered data allowed us to determine the most effective technology, which happened to be Apache Kafka.

Keywords: message broker; Apache Kafka; RabbitMQ

Streszczenie

W pracy przeprowadzono analizę porównawczą dwóch najpopularniejszych usług strumieniowego przesyłania danych: Apache Kafka oraz RabbitMQ. Celem było wykonanie analizy porównawczej wybranych technologii oraz określenia ich wydajności czasowej. Do badań wykorzystano cztery aplikacje (po dwie dla każdej badanej technologii) przesyłające oraz odbierające wiadomości. Badania uzupełniono testami z użyciem pomocniczych narzędzi oraz teoretycznym porównaniem. Analiza porównawcza uzyskanych wyników pozwoliła wyłonić wydajniejsze rozwiązanie, którym jest Apache Kafka.

Słowa kluczowe: broker wiadomości; Apache Kafka; RabbitMQ

*Corresponding author

Email address: mateusz.kaczor@pollub.edu.pl (M. Kaczor)

1. Wstęp

Usługi strumieniowego przesyłania danych służą do komunikacji pomiędzy aplikacjami. Różnica w porównaniu do tradycyjnego podejścia wymiany danych typu punkt-punkt polega na tym, że istnieje broker pomiędzy komunikującymi się obiektami, który odpowiada za walidację, przechowywanie, trasowanie oraz dostarczanie wiadomości do konkretnych odbiorców. Wykorzystanie usług strumieniowego przesyłania danych umożliwia luźne powiązanie i łączenie ze sobą aplikacji używających różnych języków oraz protokołów.

Kolejki wiadomości umożliwiają asynchroniczną komunikację między aplikacjami. Oznacza to, że systemy mogą się ze sobą komunikować w sposób nieblokujący - encje wysyłające oraz odbierające komunikaty nie są od siebie zależne. Ten sposób wymiany danych jest najbardziej pożądany w systemach rozproszonych. Asynchroniczna komunikacja zapobiega utracie danych i umożliwia działanie systemów nawet w przypadku problemów z połączeniem lub opóźnieniem sieci. Ten sposób komunikacji gwarantuje, że wiadomość będzie dostarczona raz (i tylko raz) we właściwej kolejności względem innych komunikatów.

1.1. Cel i zakres badań

Celem badań było porównanie dwóch usług strumieniowego przesyłania danych: Apache Kafka oraz RabbitMQ. Badaniom poddane zostaną aspekty usług strumieniowego przesyłania danych takie jak architektura, przypadki użycia, unikalne cechy oraz wydajność. Eksperyment wydajnościowy przeprowadzono przy pomocy aplikacji wysyłających oraz odbierających wiadomości, oraz pomocniczych narzędzi.

Zakres badań objął utworzenie czterech aplikacji (dwóch wysyłających wiadomości oraz dwóch odbierających wiadomości – dla obu badanych technologii) w celu zbadania wydajności Apache Kafka oraz RabbitMQ. Dodatkowo przeprowadzono badania literaturowe dotyczące usług strumieniowego przesyłania danych. Badania wydajnościowe zostały uzupełnione teoretycznym wprowadzeniem do porównywanych technologii, wykorzystywanej architektury systemów, unikalnych cech oraz różnic badanych technologii.

1.2. Przegląd literatury

W pracy dokonano [1] porównania technologii Apache Kafka oraz RabbitMQ. Autorzy przedstawili podobieństwa pomiędzy badanymi technologiami. Dodatkowo wyróżniono unikalne funkcjonalności dostarczone przez wybrane brokery. Po wstępnej prezentacji usług strumieniowego przesyłania danych porównane zostały wybrane cechy badanych technologii. W dalszej części pracy autorzy przeprowadzili szereg eksperymentów w celu zbadania wydajności/efektywności Apache Kafka oraz RabbitMQ. Do przeprowadzenia testów wykorzystano narzędzia, które dostarczone są przez owe technologie. Wszystkie testy zostały uruchomione na 60 sekund, a zbieranie rezultatów zaczynało się po 30 sekundach. Wyniki badań opóźnień pokazują, że obie

badane technologie dostarczają wiadomości z niewielką zwłoką. Wyniki badań wydajności pokazują, że Apache Kafka dostarcza dane nieznacznie szybciej niż RabbitMQ. Autorzy dodają jednak, że na wydajność obu technologii może wpływać wiele czynników m.in. konfiguracyjnych.

W artykule [2] dokonano porównania trzech usług strumieniowego przesyłania danych: Apache Kafka, RabbitMQ oraz NATS streaming. W pracy przedstawiono główne założenia brokerów wiadomości, scharakteryzowany został wzorzec publikowania–subskrypcji oraz opisane zostały badane technologie. W dalszej części artykułu porównano badane brokery w oparciu o najważniejsze aspekty, pozwalające na wybór odpowiedniego narzędzia w zależności od potrzeb klienta. W podsumowaniu stwierdzono, że Apache Kafka jest narzędziem preferowanym dla systemów rozproszonych do analizy danych w czasie rzeczywistym ze względu na wysoką przepustowość oraz niskie opóźnienie. RabbitMQ jest natomiast technologią preferowaną dla „Internetu rzeczy" z powodu możliwości obsłużenia skomplikowanego trasowania (ang. routing).

W artykule [3] szczegółowo opisano brokera wiadomości Apache Kafka. W początkowej części pracy opisano architekturę oraz główne założenia projektowe badanej technologii. Przedstawiono rozwiązania pozwalające na osiągnięcie wysokiej efektywności badanego narzędzia. Po opisaniu architektury Apache Kafka przeprowadzono testy wydajnościowe, porównując je z RabbitMQ oraz ActiveMQ. Eksperyment polegał na wysłaniu i odebraniu 10 milionów wiadomości o rozmiarze 200 bajtów. Wydajność określono na podstawie czasu, jaki każdy badany broker potrzebował na przetworzenie wszystkich wiadomości. Na podstawie uzyskanych wyników stwierdzono, że Apache Kafka jest wydajniejsza niż RabbitMQ oraz ActiveMQ.

W pracy [4] opisano usługę strumieniowego przesyłania danych RabbitMQ. Głównym celem artykułu była analiza skalowalności architektury badanej technologii. Eksperyment przeprowadzono dla wiadomości o rozmiarze 512 bajtów oraz 2 kilobajtów w konfiguracjach:

- pojedynczy producent, pojedynczy subskrybent,
- wielu producentów, pojedynczy subskrybent,
- pojedynczy producent, wielu subskrybentów,
- wielu producentów, wielu subskrybentów.

Na podstawie uzyskanych wyników stwierdzono, że RabbitMQ jest narzędziem, które potencjalnie może być użyte jako mechanizm do komunikacji w systemach rozproszonych. W kontekście badań wydajnościowych uzyskano niejednoznaczne wyniki. Opisano, że wraz ze zwiększeniem liczby „gałęzi" (ang. node) w klastrze RabbitMQ, szybkość przesyłania wiadomości zmalała.

W artykule [5] dokonano analizy porównawczej dwóch najpopularniejszych protokołów dla brokerów wiadomości: Apache Kafka oraz AMQP. W początkowej części pracy scharakteryzowano architekturę oraz główne założenia projektowe badanych technologii. W dalszej części artykułu opisany został przebieg eksperymentu wydajnościowego dla Apache Kafka oraz implementacji protokołu AMQP - RabbitMQ. Test polegał na pomiarze przepustowości i opóźnienia dla dwóch konfiguracji: pojedynczy producent, pojedynczy subskrybent oraz wielu producentów, wielu subskrybentów. Testowa wiadomość o rozmiarze 50B została przesłana milion razy. Na podstawie uzyskanych wyników stwierdzono, że Apache Kafka jest wydajniejsza i jest lepszym wyborem w aplikacjach, gdzie niezawodność nie jest cechą krytyczną. RabbitMQ natomiast jest brokerem odpowiednim dla aplikacji, gdzie nie można pozwolić na utratę wiadomości (przykładowo transakcje finansowe). Dodatkową zaletą protokołu AMQP jest możliwość szyfrowania wiadomości.

2. Przedmiot badań

Przedmiotem badań było porównanie wydajności dwóch najpopularniejszych protokołów strumieniowego przesyłania danych: Apache Kafka oraz RabbitMQ. Skupiono się na takich parametrach jak: architektura, skalowalność czy szybkość działania aplikacji.

2.1. Apache Kafka

Apache Kafka jest usługą strumieniowego przesyłania danych początkowo utworzoną przez LinkedIn. Pod koniec 2010 została wydana jak o oprogramowanie otwartoźdódłowe, a w lipcu 2011 dołączyła do rodziny projektów Apache [6]. Głównym celem projektowym Kafki było uzyskanie możliwości przetwarzania dużej liczby wiadomości w niezawodny sposób oraz możliwości skalowania. Obecnie Apache Kafka jest najpopularniejszą, otwartoźdódłową usługą strumieniowego przesyłania danych i jest używana przez 60% firm z listy Fortune 100 [7]. Należą do nich m.in.: Goldman Sachs, Target, Cisco, Airbnb i wiele więcej.

2.2. RabbitMQ

RabbitMQ jest usługą strumieniowego przesyłania danych opartą na protokole AMQP (Advanced Message Queuing Protocol). Standard ten utworzony został w 2003 roku przez J.P. Morgan w odpowiedzi na brak wzorca umożliwiającego łączenie systemów informatycznych w bankach [8]. RabbitMQ został opracowany w 2007 roku i jest aktualnie utrzymywany przez Pivotal. Omawiany broker jest jedną z najpopularniejszych usług strumieniowego przesyłania danych i jest używany m.in.: przez T-Mobile, Runtastic oraz wiele innych [9].

3. Porównanie technologii Apache Kafka oraz RabbitMQ

W przeprowadzonych badaniach dokonano porównania następujących parametrów testowanych protokołów: architektura, długość życia i kolejkowanie wiadomości, logika trasowania czy skalowalność. Dokonano również porównania unikalnych funkcjonalności dla każdego z brokerów.

3.1. Architektura

Jednostkę danych w Kafce nazywa się komunikatem. Wiadomość jest tablicą bajtów, dlatego dane w niej

zawarte nie mają określonego formatu ani znaczenia dla omawianej kolejki. Komunikat w Kafce można porównać do wiersza/rekordu bazodanowego.

W celu uzyskania wysokiej wydajności, wiadomości są grupowane w partie. Partia jest kolekcją danych, z których wszystkie są zapisywane na ten sam temat i tę samą partycję. Przesyłanie pojedynczej wiadomości byłoby mało efektywne, dlatego grupowanie ich znacznie usprawnia działanie kolejki.

Komunikaty w Kafce są zapisywane do tematów. Temat można porównać do tabeli bazodanowej. Dodatkowo są one podzielone na partycje. Wiadomości są dopisywane i odczytywane w kolejności od najstarszych do najnowszych. Partycje umożliwiają Kafce skalowalność i redundancję. Mogą one być hostowane na oddzielnych serwerach, dzięki czemu pojedynczy temat może być skalowany horyzontalnie (poziomo) na kilka serwerów.

Jednostką danych w RabbitMQ jest wiadomość, która składa się z ładunku oraz atrybutu. Ładunek zawiera treść komunikatu i jest tablicą bajtów. Powszechną praktyką jest definiowanie wiadomości w formie serializowanej np. JSON lub XML. Atrybuty to elementy pozwalające na zidentyfikowanie odbiorców i są ustawiane w momencie wysłania komunikatu przez producenta.

RabbitMQ bazuje na protokole AMQP (Advanced Message Queuing Protocol). Jest to otwartoźródłowy standard dla oprogramowania zorientowanego na przesyłanie wiadomości. AMQP definiuje zachowanie producenta oraz odbiorcy dla klientów wspierających ten protokół, dzięki czemu klienci są kompatybilni na podobnej zasadzie do HTTP, FTP, SMTP.

Protokół AMQP definiuje cztery rodzaje central wiadomości:

1. Centrala wiadomości bezpośrednich (ang. direct exchange) - dostarcza wiadomości do kolejek na podstawie klucza trasowania.
2. Centrala rozgłośni wiadomości (ang. fanout exchange) - wiadomości są wysyłane do wszystkich aktywnych kolejek, a klucz trasowania jest ignorowany.
3. Centrala wiadomości z nagłówkami (ang. headers exchange) - działa na podobnej zasadzie do centrali wiadomości bezpośrednich, ale nie wykorzystuje klucza trasowania. O przekierowaniu wiadomości decydują nagłówki komunikatów, dzięki czemu można zastosować liczby, łańcuchy znaków, funkcje skrótu.
4. Centrala wiadomości tematycznych (ang. topic exchange) - dostarcza wiadomości do kolejek na podstawie klucza trasowania oraz wzorca [9].

3.2. Podejście push-pull

Apache Kafka używa modelu pull. Oznacza to, że subskrybenci wysyłają zapytania po partię wiadomości. Dzięki temu podejściu Kafka umożliwia konsumowanie większej liczby komunikatów w danej jednostce czasu. Kolejną zaletą jest również to, że subskrybent może przetworzyć wiadomości pomimo przytłoczenia lub przerwy w działaniu. Ten model umożliwia także różnym odbiorcom konsumowanie wiadomości w różnym tempie. Wadą modelu pull jest większe zużycie zasobów ze względu na regularne odpytywanie brokera.

RabbitMQ używa modelu push. Oznacza to, że broker po otrzymaniu wiadomości od producenta przesyła ją od razu do subskrybenta. Może to skutkować przytłoczeniem odbiorcy, dlatego RabbitMQ umożliwia konfigurację parametru „prefetch limit". Opisywany model używany jest ze względu na małe opóźnienie przy przesyłaniu pojedynczych wiadomości.

3.3. Długość życia wiadomości

Apache Kafka domyślnie przechowuje wszystkie przesłane wiadomości na dysku. Każdy komunikat zawiera własność TTL (ang. time to live). Po upływie tego czasu wiadomość oznaczana jest do usunięcia i usuwana w celu zwolnienia miejsca na dysku. Własność TTL jest konfigurowalna dla komunikatów w ramach tematu.

RabbitMQ domyślnie nie przechowuje wiadomości - po wysłaniu do subskrybenta komunikaty są usuwane. Istnieje możliwość konfiguracji kolejki, tak aby komunikaty były zapisywane na dysku. Wiąże się to jednak ze spadkiem wydajności RabbitMQ [10].

3.4. Kolejność wiadomości

Apache Kafka zapewnia, że wiadomości wysłane do tej samej partycji zostaną przetworzone z zachowaniem kolejności. Aby komunikaty zostały wysyłane do tej samej partycji konieczne jest ustawienie klucza dla każdego komunikatu. Wszystkie wiadomości z tym samym kluczem umieszczane są w tej partii, przez co subskrybenci przetwarzają je z zachowaniem kolejności w jakiej zostały wysłane.

RabbitMQ zapewnia gwarancję kolejności dostarczenia wiadomości tylko w określonych warunkach. Porządek jest zachowany tylko w przypadku gdy istnieje jeden subskrybent odbierający komunikaty [9]. Jednak gdy wielu odbiorców konsumuje wiadomości z tej samej kolejki nie ma gwarancji co do kolejności przetwarzania danych. Brak gwarancji zachowania porządku wynika z tego, że subskrybent może odesłać komunikat po odczytaniu np. w przypadku błędu. Po zwrocie wiadomości inny konsument może ją odebrać pomimo tego, że kolejne wiadomości mogły zostać przetworzone. Istnieje możliwość zapewnienia gwarancji kolejności poprzez ograniczenie współbieżności konsumentów do jednego. Jednak ograniczenie do jednowątkowego subskrybenta poważnie wpływa na zdolność do skalowania oraz na wydajność [11].

3.5. Logika trasowania

Apache Kafka nie wspiera trasowania. Tematy podzielone są na partycje, w których komunikaty przechowywane w niezmiennej sekwencji.

RabbitMQ posiada wbudowane mechanizmy pozwalające na zdefiniowane skomplikowanego trasowania. Trasy bezpośrednie lub oparte na regularnych wyrażeniach pozwalają na zdefiniowanie ścieżki dla przesyłanych wiadomości bez potrzeby dodawania kodu [12].

3.6. Skalowalność

Apache Kafka została zaprojektowana w sposób pozwalający na skalowanie horyzontalne. Oznacza to, że w celu radzenia sobie ze zwiększonym obciążeniem istnieje możliwość dołożenia dodatkowych maszyn. Wiadomości w Kafce należą do tematów i są rozdystrybuowane na kilku partycjach. Możliwość podziału tematu na partycje umożliwia skalowanie tematu poza rozmiar, który mieściłby się na pojedynczym serwerze [13].

RabbitMQ jest skalowalny wertykalnie (pionowo). Oznacza to, że w celu radzenia sobie ze zwiększonym obciążeniem należy zwiększyć moc przetwarzania [10].

3.7. Unikalne funkcjonalności

W poprzednich podrozdziałach zostały opisane różnice w poszczególnych elementach badanych usług strumieniowego danych. Jednak obie te technologie posiadają unikalne funkcjonalności. Znajomość tych funkcji może być ważnych czynnikiem przy wyborze odpowiedniego brokera wiadomości [5].

Funkcjonalności unikalne dla Apache Kafka:

- Długotrwałe przechowywanie wiadomości - Apache Kafka przechowuje wiadomości na dysku. Usuwane są one po upływie czasu, przez który komunikaty mają być przechowywane lub gdy brakuje miejsca na dysku.
- Powtarzanie wiadomości - z powodu przechowywania wiadomości na dysku, Kafka umożliwia powtórzenie wysłania wiadomości w przypadku takiej potrzeby.
- Kafka Connect - jest to szkielet aplikacji pozwalający na integrację Kafki z docelowymi systemami. Pozwala on na łatwe zdefiniowane połączeń oraz przesłanie dużej ilości danych z lub do brokera Kafki.
- Kompaktowanie logów - Apache Kafka usuwa stare rekordy, jeżeli pojawią się nowe wiadomości z tym samym kluczem partycji. Dzięki temu mechanizmowi zawsze dostępna jest ostatnia, najbardziej aktualna wartość.

Funkcjonalności unikalne dla RabbitMQ:

- Ustandaryzowany protokół - RabbitMQ jest implementacją protokołu AMQP. Z tego powodu systemy korzystające z jednej z implementacji tego standardu uzyskują interoperacyjność.
- Wsparcie wielu protokołów - Oprócz protokołu AMQP, RabbitMQ wspiera również kilka innych standardów przesyłania danych, przede wszystkim MQTT (MQ Telemetry Transport) oraz STOMP (Simple Text Oriented Message Protocol).
- Zaawansowane narzędzie UI do monitoringu - RabbitMQ posiada standardowo wbudowane narzędzie do metryk z interfejsem użytkownika, które umożliwia monitorowanie kluczowych kwestii związanych z brokerem.
- Brak zużycia dysku - RabbitMQ nie wymaga miejsca na dysku do poprawnego działania, w przypadku gdy nie zapisujemy wiadomości na dysku (domyślnie wiadomości nie są zapisywane). Jest to zaletą w systemach z ograniczeniami sprzętowymi oraz aplikacji wbudowanych.
- Długość życia wiadomości - Komunikaty mogą posiadać dodatkową metadaną definiującą życie wiadomości. Jeśli po upływie tego czasu wiadomość pozostaje na kolejce, to nie zostanie ona dostarczona do producenta.

4. Eksperyment badawczy

Eksperymenty oraz pomiary zostały przeprowadzone na maszynie wirtualnej, której przydzielono następujące zasoby wymieniowe w Tabeli 1.

Tabela 1: Parametry maszyny wirtualnej

System operacyjny	Fedora 35
Procesor	AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
Pamięć RAM	16 GB
Dysk SSD	256 GB

Badania przeprowadzono na najnowszych stabilnych wersjach systemów kolejkowych na moment przeprowadzania badań, tj.:

- Apache Kafka, wersja: 3.0.0,
- RabbitMQ, wersja: 3.9.

W celu przeprowadzenia badań wydajnościowych przygotowano cztery aplikacje napisane w języku programowania Java:

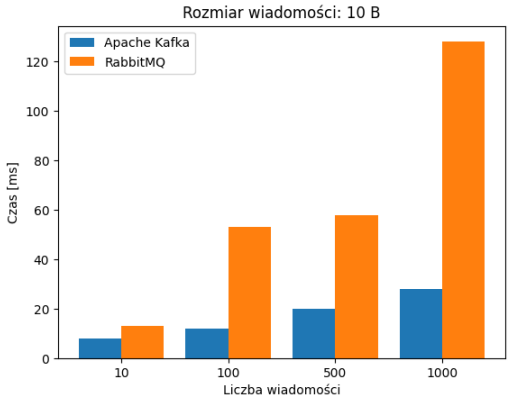
1. Aplikacja pełniąca funkcję wydawcy dla Apache Kafka.
2. Aplikacja pełniąca funkcję subskrybenta dla Apache Kafka.
3. Aplikacja pełniąca funkcję wydawcy dla RabbitMQ.
4. Aplikacja pełniąca funkcję subskrybenta dla RabbitMQ.

Wymienione aplikacje zostały napisane z użyciem szkieletu programistycznego Spring Boot w wersjach:

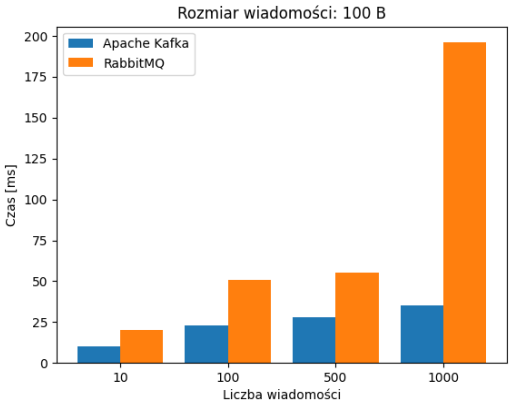
- Java JDK, wersja 11.0.8,
- Spring Boot, wersja: 2.5.

4.1. Badanie wydajności przy użyciu autorskiej aplikacji

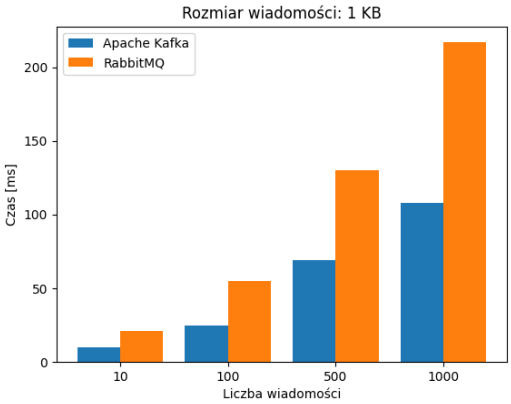
Do przeprowadzania badań wydajności brokerów Apache Kafka oraz RabbitMQ wykorzystano aplikacje producenta i subskrybenta. Badania przeprowadzono dla następujących rozmiarów wiadomości: 10B, 100B, 1KB, 10KB, 100KB, 1MB. Każdą z tych wiadomości przesłano 10 razy, 100 razy, 500 razy oraz 1000 razy. Przed wysłaniem pierwszej wiadomości oraz po odbiorze ostatniej został zapamiętany aktualny czas. Różnica pomiędzy tymi czasami pozwoliła na określenie czasu, jakiego potrzebował badany broker wiadomości na przesłanie wszystkich wiadomości.



Rysunek 1: Porównanie czasu przesyłania wiadomość o rozmiarze 10B.

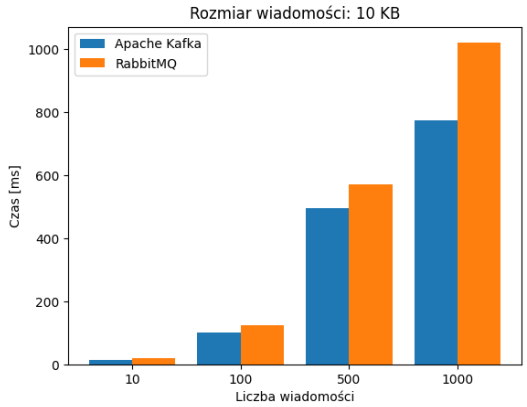


Rysunek 2: Porównanie czasu przesyłania wiadomość o rozmiarze 100B.

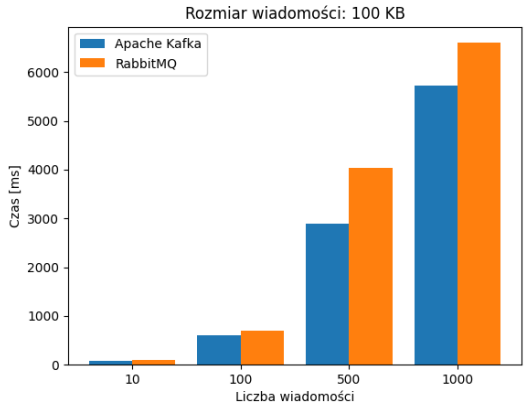


Rysunek 3: Porównanie czasu przesyłania wiadomość o rozmiarze 1KB.

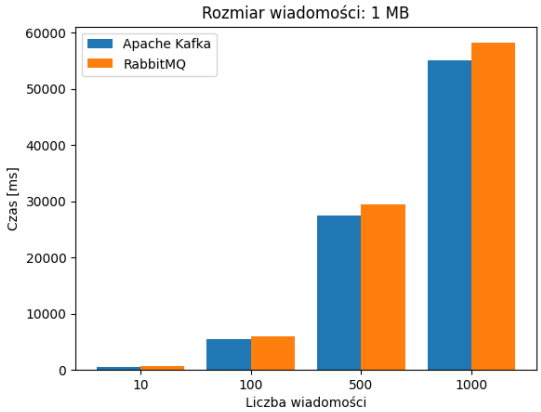
Dla każdej kombinacji rozmiaru oraz liczby przesyłanych wiadomości Apache Kafka posiada lepszą wydajność niż RabbitMQ. Najbardziej zbliżone wyniki (z nieznaczną korzyścią dla Apache Kafka) uzyskano w eksperymencie z plikiem o wielkości 1MB. Wydajność badanych brokerów (liczba przesłanych wiadomości na sekundę) przedstawiono w Tabeli 2.



Rysunek 4: Porównanie czasu przesyłania wiadomość o rozmiarze 10KB.



Rysunek 5: Porównanie czasu przesyłania wiadomość o rozmiarze 100KB.



Rysunek 6: Porównanie czasu przesyłania wiadomość o rozmiarze 1MB.

Tabela 2: Liczba przesłanych wiadomości na sekundę

Rozmiar pliku	Apache Kafka	RabbitMQ
10 B	23676	6389
100 B	1677	5000
1 KB	7594	3806
10 KB	1161	826
100 KB	173	140
1 MB	18	16

Tabela 3: Średni czas przesłania wiadomości

Rozmiar pliku	Apache Kafka	RabbitMQ
10 B	17 ms	63 ms
100 B	24 ms	80,5 ms
1 KB	53 ms	105,75 ms
10 KB	346,75 ms	424.5 ms
100 KB	2322,75 ms	2858 ms
1 MB	22166,25 ms	23589,75 ms

Średni czas przesłania wszystkich wiadomości jest niższy dla Apache Kafka.

Tabela 4: Odchylenie standardowe dla przesłanych wiadomości

Rozmiar pliku	Apache Kafka	RabbitMQ
10 B	7,69 ms	41,38 ms
100 B	9,14 ms	69,05 ms
1 KB	38,45 ms	75,37 ms
10 KB	306,65 ms	396,80 ms
100 KB	2229,51 ms	2632,64 ms
1 MB	21536,39 ms	22746,74 ms

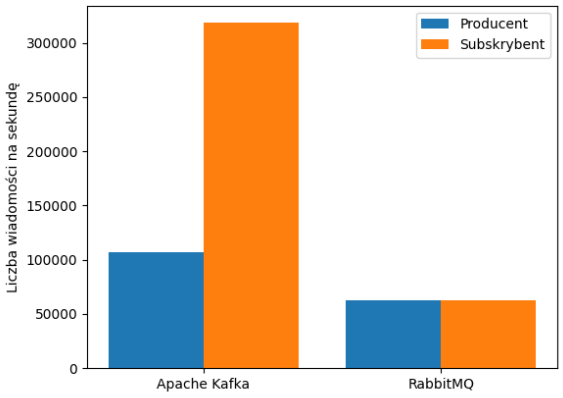
Odchylenie standardowe dla przesłanych wiadomości jest mniejsze dla Apache Kafka. Oznacza to, że wyniki badań wydajnościowych są bardziej zróżnicowane dla RabbitMQ.

4.2. Badanie wydajności przy użyciu pomocniczych narzędzi

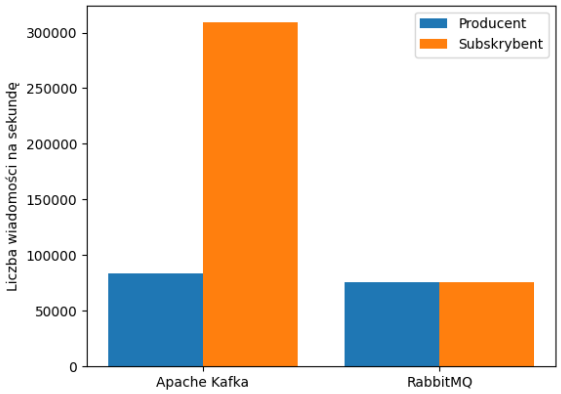
W podrozdziale 4.1 opisano eksperyment wydajnościowy z użyciem autorskich aplikacji. Scharakteryzowane zostały wykonane badania wydajnościowe z użyciem gotowych narzędzi pomocniczych dostępnych dla badanych technologii. Dla Apache Kafka są to skrypty kafka-producer-perf-test oraz kafka-producer-perf-test [14], dla RabbitMQ jest to narzędzie RabbitMQ PerfTest [15]. Celem eksperymentu jest zbadanie wydajności badanych brokerów dla modeli:

1. Jeden producent, jeden subskrybent.
2. Dwóch producentów, jeden subskrybent.
3. Jeden producent, dwóch subskrybentów.
4. Dwóch producentów, dwóch subskrybentów.

Badania zostały przeprowadzone na domyślnych konfiguracjach wybranych technologii. W ramach eksperymentu przesłano milion wiadomości o rozmiarze 1KB. Wydajność definiujemy jako liczbę wiadomości wysłanych/odebranych na sekundę.

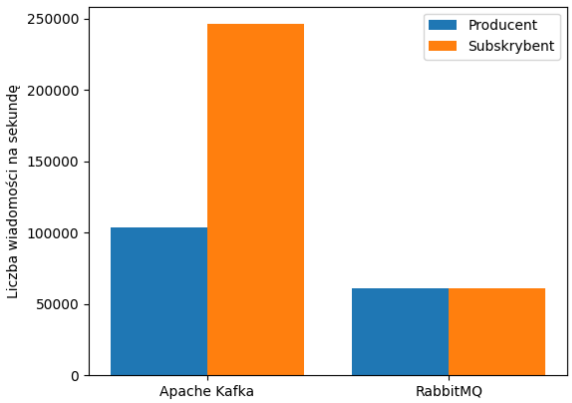


Rysunek 7: Model jeden producent, jeden subskrybent.

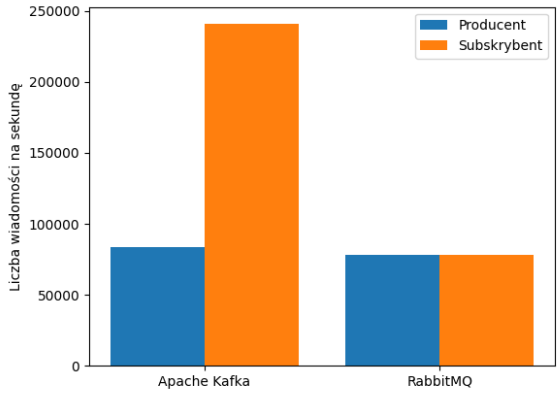


Rysunek 8: Model dwóch producentów, jeden subskrybent.

Dla wszystkich przypadków testowych Apache Kafka jest wydajniejsza niż RabbitMQ. Dla każdego modelu można również zaobserwować wysoką wydajność odbiorcy Apache Kafka. Wynika ona z wykorzystania przez badaną technologię podejścia „pull", gdzie odbiorca odpytuje broker, który wysyła wiadomości partiach.



Rysunek 9: Model jeden producent, dwóch subskrybentów.



Rysunek 10: Model dwóch producentów, dwóch subskrybentów.

5. Wnioski

Celem badań było porównanie dwóch najpopularniejszych usług strumieniowego danych - Apache Kafka oraz RabbitMQ. Technologie te zbadano pod względem wybranych aspektów, różnic w działaniu oraz oferowanych funkcjonalnościach. Badania zostały uzupełnione eksperymentami wydajnościowymi z użyciem autorskich aplikacji wysyłających oraz odbierających wiadomości, a także przy wykorzystaniu wbudowanych, pomocniczych narzędzi.

Eksperymenty polegały na analizie wydajności badanych technologii dla wiadomości o różnych rozmiarach oraz różnej liczbie wysłanych danych. Wykazały one, że Apache Kafka jest wydajniejsza niż RabbitMQ dla każdego rozmiaru i liczby przesłanych wiadomości. Warto także zauważyć, że różnice w wydajności pomiędzy badanymi technologiami były mniej znaczące wraz ze wzrostem rozmiaru przesyłanego komunikatu.

Kolejne zestawy eksperymentów przeprowadzone zostały z użyciem pomocniczych aplikacji. Ich celem było zbadanie, jaki wpływ na wydajność mają zmiany w liczbie producentów i subskrybentów. Eksperymenty wykazały, że Apache Kafka jest wydajniejsza niż RabbitMQ dla każdego badanego modelu. Warto tutaj zauważyć, że różnice na korzyść Kafki były mniej znaczące w modelach, gdzie znajdowało się dwóch producentów. Kolejną kwestią, którą trzeba podkreślić jest wysoka wydajność odbiorcy Kafki. Dzięki temu, że omawiana technologia przetwarza wiadomości w partiach, wydajność subskrybenta była dwuipółkrotnie lub nawet trzykrotnie większa niż wydawcy.

Przeprowadzone w ramach artykułu badania wydajnościowe oraz porównanie teoretyczne pokrywają się z wynikami uzyskanymi w innych pracach. W pracach [1] oraz [3] stwierdzono, że Apache Kafka jest wydajniejsza niż RabbitMQ. W artykule [2] autorzy opisali, że Apache Kafka jest narzędziem preferowanym dla systemów rozproszonych, natomiast RabbitMQ dla systemów, które wymagają obsłużenia skomplikowanego trasowania. Wyniki badań powyższych prac pokrywają się z wynikami otrzymanymi w ramach niniejszego artykułu.

Wydajność jest często bardzo ważną kwestią przy wyborze odpowiedniej usługi strumieniowego danych, ale nie zawsze kluczową. Systemy informatyczne mają określony cel i wymagania biznesowe, które mogą wpłynąć na wybór odpowiedniej technologii. Dlatego też eksperymenty wydajnościowe zostały uzupełnione porównaniem teoretycznym, gdzie zbadano usługi strumieniowego przesyłania danych pod względem wybranych aspektów.

Apache Kafka jest odpowiednim wyborem dla wskazanych scenariuszy:

- Aplikacje „big-data”, gdzie wymagane jest przetwarzanie dużej ilości danych dzięki wysokiej wydajności.
- Systemy rozproszone, aplikacje zbudowane z użyciem mikroserwisów ze względu na możliwość skalowania horyzontalnego (skalowanie poprzez dodawanie większej liczby maszyn).
- Systemy wymagające dostępu do historii wiadomości. Kafka jest magazynem wiadomości i umożliwia powtórzenie zdarzeń przez klientów.
- Pozyskiwanie zdarzeń - Apache Kafka przechowuje strumień danych związanych z danym bytem, dzięki czemu jest możliwość odtworzenia obecnego stanu obiektu na podstawie zdarzeń z nim związanych.

Scenariusze, w których można wykorzystać RabbitMQ:

- Starsze systemy informatyczne korzystające z protokołów AMQP, STOMP, MQTT.
- Aplikacje wymagające szczegółowej kontroli nad spójnością, gwarancją dostawy poszczególnych wiadomości.
- Systemy informatyczne wymagające złożonych opcji trasowania.

Literatura

[1] P. Dobbelaere, K. Esmaili, Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations, Proceedings of the 11th ACM international conference on distributed and event-based systems (2017) 227-238.

[2] T. Sharvari, K. Sowmya Nag, A study on Modern Messaging Systems - Kafka, RabbitMQ and NATS Streaming, CoRR (2019) abs/1912.03715.

[3] J. Kreps, N. Narkhede, J. Rao, Kafka: a Distributed Messaging System for Log Processing, Proceedings of the NetDB (2011) 1-7.

[4] B. Jones, S. Luxenberg, RabbitMQ Performance and Scalability Analysis, project on CS (2011) 4284.

[5] V. John, X. Liu, A Survey of Distributed Message Broker Queues, arXiv preprint (2017) arXiv:1704.00411.

[6] N. Narkhede, G. Shapira, T. Palino, Kafka: the definitive guide: real-time data and stream processing at scale, O'Reilly Media, 2017.

[7] Apache Kafka, Apache Kafka Documentation, https://kafka.apache.org/documentation.html, [2022-01-05].

[8] E. Ayanoglu, A. Yytaş, D. Nahum. Mastering RabbitMQ. Packt, 2016.

[9] RabbitMQ. RabbitMQ Documentation, https://www.rabbitmq.com/documentation.html, [2022-01-04].

[10] G. Roy, RabbitMQ in Depth, Manning Publications, 2017.

[11] E. Stiller. RabbitMQ vs. Kafka – An Architect's Dilemma, https://stiller.blog/2020/02/rabbitmq-vs-kafka-an-architects-dilemma-part-2/, [2022-01-01].

[12] L. Johansson, When to use RabbitMQ or Apache Kafka, https://www.cloudamqp.com/blog/when-to-use-rabbitmq-or-apache-kafka.html, [2022-01-02].

[13] Indexnine. Apache Kafka: What sets it Apart, https://indexnine.com/apache-kafka-what-sets-it-apart/, [2022-01-03].

[14] Cloudera. Managing Apache Kafka - kafka-*-perf-test. https://docs.cloudera.com/runtime/7.2.10/kafkamanaging/topics/kafka-manage-cli-perf-test.html, [2022-01-07].

[15] RabbitMQ PerfTest, RabbitMQ PerfTest, https://rabbitmq.github.io/rabbitmq-perf-test/stable/htmlsingle/, [2022-01-07].

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 97–104
Received: 7 March 2022
Accepted: 10 April 2022

Comparison of virtualization methods at operating system level

Porównanie metod wirtualizacji na poziomie systemu operacyjnego

Łukasz Grula*, Paweł Powroźnik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The aim of the work is comparative analysis of three tools for application container's orchestration: Kubernetes 1.2.2, Docker Swarm 1.24 and Nomad Hashicorp 1.2.0. For this purpose, test application was implemented, responding requests, then it was contenerized using Docker. For each tool, the scenario aimed at measuring pods startup time. The research was repeated three times. During each repetition number of replics were increased. Simultaneously with startup time test, CPU load and memory strain were measured. In comparison also time of regeneration was taken into consideration, what was realized by gauging time of response for GET request. The analysis showed that Docker Swarm in terms of most of the criteria examined in this work turned out as the best orchestration tool.

Keywords: virtualization; Kubernetes; Docker; Nomad

Streszczenie

Przedmiotem tej pracy jest analiza porównawcza trzech narzędzi do orkiestracji kontenerów aplikacyjnych: Kubernetes 1.2.2, Docker Swarm 1.24 oraz Nomad Hashicorp 1.2.0. Zaimplementowano w tym celu aplikację, odpowiadającą na żądania, następnie skonteryzowano ją używając technologii Docker. Dla każdego z narzędzi powtórzono trzykrotnie scenariusz, który na celu miał zmierzenie czasu startu podów. Równocześnie z badaniem czasu startu przeprowadzono badanie dotyczące obciążenia podzespołów. W porównaniach uwzględniono też czas regeneracji repliki. Ostatnim doświadczeniem było zbadanie mechanizmów równoważenia obciążenia. Z przeprowadzonych analiz wynika, że Docker Swarm pod względem dużej części kryteriów rozpatrywanych w tej pracy okazał się najlepszym narzędziem orkiestracyjnym.

Słowa kluczowe: wirtualizacja; Kubernetes; Docker; Nomad

*Corresponding author

Email address: **lukasz.grula@pollub.edu.pl** (Ł. Grula)

1. Wstęp

W dzisiejszych czasach przy tworzeniu systemów coraz częściej zauważalny jest udział architektury mikroserwisowej. W związku z czym ważnymi aspektami przy zarządzaniu takimi systemami są: łatwość konfiguracji, wysoka dostępność, reagowanie na awarię, wysoka przepustowość, możliwość szybkiego wdrażania systemów do środowisk produkcyjnych. Spowodowało to potrzebę zmiany podejścia stosowanego do tej pory - wirtualizacji, która wiązała się z dużą liczbą konfiguracji oraz małą uniwersalnością w odtwarzaniu środowiska. W dodatku wirtualizacja wymuszała większe wykorzystanie procesora, pamięci operacyjnej, a czas uruchamiania był na tyle duży, aby zapewnienie wysokiej dostępności i ciągłości funkcjonowania systemu było kłopotliwe.

Kolejnym rozwiązaniem wpływającym na usprawnienie funkcjonowania systemów było zastosowanie metod konteneryzacji, a więc odejście od używania maszyn wirtualnych i zastosowanie wirtualizacji systemów operacyjnych dzięki kontenerom. Obrazy kontenerów funkcjonują przy jądrze tego samego systemu operacyjnego w odróżnieniu od wirtualizacji.

Jednak rozwiązanie to nie oferowało wysokiej dostępności, reakcji na awarię czy automatycznego równoważenia obciążenia. Wtedy wprowadzone zostało pojęcie orkiestracji kontenerów, czyli zautomatyzowania procesów kontenerowych, a jednocześnie dostarczenie szeregu ułatwień w zapewnieniu ciągłej stabilności systemów i zarządzania nimi.

Obecnie na rynku pojawia się coraz więcej narzędzi, które wspomagają orkiestrację kontenerów, co pokazuje jak ważny jest to aspekt. Jednakże powoduje to również trudności w podjęciu decyzji, które narzędzie będzie najbardziej wydajne dla konkretnego systemu informatycznego. W związku z tym powstają artykuły oraz prace, które podejmują tą tematykę i próbują rozwiązać problemy związane z wyborem odpowiedniej technologii. W ramach przeprowadzonych badań przeanalizowano i porównano trzy narzędzia do orkiestracji kontenerów: Kubernetes (K8S), Docker Swarm oraz Nomad Hashicorp, które posiadają podobne funkcjonalności.

2. Przegląd literatury

W przypadku systemów bazujących na architekturze mikroserwisowej kluczowym jest odpowiednie ich wdrażanie i zarządzanie nimi. Wybrano i opisano więc artykuły, które skupiają się na tematyce wirtualizacji, konteneryzacji i orkiestracji.

Artykuł [1] porównuje zagadnienia wirtualizacji i konteneryzacji. Porównanie przedstawia między innymi zalety konteneryzacji w aspektach czasu startu, co autorzy przedstawiają jako jeden z głównych powodów wyparcia wirtualizacji przez kontenery. Kolejne zalety to między innymi możliwość dzielenia plików, bibliotek między kontenerami. Jest to cecha, która wpływa na znacznie mniejsze możliwości komunikacyjne między wirtualizowanymi systemami niż kontenerami. Wnioski, które wysnuto na podstawie artykułu pokazują, że konteneryzacja wyparła wirtualizację w działaniach chmurowych ze względu na swoją szybkość, większe możliwości i niezależność od systemu.

W artykule [2] zestawiono ze sobą technologię Docker, KVM, LXC oraz XenServer. Porównano między innymi czas startu dwudziestu obrazów. Wyniki badania pokazują, że kontenery Docker uruchomiły się 10 razy szybciej niż systemy uruchomione na KVM. Przedstawiono również wydajność układów wejścia-wyjścia, gdzie rezultaty pokazały, że wydajność układów kontenerów Docker jest porównywalna z rozwiązaniami natywnymi, natomiast KVM jest niemal dwukrotnie wolniejszy. Wnioski autorów pokazują, że konteneryzacja znacznie automatyzuje pracę, a w dodatku poprzez dodatkową warstwę jest o wiele szybsza niż wirtualizacja.

Praca [3] przedstawia porównanie metod orkiestracji kontenerów – takich jak Docker Swarm, Kubernetes oraz Openshift. Pokazano wady i zalety każdego z narzędzi. Jednym z ważnych aspektów Docker Swarm jest wykorzystanie wbudowanego w środowisko Docker CLI, a więc rezygnacja z konieczności instalacji dodatkowych technologii. Jako wadę i zaletę jednocześnie autorzy przedstawiają lekkość narzędzia, co powoduje mniejszą liczbę funkcjonalności w stosunku do Kubernetesa lub Openshifta. Kubernetes posiada bardzo rozbudowany system, dzięki komendom narzędzia kubectl. Ostatnim porównanym narzędziem jest Openshift, który jak mówią autorzy jest zbudowany na podstawach Kubernetesa, jednakże jego celem są aplikacje biznesowe, pod które został dostosowany. Dzięki czemu jest szybszy w instalacji oraz posiada łatwiejszy interfejs graficzny. Natomiast jest płatny, w przeciwieństwie do Kubernetesa czy Docker Swarma, które są projektami opensource, oraz oferuje mniejszą uniwersalność.

Podmiotem badań kolejnej pracy [4] było przedstawienie aspektu orkiestracji oraz porównanie narzędzi takich jak Kuberentes, Docker Swarm, Apache Mesos oraz Cattle. Pierwszy scenariusz realizował zmierzenie czasu utworzenia klastra, sprawdzając złożoność orkiestratorów. W tym porównaniu najlepiej wypadł Cattle, z uwagi na to, że jest on natywnym narzędziem do orkiestracji dla Ranchera. Największy czas tworzenia klastra uzyskał Kubernetes, ze względu na wiele możliwości, które oferuje. Kolejnym badaniem było sprawdzenie czasu startu kontenerów aplikacji takich jak Jenkins oraz Gitlab, które miały odpowiednio jedną, dwie i cztery repliki. Sprawdzono czasy, gdy obrazy były pobierane z lokalnego repozytorium oraz z rejestru Dockera. Wyniki były porównywalne, ale przy pojedynczym obrazie najlepiej zadanie zrealizował Kubernetes. Natomiast zwiększając liczbę replik, uwzględniając lokalne repozytorium, Kubernetes osiągnął najlepszy wynik, natomiast przy obrazie znajdującym się w zewnętrznym repozytorium wypadł najgorzej. Ostatnim scenariuszem było porównanie mechanizmów pracy w sytuacji awarii, zdecydowano się przetestować dwa przypadki – awarię kontenera oraz awarię hosta. W pierwszym przypadku podczas awarii kontenera najlepiej wypadł Kubernetes, natomiast gdy doszło do awarii węzła rezultaty Cattle, Docker Swarm oraz Apache Mesos były niemal identyczne, a Kubernetes potrzebował o wiele więcej czasu na przywrócenie działającego węzła. Autorzy we wnioskach stwierdzili, że obecnie Kubernetes jest najbardziej kompletnym narzędziem do orkiestracji, natomiast w niektórych przypadkach ma to przełożenie na wydajność.

Ostatni analizowany artykuł [5] przedstawia porównanie orkiestratorów kontenerów bazujących na chmurze obliczeniowej. Na początku pracy przedstawiono tematykę konteneryzacji oraz orkiestracji i opisano porównywane narzędzia, czyli Docker Swarm i Kubernetes. W kolejnej części nawiązano do wcześniej powstałych prac o tej tematyce i przedstawiono tematykę badań przeprowadzonych w ramach artykułu. W kolejnym rozdziale autorzy przedstawiają testowane narzędzia, opisują ich systemy, sposób działania oraz najważniejsze moduły i komponenty. Do przeprowadzenia badań użyto dwóch testów sprawności – Phoronix Test Suite oraz LiDAR (*Light Detection and Ranging*). Za pomocą pierwszego z nich przetestowano takie parametry jak wydajność dysku podczas dużego obciążenia, testy pamięci RAM – szybkość, przepustowość i użycie cache - kompresję plików za pomocą różnych algorytmów, kodowanie audio i wideo oraz wydajność bazy danych z użyciem SQLite. Drugi z testów odpowiadał za przetwarzanie danych pozyskanych w ramach LiDAR. Przetestowano to używając klastrów składających się z dziewięciu węzłów dla obu narzędzi. Wnioski z przeprowadzonych testów wskazały, że Docker Swarm ma niemal pomijalny narzut czasowy w stosunku do natywnego silnika Docker, natomiast w przypadku Kubernetesa wynosił on 8.3%. Autorzy wskazali, że Swarm potrzebuje o wiele mniej dodatkowych narzędzi, aby uformować klaster składający się z kontenerów swojego natywnego silnika. Zaletami Docker Swarm są jego lekkość, łatwość tworzenia i konfiguracji klastra oraz fakt, że jest to natywne rozwiązanie silnika Docker. Natomiast Kubernetes finalnie posiada o wiele więcej możliwości, dodatków, oferuje bardziej rozbudowany system i jego wsparcie, co powoduje, że już od wielu lat budowane są na nim systemy produkcyjne.

3. Porównywane narzędzia

Na przestrzeni czasu powstają kolejne narzędzia do orkiestracji kontenerów, które są mniej lub bardziej popularne. Zadaniem tych narzędzi jest m. in. ułatwienie i przyspieszenie konfiguracji środowisk, zapewnienie wysokiej dostępności, wydajności oraz bezpieczeństwa systemów.

W ramach pracy przeprowadzono analizę porównawczą trzech narzędzi: Kubernetes, Docker Swarm, Nomad Hashicorp. Wybrano te orkiestratory z uwagi na ich popularność, co potwierdza liczba artykułów opisujących te technologie.

3.1. Kubernetes

Kubernetes jest przenośną i rozszerzalną platformą oprogramowania open source, która służy do zarządzania zadaniami i aplikacjami uruchamianymi w kontenerach [6]. Najmniejszą jednostką Kubernetesa jest Pod, który może zawierać jeden lub więcej kontenerów wraz z zasobami, które są przez nie wykorzystywane. Możliwością replikacji podów zarządzają kontrolery replikacji. Kolejną wielkością są węzły. To właśnie one składają się z podów oraz narzędzi do zarządzania nimi, węzeł może być maszyną wirtualną lub fizyczną. Natomiast najwyższym poziomem abstrakcji w Kubernetesie jest klaster, jest zdefiniowany jako grupa maszyn na których działa K8s. Kubernetes bazuje na wzorcu architektonicznym nazywanym master/slave, w którym klaster menadżera odpowiada za zarządzanie węzłami (zwane minionami). Kubernetes zapewnia konteneryzację aplikacji oraz dba o jej skalowalność, równoważenie obciążenia, zdrowie czy też alokację zasobów sprzętowych.

3.2. Docker Swarm

Docker Swarm jest jednym z wewnętrznych narzędzi, które należy do orkiestracji. Jest też nazywany natywnym narzędziem Dockera do klastrowania i zarządzania aplikacjami [7]. Pomaga to w zarządzaniu większą liczbą kontenerów, skalowaniu ich czy reagowaniu na awarię. Tak jak Kubernetes bazuje na modelu master/slave, gdzie węzeł menadżera zarządza pracą jednostek – węzłów pracowniczych. Możliwe jest skonfigurowanie węzłów menadżerów tak aby nie uruchamiały one zadań, a odpowiadały jedynie za zarządzaniem pracownikami. Węzły pracownicze uruchamiają serwisy, na których pracują kontenery. W dodatku istnieje też jeden węzeł lidera, który odpowiada za decyzje zarządzające trybem swarm. Dużą zaletą Docker Swarm jest to, że jest wbudowany w bazowy silnik Docker, a więc wymaga mało konfiguracji. Swarm posiada też wbudowany mechanizm balansowania obciążenia.

3.3. Nomad Hashicorp

Nomad jest orkiestratorem, który ułatwia organizacjom wdrażanie oraz zarządzanie skonteneryzowanymi i nieskonteneryzowanymi aplikacjami. Umożliwia on programistom wdrażanie aplikacji za pomocą infrastruktury jako usługi [8]. Nomad jest zbudowany wokół całego systemu Hashicorp, który integruje się z takimi narzędziami jak Terraform, Consul czy Vault. Pomaga to m. in. we wdrażaniu aplikacji, automatycznym wykrywaniu usług czy zarządzaniu bezpieczeństwem. Klaster Nomada uruchamia w sobie tzw. agenta, który może działać w trybie serwera lub klienta. Tryb jest swego rodzaju mózgiem klastra. Zarządza on klientami czy pracami oraz alokuje zadania. Serwery replikują dane między sobą i dokonują wyboru lidera, aby zapewnić wysoką dostępność usług. Wcześniej wspomniane zadania są najmniejszą jednostką klastra. Są one uruchamiane przez wbudowane lub zewnętrzne sterowniki. Zadania mogą tworzyć ich grupy, które potem składają się w tak zwane prace i to one działają na agentach. Twórcy Nomada podkreślają, że wspierają nie tylko skonteneryzowane aplikacje, ale również ułatwiają zarządzanie starszymi systemami.

4. Metoda badań

4.1. Opis eksperymentu

Analizę porównawczą narzędzi do orkiestracji kontenerów aplikacji Kubernetes, Docker Swarm i Nomad Hashicorp przeprowadzono na podstawie czterech scenariuszy badawczych:

1. Zmierzenie czasu uruchomienia startowego skonteneryzowanej aplikacji.
2. Zbadanie obciążenia pamięci, procesora i dysku.
3. Zbadanie skalowalności, mierząc czas odpowiedzi aplikacji korzystając z mechanizmów równoważenia obciążenia.
4. Zmierzenie czasów autoregeneracji narzędzi.

Pierwszy scenariusz badawczy dotyczący czasu startu aplikacji polegał na zmierzeniu czasu między rozpoczęciem procedury tworzenia kontenerów przez narzędzie orkiestracji, a momentu, kiedy aplikacja wewnątrz kontenerów będzie gotowa do działania. Przeprowadzono takie testy dla jednej i dziesięciu replik.

Drugi scenariusz tak jak poprzedni składał się z trzech testów dla wyżej wymienionych liczb replik. Polegało to na sprawdzeniu obciążenia procesora, pamięci oraz dysku przez aplikacje. Każde z narzędzi udostępniało natywnie metryki do zmierzenia powyższych obciążeń. Natomiast w przypadku Docker Swarm dla dokładniejszych pomiarów doinstalowano narzędzie Swarmprom, które udostępniało dokładniejsze metryki.

Kolejny scenariusz dotyczył zbadania mechanizmów autoregeneracji, polegało to na zmierzeniu czasu, między zlikwidowaniem jednej z replik, a przywróceniem nowej z gotową do działania aplikacją.

Ostatni scenariusz przedstawiał testy niezawodności używając mechanizmów zrównoważenia obciążenia. Przeprowadzono je za pomocą biblioteki Gatling. Dokonano pomiarów czasów obsługi żądań. Po przeprowadzeniu testów, Gatling wygenerował raport, który zawierał wykresy i tabele przedstawiające

wyniki i statystyki opisujące żądania i odpowiedzi. Do uruchomienia aplikacji wewnątrz narzędzi użyto szablonów przeznaczonych dla każdego z orkiestratorów.

4.2. Środowisko testowe

W tabeli poniżej przedstawiono opis środowiska testowego, użytego do przeprowadzenia badań. Tabela zawiera również wersje testowanych narzędzi oraz wersje narzędzi do przeprowadzenia testów.

Tabela 1: Środowisko testowe – sprzęt i system

Sprzęt i system	
Procesor	Intel® Core™ i5-8300H
Pamięć RAM	16 GB DDR4 2444 MHz
Karta sieciowa	Realtek PCIe GbE Family Controller
System operacyjny	Ubuntu 20.04

Tabela 2: Środowisko testowe – narzędzia

Narzędzia	
Kubernetes	1.2.2
Docker Swarm	1.2.4
Nomad	1.2.0
Spring Boot	2.5.6
Gatling	3.5.1

4.3. Pomiar czasu startu aplikacji

W pierwszym scenariuszu czas startu aplikacji był mierzony od momentu zaczęcia tworzenia kontenerów dla aplikacji poprzez orkiestratora do czasu osiągnięcia przez aplikacje gotowości do działania wewnątrz kontenerów. Aplikacja, użyta także w kolejnych scenariuszach, uruchamiana jako aplikacja wewnątrz kontenerów została utworzona za pomocą szkieletu aplikacji Spring Boot. Listing 1. pokazuje fragment kodu zawierający kontroler REST, umożliwiający odbieranie żądań http typu GET pod adresem końcowym /, zwracając w odpowiedzi tekst *It works*.

Listing 1: Kod aplikacji testowej, używanej wewnątrz kontenerów

```
@SpringBootApplication
@RestController
public class TestApplication {
        @GetMapping(value = "/")
        public ResponseEntity<String> health() {
            return ResponseEntity.ok("It works!");
        }
        public static void main(String[] args) {
            SpringApplication.run(TestApplication.class, args);
        }
}
```

Pomiary startu tworzenia kontenerów uzyskano z metryk specyficznych dla każdego z narzędzi, natomiast czas startu aplikacji pozyskano z logów wewnątrz kontenera Docker.

4.4. Pomiar obciążenia podzespołów

Ważną rzeczą w dzisiejszych czasach jest to, aby aplikacje były zoptymalizowane pod kątem zużycia podzespołów. W tym scenariuszu sprawdzono jak duży narzut na użycie procesora, pamięci czy dysku ma każde z narzędzi oraz czy jest to znacząca wartość. W tym celu tego użyto specjalnych metryk dla każdych z narzędzi. Nomad udostępniał je natywnie, natomiast w przypadku Kubernetesa skorzystano z kube-state-metrics. Jest to projekt realizowany przez organizację Kubernetesa, który generuje metryki dla formatu Prometheusa na podstawie stanu aktualnych zasobów K8S [9]. W przypadku Docker Swarma skorzystano z podobnego projektu, jednakże przeznaczonego dla tej technologii – swarmprom, który również odpowiada za tworzenie metryk zgodnych z formatem Prometheusa.

4.5. Pomiar przepustowości i niezawodności

W dobie Internetu aplikacje internetowe doświadczają coraz to większego przepływu danych oraz liczby zapytań, więc bardzo ważną rzeczą jest, aby serwery posiadały wysoką przepustowość oraz były niezawodne. Oznacza to, że muszą przetwarzać dużą liczbę żądań na sekundę oraz jak najmniej z nich powinno kończyć się niepowodzeniem. W tym celu przeprowadzono badania z użyciem dziesięciu replik. Za rozprowadzenie ruchu odpowiadało narzędzie równoważenia obciążenia. W przypadku Kubernetesa był to Ingress-Nginx, dla Docker Swarma użyto wbudowanego natywnie narzędzia Ingress, a Nomad korzystał z HAProxy.

Do pomiarów czasów utworzono aplikację wykorzystującą bibliotekę Gatling. Jest to rozwiązanie open source służące do realizowania testów wydajnościowych [10]. Narzędzie to jest zaimplementowanie głównie w języku Scala. Wykonano 100 symulacji, w której zbadano ruch 900 użytkowników w ciągu 30 sekund. Listing 2 przedstawia fragment aplikacji odpowiadającej za uruchamianie symulacji.

Listing 2: Kod aplikacji wykonującej symulacje z użyciem biblioteki Gatling

```
class GetSimulation extends Simulation {
  val httpProtocol: HttpProtocolBuilder = http
    .baseUrl("http://localhost:8094")
    .header("Accept", "application/json")
    .header("Content-Type", "application/json")

  private val getRequest: HttpRequestBuilder = http("Get")
    .get("/")

  setup(
    scenario("Test performance for 500 users")
      .exec(get)
      .inject(constantUsersPerSec(500) during 10.seconds)
      .protocols(httpProtocol),
  )
}
```

4.6. Pomiar skuteczności mechanizmów autoregeneracji

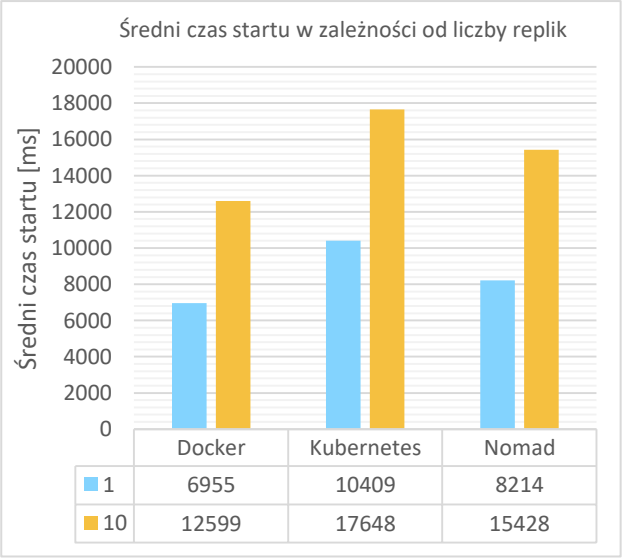
Kolejną ważną rzeczą dotyczącą systemów informatycznych jest to, aby były one projektowane

w sposób, który zapewni tolerancję potencjalnych awarii. System musi być gotowy do reakcji, gdy instancja aplikacji przestanie działać. W tym celu przygotowano badanie, które sprawdza czas działania mechanizmów autoregeneracji dla testowanych technologii. Polegało to na zmierzeniu czasu między zasymulowaniem awarii jednej z replik, czyli jej zatrzymania do czasu, kiedy nowa, uruchomiona w zastępstwie replika jest w pełni gotowa do działania. Czyli aplikacja wewnątrz kontenera potrafi przyjmować żądania i na nie odpowiadać. Testy przeprowadzono, gdy uruchomione było 10 replik aplikacji.

5. Wyniki badań

5.1. Pomiar czasu startu aplikacji

Na rysunku 1 przedstawiono średnie czasy startu aplikacji dla każdego z narzędzi, przedstawiają one pomiary przeprowadzone odpowiednio dla jednej oraz dziesięciu replik. Przeprowadzono sto takich pomiarów dla obu przypadków, a następnie wyciągnięto z nich średnią. Wykres z rysunku pierwszego pokazuje, ile czasu zajęło utworzenie aplikacji z jedną repliką. Pozwala to zaobserwować narzut infrastruktury każdego z narzędzi na uruchomienie pojedynczej aplikacji. Czasy zostały podane w milisekundach, aby zauważyć dokładniejsze różnice między badanymi technologiami.



Rysunek 1: Średnie czasy startu aplikacji testowej.

Jak można zauważyć Docker Swarm osiągnął najlepsze wyniki dla obu pomiarów. Zawdzięcza to swojej małej infrastrukturze oraz temu, że jest natywnym narzędziem silnika Docker, co pozwala na lepsze dostosowanie się do startu aplikacji umieszczonych w kontenerach. Wyniki Nomada były gorsze jedynie o 18% w przypadku jednej repliki i o 22% w przypadku dziesięciu replik. Kubernetes uzyskał najgorsze rezultaty, aczkolwiek jest to spowodowane najpotężniejszą infrastrukturą spośród narzędzi.

Tabela 3: Odchylenie standardowe i mediana czasu startu aplikacji dla jednej repliki

	Odchylenie stand. [ms]	Mediana [ms]
Docker Swarm	930,39	6 917,00
Kubernetes	882,49	10 694,00
Nomad	1 064,75	8 149,00

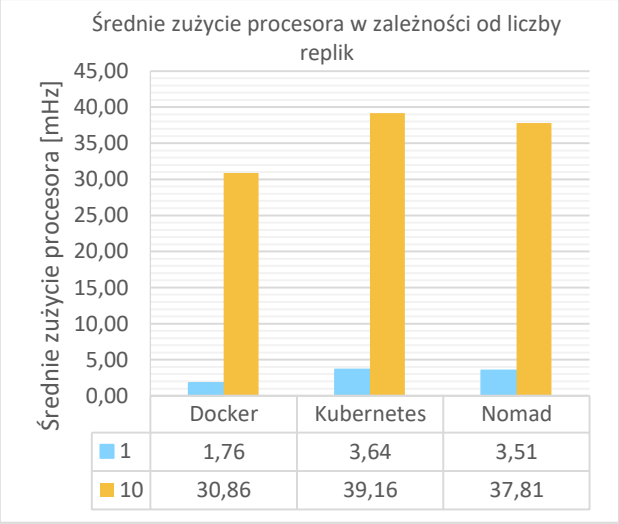
Tabela 4: Odchylenie standardowe i mediana czasu startu aplikacji dla dziesięciu replik

	Odchylenie stand. [ms]	Mediana [ms]
Docker Swarm	456,88	12 558,00
Kubernetes	1 051,78	17 637,00
Nomad	1 236,14	15 307,00

Tabele 3 i 4 przedstawiają odchylenia standardowe i mediany czasu startu aplikacji dla odpowiednio jednej i dziesięciu replik. W obu przypadkach zauważalne bardzo zbliżone do siebie miary średniej i mediany. Świadczy to o stabilnych czasach startu każdego z narzędzi. Odchylenia standardowe potwierdzają to pokazując niski i porównywalny rozrzut wyników.

5.2. Pomiar obciążenia podzespołów

Kolejnym pomiarem było zmierzenie obciążenia podzespołów przez aplikacje uruchomione za pomocą testowanych orkiestratorów. W przypadku obciążenia procesora wyniki podano w mHz, a w przypadku zużycia pomięci podano je w MB. Tak jak w poprzednim teście, badanie wykonano dla liczby jednej i dziesięciu replik i powtórzone 100 razy dla każdej z nich. W pierwszym przypadku pomogło to sprawdzić, czy narzędzia potrafią przyznawać odpowiednią ilość zasobów uruchomionej aplikacji. Drugi przypadek pozwolił na sprawdzenie czy orkiestratory są wstanie zoptymalizować zużycie podzespołów, gdy uruchomione jest kilka instancji tej samej aplikacji.



Rysunek 2: Średnie obciążenia procesora przez testową aplikację.

Rysunek 2 przedstawia wykresy ilustrujące średnie obciążenie procesora dla odpowiednio jednej i dziesięciu replik aplikacji testowej. W pierwszym przypadku widać, że Docker Swarm uzyskał wyniki ponad dwa razy niższe od pozostałych testowanych narzędzi. Pokazało to, że Docker Swarm poprzez bycie natywnym rozwiązaniem silnika Docker dobrze zoptymalizował utworzenie pojedynczego kontenera i narzut infrastruktury był najmniejszy. Drugi przypadek pokazuje już mniejsze różnice. Widać, że w porównaniu do jednej repliki aplikacja zreplikowana dziesięciokrotnie obciążyła procesor o wiele bardziej w przypadku Swarma. Natomiast wyniki Kubernetesa i Nomada pokazały, że obciążenie rośnie proporcjonalnie wraz z ilością replik. Jest to pozytywną informacją w przypadku dużych systemów, w których liczby replik potrafią sięgać setek, co pozwala na oszacowanie potencjalnego obciążenia procesora maszyny serwerowej. Zauważalne są tutaj zalety dla małych systemów, które nie posiadają wielu replik. Spowodowane jest to bardzo dobrym zarządzaniem zużyciem procesora dla małej liczby podów przez Docker Swarm.

Tabele 5 i 6 zawierają miary odchylenia standardowego i mediany dla zużycia procesora z użyciem odpowiednio jednej i dziesięciu replik. Odchylenie od wartości przeciętnej jest pomijalne. Pokazuje to stabilność zarządzania użyciem procesora przez narzędzia. Mediany dla obu przypadków są porównywalne do miar średnich, a w przypadku Kubernetesa dla dziesięciu replik równa średniej.

Tabela 5: Odchylenie standardowe i mediana zużycia procesora dla jednej repliki

	Odchylenie stand. [mHz]	Mediana [mHz]
Docker Swarm	0,15	1,74
Kubernetes	0,13	3,65
Nomad	0,14	3,47

Tabela 6: Odchylenie standardowe i mediana zużycia procesora dla dziesięciu replik

	Odchylenie stand. [mHz]	Mediana [mHz]
Docker Swarm	1,06	30,82
Kubernetes	0,55	39,16
Nomad	2,18	38,14

Tabele 7 i 8 przedstawia odchylenie standardowe i medianę dla zużycia pamięci przez uruchomione aplikacje. Istotną informacją, którą można odczytać z tabel jest pomijalne odchylenie.

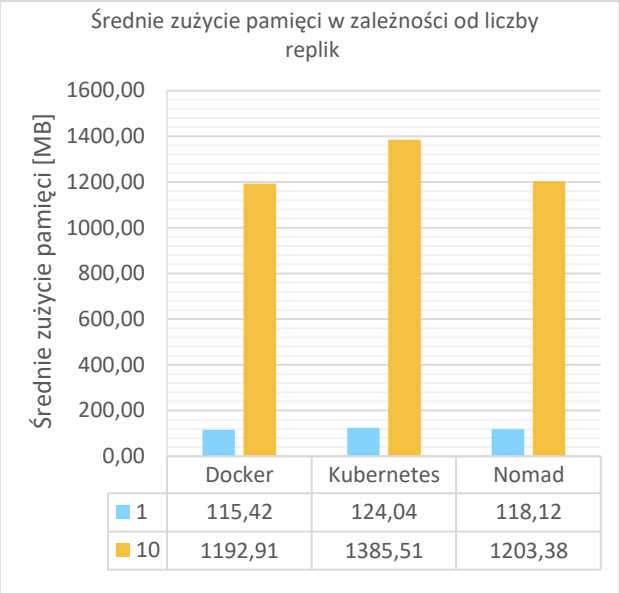
Rysunek 3 ilustruje średnie zużycie pamięci ponownie dla jednej i dziesięciu replik. Widać, że na pierwszym wykresie wyniki są niemal identyczne dla wszystkich narzędzi i każdy z orkiestratorów podobnie radzi sobie z zarządzaniem pamięcią dla kontenerów, które uruchamia. Analogicznie dla dziesięciu replik wyniki były znów niemal identyczne i nie zauważono znaczących optymalizacji, gdy zwiększono liczbę replik tej samej aplikacji. Wyniki były dziesięć razy większe niż dla pojedynczej replik. Oznacza to, że każda replika zużywała tyle samo pamięci co w przypadku pojedynczego uruchomienia aplikacji.

Tabela 7: Odchylenie standardowe i mediana zużycia pamięci dla jednej repliki

	Odchylenie stand. [MB]	Mediana [MB]
Docker Swarm	4,54	115,00
Kubernetes	6,87	123,00
Nomad	5,73	118,00

Tabela 8: Odchylenie standardowe i mediana zużycia pamięci dla dziesięciu replik

	Odchylenie stand. [MB]	Mediana [MB]
Docker Swarm	51,79	1 189,00
Kubernetes	31,86	1 385,00
Nomad	44,18	1 206,00



Rysunek 3: Średnie zużycie pamięci przez testową aplikację.

Podsumowując każde z narzędzi dobrze sprawdziło się w aspekcie zarządzania pamięci orkiestrowanych aplikacji. Jedynie w przypadku dziesięciu replik narzut Kubernetesa wzrósł o nieznaczącą wartość, co mimo wszystko nie powoduje różnic. Żadne z narzędzi nie uzyskało tutaj przewagi, co pokazuje, że coraz trudniej optymalizować aplikacje pod względem zużycia pamięci i programiści muszą tworzyć przemyślane systemy.

5.3. Pomiar przepustowości i niezawodności

Kolejnym testem było sprawdzenie przepustowości i niezawodności narzędzi, które współpracują z mechanizmami odpowiedzialnymi za równoważenia obciążenia. W tym przypadku stworzono 10 replik

aplikacji testowej. Rysunki 4, 5 i 6 przedstawiają fragment raportu wygenerowanego przez narzędzie Gatling po wykonaniu testu obciążeniowego dla każdego z orkiestratorów.

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	27000	26970	30	0%	519.231	276	12536	14154	17024	19289	22115	12122	3232
Get	27000	26970	30	0%	519.231	276	12536	14154	17024	19289	22115	12122	3232

Rysunek 4: Fragment tabeli z raportu wygenerowanego przez narzędzie Gatling dla Docker Swarm.

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	27000	27000	0	0%	400	1572	10801	13164	19234	22311	22910	11266	3905
Get	27000	27000	0	0%	400	1572	10801	13164	19233	22311	22910	11266	3905

Rysunek 5: Fragment tabeli z raportu wygenerowanego przez narzędzie Gatling dla Kubernetes.

STATISTICS — Expand all groups | Collapse all groups

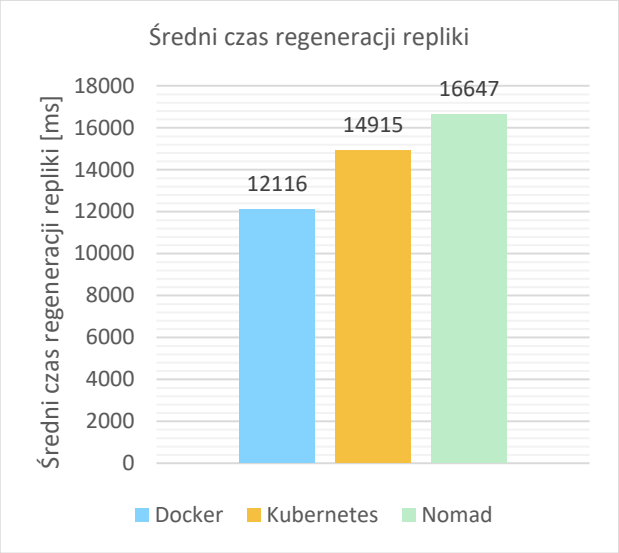
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	27000	27000	0	0%	586.957	2280	10928	15432	19147	20999	26283	11449	4664
Get	27000	27000	0	0%	586.957	2280	10930	15446	19146	20999	26283	11449	4664

Rysunek 6: Fragment tabeli z raportu wygenerowanego przez narzędzie Gatling dla Nomad Hashicorp.

Dla każdego przypadku wykonano sumarycznie 27000 żądań w czasie 30 sekund, co daje 900 żądań na sekundę. Tabela przedstawia informacje na temat liczby obsłużonych poprawnie żądań oraz tych które zakończyły się niepomyślnie. Ponadto tabela zawiera również dane statystyczne takie jak średnia, mediana, minimalny i maksymalny czas żądania, jak również rozkład percentyli. Percentyle przedstawiają wartości, poniżej których znajduje się określony procent danych. Dla powyższego przykładu 99-ty percentyl to czas odpowiedzi, dla którego 99% przypadków osiągał krótszy czas odpowiedzi serwera. Dzięki przedstawionym wynikom widać, że największy minimalny i maksymalny czas odpowiedzi uzyskał Nomad, a mimo to zaliczył najwyższy współczynnik zapytań na sekundę. Średnie czasy odpowiedzi były niemal identyczne dla każdego z narzędzi. Podobna sytuacja ma miejsce w przypadku rozkładu percentyli. Są to wartości w większości takie same dla wszystkich orkiestratorów, różnią się pojedynczymi wynikami np. dla 95-tego percentyla na korzyść Docker Swarma lub 50-tego percentyla na jego niekorzyść. Ważną rzeczą, którą można zauważyć jest to, że 30 zapytań dla systemu orkiestracyjnego Dockera zakończyło się niepowodzeniem. Z uwagi na problemy z połączeniem doszło do przekroczenia maksymalnego, dozwolonego czasu. Jest to poważny minus, ponieważ mimo podobnych wyników czasowych, Docker Swarm nie przetworzył wszystkich zapytań, co w dobie dzisiejszych systemów jest dużą wadą, mimo że to jedynie 0.1% wszystkich zapytań.

5.4. Pomiar skuteczności mechanizmów autoregeneracji

Ostatnim badaniem było sprawdzenie skuteczności i sprawności mechanizmów autoregeneracji testowanych narzędzi. W tym celu znów utworzono 10 replik aplikacji testowej, a następnie usunięto jedną z nich i zmierzono czas, w którym każdy z orkiestratorów uruchomił nową instancję aplikacji, gotową do działania, czyli zdolną do przyjęcia żądań. Rysunek 7 przedstawia średni czas ze stu pomiarów dla każdego z narzędzi.



Rysunek 7: Średni czas regeneracji repliki dla narzędzi.

Jak widać na rysunku 7 różnice między wynikami nie są bardzo duże, jednak najlepszy rezultat uzyskał Docker Swarm. Znaczenie miało to, że jest to natywne rozwiązanie do orkiestracji silnika Docker, dzięki czemu przywrócenie do życia pojedynczego kontenera zostało zrealizowane najszybciej. Na drugim miejscu znalazł się Kubernetes, a nieznacznie gorszy wynik uzyskał Nomad.

Tabela 9: Odchylenie standardowe i mediana czasu regeneracji repliki

	Odchylenie stand. [ms]	Mediana [ms]
Docker Swarm	841,68	12 176,00
Kubernetes	631,84	14 886,00
Nomad	574,28	16 634,00

Tabela 9 przedstawia odchylenie standardowego i medianę dla czasów regeneracji repliki. Otrzymane wyniki wskazują na jednolite czasy uzyskane w trakcie badań. Potwierdza to stosunkowo małe odchylenie oraz mediana, której miara jest zbliżona do miary średniej. Oznacza to stabilność mechanizmów przywracania, dzięki czemu programiści mogą przewidzieć, ile czasu zajmie potencjalne naprawienie awarii.

Dodatkowym narzutem dla obu technologii było to, że narzędzia nie korzystają z natywnych rozwiązań do równoważenia obciążenia. Przywróconą do życia replikę należało ponownie dodać do odpowiednio Nginxa dla Kubernetesa oraz HAProxy dla Nomada, co mogło nieznacznie wpłynąć na uzyskany rezultat końcowy. Mimo to mechanizm autoregeneracji dla

każdego z orkiestratorów zadziałał poprawnie, czyli przywrócił uszkodzoną replikę.

6. Wnioski

Trudno jest podjąć optymalną decyzję podczas wyboru odpowiedniego orkiestratora, ponieważ to właśnie tego typu narzędzia odpowiadają za zarządzanie całym naszym systemem. Niniejsza praca, skupiła się na porównaniu wydajności i funkcjonalności trzech narzędzi służących do orkiestracji kontenerów. Miało to na celu ułatwienie podjęcia decyzji dotyczącej wyboru odpowiedniego narzędzia dostosowanego do systemu. W analizie porównawczej wzięto pod uwagę kryteria takie jak: czas startu aplikacji, obciążenie podzespołów, niezawodność działania i przepustowość oraz czas autoregeneracji.

Biorąc pod uwagę kryterium czasu startu aplikacji Docker Swarm uzyskał najlepsze rezultaty, szczególnie dla aplikacji, które uruchamiane są jako pojedyncze repliki. Duży wpływ na to ma wcześniej wspomniany fakt, że Swarm jest natywnym narzędziem silnika Docker.

Porównując wyniki dotyczące obciążenia podzespołów przez aplikacje, którymi zarządzały orkiestratory widać znów dużą przewagę Swarma w aspekcie użycia procesora dla pojedynczej repliki. Natomiast przy większej liczbie replik różnice zacierają się. Na korzyść Nomada i Kubernetesa przemawia fakt, że zużycie pamięci rośnie proporcjonalnie dla kolejnych replik, natomiast w przypadku natywnego narzędzia do wsparcia trybu klastrowania Dockera, zużycie dla większej liczby replik wzrosło znacząco.

Pod względem przepustowości i niezawodności można zauważyć, że biorąc pod uwagę średni czas odpowiedzi na zapytanie przez każde z narzędzi, orkiestratory zrealizowały zadanie niemal identycznie. Jednakże Kubernetes osiągnął najgorszy wynik, jeśli chodzi o liczbę zapytań na sekundę, a najlepiej wypadł Nomad, choć otrzymał on największy maksymalny wynik odpowiedzi. Największym problemem Docker Swarma okazało się jednak to, że pewna część zapytań zakończyła się niepowodzeniem, ponieważ został przekroczony maksymalny czas oczekiwania, co nie zdarzyło się w przypadku pozostałych dwóch narzędzi. Jest to duża wada, ponieważ w przypadku niektórych systemów jest niedopuszczalnym, aby serwis nie był gotowy przetworzyć żądań.

W ostatnim badaniu dotyczącym mechanizmów autoregeneracji Swarm uzyskał najkrótszy czas uruchomienia nowej aplikacji w miejsce uszkodzonej repliki. Na drugim miejscu uplasował się Kubernetes, a ostatnie miejsce zajął Nomad. Jednakże czasy nie różniły się znacząco, a wszystkie narzędzia dobrze poradziły sobie z tym zadaniem, więc żadne z nich nie uzyskało znaczącej przewagi w tym aspekcie.

Z przeprowadzonej analizy można wysnuć wniosek, że najlepszym narzędziem orkiestracyjnym z trzech badanych orkiestratorów jest Docker Swarm. Jednakże nie zostały porównane funkcjonalności każdego z narzędzi, które w przypadku Kubernetesa i Nomada są o wiele większe niż możliwości Docker Swarm. W celu dokonania dokładniejszej analizy i oceny narzędzi trzeba by uwzględnić kolejne aspekty oraz przeprowadzić badania na różnorodnych aplikacjach.

Literatura

[1] R. Dua, A. Raja, D. Kakadia, Virtualization vs containerization to support paas, 2014 IEEE International Conference on Cloud Engineering (2014) 610-614.

[2] B. Rad, H. Bhatti, M. Ahmadi, An introduction to docker and analysis of its performance, International Journal of Computer Science and Network Security 17(3) (2017) 228.

[3] M. Moravcik, M. Kontsek, Overview of Docker container orchestration tools, 2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA) (2020) 475-480.

[4] I. Al Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, A. Palopoli, Container orchestration engines: A thorough functional and performance comparison, ICC 2019-2019 IEEE International Conference on Communications (ICC) (2019) 1-6.

[5] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera, R. Sinnott, A performance comparison of cloud-based container orchestration tools, 2019 IEEE International Conference on Big Knowledge (ICBK) (2019) 191-198.

[6] What is Kubernetes?, https://kubernetes.io/pl/docs/concepts/overview/what-is-kubernetes/. , [30.01.2022]

[7] P. Pedamkar, What is Docker Swarm?, https://www.educba.com/what-is-docker-swarm/ , [02.02.2022]

[8] Introduction to Nomad, https://learn.hashicorp.com/tutorials/nomad/get-started-intro?in=nomad/get-started , [02.02.2022]

[9] kube-state-metrics – Introduction to Kubernetes metrics, https://kubernetes.io/blog/2021/04/13/kube-state-metrics-v-2-0/ , [02.02.2022]

[10] What is Gatling?, https://gatling.io/open-source/, [08.02.2022]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 105–111
Received: 11 March 2022
Accepted: 29 March 2022

Detrimental Starfish Detection on Embedded System: A Case Study of YOLOv5 Deep Learning Algorithm and TensorFlow Lite framework

Nguyen Quoc Toan*

Department of Electronic and Electrical Engineering, Hongik University, Wausan-ro 94, Mapo-gu, Seoul, South Korea

Abstract

There is a great range of spectacular coral reefs in the ocean world. Unfortunately, they are in jeopardy, due to an over-abundance of one specific starfish called the coral-eating crown-of-thorns starfish (or COTS). This article provides research to deliver innovation in COTS control. Using a deep learning model based on the You Only Look Once version 5 (YOLOv5) deep learning algorithm on an embedded device for COTS detection. It aids professionals in optimizing their time, resources, and enhances efficiency for the preservation of coral reefs worldwide. As a result, the performance over the algorithm was outstanding with Precision: 0.93 - Recall: 0.77 - $F1_{score}$: 0.84.

Keywords: deep learning; computer vision; YOLO; embedded system

*Corresponding author
Email address: **quoctoann3@gmail.com** (N. Q. Toan)

1. Introduction

COTS (crown-of-thorns starfish) is a natural coral predator. However, when populations reach epidemic levels (about 15 starfish per hectare), corals are eaten quicker by COTS than they can develop. Crown-of-thorns starfish may devour up to 90% of a reef's living coral tissue during an epidemic. This adds to the pressures already exerted on the reef by problems such as bleaching and climate change. Many environmental organizations, universities, government agencies, and members of the public have contributed a vast amount of information and resources to help understand and monitor COTS outbreaks. On the other hand, new outbreaks occur in 1–15-year cycles, making it impossible to pinpoint exact causes or even keep its numbers under control. COTS outbreaks have been related to a variety of reasons, including ocean "stressors" such as surges in ocean nutrients generated by coastal and agricultural run-off into the ocean, as well as a loss of predators due to overfishing, according to decades of studies on the Great Barrier Reef. In this study, I trained and evaluated the COTS detection model (YOLOv5 small version 6) with the You Only Live Once algorithm which is used for embedded systems and mobile devices. It has a state-of-the-art convolutional neural network (CNN) at its core with the required configuration that parameters were optimized. On top of that, I applied advanced data augmentation methods for enhancing the quality and quantity of the dataset from 'The CSIRO Crown-of-Thorn Starfish Detection Dataset' [1].

2. Methodology

Deep learning has been well-known in recent years for its capacity to learn from experience and is now being utilized to solve complicated issues. Deep convolutional neural networks (CNNs) have made significant progress in large-scale object recognition in particular [7]. Figure 2 shows the workflow of a deep CNN-based object detection system with components.

Figure 1: Crown-Of-Thorns Starfish (COTS) on coral.

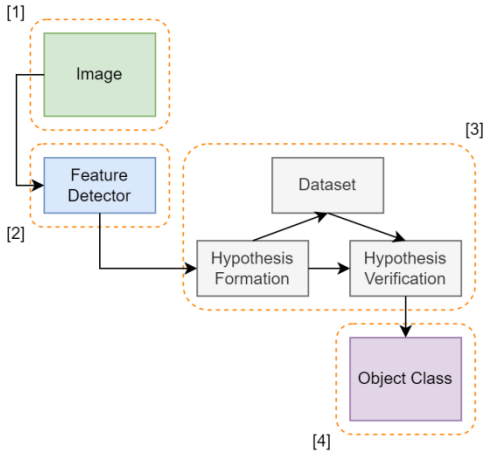


Figure 2: Deep CNN-based object detection system's workflow.

2.1. You Only Look Once algorithm (YOLO)

YOLO is a one-stage object detection algorithm that has been around for a long time. It transforms the detection issue into a regression problem. Instead of extracting RoI, it uses the regression approach to obtain the bounding box coordinates and probability of each class. It considerably enhances detection speed when compared to faster R-CNN. It doesn't employ a region proposal or a sliding window like other networks; It reframes object detection as a single regression problem. After only one

look at the input image, YOLO turns it into a grid of S x S cells. Each grid cell predicts B bounding boxes and a confidence score that shows if the predicted bounding box intersects with the ground truth bounding box and whether the predicted bounding box includes some objects:

$$Confidence = Pr(object) * IOU_{pred}^{truth} \quad (1)$$

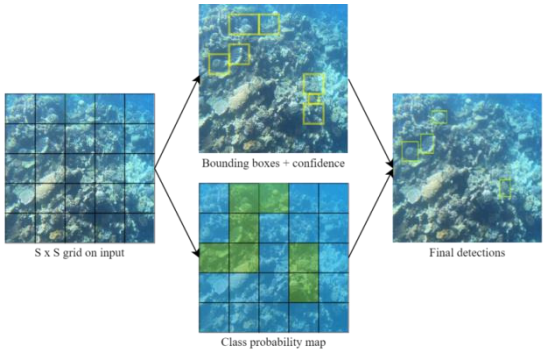


Figure 3: YOLO algorithm's anchor box.

The YOLO network predicts bounding boxes as deviations from a set of 'anchor box' dimensions to produce box predictions.

2.2. YOLOv5 Network

YOLOv5 outperforms all prior versions in terms of speed and accuracy. YOLOv5 version 6 was applied for this research. The YOLOv5 algorithm adjusts the width and depth of the backbone network using the depth-multiple and width-multiple parameters, yielding five different models: YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x.

As a backbone, it uses CSPDarknet53. This backbone handles the repeating gradient information in big backbones and incorporates gradient change into feature maps, which speeds up inference, improves accuracy, and decreases the model size by lowering parameters. It boosts information flow by using a path aggregation network (PANet) as a neck. PANet uses a novel feature pyramid network (FPN) with many bottom-up and top-down layers. This enhances the model's low-level feature propagation. PANet increases the object's localization accuracy by improving localization in lower levels. Moreover, the focus structure using CSPdarknet53 as a backbone reduces the amount of CUDA memory required, increases forward propagation, and decreases back-propagation.

2.3. TensorFlow Lite

TensorFlow Lite was employed in this research for a variety of reasons:

- Light-weight: In terms of storage and compute capacity, embedded devices have minimal resources. Deep learning models consume a lot of resources, thus the models we deploy on embedded devices should be light and have reduced binary sizes.
- Low Latency: Regardless of network connectivity, deep learning models on the embedded systems should produce faster inferences irrespective.
- Pre-trained: Models can be trained on-premise or cloud for various deep learning tasks. Such as image classification, object detection, speech recognition, etc. and can be simply deployed to make inferences.

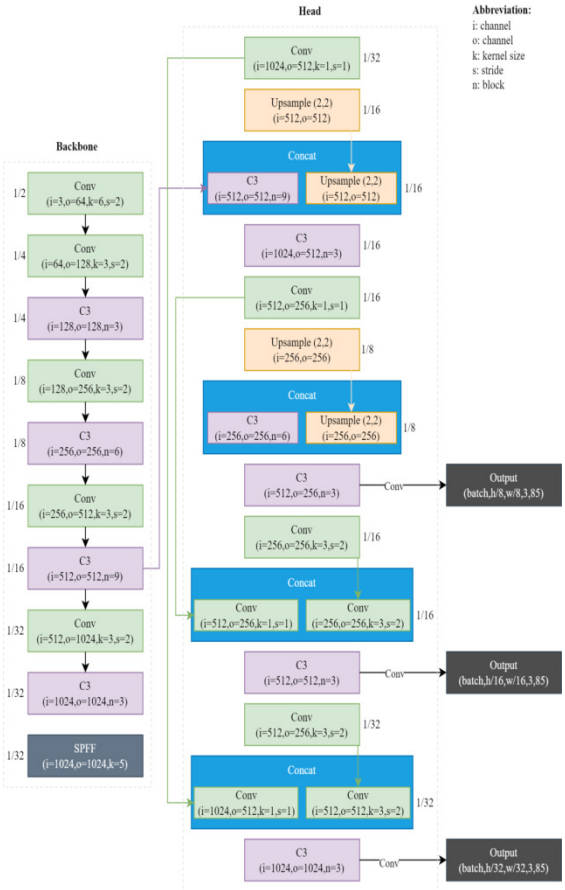


Figure 4: YOLOv5s version 6 network architecture.

It has all the capabilities needed to make inferences on the embedded systems. It is a cross-platform, open-source deep learning framework that transforms a pre-trained model into its format. It is a specific format model that is efficient in terms of performance and a lightweight version that will take up less space, these features make it ideal for use on mobile and embedded devices. Therefore, I decided to use it because it is not only easy to use but also optimized to execute.

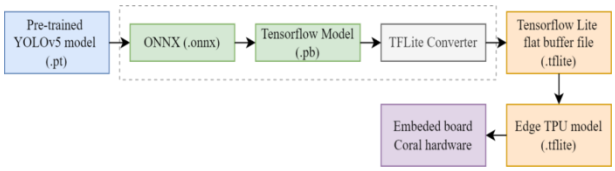


Figure 5: YOLOv5 to TensorFlow Lite weight conversion proces.

The TensorFlow Lite model can be deployed on mobile devices. Such as Android and iOS, or on embedded devices like Raspberry and Microcontrollers in general. The following are the steps to making an inference from embedded devices:

- Initialize the interpreter and load the model into it.
- Allocate the tensor and obtain the input and output tensors.

- Preprocess the images by reading them into a tensor.
- Invoke the interpreter to make the inference on the input tensor.
- Get the image's result by mapping the inference's results.

3. Experiments

The flowchart below depicts the experiment steps in my proposed research:

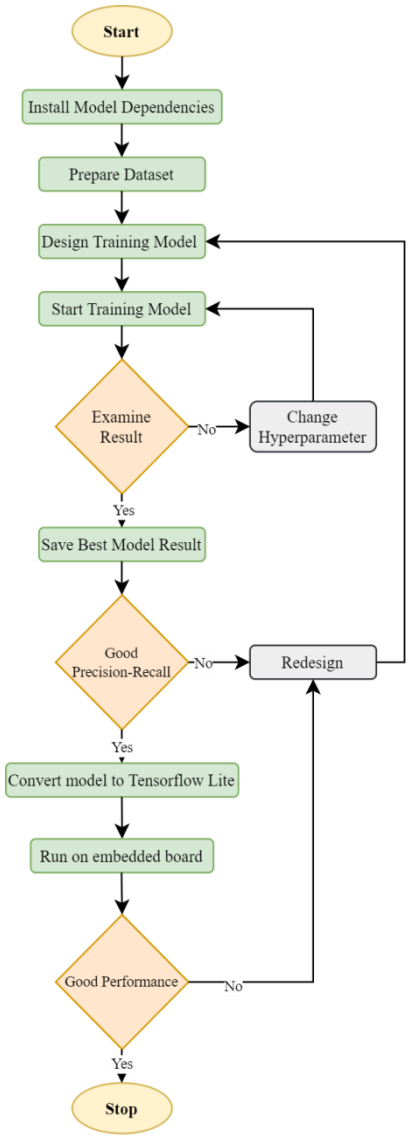


Figure 6: Flowchart of the proposed research.

3.1. Data Collection and Processing

The dataset [1] was gathered by a CSIRO team using a GoPro Hero9 camera that had been modified for use in the Manta Tow technique. The camera was specifically affixed to the bottom of the manta tow board, which the snorkeller-diver held during the surveys. As the diver explores the reef, the camera displays an oblique field of view of the reef below him, with the distance to the reef constantly changing. The distance from the bottom is usually several meters, although it might be as little as a few tens of centimeters or as much as 10 meters or more. The survey boat travels at a top speed of 5 knots and pulls the diver for two minutes, covering a distance of around 200 meters. The boat then comes to a complete halt to allow the diver to record data collected throughout the transect on a sheet of paper. Expert annotators used pre-trained COTS detection models to identify every COTS in the photos, which was followed by an AI-assisted annotation and quality assurance procedure. Using the annotation program, all of the COTS detection in a picture was marked with a box. The data was gathered on a reef in the Swain Reefs section of the GBR on a single day in October 2021. Lighting, visibility, coral habitat, depth, distance from the bottom, and viewpoint all vary.

The collection comprises underwater picture sequences recorded at five distinct locations on a reef in the GBR. There are almost 35000 photos in all, with hundreds of individual COTS visible. However, only 4888 photos, including COTS, which I utilized to augment for the dataset to train the proposed model.

The size of the original images has been normalized to 640x640 to give a detection technique that fulfills the requirement for real-time and precise operation. This boosts the detection speed without deleting any important data in the image. The normalized photos have also been divided into two groups: a test set and a training set. After including the detection model's anchor box size for clustering purposes, the pre-processing criteria would be finished. I employed certain processes for data augmentation. Such as rotation, horizontal and vertical shear, mosaic. Those approaches do not change the image's pixel values; they merely change the image's position. As a result, the learning ability of the convolutional neural network would be modified to learn high-quality features while avoiding overfitting. Table 1 depicts the description of the augmented dataset after my data processing step. Figure 7 depicts a flowchart of the data augmented implementation process.

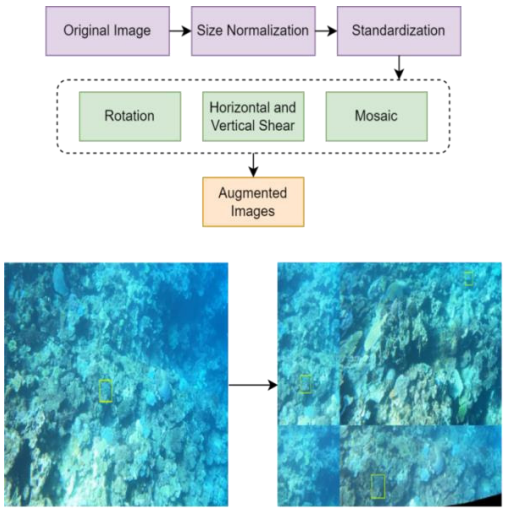


Figure 7: Data augmentation proces.

Table 1: Augmented dataset summary

Total Image	Size	Class	Labelled box
14.646	640x640	COTS	40.000

3.2. Training

Model training was carried out on 2 GPUs - NVIDIA GeForce RTX 2080 Ti (11GB-GDDR6).

Algorithm 1: Training model

```
Input: Algorithm (YOLOv5s) with installed necessary libraries and
       dependencies
Output: COTS detection model weights
Data: Dataset (Train/Val/Test)
start
for YOLOv5s Model do
    Train Dataset
    (img 640 –batch 24 –epochs 100 –device 0,1)
    Evaluate The model
    Compute From TensorBoard
    (Precision, Recall, F1-score, mAP)
    Display Training Performance
    Run Model inference on test images
    Visualize Inference on test images
    Save Model weights
end
```

4. Results

4.1. Evaluation metrics

To evaluate YOLOv5's algorithm, F1score and mAP were employed. The F1score is the harmonic mean of accuracy and recall. It is also the model's test accuracy. The maximum possible F1score value is 1, indicating flawless accuracy and memory, while the lowest possible score is 0, indicating that either precision or recall is zero. Furthermore, mAP is determined by averaging the average precision (AP), where q is the number of queries and AveP(q) is the average precision for that particular query. The mean of AP may then be used to determine mAP. mAP may also be thought of as a metric for calculating the accuracy of machine learning algorithms. In COTS detection, True Positive (TP) is the predicted COTS was correctly detected and the actual class was the same as the predicted one. False Positive (FP) is the model that predicted COTS and the actual class was not. False Negative (FN) is predicted class was not COTS and the actual class was COTS. True Negative (TN) is the predicted class was not COTS and the actual class was also not COTS.

$$F1_{score} = \frac{2*Precision*Recall}{Precision+Recall} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$mAP = \sum_{q=1}^{Q} \frac{AveP(q)}{Q} \quad (5)$$

It is worth noting that precision is computed as the ratio of true prediction to the total number of predictions. For instance, if a model generates 50 predictions and all of them are true, the precision is 100%. Precision does not take into account the actual number of true objects present in an image; whereas recall computes the ratio of true predictions to the total number of objects in an image. For example, if a model detects 75 true objects and the image contains 100 true objects, recall is determined to be 75%. The presence of simply high precision or only high recall does not imply that the model is a good one. It must strike a balance between precision and recall for an object detection algorithm to be considered outstanding. Therefore, we use the F1score to determine if a model is a good one or not.

4.2. Model's weight information

Table 2: Model's weight information

Item	Information
Model	YOLOv5s.pt
Precision	0.93
Recall	0.77
F1score	0.84
mAP(%)	83
Training time	4 hours + 53 minutes

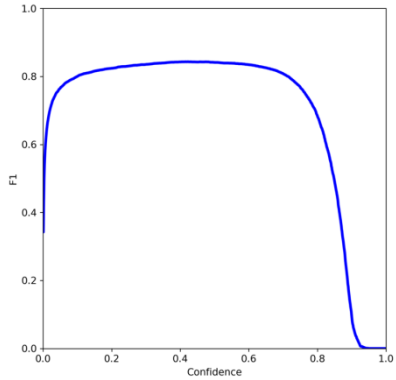


Figure 8: Model's $F1_{score}$ curve.

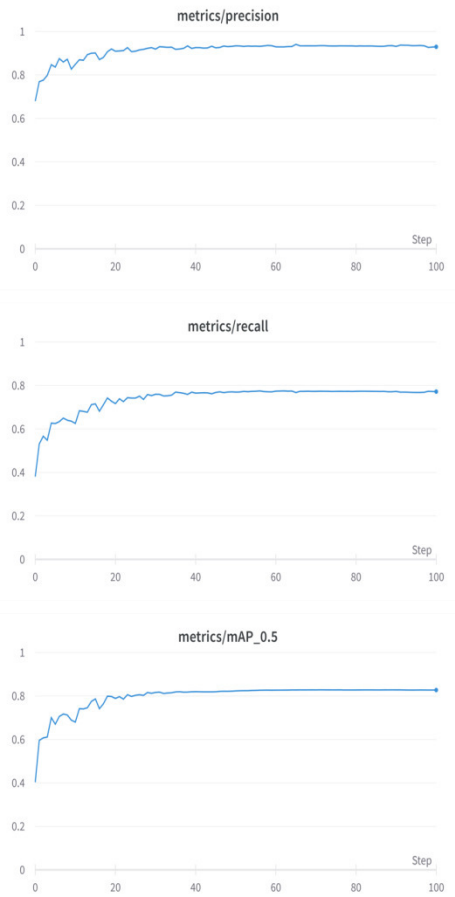


Figure 9: Model's Precision-Recall-mAP.

4.3. Model's performance on embedded system

Figure 10: Setup of the proposed embedded system: 1) Odroid XU4 - 2) Microsoft Camera C3000 - 3) Google Edge TPU coprocessor - 4) Adafruit 7-inch monitor.

Table 3: Proposed embedded system in detail

Item	Property
Microcontroller	Odroid XU4
Identifier	ARM implementer 65 architecture 7 variant 2 part 3087 revision 3
CPU	Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa-core
GPU	ARM Mali-T628 6 Cores
RAM	2GB LPDDR3
TPU	Google Edge Coral
Camera	Mircosoft C3000 (720p)
Monitor	Adafruit 7 inch (720p)
OS	Ubuntu 18.04 MATE
Language	Python 3.7.3

Table 4: Performance on the proposed embedded system

Item	Property
Model	COTS.tflite
Model size	7.3 MB
Input size	640x480
FPS	30
Inference time (Single-core)	30 milliseconds (0.03 seconds)
Inference time (Multiple-cores)	8 milliseconds (0.008 seconds)
Accuracy	93%

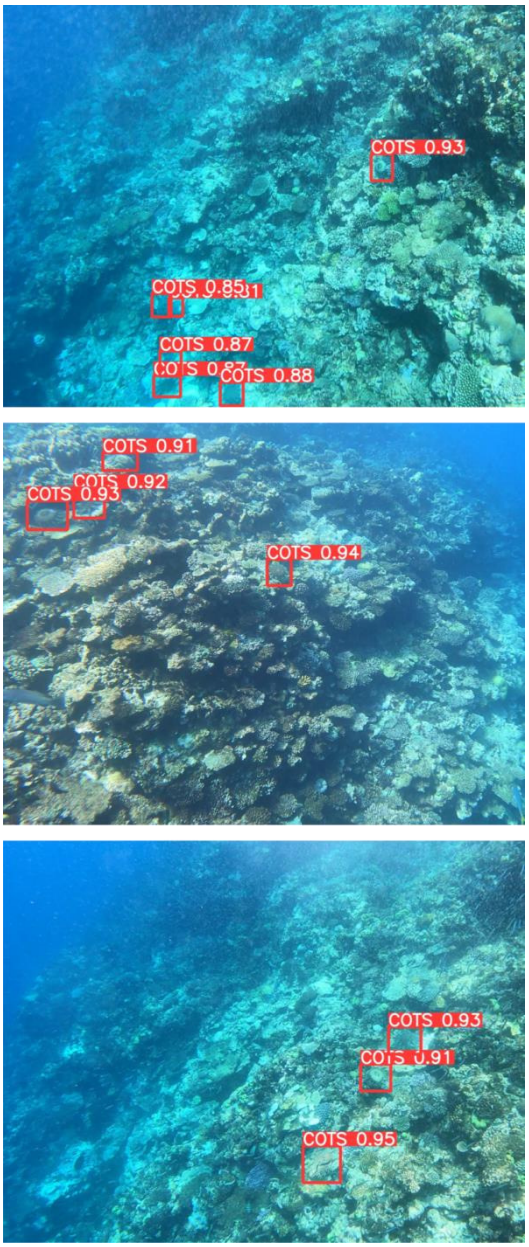


Figure 12: COTS detection inferences on the proposed system.

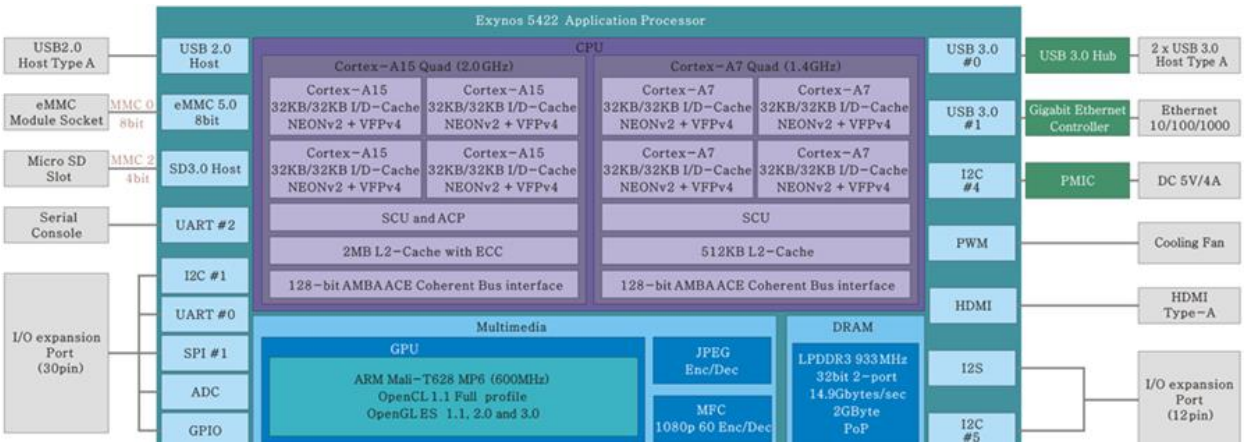


Figure 11: Proposed embedded board block diagram.

5. Conclusion

In this paper, I provided a Crown-of-thorns starfish (COTS) detection system based on the YOLOv5 algorithm. This approach can identify COTS in real-time, videos or images, and recognize whether it appears on the scene. I trained a model to provide a real-time system, but still, be able to maintain high accuracy. Furthermore, the size of my model is quite favorable; it is only 7.3 MB in size. It can undoubtedly be applied to real-world scenarios. It demonstrated that a deep CNN can indeed be applied to the task with promising results in evaluated metrics. I expect that the outcomes will be far better than those shown here with more training data. A restriction of embedded system configuration is also a concern. In the future, I will do research to develop a system that balances the needed configuration and model performance.

6. Acknowledgment

I would like to send my sincere appreciation to HAIL (Hongik University Artificial Intelligence Laboratory) which is advised by Prof. Seongwon Cho for supporting me in this research.

References

[1] L. Jiajun, K. Brano, M. Ross, D. Brendan, M. Torsten, C. Joey, S. Andy, H. Nic, V. R. Karl, T. S. Lachlan, A. A. David, A. A. Mohammad, C. Geoffrey, B. Russ, M. Peyman, S. Daniel, D. Tim, E.M. Kemal, W. Martin, M. Megha, The CSIRO Crown-of-Thorn Starfish Detection Dataset, arXiv, 2021, https://doi.org/10.48550/arXiv.2111.14311

[2] W. Junlong, K. Wei, Z. Wei, H. Fengbiao, T. Xuefeng, W. Qiong, Helmet Detection Algorithm Based on the Improved YOLOv5 and Dynamic Anchor Box Matching, Proceedings of the IEEE International Conference on Emergency Science and Information Technology (ICESIT), (2021) 79-83, http://dx.doi.org/10.1109/ICESIT53460.2021.9696525.

[3] Y. Zhong, J. Wang, J. Peng, L. Zhang, Anchor Box Optimization for Object Detection, Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV), (2020) 1275-1283, http://dx.doi.org/10.1109/WACV45572.2020.9093498.

[4] T. F. Dima, M. E. Ahmed, Using YOLOv5 Algorithm to Detect and Recognize American Sign Language, Proceedings of the International Conference on Information Technology (ICIT), (2021) 603-607, http://dx.doi.org/10.1109/ICIT52682.2021.9491672.

[5] G. Verma, Y. Gupta, A. M. Malik, B. Chapman, Performance Evaluation of Deep Learning Compilers for Edge Inference, Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), (2021) 858-865, http://dx.doi.org/10.1109/IPDPSW52791.2021.00128.

[6] T. Zhi, S. Chunhua, C. Hao, H. Tong, FCOS: Fully Convolutional One-Stage Object Detection, arXiv, 2019, https://doi.org/10.48550/arXiv.1904.01355.

[7] L. Wei, A. Dragomir, E. Dumitru, S. Christian, R. Scott, F. Cheng-Yang, B. C. Alexander, SSD: Single Shot MultiBox Detector, Lecture Notes in Computer Science, 2016, https://doi.org/10.1007/978-3-319-46448-0_2.

[8] Z. Ni, J. Chen, N. Sang, C. Gao, L. Liu, Light YOLO for high-speed gesture recognition, Proceedings of The 2018 25th IEEE International Conference on Image Processing (ICIP), (2018) 3099-3103, http://dx.doi.org/10.1109/ICIP.2018.8451766.

[9] A. Aleena, S. Ayesha, J. Tauseef, U.K. Asif, Small Object Detection using Deep Learning, arXiv, 2022, https://doi.org/10.48550/arXiv.2201.03243.

[10] B. G. Han, J. G. Lee, K. T. Lim, D. H. Choi, Design of a scalable and fast YOLO for edge-computing devices, Sensors, 2020, https://doi.org/10.3390/s20236779.

[11] B. Liang, S. Wu, K. Xu, J. Hao, Butterfly detection and classification based on integrated YOLO algorithm, arXiv, 2020, https://doi.org/10.48550/arXiv.2001.00361.

[12] C. Shaobin, L. Wei, Embedded System Real-Time Vehicle Detection based on Improved YOLO Network, Proceedings of the IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), (2019) 1400-1403, http://dx.doi.org/10.1109/IMCEC46724.2019.8984055.

[13] Y. Zhu, C. Yao, X. Bai, Scene text detection and recognition: recent advances and future trends, Frontiers of Computer Science, 2015, http://dx.doi.org/10.1007/s11704-015-4488-0.

[14] Q. Lu, Y. Yuan, Improved YOLO Algorithm for Object Detection in Traffic Video, Proceedings of the International Conference in Communications, Signal Processing and Systems, 2019, http://dx.doi.org/10.1007/978-981-13-9409-6_198.

[15] R. Shaoqing, H. Kaiming, G. Ross, S. Jian, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, arXiv, 2016, https://doi.org/10.48550/arXiv.1506.01497.

[16] H. Shijie, W. Zhonghao, S. Fuming, LEDet: A Single-Shot Real-Time Object Detector Based on Low-Light Image Enhancement, The Computer Journal, 2021, https://doi.org/10.1093/comjnl/bxab055.

[17] A. A. Choudhury, R. Saha, S. Z. Shoumo, S. R. Tulon, J. Uddin, M. K. Rahman, An efficient way to represent braille using YOLO algorithm, Proceedings of the Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV), (2018) 10-19, http://dx.doi.org/10.1109/ICIEV.2018.8641038.

[18] D. Volivier, M. Saïd, A. M. Sidi, M. Pierre, L. Frédéric, A new Edge Architecture for AI-IoT services deployment, Proceedings of the 17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC), (2020) 10-19, https://doi.org/10.1016/j.procs.2020.07.006.

[19] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, (2012) 1097-1105, https://doi.org/10.1145/3065386.

[20] D. S. Viraktamath, P. Navalgi, A. Neelopant, Comparison of YOLOv3 and SSD Algorithms,

Proceedings of the International Journal of Engineering Research & Technology (IJERT), (2021) 1156–1160.

[21] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, (2016) 779–788, http://dx.doi.org/10.1109/CVPR.2016.91.

[22] A. Bochkovskiy, C. Y. Wang, H. Y. M. Liao, YOLOv4: Optimal speed and accuracy of object detection, arXiv, 2020, https://doi.org/10.48550/arXiv.2004.10934.

[23] B. Jiang, R. Luo, J. Mao, T. Xiao, Y. Jiang, Acquisition of Localization Confidence for Accurate Object Detection, In Proceedings of the European conference on computer vision (ECCV), (2018) 8-14, http://dx.doi.org/10.1007/978-3-030-01264-9_48.

[24] Z. Zheng, P. Wang, D. Ren, W. Liu, R. Ye, Q. Hu, W. Zuo, Enhancing Geometric Factors in Model Learning and Inference for Object Detection and Instance Segmentation, arXiv, 2021, https://doi.org/10.48550/arXiv.2005.03572.

[25] K. K. Reddy, M. Shah, Recognizing 50 human action categories of web videos, Machine Vision and Applications, (2016) 971-981, https://doi.org/10.1007/s00138-012-0450-4.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 112–117
Received: 15 March 2022
Accepted: 5 April 2022

An Analysis of the Knowledge about the Aspects of Cybersecurity and Two-Factor Logging in the Society

Analiza wiedzy o aspektach cyberbezpieczeństwa i logowania dwuetapowego w społeczeństwie

Kamil Piłat*, Michał Tomasz Pawłowski*, Grzegorz Kozieł

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article presents the course of research on the safe use of the Internet, carried out on a group of 200 respondents aged 10-60 and more. Particular attention was paid to the threat of ransomware and other threats on the Internet. Computer-aided survey research was carried out. Survey results were analyzed using a frequency analysis and a Pearson-Spearman correlation analysis. The conducted research showed an unsatisfactory level of knowledge in the field of cybersecurity for the age groups 10-15 and 60 and more, and it was sufficient for the age groups 20-24. The problem has become especially topical given the impact of the COVID-19 pandemic on teaching and remote work and remote communication in families.

Keywords: cybersecurity; two-factor login; ransomware; electronic signature

Streszczenie

W artykule przedstawiono przebieg badań dotyczących bezpiecznego korzystania z Internetu przeprowadzonych na grupie 200 respondentów w wieku 10-60 i więcej lat. Szczególną uwagę zwrócono na zagrożenie oprogramowaniem szpiegującym i inne zagrożenia w sieci Internet. Przeprowadzono badania ankietowe wspomagane komputerowo. Wyniki ankiet były analizowane z użyciem analizy częstości oraz analizy korelacji Pearsona i Spearmana. Przeprowadzone badania wykazały niezadowalający poziom wiedzy z zakresu cyberbezpieczeństwa dla grup wiekowych 10-15 oraz 60 i więcej lat oraz wystarczający dla grup wiekowych 20-24 lat. Problem stał się szczególnie aktualny, biorąc pod uwagę wpływ pandemii COVID-19 na nauczanie i pracę zdalną oraz komunikację zdalną w rodzinach.

Słowa kluczowe: cyberbezpieczeństwo; logowanie dwuetapowe; oprogramowanie szantażujące; podpis elektroniczny

*Corresponding author

Email address: kamil.pilat2@pollub.edu.pl (K. Piłat), michal.pawlowski1@pollub.edu.pl (M. T. Pawłowski)

1. Wprowadzenie

We współczesnym świecie znaczącą rolę odgrywa dostęp do sieci Internet. Wraz z każdym kolejnym rokiem dostęp do niej staje się coraz bardziej powszechny. Obecnie około 60% populacji korzysta z Internetu [1]. Ten fakt wiąże się z dostępem do różnego rodzaju stron internetowych, usług świadczonych drogą elektroniczną, ale także z rozrywką np. gry komputerowe, usługi przesyłania strumieniowego itp. Taka obfitość możliwości oferowanych społeczeństwu pozwala na realizację wielu pożytecznych celów. Warto jednak wspomnieć, że każde działanie w Internecie obarczone jest ryzykiem, które może wystąpić lub można mu zapobiec. Zarówno jedna jak i druga możliwość jest bardziej lub mniej prawdopodobna w zależności od stanu wiedzy jednostki na temat bezpiecznej obsługi komputera oraz poruszania się po sieci.

Kluczowym zagadnieniem, które powinno być bliskie każdemu użytkownikowi komputera oraz Internetu jest cyberbezpieczeństwo. Termin ten wskazuje metody oraz możliwości, jak ochronić się przed różnego rodzaju zagrożeniami takimi jak np. phishing, wirus komputerowy, kradzież tożsamości, spyware itp. W obecnych czasach świadomość czyhających niebezpieczeństw jest bardzo istotna dla każdego użytkownika - zarówno dla osób młodszych jak i starszych. Każdy człowiek może paść ofiarą internetowego przestępcy. Konsekwencje zaistniałej sytuacji mogą być różne - od wycieku danych wrażliwych, poprzez przejęcie kontroli nad sprzętem lub kontem bankowym po szantaż i zastraszanie osoby [9].

W innych publikacjach można spotkać się z analizami wskazującymi na wykluczenie cyfrowe osób starszych, szczególnie żyjących na wsi [2], jak również skrajność tego zjawiska – cyfrowe dzieciństwo osób młodych lub dzieci, które nie wyobrażają sobie funkcjonowania we współczesnym świecie bez dostępu do Internetu [3]. Chcąc zweryfikować obie te skrajności jak również stany pośrednie czy występują w społeczeństwie postanowiono przeprowadzić badanie. Jego celem było zbadanie obecnego stanu wiedzy Polaków w zakresie bezpiecznego korzystania z sieci Internet oraz świadomości zagrożeń, jakie występują.

Na potrzeby badań postawiono następujące tezy badawcze:

T1: Świadomość i podatność na zagrożenia wynikające z korzystania z Internetu jest najmniejsza dla skrajnych grup wiekowych (najmłodsi i najstarsi użytkownicy)

T2: Osoby młodsze (wiek 18-29) mieszkające w mieście mają wyższy poziom wiedzy z zakresu cyberbezpieczeństwa niż osoby starsze (40+) mieszkające zarówno w mieście jak i na wsi. Wyniki w identycznych przedziałach wiekowych są zbliżone.

2. Przegląd literatury

Portal internetowy dobreprogramy.pl udostępnił wyniki badań ankiety dotyczącej cyberbezpieczeństwa w społeczeństwie. Na jej podstawie można wykonać analizę porównawczą danych z jednego jak i z drugiego badania. Po wstępnym porównaniu wyników można stwierdzić, że przeprowadzony scenariusz eksperymentu w ramach pracy magisterskiej nie odbiega znacząco od badania przeprowadzonego przez dobreprogramy.pl.[12]

Strona internetowa controlengineering.pl przeprowadziła ankietę na temat cyberbezpieczeństwa w systemach sterowania w 2016 roku. We wnioskach można przeczytać, że według 37% respondentów największym zagrożeniem jest złośliwe oprogramowanie, 21% badanych osób martwi się możliwością ataku przez hakerów. Co więcej sześciu na dziesięciu respondentów doświadczyło w ciągu ostatnich 24 miesięcy zarejestrowanych incydentów związanych ze złośliwymi cyberatakami [13].

O tym jak bardzo istotna jest ochrona informacji w sieci można było przekonać się na przykładzie służb wojskowych. W związku z wyciekiem danych z Wojska Polskiego przez aplikację Pegasus, można było zdobyć dane o stanach magazynowych polskiej armii [14].

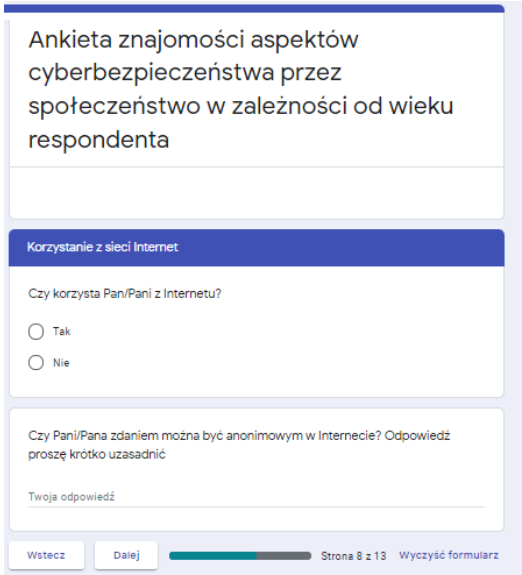
Sprawdzono również jakie skutki może spowodować wyciek danych. Może być to utrata zaufania i wizerunku przez instytucję, duże kary finansowe. Takie wycieki danych mogą powodować kilkuset milionowe straty tak jak to było w przypadku spółki Heartland Payment Systems w 2009 roku i wycieku danych kart płatniczych jej klientów. Straty zostały oszacowane na ponad 100 milionów dolarów [15].

3. Materiały i metody

Eksperyment zrealizowano za pomocą badań ilościowych. W omawianym przypadku zastosowano technikę badań ankietowych wspomaganych komputerowo (CAPI) na grupie 200 respondentów w różnych grupach wiekowych. Aby właściwie przeprowadzić eksperyment przygotowano kwestionariusz ankiety sporządzony w Google Forms. Badanie było anonimowe i zostało przeprowadzone jednorazowo – istotny w tym badaniu jest stan obecny wiedzy respondentów, bez porównywania w okresie czasowym. Poniżej zaprezentowano scenariusz badawczy:

1. Udostępnienie linku do ankiety elektronicznej
2. Respondent otwiera link – otwiera się ankieta
3. Respondent zapoznaje się z opisem na stronie tytułowej, następnie klika przycisk „Dalej”
4. Prezentowana jest ankieta właściwa z pytaniami:
   a) Metryczka (płeć, przedział wiekowy, zawód, wielkość miejsca zamieszkania)
   b) Sekcje:
      - Poziom znajomości informatyki;
      - Korzystanie z sieci Internet;
      - Zagrożenia w sieci;
      - Płatności elektroniczne;
      - Bankowość elektroniczna;
      - Usługi zdrowotne w czasach pandemii;
      - Bezpieczeństwo informacji.
5. Respondent kończy ankietę klikając na przycisk „Wyślij”

Oprócz zaprezentowanego scenariusza, z racji ograniczeń niektórych respondentów wzięto pod uwagę scenariusz alternatywny, który umożliwiał dodatkowo skorzystanie ze wsparcia asystenta – docelowo przeznaczony dla osób z grupy wiekowej 60+. Na Rysunku 1 przedstawiono wybrane pytania z ankiety badawczej:



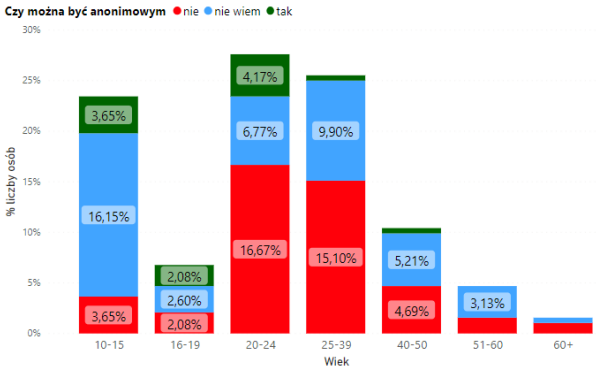
Rysunek 1: Pytanie dotyczące anonimowości w Internecie.

4. Rezultaty i dyskusja

W wyniku przeprowadzonych badań, które poddano analizie uzyskano następujące rezultaty:

4.1. Wykresy

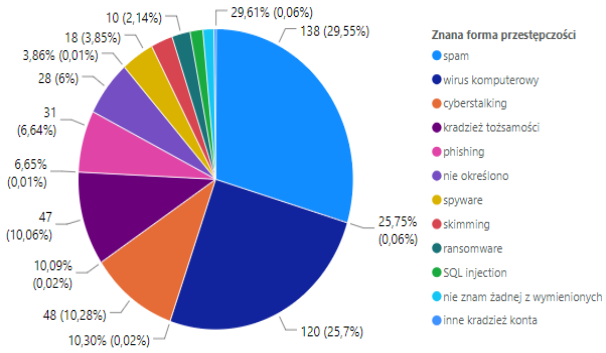
Na wykresie przedstawionym na Rysunku 2 zaprezentowano zależność odpowiedzi na pytanie „Czy można być anonimowym w Internecie?” względem badanych grup wiekowych



Rysunek 2: Wykres zależności wiedzy o anonimowości względem wieku.

Z przeprowadzonych badań wynika że największy procent osób uważa że nie można być anonimowym w Internecie – szczególnie dominują tu osoby z przedziału wiekowego 20-24 (16,67% wszystkich badanych) i 25-39 lat (15,10% wszystkich badanych). Największy odsetek osób nie mających wiedzy w tym zagadnieniu jest w wieku 10-15 lat – te osoby stanowią 16,15% wszystkich badanych osób.

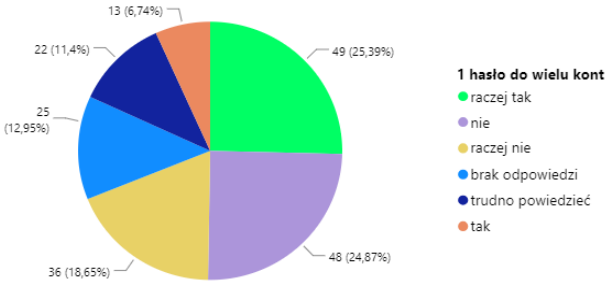
Kolejnym zagadnieniem jakie sprawdzono była wiedza z zakresu znanych form przestępczości. Rezultat przedstawiono na wykresie na Rysunku 3.



Rysunek 3: Znane formy przestępczości – wyniki ogólne.

Ankietowani najczęściej wskazywali na znajomość spamu (29,55%) oraz wirusa komputerowego (25,7%). Najmniej znane były zagrożenia, których nazwy dotyczą oprogramowania szantażującego (ransomware – 2,14%) i ataków na strony lub aplikacje internetowe (SQL injection – 1,5%).

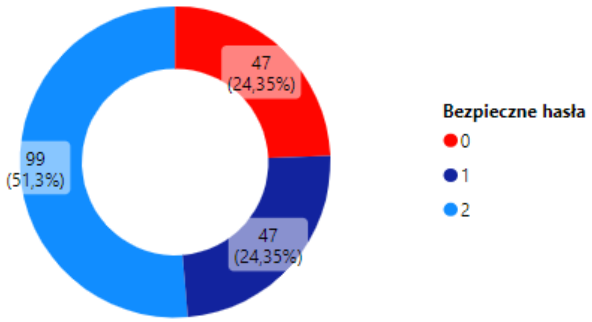
Jak widać na Rysunku 4 blisko 32% ankietowanych zaznaczyło odpowiedź "Raczej tak" lub "Tak", z drugiej strony blisko 42% osób zaznaczyło odpowiedź "Nie" lub "Raczej nie".



Rysunek 4: Czy posiada Pan/Pani jedno hasło do większości kont internetowych? - wyniki ogólne.

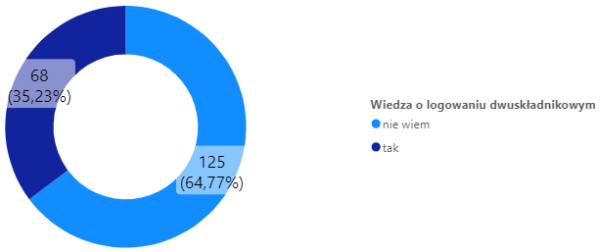
Kolejne pytanie dotyczyło znajomości ustawiania silnych haseł. Ustalono, że maksymalna liczba punktów do zdobycia za w pełni poprawną odpowiedź to 2 punkty. Każda błędna odpowiedź zmniejszała tę wartość o 1 punkt.

Prawie 52% osób uzyskało maksymalną liczbę punktów za to pytanie, 1 punkt zdobyło ponad 24% respondentów (Rysunek 5). Wyniki pomiędzy kobietami a mężczyznami praktycznie się nie różnią.

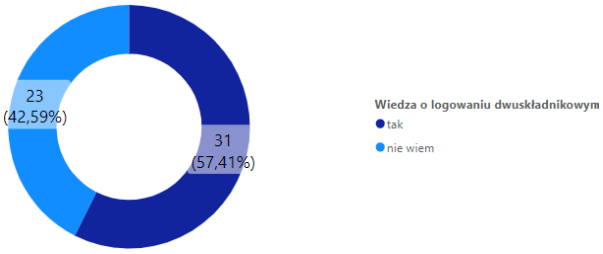


Rysunek 5: Wskazanie poprawnych haseł - wyniki ogólne.

Prawie 65% respondentów nie wie czym jest logowanie dwuskładnikowe (Rysunek 6). Analizując poszczególne grupy wiekowe można zauważyć, że wszystkie mają podobne wyniki do wyników ogólnych, z wyjątkiem przedziału wiekowego 20-14 lat (Rysunek 7), w którym 57% ankietowanych wie co to jest logowanie dwuskładnikowe. Największą wiedzę z grupy zawodów mają studenci bo 67% studentów wie co to jest logowanie dwuskładnikowe. Analizując wielkość zamieszkania respondentów można stwierdzić, że osoby mieszkające w mieście powyżej 300 tys. posiadają dużo większą wiedzę z zakresu logowania dwuskładnikowego ponieważ ich wiedza na ten temat jest wyższa o blisko 14 punktów procentowych większa od osób mieszkających w miastach o wielkości 30 tys. do 300 tys., natomiast porównując tę wiedzę do pozostałych grup osób to wynik jest lepszy o blisko 32 punkty procentowe.

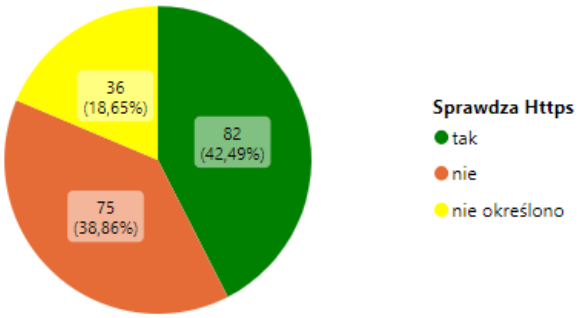


Rysunek 6: Wiedza o logowaniu dwuskładnikowym - wyniki ogólne.



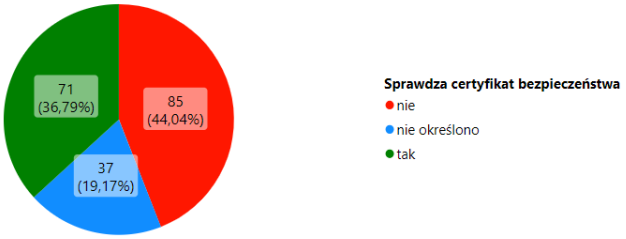
Rysunek 7: Wiedza o logowaniu dwuskładnikowym - wyniki grupy wiekowej 20-24.

Respondenci na pytanie odnośnie sprawdzania przedrostka HTTPS na stronie internetowej banku odpowiedzieli "Tak" w 42%. Prawie 19% ankietowanych nie odpowiedziało na to pytanie, może to oznaczać, że nie znają tego zagadnienia (Rysunek 8).

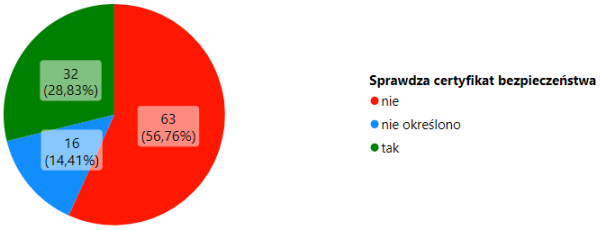


Rysunek 8:Sprawdzanie przedrostka HTTPS na stronie internetowej banku - wyniki ogólne.

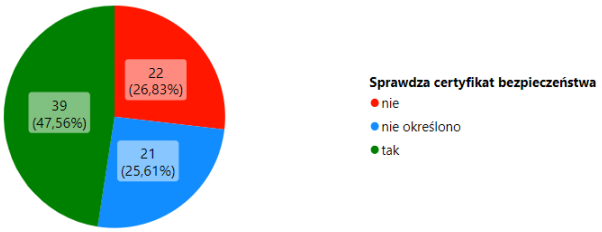
Tylko nieco ponad 44% ankietowanych sprawdza certyfikat bezpieczeństwa po wejściu na stronę internetową swojego banku (Rysunek 9). Warte zauważenia jest, że 56,76% kobiet (Rysunek 10) nie sprawdza certyfikatu bezpieczeństwa na stronach internetowych banków, natomiast 47,56% mężczyzn sprawdza wspomniane certyfikaty bezpieczeństwa (Rysunek 11).



Rysunek 9: Sprawdzanie certyfikatu bezpieczeństwa na stronie internetowej banku - wyniki ogólne.

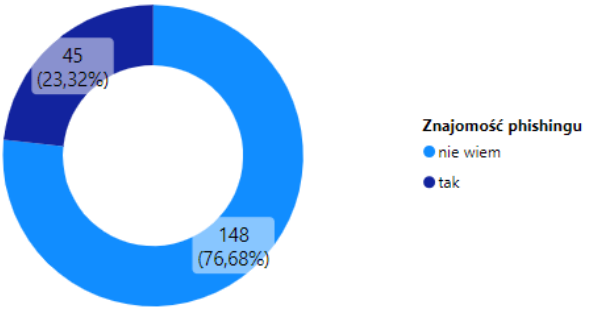


Rysunek 10: Sprawdzanie certyfikatu bezpieczeństwa na stronie internetowej banku - wyniki kobiet.



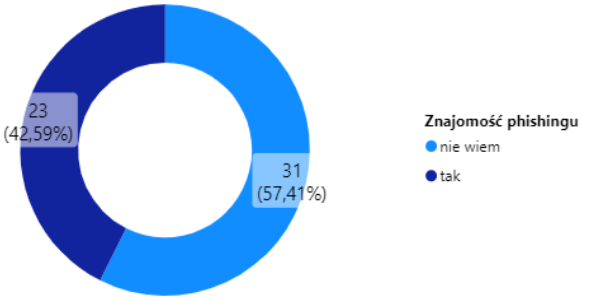
Rysunek 11: Sprawdzanie certyfikatu bezpieczeństwa na stronie internetowej banku - wyniki mężczyzn.

Zweryfikowano wiedzę z zakresu phishingu, aby respondenci podali jak rozumieją to pojęcie. Aby to sprawdzić poproszono o podanie krótkiej definicji, jak respondent rozumie dane zagadnienie. Tylko 23% ankietowanych było w stanie prawidłowo je wyjaśnić, pozostałe wyniki stanowią odpowiedzi błędne lub odpowiedzi "nie wiem" (Rysunek 12).



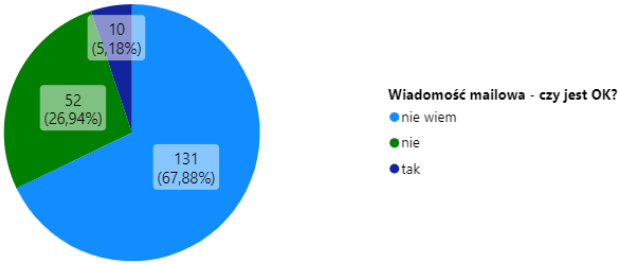
Rysunek 12: Czym jest phishing- wyniki ogólne.

Największą znajomością phishingu okazała się grupa wiekowa 20-24 lat (Rysunek 13) - blisko 43%. Podobny wynik uzyskały osoby z grupy student.

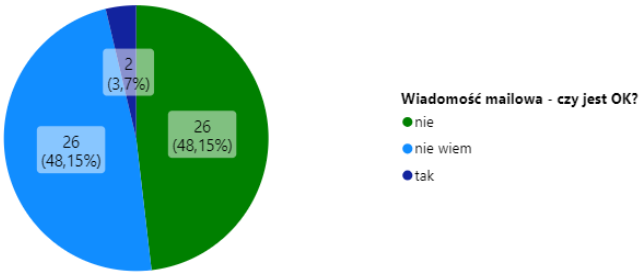


Rysunek 13: Czym jest phishing- wyniki grupy wiekowej 20-24 lat.

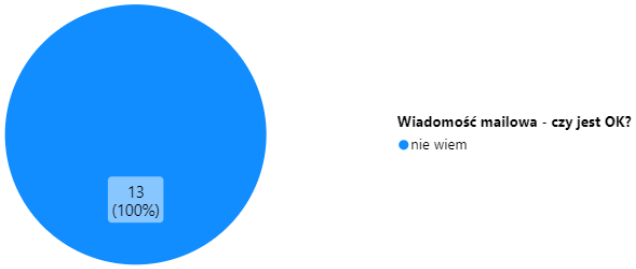
Następnie postanowiono sprawdzić czy respondenci są w stanie określić czy proponowana wiadomość jest próbką phishingu czy też nie. Odpowiedź prawidłowa na to pytanie brzmi – tak, jest to phishing. Dodatkowo ankietowani powinni wskazać dlaczego wydaje im się dlaczego tak wiadomość jest podejrzana lub też dlaczego nie jest. Tylko blisko 27% wskazało poprawną odpowiedź oraz wskazało elementy podejrzane, natomiast ponad 67% respondentów nie wiedziało, o co są pytani w tym pytaniu (Rysunek 14). Porównując wyniki poszczególnych grup wiekowych kolejny raz najlepszą znajomością danego zagadnienia okazała się grupa z przedziału wiekowego 20-24 lat – ponad 48% respondentów wskazało prawidłową odpowiedź i ją uzasadniło (Rysunek 15). Pozostałe grupy wiekowe odpowiadały w podobny sposób, z mniejszą częścią prawidłowych odpowiedzi, z wyjątkiem grupy 16-19 lat, w której nikt nie był w stanie określić czy ta wiadomość jest poprawna czy też nie (Rysunek 16).



Rysunek 14: Wiadomość mailowa phishing- wyniki ogólne.

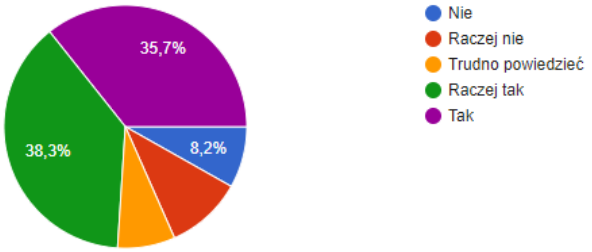


Rysunek 15: Wiadomość mailowa phishing- wyniki grupy wiekowej 20-24 lat.



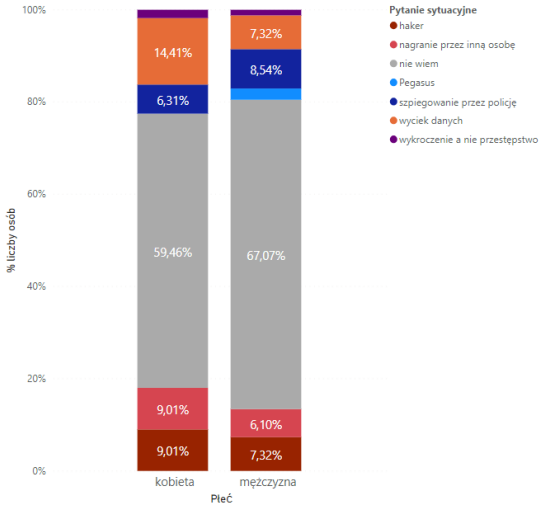
Rysunek 16: Wiadomość mailowa phishing- wyniki grupy wiekowej 16-19 lat.

Kolejne pytanie dotyczyło sprawdzania uprawnień przez użytkowników w instalowanych aplikacjach na ich telefonach. Można zauważyć, że 74% ankietowanych sprawdza lub raczej sprawdza uprawnienia instalowanych aplikacji internetowych (Rysunek 17).



Rysunek 17: Sprawdzenie uprawnień podczas instalowania aplikacji na telefonie- wyniki ogólne.

Ostatnie pytanie w tej sekcji jak i podsumowujące całą ankietę dotyczyło przeczytania pewnej historii, która wydarzyła się w życiu codziennym, i określenia co mogło spowodować taki a nie inny przebieg wydarzeń. Przytoczona historia miała na celu sprawdzić kreatywność ankietowanych pod względem świadomości pozostawania pod obserwacją instytucji oraz innych osób trzecich. Jak można zauważyć na Rysunku 18 wyniki kobiet oraz mężczyzn są bardzo zbliżone do siebie. Najczęściej ankietowani wskazywali na wyciek danych oraz działalność hakerów – blisko 25% kobiet wskazało te dwie odpowiedzi, natomiast mężczyźni dodatkowo wskazywali częściej śledzenie przez Policję. Niestety zdecydowana większość ankietowanych nie miała pomysłu co mogło spowodować przytoczoną sytuację.



Rysunek 18: Ocena sytuacji związanej z zagrożeniem cyberprzestępstwem - wyniki płci.

5. Wnioski

Postawiona teza 1, która brzmi: „Świadomość i podatność na zagrożenia wynikające z korzystania z Internetu jest najmniejsza dla skrajnych grup wiekowych (najmłodsi i najstarsi użytkownicy)" została potwierdzona. Skrajne grupy miały trudność w znalezieniu prawidłowej odpowiedzi na większość podstawowych pytań. Współczynnik odpowiedzi poprawnych do odpowiedzi niepoprawnych oscylował w okolicach 0.3, podczas gdy dla grup z najwyższą świadomością zawierał się w okolicach 0.7.

Teza „Osoby młodsze (wiek 18-29) mieszkające w mieście mają wyższy poziom wiedzy z zakresu cyberbezpieczeństwa niż osoby starsze (40+) mieszkające zarówno w mieście jak i na wsi. Wyniki w identycznych przedziałach wiekowych będą zbliżone" również została potwierdzona. Osoby młodsze zazwyczaj korzystały częściej z Internetu w przeciwieństwie do osób 40 lat i więcej, poświęcając więcej czasu na edukację w tym zakresie. Często te osoby miały styczność z komputerem we wcześniejszym wieku niż osoby starsze, wobec czego duża część pojęć nie jest im obca.

Duży wpływ na rezultaty badań miała pandemia COVID- 19, która przyczyniła się do wzrostu aktywności respondentów sieci oraz korzystania z komputera. Ludzie niezależnie od wieku byli zmuszeni do zmiany nawyków, gdyż codzienne życie skłoniło ich do korzystania z usług elektronicznych. Szczególnie dało się to zauważyć w zakresie nauki oraz pracy zdalnej. Zwiększony ruch sieciowy na całym świecie spowodował zwiększenie częstotliwości cyberataków.

W ankiecie również zawarto pytanie kontrolne mające na celu sprawdzenie czy deklarowany poziom znajomości informatyki i cyberbezpieczeństwa pokrywa się ze stanem rzeczywistym. Osoby młodsze częściej zaznaczały odpowiedź o wysokim lub bardzo wysokim stopniu znajomości informatyki, co nie miało odzwierciedlenia w uzyskanym wyniku. Zdecydowana większość ankietowanych zaznaczyła odpowiedź średni stopień znajomości informatyki, co odpowiada stanowi faktycznemu. Pomimo tego, że osoby z grupy wiekowej

20-24 lata wypadły najlepiej w tym badaniu to ich wiedza mogłaby być znacznie lepsza. Warto wziąć pod uwagę poprawę edukacji z zakresu informatyki oraz wiedzy o cyberbezpieczeństwie. Duży nacisk można by położyć na samorozwój poprzez kursy i szkolenia oraz obowiązkowe zajęcia na uczelniach lub w pracy z tego zakresu.

6. Podziękowanie

Autorzy przeprowadzanego badania dziękują wszystkim uczestnikom, którzy ochoczo i dobrowolnie wzięli w nim udział. Podziękowanie szczególnie kierują do wszystkich nauczycieli i dyrekcji szkół, w których umożliwiono przeprowadzenie ankiety.

Bibliografia

[1] M. Warner, Cybersecurity: A Pre-history, Intelligence and National Security 27(5) (2012) 781-799.

[2] A. Jastrzębska, W. Jastrzębska, Wykluczenie cyfrowe–przyczyny, zagrożenia i bariery jego pokonania. Studium przypadku, Nierówności społeczne a wzrost gospodarczy 25 (2012) 91-104.

[3] S. Wójcik, Zagrożenia dzieci i młodzieży w Internecie. Dziecko krzywdzone. Teoria, badania, praktyka 16(1) (2017) 270-287.

[4] E. Krok, Budowa kwestionariusza ankietowego a wyniki badań, Zeszyty Naukowe Studia Informatica/Uniwersytet Szczeciński, 2015.

[5] R. G. Brody, E. Mulig, V. Kimball, Phishing, pharming and identity theft, Academy of Accounting & Financial Studies Journal 11(3) (2007).

[6] D. Mielnik, Anonimowość w Internecie a problem cyberbezpieczeństwa, Aspekt prawny, 2019

[7] B. Buraczyńska, Zagrożenia bezpieczeństwa w sklepach internetowych, Monografie–Politechnika Lubelska 69 (2020).

[8] B. Kaczmarczyk, P. Szczepański, M. Dąbrowska, Wybrane zagadnienia cyberbezpieczeństwa, Zeszyty Naukowe Państwowej Wyższej Szkoły Zawodowej im. Witelona w Legnicy 3(32) (2019).

[9] I. Hejduk, Rozwój technologii cyfrowych a wykluczenie społeczne osób 65 plus. Zeszyty Naukowe Uczelni Vistula, (46 (1) Ekonomia X. Pracownicy wiedzy 65 plus-nowe szanse (czy kontrowersje) wobec wyzwań współczesności) (2016) 64-78.

[10] R. Pitera, Współczesne problemy i zagrożenia cyberbezpieczeństwa w sektorze usług bankowości elektronicznej. Przegląd Nauk o Obronności 2 (2017) 181-192.

[11] M. K. Maciejczuk, K. Wnorowski, M. Olchanowski, Cyberbezpieczeństwo dzieci i młodzieży w prawie polskim, 2019

[12] Ankieta dotycząca cyberbezpieczeństwa w społeczeństwie – portal internetowy dobreprogramy.pl https://www.dobreprogramy.pl/wyniki-ogolnopolskiego-badania-wskazuja-ze-polacy-nie-doceniaja-wartosci-swoich-danych,6656565069908544a [15.03.2022r]

[13] Ankieta portalu internetowego controlengineering.pl cyberbezpieczeństwo w systemach sterowania - https://www.controlengineering.pl/cyberbezpieczenstwo-6-kluczowych-wnioskow-z-redakcyjnej-ankiety/ [15.03.2022r]

[14] Artykuł o wycieku danych Wojska Polskiego https://www.telepolis.pl/wiadomosci/wydarzenia/pegasus-wojsko-polskie-wyciek [15.03.2022r]

[15] Blog opisujący skutki wycieku danych https://www.ekransystem.com/pl/blogpolska/ile-kosztuje-wyciek-danych [15.03.2022r]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 118–122
Received: 22 March 2022
Accepted: 27 April 2022

Analysis of the application of brain-computer interfaces of a selected paradigm in everyday life

Analiza zastosowania interfejsów mózg-komputer o wybranym paradygmacie w życiu codziennym

Katarzyna Mróz*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The main objective of this paper is to carry out a research on the analysis of the use of brain-computer interface in everyday life. The article presents the method of recording brain activity, electroencephalography, which was used in the study. The brain activity used in the brain-computer interface and the general principle of brain-computer interface design are also described. The performed study allowed to develop an analysis of the obtained results in the matter of evaluating the usability of brain-computer interfaces using motor imagery. As a result of the process of analyzing the results obtained during the research, it was found that each subsequent experiment allowed for obtaining more favourable results than the previous one. The reason for this was the use of an additional training session for the next test person. In the final stage, it was possible to evaluate the usability of the brain-computer interface in everyday life.

Keywords: brain-computer interface; electroencephalography; motor imagery

Streszczenie

Głównym celem artykułu jest przeprowadzenie badania nad analizą wykorzystania interfejsu mózg-komputer w życiu codziennym. W artykule przedstawiono metodę rejestrowania aktywności mózgu, elektroencefalografię, która została wykorzystana w badaniu. Opisano również aktywność mózgu wykorzystywaną w interfejsie mózg-komputer oraz ogólną zasadę projektowania interfejsu mózg-komputer. Przeprowadzone badanie pozwoliło na opracowanie analizy uzyskanych wyników w zakresie oceny użyteczności interfejsów mózg-komputer z wykorzystaniem obrazowania motorycznego. W wyniku procesu analizy wyników uzyskanych podczas przeprowadzania badań ustalono, iż każdy następnie zrealizowany eksperyment pozwalał na uzyskanie korzystniejszych wyników od poprzedniego. Powodem tego było zastosowanie dodatkowej sesji treningowej dla kolejnych badanych osób. W końcowym etapie można było ocenić przydatność interfejsu mózg-komputer w życiu codziennym.

Słowa kluczowe: interfejs mózg-komputer; elektroencefalografia; wyobrażenia ruchowe

*Corresponding author

Email address: katarzyna.mroz@pollub.edu.pl (K.Mróz)

1. Introduction

The brain-computer interface is a variant of the human-machine interface, which is a direct communication path between the human brain and an external device without the involvement of the peripheral nervous system and muscles [1,2]. Brain-computer interface requires a combination of knowledge from several sciences: artificial intelligence, biomedical engineering, neuroscience and electronics. The continuous development of these fields of science has resulted in an increase in research and publications on the issue of the brain-computer interface. The brain-computer interface is defined as interdisciplinary, therefore the article covers only its individual fragments related to processing, analysis and classification [3].

Brain-computer interfaces are systems with practical use in everyday activities, communication, entertainment, rehabilitation and prosthetics. For people with disabilities, they offer the possibility to control the device that performs the movement for them, and for people for whom communication with the environment is impossible, they offer the ability to write only from the activity of the nervous system, without the involvement of muscles [4]. In the research carried out for the purposes of this article, the potential related to the image of movement was used. In order to apply it correctly, the user had to learn to visualize the movement, and it was also necessary to construct the experiment requiring many repetitions to ensure that the signal-to-noise ratio of the synchronization and desynchronization of the signal was low. This potential is related to the ERD/ERS stimulus, ERD means a decrease, while ERS means an increase in the average signal strength in a given frequency band. The occurrence of desynchronization and synchronization takes place within the motor cortex of the brain.

2. Purpose of the work

The main purpose of this article was to process, analyze and classify EEG signals in terms of the use of the brain-computer interface. The obtained results allowed for the assessment of the usefulness of brain-computer interfaces in general human life. For the purposes of the study, a thesis was formulated: Adequate training before using the brain-computer interface and its subsequent

use allows for a significant improvement in the quality of human life. In order to achieve the aim of the work, a selected paradigm of the motor imagery was used, the operation of which was preceded by the analysis of its data, extraction and selection of features.

3. Materials and methods

3.1. Electroencephalography (EEG)

During the research, the method of electroencephalography was used. It is a non-invasive method of recording the brain's bioelectric signals with the use of electrodes placed on the subject's scalp. EEG is characterized by the highest time resolution, safety and low cost of carrying out a brain activity test [7].

3.2. Preparation for the EEG examination

The preparation of the EEG cap is very important steps. This is due to the fact that the electrodes collect potentials from the head of the examined person, so the EEG cap must be perfectly fitted to it. Therefore, the head surface at the electrode site should be cleaned of the secreted sebum and exfoliating epidermis. Then, using a tailor's tape measure is the distance between the point of the seeds (nose bridge) and the inion (place of the bulge in the center line of the skull base). Then the circumference of the head is measured in order to select the appropriate size of the EEG cap to the tested person's head based on the measurement result.

3.3. The 10-20 system of EEG electrode placement

The 10-20 assembly system, which is the arrangement of the arrangement of 21 electrodes on the surface of the subject's head. The 10-20 system is characterized by the necessity to divide the head into uniform sections in relation to the following places on the skull: right and left ear section, external occipital tumor, nose bridge. The name of the assembly system 10-20 indicates equal distances, expressed as a percentage, between individual points such as: the base of the nose (seeds), the occipital tumor on the antero-posterior plane (inion), right and left segment of the ear on the dorso-abdominal plane. The reference point is defined as 10% of the total distance measured in cm from the root of the nose. Figure 1 shows the 10-20 assembly system in theory [3,8].

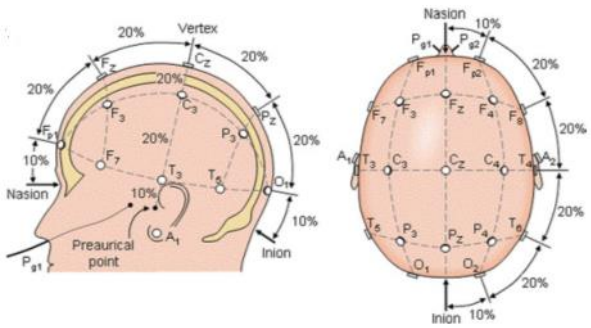


Figure 1: Ideal electrode placement in a 10-20 system [12].

3.4. Brain-computer interface

BCI can be defined as a computer system that obtains signals from the human brain, analyzes them and translates into commands. An important aspect of BCI is the conversion of a signal from human brain activity into a digital signal that is passed to an output device that performs the desired action. Hence, the relationship between humans and computers, especially in terms of communication, takes on a new meaning [2].

3.5. The stages of the brain-computer interface

The brain-computer interface works by measuring the activity of the user's brain. The detected brain activity through the BCI system does not provide many perspectives because it is impossible to read human thoughts. The only use of the resulting models of brain activity is to classify them with certain events. Therefore, during the study, the participant had to focus to generate the appropriate brain activity using a specific event or stimulus. The user uses the so-called mental strategies of focusing on a certain event or image in order to generate useful EEG waves.

The first stage of the brain-computer interfaces is the training session. This is an action taken by the user prior to the application of the brain-computer interface. Training is an element that is often required from the user along with learning about the mental strategy. It consists in learning to consciously manipulate a given feature with the use of simulators [8]. When learning the system, it is sometimes necessary to carry out even several training sessions. This is required in many cases due to the easy possibility of making a mistake during the human-device communication process. During training, the user performs a specific task, e.g. imagining the movement of the right hand.

After completing this mental task, the signal parameters needed at a later stage called classification are obtained. The training stage is followed by teaching, more precisely training the classifier. The brain-computer interface used during the research worked online, thanks to which a threshold classifier was used. Its operation is based on the fact that after reaching a certain value of a certain parameter, the signal will remain qualified to a certain class [4,9].

After training the classifier, the system is ready for further tests. The next stage consists of training sessions repeated several times. The test person tries to generate perfect signals by observing the results appearing on the computer monitor. This action is called user training based on feedback [6]. This process allows the test subject to receive real-time information on how to classify his or her reactions. It is a facility that allows the user to influence his or her next reactions and obtain the best possible results of the study [5,10].

The last stage consists in appropriate use of the interface, to control devices or communicate with the environment by people with disabilities. The person using the interface does not need additional help from another person, and if there is such a need, it is usually a small help. a person with a disability, e.g. paralyzed, is able to: control the movement of his own wheelchair, guide the cursor on a monitor screen, use a whiteboard with letter symbols to communicate his thoughts, control an artificial prosthesis [10,13].

3.6. Motor imagery (MI)

During the study, the motor imagination paradigm was used. The sensation of movement is a cognitive-perceptual process consisting in the mental performance of movement without the use of muscle activity [8]. It is an experience that suggests that the subject feels, the performance of which he imagines. Motor imagery require the conscious activation of certain areas of the brain that are directly related to the exercise and preparation for movement. It is used in the neurological rehabilitation of patients who have suffered a stroke and have paresis in moving certain parts of the body.

4. Experiment description

During the experiment, the subjects were presented with an interface consisting of a board with an arrow on it. Right and left arrows were displayed randomly during the presentation. Depending on the stage of research, the commands and requirements for the user of the brain-computer interface changed. During one stage, the participant was asked to focus on the monitor screen and imagine the bend of the right or left hand, depending on the appearance of arrows on the monitor. Alternatively, the participant had to physically bend the individual hands. The main stage of the study was preceded by registration with eyes closed and open while the subject was resting. During the experiment, the accuracy and additional parameters of the algorithm, such as efficiency, were tested [9,11].

The electroencephalography study was carried out using 21 measuring electrodes connected according to the 10-20 assembly system. The 10-20 system is characterized by the necessity to divide the head into uniform sections in relation to the following places on the skull: right and left ear section, external occipital tumor, nose bridge. The actual arrangement of the electrodes in the 10-20 system is shown in Figure 2.

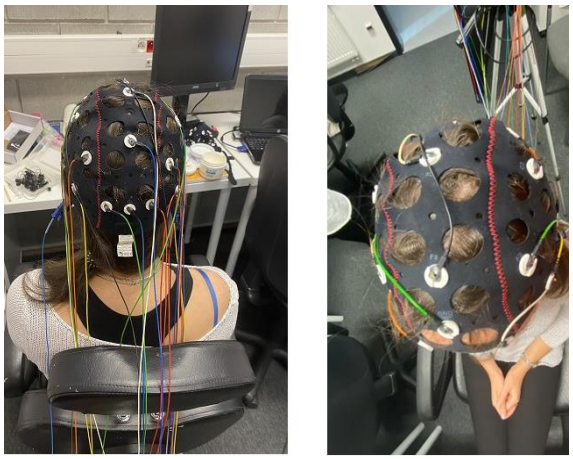


Figure 2: Placement of electrodes during the test.

The Mitsar amplifier shown in Figure 3 was used to transmit the signals from the electrodes to the computer. The procedure of recording the brain activity was repeated twice for each person: the first study for the hand flexion with the use of muscles, and the second one exclusively with the use of the hand movement image paradigm [7,8,12].

Figure 3: Test stand.

4.1. Data analysis process

The data analysis process involves the processing of EEG signals, which is the main component of the brain-computer interface system. As a result, EEG signal processing is very often the subject of research, and the general structure of a typical BCI solution is shown in stages which include: pre-processing, extraction and feature selection. Data from EEG channels are considered input. After the signal pre-processing stage, feature extraction begins. This step is to provide the most important set of values representing the characteristics of the signals. After that, the collected data are subjected to a selection of features, so that they can be classified [7].

4.2. Research scenario

The "Motor imagery" scenario available in the OpenViBE program was used to achieve the research goal. The scenario consists of six stages: signal monitoring, acquisition, interface training, classifier training, online testing, replay. For each test subject, two trials of the research experiment were used. The first attempt consisted in training the participant in which he squeezed rubber balls using the muscular activity of his right and left hands. However, during the second attempt, balls were not used and the user imagined the movement of his hands. The second attempt was based on the use of the motor imagination paradigm [8].

The "Motor imagery" scenario uses a linear discriminant analysis classifier with a k-fold cross-validation algorithm to classify data. The LDA classifier is used to reduce the number of features for easier value management before classification. On the other hand, the algorithm of k-fold cross-validation ensures that the full data set is used both for model validation and its training. In addition, while training the model, it determines its quality to prevent the problem of overfitting.

One of the Motor imagery scenarios written in the graphic language is shown in Figure 4.

An interface is required to gather the necessary information along with the events that have occurred while running the experiment. Its operation is presented below through the visualization presented to the respondents in Figure 5.

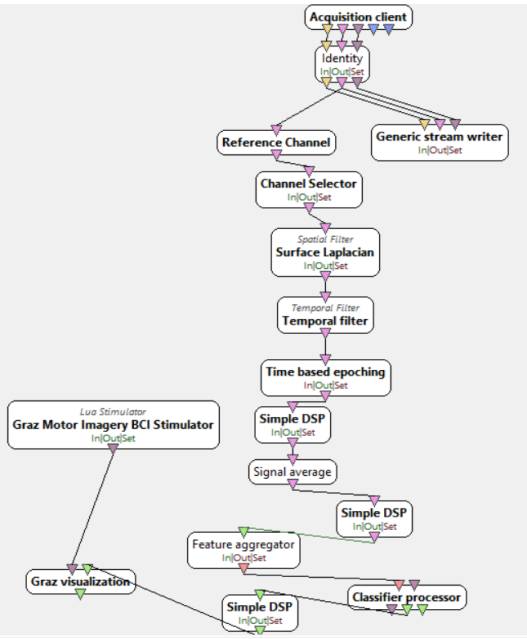


Figure 4: Motor imagery – online.

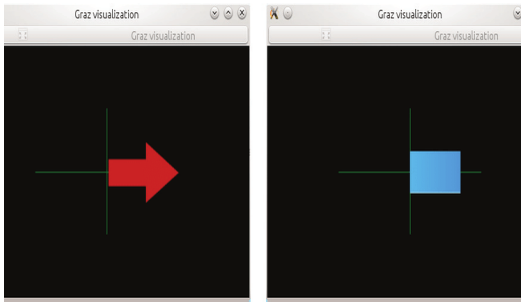


Figure 5: Motor imagery scenario in OpenViBE.

5. Results

During the experiment described in point 3, each of the tested algorithms went through the stages of learning and testing twice. Its parameters were recorded at each stage. This was for the paradigm of imagining the movement and flexing of the arms with the use of muscle activity. The data obtained during the learning of the classifier with the use of the k-fold cross-validation algorithm were extracted was recorded during each session of individual users. All results were averaged and displayed in this section.

5.1. Accuracy results

Accuracy results are the results of a trained classifier running online against guessed user intent. The task of the learned classifier was to cause an impulse in the form of an arrow in relation to the defined stimulus, which means determining the side of triggering the arrow on which the subject imagines the bending movement of his hand. At the end of the training, a matrix was obtained, its results are presented in Table one.

The obtained results for both the accuracy and the value of correct forecasts per class were favorable. High accuracy values mean the precision of the feature vectors classification and the effectiveness of the online classifier method. For the first person tested, in trial 1, who obtained the following results: 70.6% and 29.4%. This meant that 70.6% of the feature vector for Objective 1 was actually intended as Objective 1, the right hand flexion. For sample 2, which obtained the following results: 39.4% and 60.6%. This meant that 60.6% of the feature vector for Objective 2 was actually envisioned as Objective 2, the left hand flexion. For the next respondents, the obtained results are much higher, which was caused by the use of an additional training session.

Table 1: Class accuracy results by class

	cls vs cls [%]					
Person	Input 1		Input 2		Average [%]	Accuracy [%]
	1	2	1	2		
1	70,6	29,4	39,4	60,6	65,61	50
2	77,6	22,4	24,9	75,1	76,33	85
3	80,4	19,6	14,7	85,3	82,86	75

Performance results

Based on the k-fold cross-validation, an analysis of the LDA classifier performance was developed. The results of learning the LDA classifier using k-fold cross-validation against the training algorithm, in which the subjects used the motor image paradigm, are presented in Table two. The value of k is 7 according to the assumptions of k-fold cross-validation where k is the number of patterns of the data set that they are learning.

Table 2: Results of k-fold cross-validation

	k-fold cross validation [%]								
Person	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Average	Sigma [%]
1	63,92	79,64	79,64	63,21	82,14	80,35	68,21	73,87	7,76
2	70	80,35	56,42	66,78	79,28	83,57	44,28	68,67	13,20
3	95	86,78	97,5	99,28	97,5	78,57	86,42	91,58	7,17

The obtained results of seven sets show the efficiency of the training algorithm, which can be understood as the level of acquiring knowledge during the learning stage. The achieved results for the LDA classifier are very good. The mean value of the k-fold cross-validation for all interface users is greater than 60%.

The results of learning the LDA classifier with the use of k-fold cross-validation against the training algorithm, in which the subjects used their muscle activity and bent their hands alternately, are presented in Table three.

The achieved results are much worse than the previous ones, from Table 2, despite the fact that the obtained data came from the same users and the same classification procedure was used. The mean k-fold cross-validation for two interface users is greater than 40%. The reason for such poor results is that it was user training, consisting in the actual bend of the hand, and above all, it was the first approach of the respondents with BCI.

Table 3: Results of k-fold cross-validation

	k-fold cross-validation [%]								
Person	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Average	Sigma [%]
2	50,71	37,85	34,28	35,35	68,21	17,14	42,14	40,81	14,59
3	44,28	63,57	47,5	52,85	48,92	72,5	52,85	54,64	9,22

6. Conclusions

At the beginning of the article it was mentioned that the aim of the experiment was to analyze the use of brain-computer interfaces in everyday life. As a result of the process of analyzing the results obtained during the research, it was found that each subsequent experiment allowed for obtaining more favorable results than the previous one. The reason for this was the use of an additional training session for subsequent test persons, as well as the use of new cosmetics to prepare the head surface for examination. The results for the first test person may be burdened with technical and physiological artifacts. They result from the subject's thick hair, so that the electrodes may not be connected precisely during the test. Nevertheless, the results obtained during the classifier learning process and the main study with the use of the motor imagination paradigm were satisfactory for all the subjects.

The collected data allowed to state that the LDA classifier is very effective and equivalent to the feature vectors on which it carries out the classification process. The LDA classifier perfectly guessed the users' intentions, which were based on the motion image paradigm, which in effect contributed to the display of arrows on the monitor according to the image of hand movement. This means the effective learning of the classifier and the correctness of the data obtained during the EEG test. Thus, the designed brain-computer interface successfully accomplished its task.

Currently, BCI systems are no longer perceived as advanced projects implemented only in a scientific laboratory, but are gaining importance as opportunities to improve the quality of human life. The main aspect is improving the performance of the central nervous system in people suffering from neurological diseases. However, the implementation of BCI systems for use is associated with some challenges, such as: numerous training sessions, observation of feedback, the presence of a trained person during the test, access to research equipment and appropriate cosmetics to prepare for the test. Research on BCI technology gives hope for promising innovations that will surely have a huge impact on all of us.

References

[1] B. Michalik, Analiza porównawcza wydajności klasyfikatorów pod nadzorem w realizacji interfejsu mózg-komputer opartego o paradygmat P300, Praca inżynierska, Politechnika Lubelska, Lublin, 2018.

[2] D. Aminaka, S. Makino, T. M. Rutkowski, Chromatic SSVEP BCI paradigm targeting the higher frequency EEG responses, Asia-Pacific Signal and Information Processing Association, Annual Summit and Conference (APSIPA) (2014) 1-7.

[3] F. Aloise, P. Aricò, F. Schettini, S. Salinari, D. Mattia, F. Cincotti, Asynchronous gaze-independent event-related potential-based brain–computer interface, Artificial Intelligence in Medicine 59(2) (2013) 61–69.

[4] R. A. Robbins, Z. Simmons, B. A. Bremer, S. M. Walsh, S. Fischer, Quality of life in ALS is maintained as physical function declines, Neurology 56(4) (2001) 442–444.

[5] T. Cedro, A. Grzanka, CeDeROM Brain Computer Interface, Information Technologies in Biomedicine (2012) 219-231.

[6] M. M. Jackson, R. Mappus, Applications for Brain-Computer Interfaces, Brain-Computer Interfaces (2010) 89-103.

[7] H. Suryotrisongko, F. Samopa, Evaluating OpenBCI Spiderclaw V1 Headwear's Electrodes Placements for Brain-Computer Interface (BCI) Motor Imagery Application, Procedia Computer Science 72 (2015) 398–405.

[8] K. Cieślak-Blinowska, R. Kuś, J. Ginter, Propagation of EEG activity during finger movement and its imagination, Acta Neurobiologiae experimentalis 66 (2006) 195-206.

[9] C. W. Anderson, E.A. Stolz, S. Shamsunder, Multivariate autoregressive models for classification of spontaneous electroencephalographic signals during mental tasks, IEEE Transactions on Biomedical Engineering 45(3) (1998) 277-286.

[10] M. Kołodziej, A. Majkowski, R. J. Rak, Linear discriminant analysis as a feature reduction technique of EEG signal for brain-computer interfaces, Przegląd Elektrotechniczny 88 (2012) 28-30.

[11] S. Aggarwal, N. Chugh, Review of Machine Learning Techniques for EEG Based Brain Computer Interface, Archives of Computational Methods in Engineering (2022) 1-20.

[12] A. Hulewicz, M. Jukiewicz, Analiza sygnałów EEG na potrzeby interfejsu mózg-komputer, Acta Bio-Optica et Informatica Medica. Inżyneria Biomedyczna 3 (2014) 137-143.

[13] A. Cegielska, M. Olszewski, Nieinwazyjny interfejs mózg-komputer do zastosowań technicznych, Pomiary Automatyka Robotyka 3 (2015) 5-14.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 123–127
Received: 24 March 2022
Accepted: 6 April 2022

Performance Comparison of Unit Test Isolation Frameworks

Porównanie wydajności szkieletów programistycznych do izolacji kodu w testach jednostkowych

Mateusz Domański*, Michał Dołęga*, Grzegorz Kozieł

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The goal of unit testing is to verify that individual parts of application are correct. For external dependencies mock objects should be created. This process is supported by dedicated libraries. The paper compares three unit test isolation frameworks for .NET: Moq 4.16.1, FakeItEasy 7.2.0 and NSubstitute 4.2.2. The performance research included comparison of benchmark execution times and comparison of unit test execution times in which selected methods of tested libraries were used. The results are shown on box plots. The analysis shows that Moq is optimal mocking framework.

Keywords: code isolation; unit testing; mock objects

Streszczenie

Celem testów jednostkowych jest weryfikacja poprawności działania pojedynczych elementów programu. Dla zależności wychodzących poza ten zakres powinny zostać utworzone atrapy obiektów. Proces ten wspomagają dedykowane biblioteki. W niniejszej pracy przedstawiono porównanie trzech szkieletów programistycznych do izolacji kodu w testach jednostkowych dla platformy programistycznej .NET: Moq 4.16.1, FakeItEasy 7.2.0 oraz NSubstitute 4.2.2. Badanie wydajności objęło porównanie czasów wykonania testów wydajnościowych oraz porównanie czasów wykonania testów jednostkowych, w których wykorzystane zostały wybrane metody badanych bibliotek. Wyniki przedstawiono na wykresach pudełkowych. Z przeprowadzonej analizy wynika, że optymalnym szkieletem programistycznym do tworzenia atrap obiektów jest Moq.

Słowa kluczowe: izolacja kodu; testy jednostkowe; atrapy obiektów

*Corresponding author

Email address: mateusz.domanski1@pollub.edu.pl (M. Domański), michal.dolega@pollub.edu.pl (M. Dołęga)

1. Wstęp

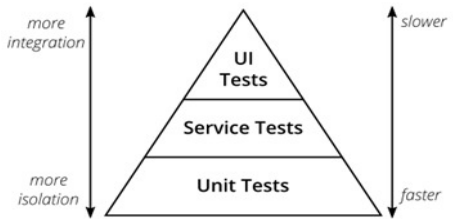
Testy jednostkowe są jednym ze sposobów poprawy jakości oprogramowania. Są metodami testowymi, które weryfikują poprawność pojedynczych metod programu. Ich niewątpliwą zaletą jest bardzo krótki czas wykonania oraz możliwość wykrywania błędów podczas początkowych faz tworzenia oprogramowania. W dużej mierze ma to wpływ na zmniejszenie kosztów projektu, ponieważ im defekt zostanie wcześniej zauważony, tym koszt jego naprawy będzie mniejszy. Dodatkowo testy jednostkowe wpływają także na skrócenie całego procesu testowania. Związane jest to z tym, że zespoły złożone z testerów mogą skupić się w większej mierze na sprawdzaniu funkcjonalności oraz integracji pomiędzy modułami.

Testy jednostkowe nie powinny wykorzystywać zewnętrznych zależności w celu sprawdzenia poprawności działania aplikacji. Dlatego istnieją biblioteki, które ułatwiają izolacje testów poprzez tworzenie atrap obiektów. Są to obiekty, które imitują wymagane zależności. Programista może zdefiniować zachowanie atrapy oraz zwracane przez nią wartości na potrzeby danego testu. Dzięki temu test jednostkowy sprawdza poprawność działania pojedynczego fragmentu logiki aplikacji. Wykorzystanie tego typu narzędzi znacząco ułatwia i przyspiesza proces tworzenia testów.

W niniejszym artykule przedstawiono porównanie szkieletów programistycznych do izolacji kodu w testach jednostkowych. Ma on na celu zbadanie najpopularniejszych bibliotek dla platformy programistycznej .NET pod względem wydajności.

2. Testy jednostkowe

Wyróżnia się cztery typy testów automatycznych: akceptacyjne, systemowe, integracyjne oraz jednostkowe. Głównym podejściem do ich tworzenia jest tzw. piramida testów, która została opisana przez H. Vocke [1]. Liczba testów jednostkowych powinna być największa, ponieważ ich wytworzenie kosztuje najmniej oraz działają najszybciej. Ich zaletą jest również fakt, że tworzone są równolegle z testowaną metodą. Uruchomienie testu bezpośrednio po zaimplementowaniu metody pozwala na wykrycie błędów na wczesnym etapie, a co za tym idzie, obniżenie kosztów ich usunięcia. Opisane podejście przedstawione jest na Rysunku 1.



Rysunek 1: Piramida testów [1].

Testy jednostkowe są funkcjonalną metodą testowania. Polegają na sprawdzaniu pojedynczych metod w celu potwierdzenia, że działają zgodnie z przeznaczeniem. Są one wytwarzane przez programistów w celu poprawy jakości kodu oraz zniwelowania potencjalnych błędów, które mogą przydarzyć się podczas tworzenia oprogramowania. J. V. Petersen w swoim artykule przedstawia 10 powodów, dlaczego testy jednostkowe są ważne [2]. Są to między innymi:

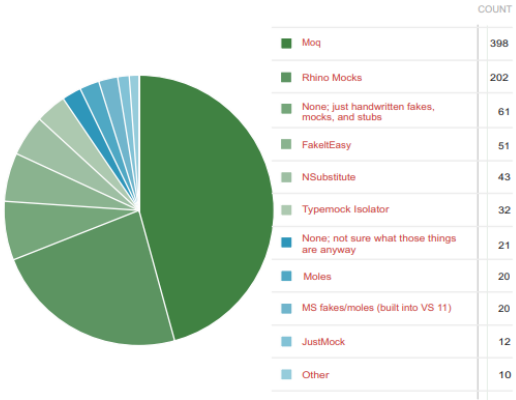
- sprawdzenie poprawności oprogramowania,
- zmniejszenie złożoności kodu,
- traktowanie testów jednostkowych jako formę dokumentacji,
- pomiar wymaganego wysiłku niezbędnego do zmiany metody,
- wymuszenie stosowania wzorców projektowych, wstrzykiwanie zależności pomiędzy nimi,
- sprawdzenie pokrycia kodu,
- sprawdzenie wydajności,
- przyspieszenie procesu wytwarzania oprogramowania poprzez możliwość wykorzystywania testów w przypadku ciągłej integracji (CI).

Współczesne metodyki pracy kładą duży nacisk na odpowiednie pokrycie aplikacji testami, dlatego zazwyczaj projekt zawiera bardzo dużą liczbę testów jednostkowych. Są one często uruchamiane, zarówno w lokalnym środowisku jak i w procesie ciągłej integracji, za każdym razem, gdy w repozytorium kodu zostanie wprowadzona zmiana. Właśnie z powodu dużej liczby testów jednostkowych oraz potrzeby częstego ich uruchamiania istnieje konieczność rozważenia kwestii czasu uruchomienia owych testów. Zbyt długi czas potrzebny na ich wykonanie może mieć negatywny wpływ na proces wytwarzania oprogramowania. Z tego względu ważny jest dobór bibliotek, które nie tylko będą oferowały szeroki wachlarz możliwości ale również zapewnią efektywność testów.

3. Analiza literatury

W niniejszym artykule dokonano przeglądu literatury dotyczącej testów jednostkowych oraz bibliotek do tworzenia atrap obiektów. W pracy J. Copliene próbuje udowodnić, że tworzenie testów jednostkowych jest niewydajne [3]. Autor stara się pokazać, że są one kwestią przeszkadzającą w jego pracy. Po analizie okazuje się jednak, że to nie same testy stwarzają problem, lecz sposób podejścia do nich, narzucany przez klientów. M. Fowler w swoim artykule skupia się zaś na charakterystyce obiektów typu mock i stub [4]. Ponadto pokazuje czytelnikowi różnice pomiędzy danymi typami, jednak nie wskazuje konkretnie, który rodzaj atrapy jest lepszy w użytkowaniu.

Istnieje wiele bibliotek do izolacji kodu w testach jednostkowych, jednak dostępność publikacji, porównujących konkretne szkielety programistyczne w ramach jednego języka programowania jest niewielka. R. Osherove jest jednym z nielicznych autorów poruszających ten temat [5]. W swojej książce przedstawił zestawienie bibliotek FakeItEasy oraz NSubstitute. Drugie wydanie książki zostało rozszerzone o bibliotekę Moq. Warto wspomnieć, że autor przeprowadził ankietę badającą popularność bibliotek. Jej wyniki zaprezentowano na Rysunku 2. Kolumna COUNT określa liczbę osób, które używały danego szkieletu programistycznego.



Rysunek 2: Wykres popularności bibliotek [5].

T. Haukilehto jest z kolei autorem porównania szkieletów Moq i FakeItEasy [6]. Twórca sugeruje, aby nie skupiać się na wyborze jednej, konkretnej biblioteki do izolacji testów jednostkowych, lecz używać ich obu. We wpisie autorstwa T. Ardal znajduje się zaś opis przedstawiający wady i zalety bibliotek badanych w ramach niniejszego artykułu [7]. W artykule T. Hyttinen można zaś poznać sposób wykorzystania szkieletu programistycznego BenchmarkDotNet, z pomocą którego tworzy się testy wydajnościowe dla rozwiązań opartych na platformie .NET [8].

4. Wybór rozwiązań do testów

Na rynku dostępnych jest wiele bibliotek do izolacji kodu w testach jednostkowych, są to między innymi:

- Moq – umożliwia definiowanie atrap przy użyciu dwóch podejść: imperatywnego lub funkcyjnego. Pozwala także na imitowanie zarówno interfejsów jak i klas, jednak rekomendowane jest tworzenie atrap dla interfejsów. Bardzo popularna, ogólna liczba jej pobrań wynosi ponad 230 miliony.
- FakeItEasy – charakteryzuje się nietypową semantyką, gdyż posiada tylko obiekty zwane fake. To w jaki sposób są używane determinuje jaka atrapa obiektu została użyta. Dokumentacja tej biblioteki jest dokładna i przejrzysta. Na stronie głównej dostępna jest opcja czatu, z którego można skorzystać w razie problemów.
- NSubstitute – wyróżnia ją składnia, która jest najłatwiejsza do nauki, ponieważ w założeniu ma przypominać naturalny język. Użycie wyrażeń lambda zostało zmniejszone do minimum. Warta uwagi jest również dokumentacja, która jest bardzo dobrze napisana.
- Rhino Mocks – jest to biblioteka programistyczna wydana na zasadach licencji BSD (Berkeley Software Distribution). Nie pozwala ona na tworzenie imitacji statycznych metod. Na rynku znajduje się już od 2011 roku, a od 9 lat nie wydano nowej wersji tego szkieletu programistycznego, co sugeruje zaprzestanie prac nad nią.

- JustMock Lite, JustMock – wersja Lite jest darmowa i zawiera przejrzystą dokumentację, jednak nie posiada wielu ważnych funkcjonalności, które są zawarte w Moq, FakeItEasy i NSubstitute. Wynika to z tego, że nie jest ona pełną wersją oprogramowania. Aby uzyskać dostęp do pozostałych funkcji, należy wykupić komercyjną wersję, o nazwie JustMock, która jest jedyną płatną biblioteką w tym zestawieniu.
- AutoFixture – spośród innych bibliotek wyróżnia ją fakt, że generowanie domyślnych wartości imitowanych obiektów może zostać zautomatyzowane. Ułatwia to pisanie testów, aczkolwiek takie działanie jest ryzykowne, ponieważ każde uruchomienie testu generuje różne wyniki. Mogą one negatywnie wpływać na końcowe sprawdzenie oczekiwanych rezultatów. Istnieje jednak sposób, dzięki któremu można otrzymywać te same instancje obiektów cały czas. Należy do tego wykorzystać typ Frozen.

Powyżej opisano najciekawsze biblioteki do tworzenia atrap obiektów w testach jednostkowych dla technologii .NET. Jednak na potrzeby niniejszej pracy zdecydowano się na wybranie trzech szkieletów programistycznych. Biblioteka Rhino Mocks nie zostanie wzięta pod uwagę, ponieważ nie jest już wspierana. Również narzędzie JustMock zostało odrzucone z powodu, że jest to rozwiązanie komercyjne i wymaga wykupienia licencji. Wersja Lite jest natomiast mocno ograniczona. Narzędzie AutoFixture ma trochę inne przeznaczenie w porównaniu do pozostałych bibliotek, ponieważ umożliwia generowanie domyślnych wartości, co nie jest do końca dobrą praktyką. Do badań zostały wytypowane biblioteki Moq, FakeItEasy oraz NSubstitute, ponieważ są to rozwiązania najpopularniejsze, darmowe, oferują podobne możliwości oraz przejrzystą dokumentację.

5. Plan badań

Celem niniejszej pracy jest porównanie wydajności wybranych szkieletów programistycznych do izolacji kodu. Z tego powodu przeprowadzone zostały badania na podstawie dwóch scenariuszy:

- pomiar czasów wykonania testów wydajnościowych (benchmarki) dla wybranych metod badanych bibliotek,
- pomiar czasów wykonania testów jednostkowych w których wykorzystane zostały wybrane metody badanych szkieletów programistycznych.

Na potrzeby badań wybrane zostały podstawowe funkcjonalności, które oferowane są przez wszystkie analizowane biblioteki do izolacji kodu i dla których zostały zaimplementowane powyższe scenariusze badawcze. Są to:

- wywołanie zwrotne,
- zwrócenie zadanej wartości,
- weryfikacja jednokrotnego wywołania,
- wywołanie metody bez parametrów,
- wywołanie metody z jednym parametrem,
- wywołanie metody z dwoma parametrami.

W celu uzyskania wiarygodnych wyników, zarówno w przypadku testów wydajnościowych, jak i testów jednostkowych, zadbano o powtarzalność eksperymentu. Każdy z przypadków został zaprojektowany tak, by realizował tę samą operację i uzyskiwał ten sam rezultat przy każdym uruchomieniu. Dla obu scenariuszy wyznaczono liczbę pomiarów, która umożliwi porównanie bibliotek. Następnie każdy z przypadków wykonano określoną liczbę razy. Spośród uzyskanych wyników odrzucono odstające, a pozostałe wzięto pod uwagę podczas oceniania. W ramach analizy wyznaczone zostały wielkości statystyczne, takie jak średni czas wykonania, odchylenie standardowe, mediana, minimum, maksimum oraz pierwszy i trzeci kwartyl.

5.1. Testy wydajnościowe

Pierwszy scenariusz badawczy to pomiar czasów wykonania testów wydajnościowych dla wybranych metod badanych bibliotek. Dla każdej z rozpatrywanych funkcjonalności zostały utworzone trzy benchmarki, po jednym dla każdego szkieletu programistycznego. Wywołują one odpowiednie metody, które realizują daną funkcjonalność. Następnie metody te zostały przekształcone w testy wydajnościowe przy użyciu biblioteki BenchmarkDotNet. Umożliwia ona uruchomienie poszczególnych metod określoną liczbę razy oraz pomiar ich czasów wykonania. W tym przypadku zastosowano następującą konfiguracje: 3 uruchomienia, 10 powtórzeń, 100000 iteracji dla każdego powtórzenia. Dzięki czemu wykonano 3000000 pomiarów dla każdej z badanych funkcjonalności. Scenariusz ten pozwoli na porównanie wydajności badanych szkieletów w warunkach izolowanych.

5.2. Testy jednostkowe

Drugi scenariusz badawczy obejmuje stworzenie prostych testów jednostkowych, które wykorzystują wybrane funkcjonalności badanych szkieletów programistycznych. Testy zostały przygotowane z wykorzystaniem biblioteki NUnit i były uruchamiane przy użyciu środowiska programistycznego Visual Studio z zainstalowanym pakietem NUnit3TestAdapter. Daje to możliwość rejestrowania czasu wykonania w łatwy sposób. Przy realizacji tego scenariusza istotne było uruchamianie testów pojedynczo w celu uzyskania niezależnych wyników. Wykonanie pomiarów nie było zautomatyzowane, każdy test był uruchamiany manualnie. Wykonano po 50 pomiarów dla każdej z badanych funkcjonalności. Rezultaty uzyskane przy użyciu tego scenariusza pozwolą porównać wpływ badanych bibliotek na czas wykonania testów.

5.3. Środowisko testowe

W celu uzyskania powtarzalności wyników wszystkie pomiary wykonano na tym samym środowisku. Specyfikacje stanowiska badawczego przedstawiono w Tabeli 1. Przed rozpoczęciem eksperymentu, ponownie zainstalowany został system operacyjny oraz niezbędne oprogramowanie zaprezentowane w Tabeli 2. Podczas wykonywania badań wyłączone pozostawały wszelkie

inwazyjne aplikacje oraz procesy, na przykład program antywirusowy. Stanowisko było także odłączone od sieci internetowej.

Tabela 1: Specyfikacja stanowiska badawczego

Element	Specyfikacja
System operacyjny	Windows 10 Education
Procesor	Intel Core i5-6300HQ 2.30GHz (4679 punktów według strony www.cpubenchmark.net)
Pamięć RAM	16 GB
Dysk	Plextor PX-256M7VG

Tabela 2: Wykorzystane narzędzia i biblioteki

Narzędzie	Wersja
Visual Studio	2019 Community
.NET	5.0
Język C#	9.0
NUnit	3.13.2
NUnit3TestAdapter	4.0.0
BenchmarkDotNet	0.13.1
FakeItEasy	7.2.0
Moq	4.16.1
NSubstitute	4.2.2

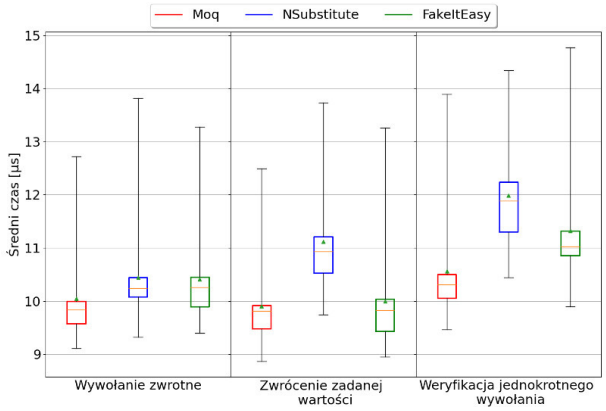
6. Wyniki badań

Na podstawie rezultatów uzyskanych zarówno dla testów wydajnościowych jak i dla testów jednostkowych, wyznaczono podstawowe wielkości statystyczne. Wyniki zostaną przedstawione na wykresach, co ułatwi ich porównanie.

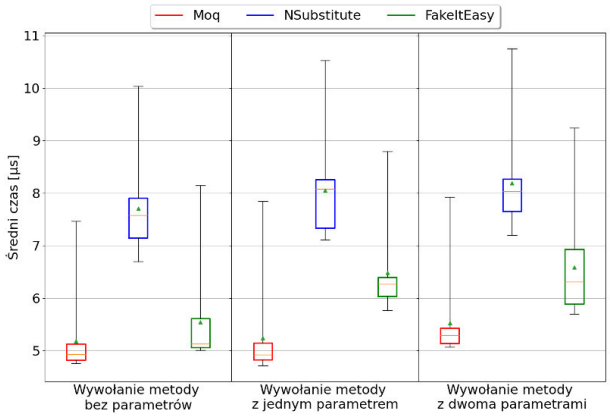
6.1. Testy wydajnościowe

Na Rysunku 3 oraz Rysunku 4 przedstawiono wykresy pudełkowe obrazujące wyniki otrzymane dla testów wydajnościowych. Rysunek 3 zawiera wykres przedstawiający rozkład wartości dla trzech funkcjonalności: wywołania zwrotnego, zwrócenia zadanej wartości oraz weryfikacji jednokrotnego wywołania. We wszystkich przypadkach najmniejszy czas osiągnęła biblioteka Moq, a nieco gorsze wyniki uzyskał szkielet programistyczny FakeItEasy. Warto jednak zaznaczyć, szczególnie biorąc pod uwagę fakt, że przedstawione wartości są rzędu mikro sekund, że różnice pomiędzy bibliotekami są bardzo niewielkie.

Na Rysunku 4 zaprezentowano wykres pudełkowy przedstawiający wyniki dla trzech kolejnych funkcjonalności. Jest to wywołanie metody bez parametrów, wywołanie metody z jednym parametrem oraz wywołanie metody z dwoma parametrami. Również w tym przypadku Moq osiąga najkorzystniejsze wyniki. Bardzo małe pudełko oraz mała odległość od wartości minimalnej, dla tego szkieletu programistycznego, świadczy o tym, że większość (co najmniej 75%) pomiarów ma bardzo zbliżone wartości. Można także zauważyć, że im większa jest liczba przekazanych parametrów, tym większy jest średni czas wykonania metod atrapy obiektu dla wszystkich badanych bibliotek.



Rysunek 3: Wykres pudełkowy – testy wydajnościowe – wywołanie zwrotne, zwrócenie zadanej wartości oraz weryfikacja jednokrotnego wywołania.



Rysunek 4: Wykres pudełkowy – testy wydajnościowe – wywołanie metody bez parametrów, wywołanie metody z jednym parametrem oraz wywołanie metody z dwoma parametrami.

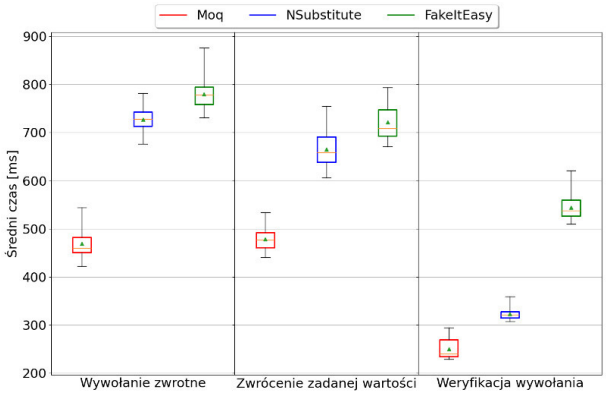
W przypadku wykresów przedstawionych na Rysunku 3 oraz Rysunku 4 wyraźnie widać, że rozkład wartości jest asymetryczny prawostronnie. Oznacza to, że odległość wartości maksymalnej od mediany jest znacząco większa niż odległość wartości minimalnej od mediany. Świadczy to o sporym rozproszeniu wartości większych od mediany.

6.2. Testy jednostkowe

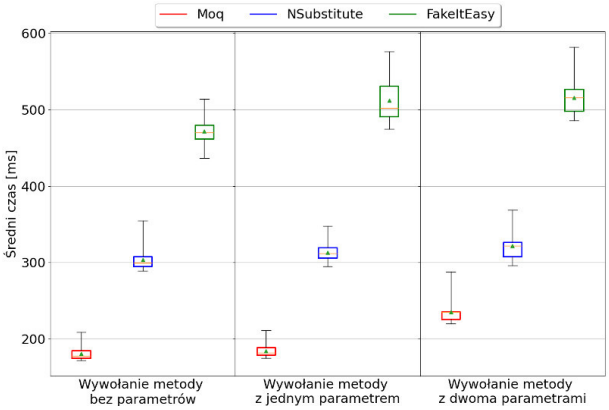
Drugi scenariusz badawczy to sprawdzenie wpływu wyboru biblioteki do izolacji kodu na czas wykonania testów jednostkowych. Na Rysunku 5 oraz Rysunku 6 zaprezentowano wykresy pudełkowe przedstawiające wyniki czasów uruchomienia owych testów.

Podobnie jak w przypadku benchmarków, Moq osiąga najlepsze wyniki. Różnicą jest jednak to, że w tym scenariuszu, szkielet ten ma większą przewagę nad pozostałymi narzędziami. Biblioteką z najgorszymi wynikami nie jest, jak dla testów wydajnościowych, NSubstitute. Największe czasy wykonania uzyskał szkielet programistyczny FakeItEasy. Dla czterech funkcjonalności jest to czas ponad dwa razy dłuższy niż czas uzyskany przez bibliotekę Moq. Jest to zaskakujący wynik, biorąc pod uwagę fakt, że narzędzie FakeItEasy osiągało niedużo gorsze wyniki niż Moq w testach wydajnościowych. Może to oznaczać słabe zarządzanie

cyklem życia obiektów przez bibliotekę FakeItEasy, ponieważ w przypadku testów jednostkowych atrapa jest wykorzystywana wielokrotnie w obrębie jednego testu.



Rysunek 5: Wykres pudełkowy – testy jednostkowe – wywołanie zwrotne, zwrócenie zadanej wartości oraz weryfikacja wywołania.



Rysunek 6: Wykres pudełkowy – testy jednostkowe – wywołanie metody bez parametrów, wywołanie metody z jednym parametrem oraz wywołaniem metody z dwoma parametrami.

7. Wnioski

Celem niniejszej pracy było porównanie wydajności szkieletów programistycznych do izolacji kodu w testach jednostkowych, takich jak: FakeItEasy, Moq i NSubstitute. Są to najpopularniejsze tego typu narzędzia dla platformy programistycznej .NET.

Przeprowadzone badania obejmują benchmarki oraz testy jednostkowe wykorzystujące analizowane biblioteki. Wskazały one, że narzędzie Moq osiąga najlepsze wyniki dla obu scenariuszy badawczych. W przypadku pozostałych szkieletów programistycznych wyniki nie są jednoznaczne. W testach wydajnościowych drugie miejsce zajęła biblioteka FakeItEasy, jednak w testach jednostkowych drugie najmniejsze czasy wykonania uzyskała biblioteka NSubstitute. Pomiar czasów uruchomienia testów jednostkowych pokazuje wpływ szkieletów programistycznych na wydajność testów. Są one głównym przeznaczeniem badanych bibliotek, dlatego lepszy rezultat NSubstitute dla tego scenariusza wskazuje, że jest to bardziej efektywne narzędzie niż FakeItEasy.

Na podstawie przeprowadzonej analizy można stwierdzić, że Moq to najlepszy szkielet programistyczny do izolacji kodu w .NET pod względem wydajności. Mimo to, pozostałe dwie biblioteki również są godnymi uwagi możliwościami. Przedstawione porównanie można by rozszerzyć o inne, mniej popularne oraz płatne rozwiązania.

Literatura

[1] H. Vocke, The Practical Test Pyramid, martinfowler.com, 2018.

[2] J. Petersen, 10 Reasons Why Unit Testing Matters, CODE Magazine, 2019 January/February.

[3] J. Coplien, Why Most Unit Testing is Waste, RBCS-US.com, 2015.

[4] Porównanie obiektu typu mock a stub, https://martinfowler.com/articles/mocksArentStubs.html, [07.03.2022]

[5] R. Osherove, Testy jednostkowe. Świat niezawodnych aplikacji. Wydanie II, Helion, 2014.

[6] T. Haukilehto, Isolated unit tests in. Net, Seinäjoki University of Applied Sciences, 2013.

[7] Porównanie testów jednostkowych z wykorzystaniem bibliotek Moq, NSubstitute i FakeItEasy, https://blog.elmah.io/moq-vs-nsubstitute-vs-fakeiteasy-which-one-to-choose/, [07.03.2022]

[8] T. Hyttinen, .NET Core 3.1 & .NET 5, Performance benchmarking in Web API use, JAMK University of Applied Sciences, May 2021.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 128–134
Received: 29 March 2022
Accepted: 27 April 2022

Comparative analysis of frameworks using TypeScript to build server applications

Analiza porównawcza szkieletów programistycznych wykorzystujących TypeScript do tworzenia aplikacji serwerowych

Marcin Golec*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The subject of the research was a comparative analysis of programming frameworks that are intended for building applications. NestJS (version 8.1.1), FoalTS (version 2.5.0) and Ts.ED (version 6.69.1) were put together. An experiment was prepared based on scenarios that focused on the response time of test applications to requests. Each of them had the same set of functionalities. NestJS turned out to be the most efficient of the compared skeletons. It achieved the best results. The worst results in each scenario were achieved by Ts.ED, especially with higher loads. The biggest differences in the comparison can be seen in studies conducted according to a scenario based on GET-type requests, and in particular with more objects in response.

Keywords: Typescript frameworks; node.js; performance comparative analysis; server application

Streszczenie

Przedmiotem badań była analiza porównawcza szkieletów programistycznych, które są przeznaczone do budowania aplikacji. Zestawiono ze sobą NestJS (wersja 8.1.1), FoalTS (wersja 2.5.0) oraz Ts.ED (wersja 6.69.1). Został przygotowany eksperyment przeprowadzony według scenariuszy, które skupiały się na czasie odpowiedzi na żądania przez aplikacje testowe. Każda z nich posiadała ten sam zestaw funkcjonalności. Z porównywanych szkieletów najwydajniejszym okazał się NestJS. Osiągał on najlepsze wyniki. Najgorsze wyniki w każdym scenariuszu osiągał Ts.ED, a w szczególności przy większych obciążeniach. Największe różnice przy porównaniu widać w badaniach przeprowadzonych według scenariusza opierającego się na żądaniach typu GET, a w szczególności z większą ilością obiektów w odpowiedzi.

Słowa kluczowe: szkielety programistyczne TypeScript; node.js; porównanie wydajności; aplikacje serwerowe

*Corresponding author

Email address : marcin.golec@pollub.edu.pl (M. Golec)

1. Wstęp

Zainteresowanie aplikacjami internetowymi utrzymuje się na bardzo wysokim poziomie. Rezultatem tego trendu jest zwiększone zainteresowanie technologiami do tworzenia tego typu aplikacji. Przybywa narzędzi, które usprawniają pracę przy tworzeniu elementów składowych aplikacji internetowych.

Możliwość korzystania z tego samego języka programowania przy rozwijaniu zarówno części dla użytkownika jak i części do zadań serwerowych stanowi zdecydowane ułatwienie dla pracy programisty. Takim językiem programowania, popularnym od wielu lat, jest JavaScript. Jednak język ten może być trudny dla osób początkujących. Firma Microsoft wydała język programowania TypeScript, który jest kompilowany do JavaScript. Dzięki zastosowaniu TypeScript wytwarzanie kodu pozbawione jest błędów, które zostają wyłapane na etapie kompilacji. Obecnie wiele szkieletów programistycznych korzysta lub umożliwia wykorzystanie tego języka programowania.

Niniejszy artykuł powstał w odpowiedzi na rosnące zainteresowanie szkieletami programistycznymi tworzonymi w oparciu o TypeScript. Jego celem jest porównanie trzech wybranych rozwiązań: NestJS [1], FoalTS [2] oraz Ts.ED [3]. Rezultatem artykułu jest utworzenie rankingu ocenianych szkieletów, pod względem wydajności. W tym celu opracowano scenariusze badawcze, w których dokonano pomiarów czasów odpowiedzi na żądania przez serwerowe aplikacje testowe. Do realizacji tego zadania wykorzystano wybrane szkielety programistyczne. Z ich pomocą wykonano trzy aplikacje zawierające te same funkcjonalności.

Każdy z wybranych szkieletów wykorzystuje Node.js [4] jako środowisko uruchomieniowe. Jest ono już znane wśród twórców aplikacji internetowych od kilkunastu lat. Popularność zyskuje z powodu wykorzystywanego języka programowania JavaScript oraz uruchamiania na wydajnym silniku V8 utrzymywanego przez Google. Coraz więcej firm decyduje się na wykorzystanie Node.js w projektach komercyjnych.

Celem badań było odwzorowanie sytuacji, które są najczęściej spotykane podczas realizacji zadań, do których wybrane szkielety programistyczne zostały stworzone. Nieodłączny element aplikacji serwerowej to komunikacja z zewnętrznymi serwisami. Z tego powodu do badań zaimplementowano obsługę żądań typu GET,

POST, PUT i DELETE. Wykorzystane dane do badań były przechowywane w pamięci komputera.

Niniejszy artykuł poświęcony jest analizie szkieletów programistycznych wykorzystujących TypeScript. Głównym celem artykułu jest porównanie wydajności definiowanej czasem obsługi żądań wybranych szkieletów programistycznych.

2. Przegląd literatury

Język PHP był i wciąż jest najpopularniejszym językiem programowania wykorzystywanym do tworzenia aplikacji internetowych. W artykule [6] porównany został on z aplikacją napisaną w języku JavaScript, pracującą na platformie Node.js. Przedstawione zostały scenariusze badawcze, które opierały się na obliczeniach liczby Fibonacciego oraz odczycie dużych plików tekstowych. Wyniki badań były korzystniejsze dla Node.js. Jedynie w przypadku obliczeń matematycznych przegrywał on z PHP.

W artykule [7] autor porównuje szkielety programistyczne do tworzenia aplikacji serwerowych. Wykorzystują one język programowania JavaScript. Porównywanymi szkieletami są: Hapi, Koa i Express. W celu porównania zmierzone zostały czasy odpowiedzi na żądania. Danymi, które były wykorzystane przy badaniach to prosty ciąg znaków oraz zmienna liczba obiektów. W rezultacie można zauważyć, że przy większych obciążeniach różnice pomiędzy porównywanymi szkieletami zwiększają się. Przy mniejszej liczbie obiektów rozbieżności wyników nie były aż tak widoczne.

Porównanie Node.js, PHP i Python jest elementem artykułu [8]. Jego głównym założeniem było porównanie ich wydajności. Nie wykorzystano tutaj żadnych szkieletów programistycznych, a jedynie języki programowania. Autorzy porównują czasy odpowiedzi na żądania oraz czas potrzebny na wykonanie obliczenia liczby na danej pozycji w ciągu Fibonacciego. W scenariuszach badawczych była uwzględniona liczba użytkowników korzystających jednocześnie z aplikacji. W wyniku przeprowadzonych porównań stwierdzono, że najefektywniejszy był Node.js, ale nie radził on sobie przy obliczeniach matematycznych.

Tworzenie aplikacji serwerowej z wykorzystaniem NestJS jest przedstawione w artykule [9]. Autor porusza kwestie związane z istotnymi aspektami w procesie tworzenia kodu. Jako główne elementy poddawane ocenie zostały wybrane: architektura, testowanie, wydajność, dokumentacja, społeczność korzystająca z tego rozwiązania oraz tempo uczenia się tego narzędzia. Każdemu z tych tematów został poświęcony odpowiedni fragment analizy poparty zarówno wykresami jak i badaniami opartymi o doświadczenie innych programistów. Artykuł przedstawia aspekty związane bezpośrednio z NestJS oraz metody wykorzystywane do badania wydajności szkieletów programistycznych.

Wykorzystanie języka JavaScript do tworzenia prostej aplikacji serwerowej jest przedstawione w artykule [10]. Autor przedstawia listingi zawierające kod aplikacji uruchamianej na platformie Node.js realizującej operacje: zwracanie odpowiedzi tekstowej, wysyłanie obrazów przez serwer, odczytywanie strumieni z dysku oraz asynchroniczny i synchroniczny dostęp do plików. Kolejna część artykułu zawiera fragmenty kodu przedstawiające funkcjonalności takie jak: obsługa żądań, definiowanie tras (ang. routes) oraz obsługa plików statycznych. Zaimplementowane są wykorzystując szkielet programistycznego Express.js [11], który dostarcza interfejs do wybranych podstawowych funkcjonalności Node.js. Rezultatem artykułu było przedstawienie zalet korzystania z połączenia Node.js i Express.js do budowaniu aplikacji serwerowych napisanych w języku JavaScript.

3. Technologie i narzędzia

3.1. Node.js

Node.js [4] jest środowiskiem uruchomieniowym dla kodu napisanego w języku programowania JavaScript po stronie serwerowej. Stworzone w tym środowisku aplikacje, dzięki działaniu asynchronicznym, nie wymagają tworzenia osobnych wątków dla każdego z żądań przesyłanych do serwera. Zastosowanie architektury zdarzeń ograniczyło spowolnienie działania. Fragmenty kodu odpowiedzialne za wczytywanie lub zapisywanie danych nie powodują zatrzymania wykonywania pozostałego kodu całej aplikacji. Kolejne instrukcje są przetwarzane do momentu, kiedy zapis lub odczyt danych zostanie zakończony. Wtedy wykonywany jest kod, który został zdefiniowany do zrealizowania po zakończeniu tych zdarzeń.

3.2. Postman

Postman [5] jest to narzędzie, które służy do pracy z interfejsami programistycznymi aplikacji. Posiada dużą liczbę funkcji, które ułatwiają zarówno testy i rozwój tych interfejsów. Jest narzędziem bezpłatnym do użytku osobistego. Jedną z funkcji szczególnie przydatnych w kontekście tej pracy jest możliwość wielokrotnego wysyłania żądania do serwera. Każde z tak wysłanych zapytań po wykonaniu zawiera zarówno odpowiedź od serwera jak i czas potrzebny do przetworzenia.

Możliwość utworzenia żądania, które będzie symulacją oczekiwanego przez aplikację ułatwia testowanie i wykonywanie pomiarów. Dla każdego z zapytań można określić parametry zawarte w URL, nagłówek, ciało żądania lub elementów związanych z autoryzacją. Wszystkie obecnie panujące standardy (REST, SOAP, GraphQL, WebSocket) wykorzystywane do komunikacji z aplikacjami serwerowymi są możliwe do skonfigurowania w tym programie. Powtórzenie wielokrotne dokładnie tak samo skonstruowanego żądania i wyeksportowanie wyników daje pogląd na to jak wydajnie serwer przetwarza dane zadanie.

3.3. NestJS

Jest to szkielet programistyczny, który jest ciągle rozwijany i ulepszany oraz opiera się głównie na języku

TypeScript. Używając tego narzędzia, można także programować w języku JavaScript. Szkielet ten jest wykorzystywany do tworzenia aplikacji serwerowych. Kod pisany w szkielecie jest kombinacją programowania obiektowego (OOP), programowania funkcyjnego (FP) oraz programowania funkcyjnego reaktywnego (FRP). Jako serwera http NestJS używa solidnego szkieletu programistycznego jakim jest Express.js. Proces wytwarzania aplikacji jest przyspieszony dzięki wykorzystaniu CLI, które jest dostarczone wraz ze szkieletem. Dzięki temu aplikacja już na starcie posiada uschematyzowaną strukturę.

3.4. Ts.ED

Proces rozwoju tego szkieletu programistycznego jest na bardzo zaawansowanym etapie. Autorzy przygotowali wiele gotowych pakietów startowych dla tego szkieletu. Przykładami mogą być: pakiet z gotowymi elementami do pracy z AWS, MongoDB, Vue.js, React i wiele innych. Podobnie jak w przypadku szkieletu NestJS, głównym założeniem jest programowanie obiektowe oraz dekoratory. Domyślnym serwerem HTTP jest Express.js, ale istnieje możliwość zmiany w konfiguracji na szkielet Koa.js. Programista używający tego szkieletu ma dostęp do interfejsu, przez który może sobie skonfigurować dowolne inne rozwiązanie i dzięki temu może według swoich preferencji dostosować środowisko robocze.

3.5. FoalTS

FoalTS jest szkieletem programistycznym zawierającym bogaty zestaw komponentów, które ułatwiają implementację aplikacji. Wraz ze szkieletem dostarczane są narzędzia do testowania, CLI, narzędzia do integracji z aplikacjami internetowymi, skrypty, zaawansowana autentykacja, obsługa bazy danych, środowiska wdrożeniowe, GraphQL i Swagger API, narzędzia do AWS oraz wiele innych. Podstawowa architektura aplikacji stworzonej przy pomocy FoalTS jest zorientowana na trzy główne komponenty: kontrolery, serwisy oraz hooki. Każdy z elementów może zostać rozszerzony i odpowiednio dostosowany do wymagań implementacyjnych.

4. Metoda badań

4.1. Opis eksperymentu

Do przeprowadzenia porównania szkieletów programistycznych do budowania aplikacji serwerowych opracowano pięć scenariuszy badawczych. W czterech pierwszych scenariuszach wykonano pomiary czasów odpowiedzi na żądania typu: GET, POST, PUT i DELETE. Krótszy czas potrzebny do obsługi pojedynczego żądania oznaczał lepszy wynik. Pomiary były wykonywane przy pomocy wbudowanej w program Postman funkcji o nazwie „Runner", która każde żądanie wywoływała w seriach 1000 razy.

4.2. Aplikacje testowe

Stworzone zostały trzy aplikacje testowe posiadające ten sam zestaw funkcjonalności. Każda aplikacja była wygenerowana automatycznie poprzez CLI dostarczone wraz ze szkieletami programistycznymi. Żaden kod nie był usuwany, dodane zostały jedynie linie, które służyły do obsługi żądań. Stworzone aplikacje miały na celu obsłużyć serię żądań HTTP. Do badań został przygotowany zbiór danych zawierający obiekty w formacie JSON o strukturze przedstawionej na Listingu 1. Symulowało to bazodanowe obiekty, które są najczęściej wykorzystywane w tego typu aplikacjach. Zestaw danych był wczytywany przy starcie aplikacji i przechowywany był w pamięci komputera. Wczytanie danych było wykonywane w momencie startu aplikacji, który nie powodował zakłóceń przy pomiarach czasu potrzebnego na obsługę wysyłanych żądań. Realizacja funkcji wywoływanych w serii zapytań była całkowicie odseparowana od czynników zewnętrznych, które mogłyby mieć wpływ na uzyskany wynik.

Listing 1: Struktura obiektu wykorzystywanego do badań

```
export type PhotoDto = {
        albumId: number;
        id: number;
        title: string;
        url: string;
        thumbnailUrl: string;
}
```

4.3. Stanowisko badawcze

W Tabeli 1 znajdują się parametry stanowiska, które zostało wykorzystane do przeprowadzenia badań.

Tabela 1: Parametry stanowiska badawczego

Sprzęt	
Rodzaj sprzętu	Laptop podłączony do zasilania
Procesor	Intel(R) Core(TM) i7-7700HQ
Pamięć RAM	20 GB
Karta sieciowa	Realtek 8821AE Wireless LAN 802.11ac PCI-E NIC
System operacyjny	Windows 10 Education N
Oprogramowanie	
Node.js	16.13.2
NestJS	8.1.1
FoalTS	2.5.0
Ts.ED	6.69.1
Postman	9.08

5. Wyniki badań

Badania zostały przeprowadzone z podziałem na liczbę zwracanych obiektów. W przypadku żądań GET było to dla 1, 100, 500, 1000, 5000 oraz 10000 obiektów. Dla badań dotyczących żądań typu POST, PUT i DELETE była wykonywana jedna seria 1000 powtórzeń. Zbierano wyniki dotyczące czasu obsługi zapytania, który był wyznaczany jako czas od momentu otrzymania żądania do momentu zwrócenia odpowiedzi. Na podstawie uzyskanych danych policzone zostały takie wskaźniki jak: średnia arytmetyczna, odchylenie standardowe oraz wartości minimalne i maksymalne. Odchylenie standardowe jest wskaźnikiem, który

pozwala na zobrazowanie w postaci liczbowej tego, jak bardzo wyniki uzyskane w badaniach są oddalone od średniej arytmetycznej. Im mniejszy wynik uzyskany przy obliczaniu odchylenia standardowego, tym uzyskane wyniki są bliższe średniej. Dla sprawdzenia istotności różnic pomiędzy porównywanymi szkieletami przeprowadzono analizy statystyczne przy użyciu testów: jednoczynnikowej analizy wariancji (ANOVA) oraz testu Tukeya. Wybór testu ANOVA był podyktowany faktem, iż porównywane dane pochodziły z więcej niż dwóch (w tym przypadku trzech) niezależnych grup. Natomiast test Tukeya stosuje się w momencie, kiedy wykryte w teście ANOVA różnice są istotne statystycznie dla dokładniejszego wskazania par różnych od siebie. Przed przystąpieniem do testu ANOVA dokonano sprawdzenia i potwierdzono spełnienie założeń dotyczących normalności rozkładu i jednorodności wariancji. Rezultat przeprowadzenia testu ANOVA pozwolił na zweryfikowanie czy występują różnice pomiędzy którąkolwiek parą szkieletów, a test Tukeya umożliwił wskazanie, które z nich były istotnie różne względem siebie.

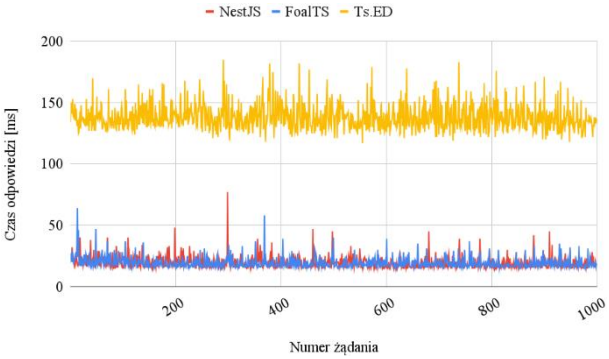
Na potrzeby tego artykułu z szeregu przeprowadzonych badań wybrano wyniki, które są najbardziej znaczące. Zawarta na końcu tego rozdziału tabela, stanowi podsumowanie zebranych wyników i jest próbą oceny badanych szkieletów programistycznych według ustalonych kryteriów.

Pierwsze przedstawione wyniki dotyczą czasu odpowiedzi serwera na żądania typu GET dla 10000 obiektów. Obrazuje to najlepiej różnice jakie wystąpiły pomiędzy badanymi szkieletami. Tabela 2 prezentuje wartości otrzymane z obliczeń statystycznych.

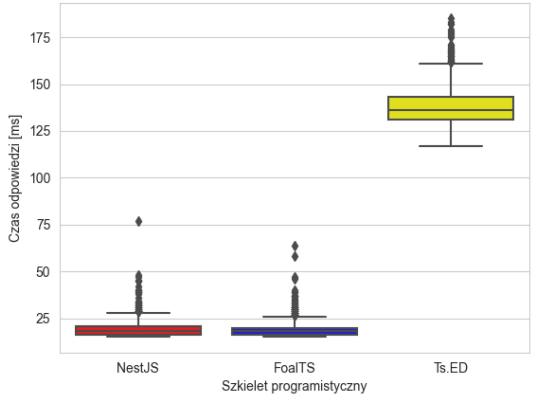
Tabela 2: Czasy odpowiedzi na żądania typu GET zwracającego tablicę 10000 obiektów

Szkielet	Czas min. [ms]	Czas maks. [ms]	Czas śr. [ms]	Odchylenie std. [ms]
NestJS	15	77	19,386	5,005
FoalTS	15	64	19,288	4,512
Ts.ED	117	185	138,217	10,584

Rysunki 2 i 3 przedstawiają wykresy dla uzyskanych wyników. Zaprezentowano wyniki w postaci dwóch typów wykresów: liniowego i pudełko-wąsy. Na wykresie liniowym został zawarty uzyskany czas odpowiedzi z każdego powtórzenia wykonanego w badaniu. Forma linii pozwala na zobrazowanie rozstrzału uzyskiwanych wartości – kształt linii odbiegający od prostego pokazuje, że wyniki są bardziej odległe względem siebie. Na wykresie pudełko-wąsy zostały zaprezentowane wartości pochodzące z obliczeń statystycznych. Rombami oznaczone są wartości odstające powyżej maksymalnych, które nie mieszczą się w ramach kwartyla. Górna linia „wąsa" odpowiada wartości maksymalnej. Górna krawędź pudełka jest rozpoczęciem trzeciego kwartyla, a odpowiednio dla pierwszego kwartyla jest to dolna krawędź. Pierwszy kwartyl jest odwzorowaniem wyniku, który jest powyżej 25% wszystkich obserwacji, a wartość trzeciego kwartylu jest powyżej 75% [12]. Istotną kwestią jest fakt, że wyniki są posortowane w kolejności rosnącej. Linia znajdująca się w środku pudełka reprezentuje medianę, a linia w dolnej części „wąsa" poniżej pudełka prezentuje wartość minimalną. W przypadku, kiedy na wykresie brakuje linii wewnątrz pudełka oznacza to, że wartość mediany jest taka sama jak jednego z dwóch kwartyli. Taką sytuację można zaobserwować przy wynikach badania czasu odpowiedzi na żądania POST, PUT i DELETE.



Rysunek 2: Czasy odpowiedzi na żądania typu GET zwracającego tablicę 10000 obiektów.



Rysunek 3: Czasy odpowiedzi na żądania typu GET zwracającego tablicę 10000 obiektów.

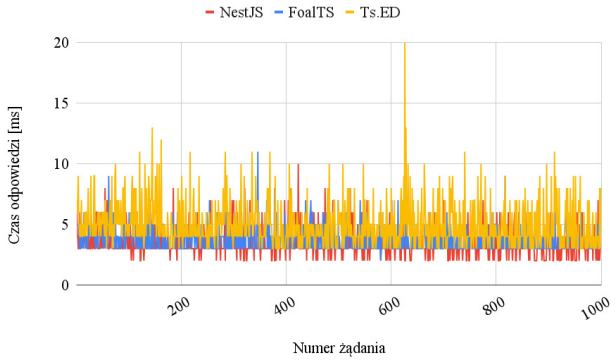
Jak widać na powyżej zaprezentowanych wynikach najwolniejszym szkieletem programistycznym okazał się *Ts.ED*. Natomiast na uwagę zasługuję tutaj fakt, że szkielety *NestJS* i *FoalTS* uzyskały wyniki zbliżone do siebie. Jest to o tyle ciekawe, że w badaniach na mniejszej liczbie zwracanych obiektów to pierwszy z nich uzyskiwał zdecydowanie lepsze wyniki. Przy tym badaniu test Tukeya nie wykazał istotnych różnic pomiędzy tymi dwoma szkieletami.

Kolejne badania dotyczyły żądań typu POST, PUT i DELETE. Wyniki dla nich zostały przedstawione w takiej samej formule jak te dla żądań typu GET, czyli najpierw tabela z wynikami liczbowymi, a potem graficzna prezentacja w postaci wykresów. Jako pierwsze zostały zaprezentowane rezultaty z badania dotyczącego żądań typu POST. Zmierzono czas od momentu wysłania do serwera zapytania zawierającego nowy obiekt JSON z taką samą strukturą jak te, które

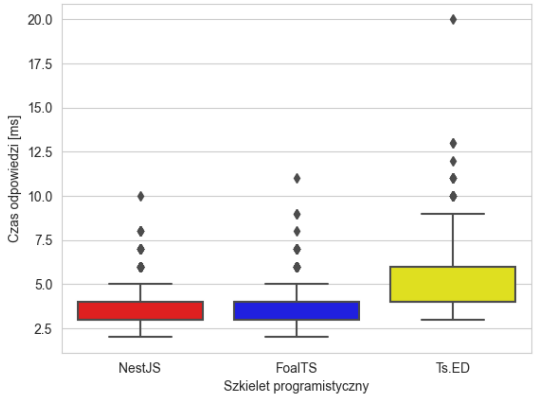
były wykorzystywane we wcześniejszych badaniach. Każdy serwer został poddany serii 1000 powtórzeń, a w każdym z nich był wykorzystany ten sam obiekt z tak samo zdefiniowanymi właściwościami. Ze względu na znacząco odstający wynik pierwszego pomiaru został on pominięty w prezentacji i obliczeniach. Wyniki zostały zaprezentowane w Tabeli 3 oraz na Rysunkach 4 i 5. Wyniki pokazują, że *NestJS* okazał się najszybszy, a *FoalTS* i *Ts.ED* były odpowiednio na drugim i trzecim miejscu. Przy tym badaniu testy statystyczne ANOVA i Tukeya wykazały istotne różnice dla każdej porównywanej pary.

Tabela 3: Czasy odpowiedzi na żądania typu POST

Szkielet	Czas min. [ms]	Czas maks. [ms]	Czas śr. [ms]	Odchylenie std. [ms]
NestJS	2	10	3,456	1,177
FoalTS	2	11	3,770	0,981
Ts.ED	3	20	4,947	1,823



Rysunek 4: Czasy odpowiedzi na żądania typu POST.



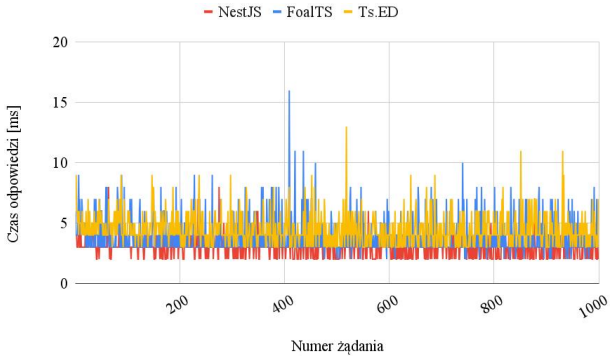
Rysunek 5: Czasy odpowiedzi na żądania typu POST.

Kolejnym badaniem było badanie czasu odpowiedzi na żądania typu PUT. Zapytanie składało się z identyfikatora istniejącego obiektu oraz nowych wartości dla obiektu, który miał zostać zaktualizowany. Czas był zmierzony od momentu wysłania żądania do momentu zwrócenia odpowiedzi przez serwer. Operacja została wykonana w serii 1000 powtórzeń aktualizujących całą tablicę wcześniej wczytanych obiektów. Podobnie jak w poprzednich porównaniach otrzymano wyraźnie odstający wynik w pierwszym pomiarze, przez co został on pominięty przy prezentacji i obliczeniach.

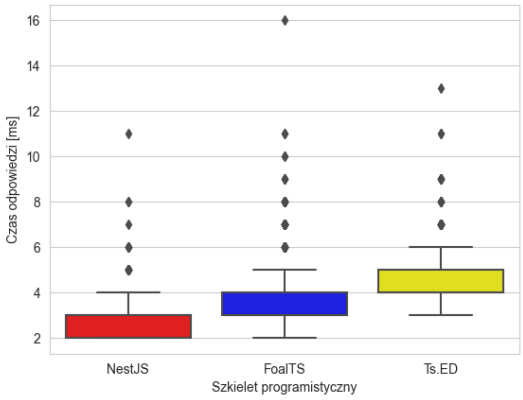
Wyniki zostały zaprezentowane w Tabeli 4 oraz na Rysunkach 6 i 7. Po przeanalizowaniu wyników można zauważyć, że najlepiej wypadł *NestJS*. Uzyskał on zarówno najszybszy średni czas oraz jego wyniki były najbardziej stabilne. Drugie miejsce zajął *FoalTS* ze średnim czasem gorszym o niecałą milisekundę. Najwolniejszy po raz kolejny był *Ts.ED,* uzyskując najsłabszy średni czas.

Tabela 4: Czasy odpowiedzi na żądania typu PUT

Szkielet	Czas min. [ms]	Czas maks. [ms]	Czas śr. [ms]	Odchylenie std. [ms]
NestJS	2	11	2,971	0,834
FoalTS	2	16	3,862	1,466
Ts.ED	3	13	4,425	1,260



Rysunek 6: Czasy odpowiedzi na żądania typu PUT.



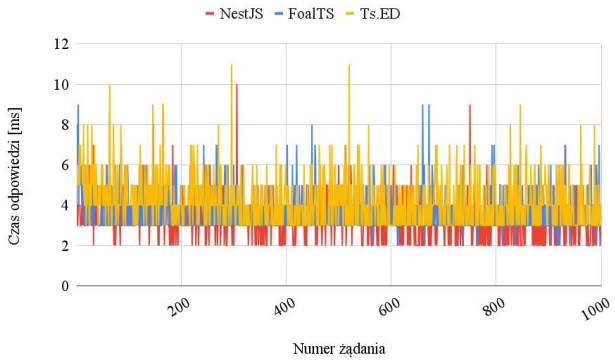
Rysunek 7: Czasy odpowiedzi na żądania typu PUT.

Ostatnim z porównań było porównanie czasów odpowiedzi na żądania typu DELETE. Pomiar czasu był wykonywany od momentu wysłania zapytania z identyfikatorem obiektu do usunięcia do momentu zwrócenia odpowiedzi przez serwer. Seria badań zaczynała się od usunięcia obiektu pierwszego aż do ostatniego, czyli tysięcznego. Po przeprowadzeniu 1000 powtórzeń pozostawała pusta tablica i badanie kończyło się. Odstający pierwszy pomiar został pominięty

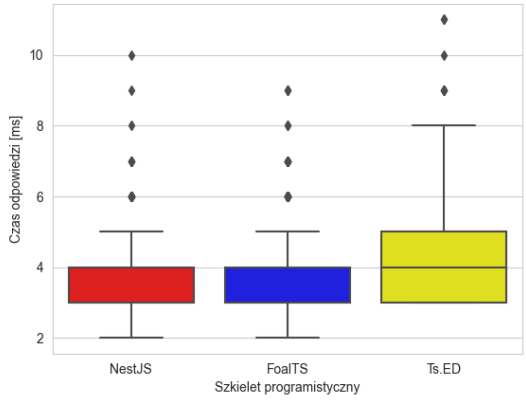
w analizach. Uzyskane wyniki zostały przedstawione w Tabeli 5 i na Rysunkach 8 i 9. Po przeanalizowaniu wyników można określić *NestJS* jako najlepszy szkielet w zestawieniu. Uzyskał on najszybszy czas oraz wykazał się najlepszą stabilnością czasu potrzebnego na obsłużenie żądania. *FoalTS* uzyskał czas nieco dłuższy. Przeprowadzone testy ANOVA i Tukeya pozwoliły na stwierdzenie, że były istotne różnice dla czasów uzyskanych przez wszystkie pary porównywanych szkieletów.

Tabela 5: Czasy odpowiedzi na żądania typu DELETE

Szkielet	Czas min. [ms]	Czas maks. [ms]	Czas śr. [ms]	Odchylenie std. [ms]
NestJS	2	10	3,236	0,999
FoalTS	2	9	3,687	1,018
Ts.ED	3	11	4,187	1,248



Rysunek 8: Czasy odpowiedzi na żądania typu DELETE.



Rysunek 9: Czasy odpowiedzi na żądania typu DELETE.

Tabela 6 zawiera podsumowanie uzyskanych rezultatów dla wszystkich przeprowadzonych badań. Wyniki dla poszczególnych szkieletów przedstawione są w kolejnych wierszach. Do przyznawania punktów zastosowano następujące kryteria:

- 0 pkt. – trzecie miejsce,
- 0,5 pkt. – drugie miejsce,
- 1 pkt. – pierwsze miejsce.

Powyższe kryteria oceniania były stosowane w momencie, kiedy występowały różnice istotne statystycznie. W przypadku, kiedy pomiędzy wszystkimi szkieletami nie występowały różnice, przyznawane było 0 punktów. W sytuacji, kiedy pomiędzy dwoma z trzech nie było istotnych statystycznie różnic, otrzymywały one punktację odpowiadającą uzyskanemu przez nie miejscu.

Tabela 6: Zestawienie wyników uzyskanych w badaniach

Badanie	NestJS	FoalTS	Ts.ED
Czas odpowiedzi żądanie typu GET, 1 obiekt	1	0,5	0,5
Czas odpowiedzi żądanie typu GET, 100 obiektów	1	0,5	0
Czas odpowiedzi żądanie typu GET, 500 obiektów	1	0,5	0
Czas odpowiedzi żądanie typu GET, 1000 obiektów	1	0,5	0
Czas odpowiedzi żądanie typu GET, 5000 obiektów	1	1	0,5
Czas odpowiedzi żądanie typu GET, 10000 obiektów	1	1	0,5
Czas odpowiedzi żądanie typu POST	1	0,5	0
Czas odpowiedzi żądanie typu PUT	1	0,5	0
Czas odpowiedzi żądanie typu DELETE	1	0,5	0
Suma	9	5,5	1,5

6. Wnioski

Celem tego artykuły było porównanie pod względem wydajności szkieletów programistycznych wykorzystywanych do budowy aplikacji serwerowych przy pomocy języka programowania TypeScript.

Najważniejszym czynnikiem wziętym pod uwagę w badaniach był czas odpowiedzi na żądania, ponieważ na jego podstawie była określana wydajność aplikacji. Odwzorowanie naturalnych warunków pracy, w których wykorzystywane są porównywane technologie zostało uwzględnione w stworzonych scenariuszach badawczych. Przeprowadzone testy ANOVA i Tukeya wykazały istotne statystycznie różnice w porównywanych czasach uzyskanych przez szkielety.

Na podstawie uzyskanych wyników można stwierdzić jednoznacznie, że wielkość przesyłanych danych ma wpływ na czas odpowiedzi. Najbardziej wyraźnie widać to w przypadku obsługi żądań typu GET. Jednocześnie, wraz ze wzrostem liczby zwracanych obiektów, zwiększały się różnice pomiędzy porównywanymi szkieletami. W pomiarach opartych o żądania typu GET to NestJS wypadł najlepiej uzyskując najkrótsze czasy obsługi. Najwolniejszy dla każdego rozmiaru danych był Ts.ED i różnica względem pozostałych konkurentów w kolejnych porównaniach wzrastała na niekorzyść tego szkieletu. Co ciekawe, FoalTS przy większym rozmiarze danych pomniejszał rozbieżność względem NestJS i przy zwracanych 5000 i 10000 obiektów różnice statystyczne nie występowały. W przypadku żądań typu POST, PUT

i DELETE to NestJS uzyskał najlepsze wyniki zajmując pierwsze, a drugie i trzecie miejsce zajęły kolejno FoalTS i Ts.ED.

Po przeprowadzeniu analiz można było stwierdzić, że najbardziej wydajnym szkieletem spośród wszystkich porównywanych był NestJS, a FoalTS i Ts.ED zajęły odpowiednio drugie i trzecie miejsce.

Literatura

[1] NestJS - szkielet programistyczny dla node.js, https://nestjs.com/, [26.11.2021].

[2] FoalTS - szkielet programistyczny dla node.js, https://foalts.org/, [26.11.2021].

[3] Ts.ED – szkielet programistyczny dla node.js, https://tsed.io/, [26.11.2021].

[4] Node.js - oficjalna strona, https://nodejs.org/, [26.11.2021].

[5] Postman - platforma do pracy z API, https://www.postman.com/, [26.11.2021].

[6] N. Chhetri, A comparative analysis of node.js (serverside javascript) (praca magisterska), Culminating Projects in Computer Science and Information Technology 5 (2016).

[7] B. Miłosierny, M. Dzieńkowski, The comparative analysis of web applications frameworks in the Node.js ecosystem, Journal of Computer Sciences Institute 18, (2021) 42–48.

[8] K. Lei, Y. Ma, Z. Tan, Performance comparison and evaluation of web development technologies in PHP, Python, and Node.js, Proceedings of 17th international conference on computational science and engineering, IEEE (2014) 661-668.

[9] A. D. Pham, Developing back-end of a web application with NestJS framework: Case: Integrify Oy's student management system (praca licencjacka), (2020).

[10] C. Peters, Building Rich Internet Applications with Node.js and Express.js, Rich Internet Applications w/HTML and Javascript Feb 6, (2017) 15-20.

[11] Express - szkielet programistyczny dla node.js, https://expressjs.com/, [26.04.2021].

[12] M. Major, J. Niezgoda, Elementy Statystyki. Część I. Statystyka opisowa, Oficyna Wydawnicza AFM, Kraków, 2003.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 135–139
Received: 30 March 2022
Accepted: 25 April 2022

C++ and Java performance on the Android platform

Wydajność języków C++ oraz Java na platformie Android

Paweł Wlazło*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article presents a comparative analysis of Java and C ++ technologies in terms of performance on the Android platform. The purpose of this work was to point to a more efficient language for developing mobile applications. The study was carried out on custom applications. The tests concerned data sorting, prime numbers determination, bitmap modification, saving to the database and reading from a text file. The series of repetitions of each test were performed on Samsung and Xiaomi devices. The following criteria were used: test execution time, CPU load, and RAM usage. The performance, in most of the carried out tests, was in favor of the C++ language, and the main difference and with the greatest discrepancy between the technologies tested was the execution time, where C++ scored 18 points, and Java 3 points. For the CPU usage, the result was the same, but value differences were smaller. A nondiscerning parameter that was the use of RAM. C++ received 11 points and Java 10.

Keywords: Java; C++; performance; Android

Streszczenie

W artykule przedstawiono analizę porównawczą technologii Java i C++ w kontekście wydajności na platformie Android. Celem tej pracy było wskazanie wydajniejszego języka do tworzenia aplikacji mobilnych. Badania przeprowadzono na autorskich aplikacjach. Testy dotyczyły sortowania danych, wyznaczania liczb pierwszych, modyfikacji bitmapy, zapisu do bazy danych i odczytu z pliku tekstowego. Serie powtórzeń każdego testu wykonane zostały na urządzeniach marki Samsung oraz Xiaomi. Kryteria, którymi się posłużono to: czas wykonania testu, obciążenie procesora, wykorzystanie pamięci RAM. Wydajność w większości przeprowadzonych testów była na korzyść języka C++. Cechą wykazującą największe różnice między badanymi technologiami był czas wykonania, gdzie C++ uzyskał 18 punktów, a Java 3 punkty. Dla wykorzystania procesora wynik był taki sam, jednak różnice wartości mniejsze. Parametrem niewskazującym faworyta było wykorzystanie pamięci RAM. Uzyskano 11 punktów dla języka C++ i 10 punktów dla Javy.

Słowa kluczowe: Java; C++; wydajność; Android

*Corresponding author

Email address: wlazlopwl@gmail.com (P. Wlazło)

1. Wstęp

Ostatnie lata to ciągły bój o lidera w branży producentów telefonów komórkowych. Regularnie słyszy się o premierach smartfonów, które w najbliższej przyszłości trafić mają do sprzedaży. Główną zaletą, rzeczą, która wpływa na atrakcyjność danego urządzenia jest jego wydajność. I tak, kiedy producenci chwalą się lepszymi parametrami telefonu od konkurencji, tak i klienci szukając nowego, starają się wybrać ten posiadający więcej pamięci RAM (*ang. Random Access Memory*) lub mający lepszy procesor. Dla potencjalnego klienta smartfon ma po prostu działać. Inaczej jednak ma się sprawa z punktu widzenia programistów na platformę mobilną. Muszą oni tworzyć aplikacje w sposób umożliwiający jej płynne działanie. Nie może, a przynajmniej nie powinno dochodzić do sytuacji, w których aplikacja działa, ale zacina się, czy jest powolna – wtedy staje się ona wręcz bezużyteczna. Wybór rozwiązania, technologii, języka programowania nie jest sprawą łatwą i oczywistą. Przeskok technologiczny w ostatnich latach wzrósł do olbrzymich rozmiarów. Smartfony posiadają podzespoły niekiedy nawet lepsze niż niejeden laptop. Jednakże to nie rozwiązuje problemu wydajności. Wraz ze wzrostem technologii wzrasta zapotrzebowanie klientów, aplikacje mają za zadanie wykonywać co raz to bardziej skomplikowane operacje, wymagające sporych nakładów pracy podzespołów.

Poniższa praca porównuje języki C++ i Java na platformie Android. Kryteria, którymi kierowano się podczas oceny wydajności obu technologii to czas wykonania danego testu, obciążenie procesora i wykorzystanie pamięci RAM. Artykuł próbuje odpowiedzieć na pytanie, który spośród dwóch języków C++ i Java zapewnia większą wydajność aplikacji działających na platformie Android.

2. Cel i zakres badań

Celem badań jest analiza wydajności języków programowania Java i C++ na platformie Android poprzez wykonanie określonych zadań, tj. sortowanie danych, obliczanie liczb pierwszych, modyfikacja bitmapy na ekranie urządzenia, zapis do bazy danych i odczyt danych zawartych w pliku txt.

Zakres badań:

- opracowanie dwóch aplikacji w języku Java i C++ na platformę Android,
- przedstawienie metodyki badań,

- przeprowadzenie testów wydajnościowych na opracowanych aplikacjach,
- analiza wyników pomiarów: czasy wykonania, wykorzystania pamięci RAM i obciążenia procesora.

3. Przegląd literatury

W publikacji *FFT benchmark on Android devices: Java versus JNI* autorzy swoją uwagę skupili na implementacji algorytmu Szybkiej Transformacji Fouriera (*ang. Fast Fourier Transform*). Badania wykonano na 35 różnych jednostkach badawczych. Autorzy w swoich badaniach pod uwagę wzięli również obszary wielowątkowości. Zauważyli, że podczas korzystania z wielu wątków język natywny radzi sobie gorzej od języka Java, a spowodowane jest to jądrem systemu Android. Jak stwierdzają twórcy eksperymentu, korzystanie z JNI (*ang. Java Native Interface*) jest w stanie zwiększyć wydajność, ale wiąże się to ze wzrostem kosztów wywołania kodu natywnego względem języka Java, a to może negatywnie wpłynąć na działanie aplikacji [1].

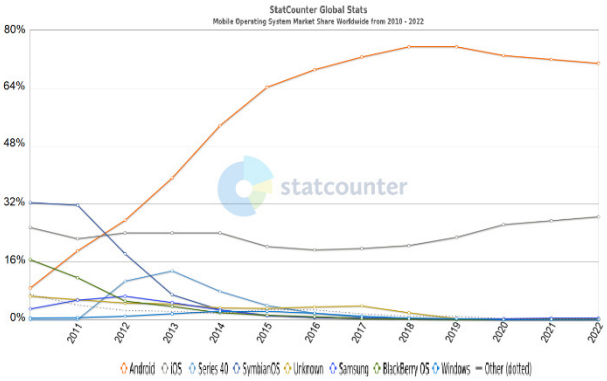
Kolejnym artykułem, w którym naukowcy starali się znaleźć odpowiedź na temat wydajności, a dokładnie odnośnie czynników wpływających na tą wydajność jest *Towards Performance-Enhancing Programming for Android Application*. Okazuje się, że to nie tylko wybór technologii w danej implementacji ma znaczenie, a czynników powodujących mniejszą bądź większą wydajność na platformie Android jest więcej. Autor Dong Kwan Kim zbadał wpływ różnych technik wykorzystywanych w programowaniu m.in. na systemy mobilne. Porównał fragmenty kodu takie jak pętla *for* w zapisie skróconym oraz pełnym, typy list, stosowanie metod rekurencyjnych, tworzenie zmiennych lokalnych, globalnych i ich wpływ na wykorzystanie procesora. Dwie, stworzone przez autora proste gry wykazały, że wybór sposobu implementacji ma znaczący wpływ na efektywność działania aplikacji na platformie Android. Nieodpowiednie, losowe stosowanie schematów zapisu funkcji przyczynić się może nawet do pogorszenia wydajności w teoretycznie lepszej technologii, a w konsekwencji zamiast przynieść korzyść, może spowodować jedynie szkody [2].

W publikacji *A performance comparision of Java and native C on Android* autorzy implementując, a następnie badając wyniki z testów, wysuwają wnioski mówiące o tym, że wydajność języków Java i C++ jest zbliżona. Według nich zapotrzebowanie na natywne technologie z każdą kolejną wydaną wersją platformy Android maleje, a jedyną potrzebą wykorzystywania narzędzia Android NDK może być użycie wcześniej stworzonego kodu w C/C++, tak by nie przepisywać danej funkcjonalności do języka Java [3].

4. Omówienie technologii

Początki systemu Android sięgają 2003 roku. Założyciele swe pierwsze kroki stawiali w Kalifornii gdzie założyli Android Inc. W 2005 roku firma Google kupiła Android Inc., jednak założyciele postanowili pozostać i wciąż rozwijać system. Platformę rozwijano z zastosowaniem jądra systemu Linux, co przyczyniło się do udostępnienia producentom telefonów komórkowych platformy Android bezpłatnie [4]. Firma Google stworzyła zestaw narzędzi dla programistów – SDK (*ang. Software Development Kit*). W 2008 roku opublikowano pierwszą wersję systemu Android [5].

Popularność systemu Android z roku na rok wzrasta. W roku 2021 udział platformy Android stanowił około 72% względem wszystkich dostępnych systemów na urządzenia mobilne [6]. Pomiędzy rokiem 2010, a 2022 system Android charakteryzował się największą popularnością (rysunek 1).



Rysunek 1: Udział w rynku mobilnych systemów operacyjnych na całym świecie pomiędzy 2010 a 2022 rokiem [6].

4.1. Android NDK i JNI

NDK to zestaw narzędzi, które stworzyła firma Google umożliwiających implementację wybranej części aplikacji Android w języku natywnym – C i C++. Według oficjalnej dokumentacji NDK, nie zaleca się korzystania ze wspomnianych języków na platformie Android, jeśli nie jest to uzasadnione [7]. Tworzenie aplikacji wyłącznie posługując się językami C i C++ przynieść może więcej szkód niż pożytku, a powstały kod stanie się nieczytelny.

JNI to tzw. natywny interfejs Javy. Umożliwia osobom tworzącym aplikację na system Android interakcję z językami natywnymi. JNI pozwala łączyć implementację w technologii Java z C czy C++. Jedną z głównych zalet omawianego rozwiązania jest możliwość wykorzystania kodu, który już wcześniej został stworzony w językach C/C++. Kolejną z zalet jest szybkość działania, a opóźnienie komunikacji JNI przy przekazywaniu ciągu znaków do aplikacji wynosi zaledwie 0,15 mikrosekundy [8]. Na listingu 1 przedstawiono metody JNI wykorzystane w badaniach.

Listing 1 Deklaracje metod JNI w języku Java (opracowanie własne)

```
public native String sorting(int checkedPosition);

public native void calculatePrimeNumbers(int range);

public native Bitmap rotateBitmap(Bitmap bitmap);

public native void initDb(String path);

public native String saveToFile(AssetManager mgr);
```

5. Metoda badań

Testy umożliwiające dokonania analizy przeprowadzono na trzech jednostkach badawczych – urządzeniach z systemem operacyjnym Android. Wykorzystane urządzenia to:

- Samsung A5 2016,
- Samsung A5 2017,
- Xiaomi Mi 11 Lite 4g.

Ponadto, w celu odczytania poszczególnych parametrów niezbędne okazało się użycie kolejnej jednostki badawczej – laptopa (Asus).

5.1. Opis testów

Każdy test wydajnościowy zawiera dwie implementacje, dla każdej technologii z osobna. Badania przeprowadzono przy użyciu pięciu rodzajów testów.

Pierwszy test zawiera implementacje pozwalające utworzyć zbiór liczb w kolejności malejącej, a następnie na posortowaniu uzyskanego zbioru w kolejności odwrotnej – rosnącej. Implementacja pozwala użytkownikowi wybrać 3 dostępne wielkości danych do sortowania, kolejno 25 000, 35 000 i 50 000. Do sortowania danych wykorzystano algorytm bąbelkowy, który cechuje się złożonością czasową równą $O(n^2)$, a to pozwala urządzeniom dokonywać dłuższych obliczeń wymagających większego wykorzystania podzespołów w porównaniu do innych algorytmów.

Kolejny test umożliwia wyznaczanie liczb pierwszych z podanego przedziału. W implementacji wykorzystano algorytm sita Eratostenesa. Eksperyment umożliwia wybranie losowej liczby z zakresu od 1 do 100 000 000, po czym rozpoczyna się proces detekcji liczb pierwszych, gdzie wykorzystanie parametrów jednostki badawczej zależne jest od wybranej wielkości przedziału.

Modyfikacja obrazu to kolejny test wykorzystany w badaniach. Stworzona metoda pozwala na kilkukrotny obrót bitmapy o 90 stopni w prawą stronę.

Kolejny, czwarty test pozwala na zmierzenie wydajności zapisu do bazy danych. Baza ta zawiera jedną tabelę z pięcioma kolumnami typu Integer.

Piąty, ostatni z założonych testów dotyczy odczytu danych zapisanych w pliku txt**.** W pliku znajduje się 1 180 452 linii tekstu (wartość losowa), a każda z nich zawiera 200 znaków – małych i dużych liter alfabetu. Przyjęto dany rząd wielkości ze względu na konieczność stosownego obciążenia urządzeń.

5.2. Warunki dotyczące testów

Przeprowadzenie badań wiąże się z koniecznością zachowania spójnych warunków pozwalających w końcowym etapie uzyskać miarodajne wyniki. W jednostkach badawczych przed przystąpieniem do testów wyłączono funkcjonalności takie jak: Wi-Fi, GPS, dane komórkowe, Bluetooth. W każdym z urządzeń pozostawiono jedynie aplikacje systemowe (i te odpowiedzialne za przeprowadzenie testów). Po każdej z prób wymuszano zakończenie procesów działających w tle. Każdy z zaimplementowanych testów powtórzono 10-krotnie. Działania te podyktowane były zamiarem wyeliminowania niepożądanych i zaburzonych wyników uniemożliwiających rzetelną analizę.

5.3. Opis pomiaru czasu

W badaniach, dla pomiaru czasu zdecydowano się wykorzystać metodę *System.currentTimeMillis.* Wywoływała się ona tuż przed rozpoczęciem testu i ponownie zaraz po jego zakończeniu. Posiadając pobrane dokładne czasy odejmowano pierwszy wynik od drugiego uzyskując wynik.

5.4. Wykorzystanie CPU i RAM

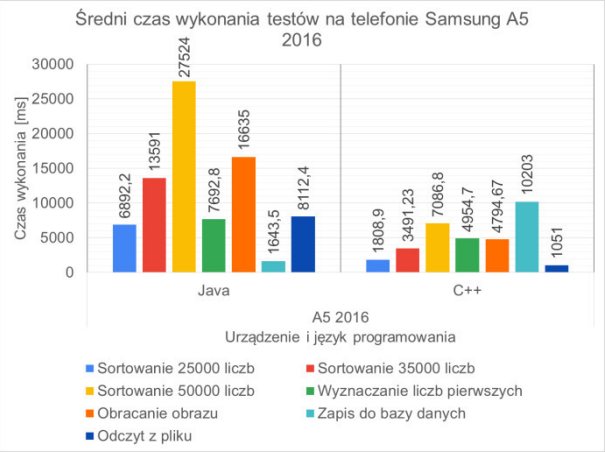
Obciążenie procesora i wykorzystanie pamięci operacyjnej RAM zmierzono za pomocą wbudowanego w kompilator Android Studio narzędzia o nazwie Android Profiler, dzięki któremu możliwe jest śledzenie wyżej wymienionych parametrów w czasie rzeczywistym zarówno w formie wykresów jak i wartości liczbowo-procentowych.

6. Analiza wyników

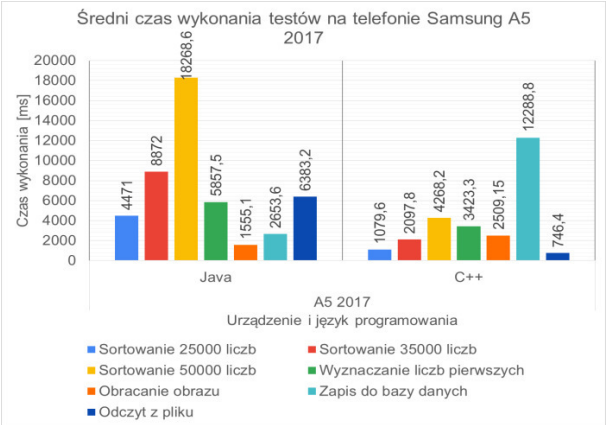
Po uzyskaniu wyników przydzielano punkty odnoszące się do każdego testu, języka oraz każdej jednostki badawczej z osobna. Badane parametry w danej technologii otrzymywały 0, 1 bądź 0,5 punktu. W każdym przypadku pod uwagę brano wyniki z jednego testu oraz odpowiedniego parametru w obu językach, a następnie porównywano wyniki, gdzie lepszy otrzymywał 1 punkt, natomiast gorszy 0. W analizie przyjęto również możliwość uzyskania bardzo zbliżonych wyników w obu badanych językach. Różnica wyników mniejsza niż 1% decydowała o przyznaniu dla danego testu i urządzenia po 0,5 punktu dla obu języków.

6.1. Czas wykonania

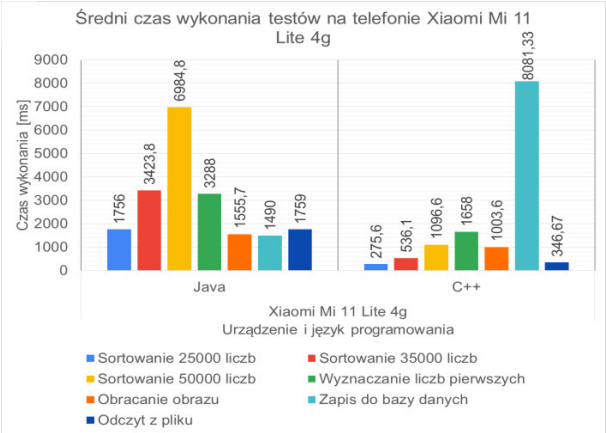
Czas wykonania danego zadania zależny jest od jednostki badawczej. Nowsze urządzenie, posiadające lepsze parametry, więcej dostępnej pamięci, wydajniejszy procesor osiąga lepsze – krótsze czasy niezbędne do ukończenia zaimplementowanego testu.



Rysunek 2: Średni czas wykonania testów na telefonie Samsung A5 2016 (opracowanie własne).



Rysunek 3: Średni czas wykonania testów na telefonie Samsung A5 2017 (opracowanie własne).

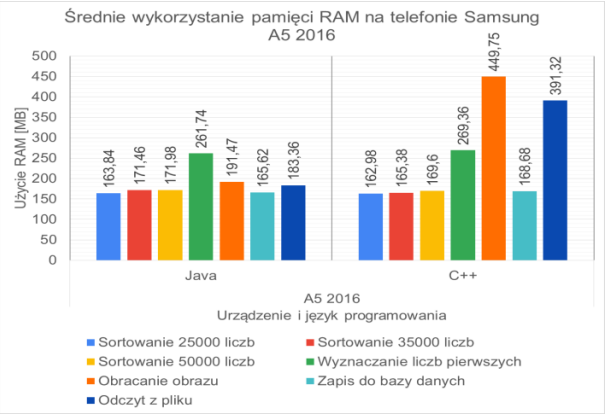


Rysunek 4: Średni czas wykonania testów na telefonie Xiaomi Mi 11 Lite 4g (opracowanie własne).

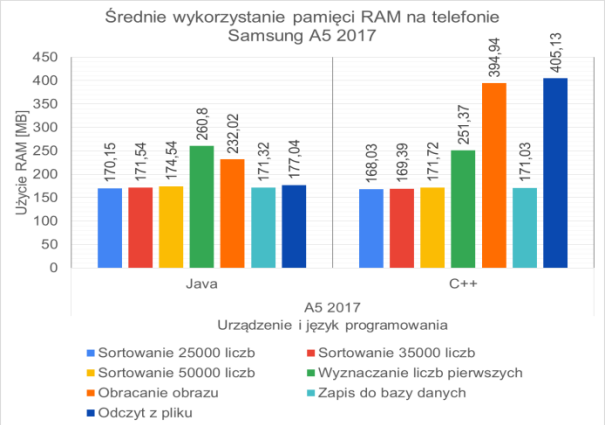
Na rysunkach 2, 3 i 4 widać, że wykorzystanie urządzenia z lepszymi parametrami powoduje krótszy czas wykonania. W większości testów język C++ wykonał zadania szybciej. Zarówno w urządzeniach marki Samsung jak i Xiaomi zapis do bazy danych okazał się być bardziej czasochłonny w języku C++ niż w Java. Ponadto telefon Samsung A5 2017 wymagał nieznacznie, ale jednak więcej czasu na ukończenie testu związanego z obracaniem bitmapy. Wszystkie pozostałe zaimplementowane testy w krótszym czasie wykonały się w języku C++.

6.2. Wykorzystanie RAM

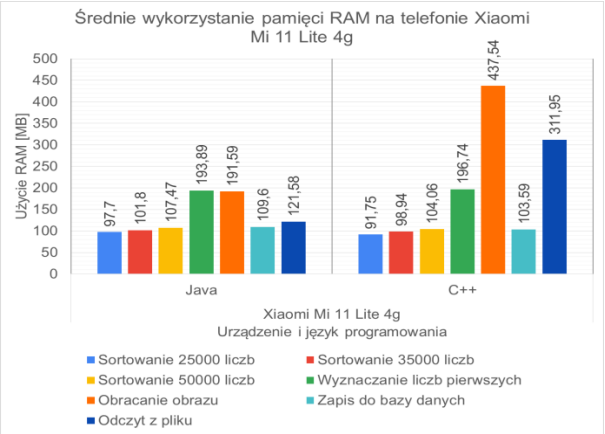
Wykorzystanie pamięci operacyjnej RAM było zróżnicowane, zależne od wykonywanego testu. Porównując słupki na rysunkach 5, 6 i 7 zauważa się, że niektóre z testów (np. sortowanie lub zapis do bazy danych) w obu technologiach uzyskały zbliżone wartości, a inne (np. obracanie obrazu) różniły się znacznie – niekiedy dwukrotnie. W przypadku pomiarów testu dotyczącego zapisu do bazy danych na urządzeniu Samsung A5 2017 oraz sortowania zbioru 25 000 liczb różnice nie przekraczały 1%. W tych konkretnych sytuacjach uznano remis obu technologii przyznając po 0,5 punktu dla każdej z nich.



Rysunek 5: Średnie wykorzystanie pamięci RAM na telefonie Samsung A5 2016 (opracowanie własne).



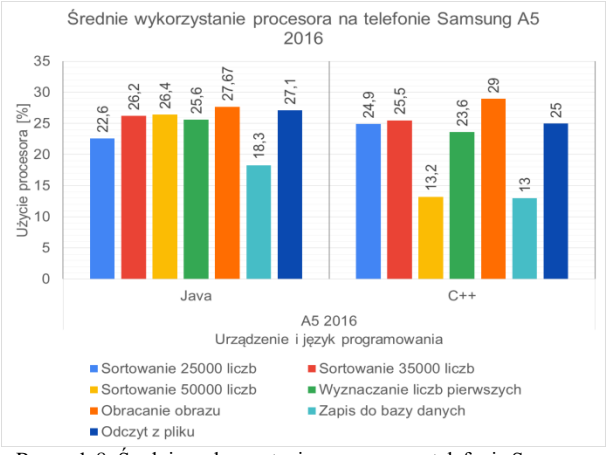
Rysunek 6: Średnie wykorzystanie pamięci RAM na telefonie Samsung A5 2017 (opracowanie własne).



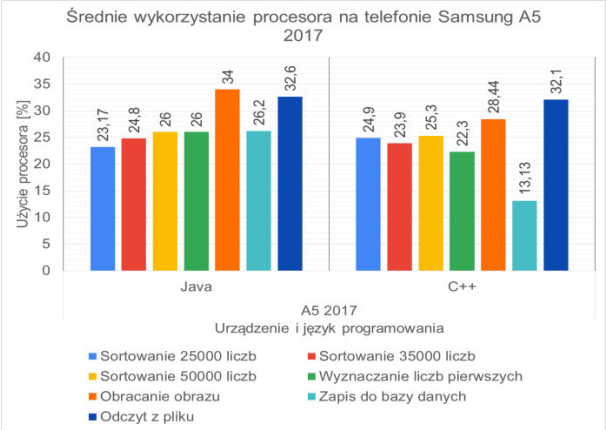
Rysunek 7: Średnie wykorzystanie pamięci RAM na telefonie Xiaomi Mi 11 Lite 4g (opracowanie własne).

6.3. Obciążenie procesora

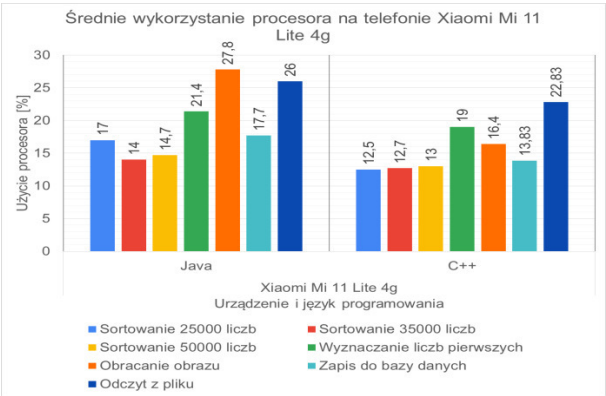
Podczas określania wydajności urządzenia, obok czasu wykonania i wykorzystania pamięci RAM, istotnym parametrem jest obciążenie procesora. Poniższe rysunki (8, 9, 10) są graficznym odzwierciedleniem średnich wyników uzyskanych podczas pomiarów obciążenia procesora na jednostkach badawczych wykorzystanych podczas badań. Obciążenie procesora w każdym z badanych urządzeń w obu technologiach kształtowało się na podobnym poziomie. W niektórych przypadkach na korzyść języka C++, w innych – Java. Sumarycznie, mimo małych różnic, pierwszy z nich był lepszy w 18 przypadkach, a drugi jedynie 3 razy.



Rysunek 8: Średnie wykorzystanie procesora na telefonie Samsung A5 2016 (opracowanie własne).



Rysunek 9: Średnie wykorzystanie procesora na telefonie Samsung A5 2017 (opracowanie własne).



Rysunek 10: Średnie wykorzystanie procesora na telefonie Xiaomi Mi 11 Lite 4g (opracowanie własne).

7. Wnioski

Badania przeprowadzone w niniejszej publikacji pozwalają stwierdzić, że w programowaniu aplikacji na platformę Android przewagę pod względem wydajnościowym ma język C++. Biorąc pod uwagę wszystkie wykonane testy oraz wszystkie jednostki badawcze, które brały udział w testach, język C++ uzyskał 46 punktów, natomiast język Java zaledwie 17. Zatem przeprowadzone testy wskazały, że wydajniejsza jest technologia C++ w aspekcie systemu Android.

Badania wykazały, że wpływ na wydajność ma nie tylko urządzenie i podzespoły, które kryją się w jego wnętrzu. Duży wpływ ma również rodzaj testu. To, że język C++ sortował dane w sposób wydajniejszy, w żaden sposób nie oznacza, że z porównywalną wydajnością będzie w stanie umieszczać i zapisywać w bazie danych. Programista powinien mieć świadomość, że technologia C++ jest wydajniejsza, ale nie nadaje się do wszystkiego. Zgubne może okazać się wykorzystywanie języka C++ do każdego złożonego obliczeniowo zadania. Mówi to zarówno dokumentacja Android [9] jak i przeprowadzone badania.

Obok wydajności, uwagę należy również zwrócić na próg wejścia i tworzenia aplikacji za pomocą języka C++ bądź narzędzia NDK. Próba tworzenia całych aplikacji w języku natywnym nie jest zalecana [7]. Osoba, szczególnie ta niedoświadczona powinna z rozwagą podejść do tematu wydajności i wiedzieć, że zgubnym jest usilna chęć wytworzenia oprogramowania bardziej wydajnego, kosztem dużo większych nakładów pracy, gdzie finalnie okazać może się, że niewystarczająca wiedza i nietrafny wybór mogą jeszcze obniżyć wydajność bądź w najgorszym wypadku doprowadzić do niepowodzenia i nieukończenia wytwarzania danej aplikacji na platformę Android.

Literatura

[1] A. Carvalho, M. Rosan, A. Bianchi, M. Queiroz, FFT benchmark on Android devices: Java versus JNI, Proceedings of the 14th Brazilian Symposium on Computer Music, Brasilia, Brazil (2013) 4-7.

[2] D. K. Kim, Towards performance-enhancing programming for Android application development, International Journal of Contents 13 (2017) 39-46.

[3] A. Ulvesand, D. Eriksson, Native code on Android: A performance comparison of Java and native C on Android. Bachelor's thesis at NADA, KTH Royal Institute of Technology (2011).

[4] J. Annuzzi, L. Darcey, S. Conder, Android. Wprowadzenie do programowania aplikacji, Helion, Gliwice, 2016.

[5] Historia i ewolucja systemu Android, https://www.androidauthority.com/history-android-os-name-789433/, [08.01.2022].

[6] Udział mobilnych systemów operacyjnych na świecie, https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2010-2022, [08.01.2022].

[7] Oficjalna dokumentacja Android NDK, https://developer.android.com/ndk, [08.01.2022].

[8] S. Lee, J.W Jeon, Evaluating performance of Android platform using native C for embedded systems, Proceedings of the International Conference on Control, Automation and Systems, ICCAS 2010, Gyeonggi-do, South Korea (2010) 1160-1163, https://doi.org/10.1109/ICCAS.2010.5669738.

[9] Oficjalna dokumentacja Android, https://developer.android.com/docs, [10.01.2022].

JOURNAL COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 140–144
Received: 31 March 2022
Accepted: 8 May 2022

A Novel Inconsequential Encryption Algorithm for Big Data in Cloud Computing

Ravi Kanth Motupalli [a,*], Krishna Prasad K.[b]

[a]*Research Scholar, Institute of Computer Science & Information Sciences, Srinivas University, Mangalore, Karnataka, India and Assistant Professor, Department of CSE, VNR VJIET, Hyderabad, Telangana, India.*
[b]*Associate Professor, Institute of Computer Science & Information Sciences, Srinivas University, Mangalore, Karnataka, India.*

Abstract
In the digitalized era of the information technology the expansion of the data usage is very high accounting for about enormous data transaction in day to day life. Data from different sources like sensors, mobile phones, satellite, social media and networks, logical transaction and ventures, etc add an gigantic pile to the existing stack of data. One of the best way to handle this exponential data production is the Hadoop network. Thus in the current scenario big industries and organizations rely on the Hadoop network for the production of their essential data. Focusing on the data generation and organization, data security one of the most primary important consideration was left unnoticed making data vulnerable to cyber attacks and hacking. Hence this article proposes an effective mixed algorithm concept with the Salsa20 and AES algorithm to enhance the security of the transaction against unauthorised access and validates the quick data transaction with minimal encryption and decryption time. High throughput obtained in this hybrid framework demonstrates the effectiveness of the proposed algorithmic structure over the existing systems.

Key words: Hadoop network; data security; cyber-attacks; Salsa20

*Corresponding author
Email address: ravikanth_m@vnrvjiet.in (Ravi Kanth. M)

1. Introduction

In this digital era the data generated every day for multiple purpose accounts for a very large pile. Data may be of any time from either sensors or devices, Public media or concerns, Cognitive data or transaction data, etc are getting added to the cascading data forming a big torrent called big data. This avalanche of data are the data expansion in the last decade from little to huge volume. As there are enormous amount of data in every information Big data receives its limelight and demonstrated outstanding performance in the field of Digital information technology. These large volume of data can be either in an organized or unorganized format. Uncertainty of the big data is estimated by the organized format of the digital data that is generated in current scenario with the advancement of the novel and modern communication techniques. This is just an initial phase there is a bright future of expansion for the big data in the upcoming decades. The estimated revenue forecast of the big data in US has been illustrated in the Figure 1. This shows the relevance of the big data investement and its progressive expansion.

Big Data is delivered in correspondence with the transmission of data in a broader spectrum accumulated from different sources.Thus Big Data is obligated to set up the data mining computations. The primary requirement of this framework is to assure the security of the data along with its structural analysis. The Information engineering has developed its analysis in an advanced way migrating from the centralized framework to the distributed framework. In this corresponding study mobile data is taken for the consideration. With the developed gadget usage and the production the amount of storage space in the mobile phones are increased to bigger levels. With high storage device there comes issues with the performance and heat dissipation factors. To avoid such issues and to centralize the storage Big data in cloud storage was introduced to the mobile users. Hence we consider the analysis of the big data stored in the Hadoop Distributed File System (HDFS) [1]. The users of the big data are concerned in gaining more knowledge about the data structure to earn a good profit but fail to consider the security of the data which is the most important attribute of the cognitive communication.

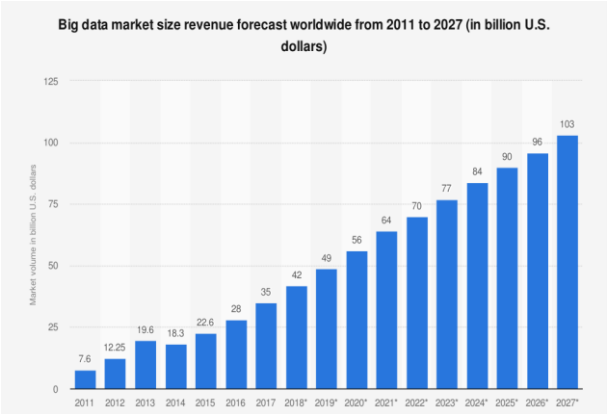


Figure 1: Revenue forecast of Big Data (Satsita, 2021).

Security is an primary factor to be assured in every storage system to save the data from the unfortunate attacks and known hackers. Upon investing it is known that HDFS has no organized security framework to secure the data. Hence the current study intends to pro-

pose a inconsequential encryption algorithm to assure the security of the data. HDFS is an distributed network which is built at less expenses with high tolerant against the faults [2]. When enormous data is being used in the HDFS, high throughput is achieved making it qualified to handle the Big Data. The size of the HDFS varies from Giga bytes to Tera Bytes which is facilitated to support the larger files [3]. Thus in a single instance larger data files.

Hadoop is anticipated to handle large amount of data , irrespective of its structure. Hadoop works on the MapReduce structure to handle the index of the web search[4,5]. Hadoop has an inbuilt security framework that makes use of cryptographic technique to mix the data and save it in the cloud. The sensitive data is encrypted to enhance the security where the raw data is transferred in to the cipher text and transformed in to arbitrary data.

The essential intension of this article is to design a technique through which the secure transformation of the data can be done with easy recovery of lost data without increasing the overhead in the computation, communication and storage. Hence in this study an hybrid algorithm mixing AES and Salsa20 algorithm was proposed. In this hybrid algorithm the size of the key is developed up to 256 bits. Apache ranger is used to achieve the cross control over the security of the data. In the proposed work Salsa20 encryption algorithm is executed before moving the HDFS and the data mining computation is done using the mapper class.

2. State of Art

Big Data is a messy, Jumbled and large which has an uncontrollable approach. The accumulated information is segregated and classified to identify the data type in an organized form to be used by a business or an organization[6,7]. Several studies and researches were done to enhance the security of the data stored in the cloud. There are broadly two types of encryption system which was commonly in practise known as symmetric and asymmetric key encryptions[8,9]. Lin.H.Y et al attempted to propose an hybrid encryption system mixing AES and pairing of the HDFS system to improve the security[10]. An novel encryption technique based on the new instruction encryption was proposed by Acharya et al [11] to upright the use of a trusted environment.

Privacy is the term which is used to denote the protected handling and storage of data in the cloud network [12]. Cloud computing has occupied the digital market due to the specialized features like consistency, less expensiveness, rigid against faults and scalability [13,14]. With the suggested usage of symmetric encryption , blowfish encryption algorithm was used in the IOT clod controlling FPGA based devices [15]. Data masking technique was identified as the simple and efficient security enhancing technique [16]. Hamid et al in his paper highlighted three important key points like security in Big Data, retrieval of information using context aware concepts and ontology integrated big data [17].

Sachin et al had made an conceptual study on the big data security and supported the arguments uplifting the use of high volume data and RDBMS [18]. Bhargavi et al proposed that the original DS is used for the effective use of Big data [19]. The biggest challenge in the Big Data security is the information security , where the data is vulnerable to data leaks, Unauthorized access, DoS attacks, etc. The impression of handling the unstructured enormous data was changed by the Hadoop.

Aditham et al [20] developed an algorithm to detect the attack on the network in two steps. Initially the control of the algorithm was framed in the first step and the process to merge the replicating nodes was developed in the second step. Reddy et al [21] specifically developed detection algorithm for the cloud environment where the access control was the main objective. But this method failed to assure security to the data when providing restricted control over the access. In the proposed framework four stages of the security was ensured. In the initial stage the security for the access control was ensured with the existing technology. In the second or registration process both the receiver and the transmitter were registered to the trust center and service provider to obtain the secret key. In the third stage authentication of the user is validated against the entry of the user with the service provider so as to ensure the access control over the data. In the last phase the secret key of the user is updated or revoked upon the detection of the leakage or misusage.

3. Challenges in the Cloud Storage Security

The data in the cloud is stored in a remote location where the attack or hacking of the cloud environment without proper security features happen, there may be collateral loss of the data assets and information. The physical damage to the cloud server machines may result in unrecoverable data loss. In order to assure the data security the information is replicated and stored in different location where it should be managed effectively to avoid the confusion and data replication issues to the end users. Though the cloud servers are handled by the third entity the service provider themselves should have restricted access to the data to ensure the privacy of the data stored. Hence in order to assure the data privacy through limiting the data access to the service providers, all the data hiding and recovery process of encryption, decryption and scrambling are given to the user side. The security and privacy of the data in big data are considered to be the two giant attributes of its performance analysis

4. Research Methods

The architecture for storing data in cloud has several framework. The proposed framework has four parameters named as trust center, data transmitter, receiver and server in cloud environment. Trust center is an sturdy, vigorous, and enormous entity that is framed for the assurance of security of data in the cloud environment. The relation between the receiver, transmitter and the server was effectively and efficiently managed by this

entity. This entity is also responsible for tracking the secret key in all the transaction and has powers to revoke the key when misused. Each transmitter should register themselves in this trust center to obtain the benefits of secure communication. Similarly each receiver should register in to the trust center to obtain authenticated access to the data.

Processes like slicing, transformation of data, encryption, decryption, scrambling and unscrambling, consolidation, and recovery are the basic processes involved in any data transfer. Using an effective secret key the transmitter uploads or stores data in the cloud.

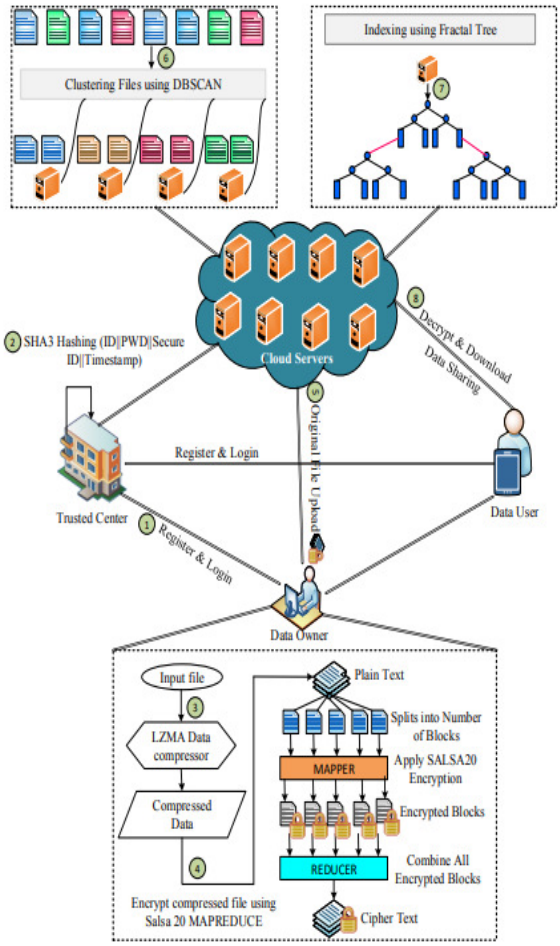


Figure 2: Proposed framework for the inconsequential encryption algorithm in Big Data.

When transferring the data raw information is scrambled and converted in to cipher text. Then the cipher text is decrypted and unscrambled at the other end using the secret key provided by the owner to recover the original data at the receiver end.

The process flowchart is illustrated in the Figure 3. Salsa20 algorithm is an efficient stream cipher which can effective transform key bits from 256 to 264 streams with occasionally susceptible 64 byte blocks per stream. This 64 byte block was represented in 16 word format which is obtained as a result of 320 reversible alterations, one word for each modification [22,23]. The corresponding output was amended to the producing 64 byte output. In each change, two words, one original word and the rotated version of the other was xor-ed twice. Thus each round is performed as 4 parallel quarter rounds.

There is separate function for each quarter round. There is row round function for each quarter which modifies the rows. Then column round function is processed changing the column. Then a double round function is performed for the specified number [24,25]. In this article 20 rounds of Salsa 20 algorithm was proposed. Then the resulting 64byte is transformed in to 16 word format using little endian method.

Thus a 64byte output is obtained. Encryption tools along with the authentication and platform manager was used to improvise the Hadoop security. In this proposed module 10 rounds of Hadoop clustering with specified configuration was done. Deployment validation is highly prioritized based on the running concerns and the Apache ranger is given steady administration platform to configure and strategize the secured data in each cluster [26,27]. The ranger key management service is used for the implementation of HDFS encryption. Apache officer is used for the authenticated data access through the centralized fine grain approval system [28].

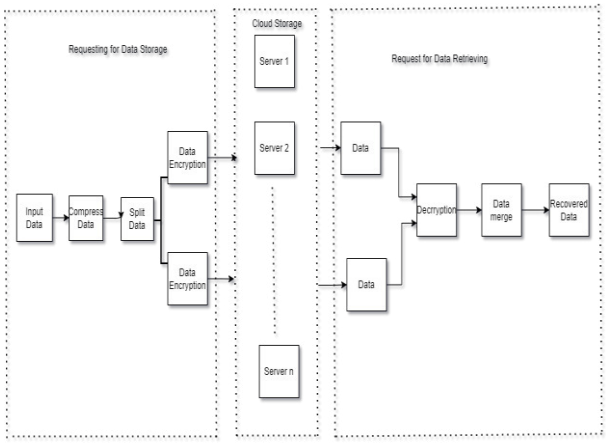


Figure 3: Proposed framework flowchart.

5. Results and Discussion

For the experimental procedure 10 cluster was created where the first 8 nodes serve as MapReduce clients and as Data Node servers. The last two nodes function as MapReduce scheduler and Name Node manager. The first eight nodes are a 8V processor operating with 8GB random access memory with Ethernet NIC. The last two nodes are 16 V processor with 32GB memory. Hadoop replica was not allowed in this system. The sequenced process of encryption and decryption of enormous data pushes and stacks them into a HDFS.

The encryption time was calculated in milli seconds. To make the system more responsive and more dynamic the execution period should be small enough. Similarly, decryption time should be less enough to accelerate the system. The third important parameter for consideration is Avalanche effect where a small change in the input has big and drastic effect at the output. The encryption time analysis of the proposed system with other encryption and decryption techniques was shown in the Fig-

ure 4. This illustrates that the proposed methodology is more suitable for the foresaid applications.

From the Figure 4, it is evident that the encryption and decryption time of the proposed model is short enough to support the processing of big data files quickly and actively. The throughput of the decryption process is observed to higher than that of the encryption process. The throughput is calculated as the ratio of plain text to the encryption time in milli seconds.

Performance analysis was done with the other encryption algorithms like AES and blowfish algorithm. From the result it is evident that the proposed technique has higher encryption than that of the other prevailing techniques.

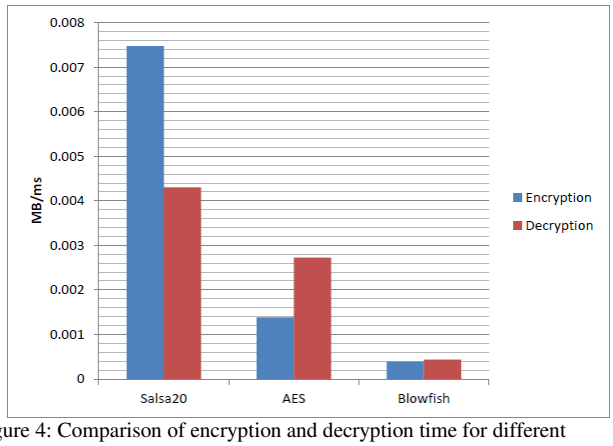


Figure 4: Comparison of encryption and decryption time for different algorithms.

Due to the smaller key size the throughput of the blowfish algorithm was very less. The security of the data in the proposed model is measured with the parameter called avalanche effect. A small change in the input stream causing a big impact on the output stream is called avalanche effect. This method allows the data scrambling in a reversible format to enhance the security of the data. Thus the overall performance of the proposed Salsa20 hybrid algorithm is found to have minimal running time, increased throughput, with the augmented avalanche effect guarantying the security of the stacked data in HDFS.

6. Conclusion

The overall concept of the proposed framework is to secure the data transaction in the cloud system through an effective encryption algorithm in the Hadoop network. In this proposed model Salsa20 hybrid algorithm was used for effective encryption and efficient key management. The overall performance efficiency of the proposed hybrid algorithm is accessed based on the speed of transmission and the less computational time for multiple nodes in a cluster. Thus the proposed framework is guaranteed for the authenticated data access, confidential data storage and controlled access of the Hadoop network. The low computational time and the improvised throughput facilitated the immediate transaction of the data with utmost security. This study is majorly focused on the mobile data transaction to the cloud storage. The future work may include designing framework for the most complicated IoT transaction like defence data storage in the cognitive cloud, Big Data accessing in the educational domain and the large database secured in the healthcare sector.

References

[1] Big Data market revenue forecast worldwide 2011-2027, by Wikibon; Silicon ANGLE Available: https://www.statista.com/statistics/254266/global-big-data-market-forecast/ [04.02.2019].

[2] Apache Hadoop 3.2.0-Hdfs Architecture, Available: https://hadoop.apache.org/docs/r3.2.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign [04.01.2019].

[3] D. Borthakur, The Hadoop distributed file system: Architecture and design, Hadoop Project Website 11 (2007) 21.

[4] M. Dhatrak, H. Panadiwal, Privacy-Preserving Mining using Data Encryption scheme for Hadoop Ecosystem, In International Journal of Advanced Research in Science, Engineering and Technology 5(4) (2018) 5578-5585.

[5] S. Singh, M. Sharma, The Prototype for Implementation of Security Issue in Big Data Application using Hadoop Server, International Journal of Computer Applications 145(13) (2016) 9-13.

[6] S. Ghemawat, J. Dean, MapReduce: Simplified data processing on large clusters, ACM Commun. Mag. 51(1) (2008) 107-113.

[7] Y. Kumar, R. Munjal, H. Sharma, Comparison of symmetric and asymmetric cryptography with existing vulnerabilities and countermeasures, (IJAFRC) 1:6 (2014) 2348-4853.

[8] B.H. Lee, E.K. Dewi, M.F. Wajdi, Data security in cloud computing using AES under HEROKU cloud, In The 27th Wireless and Optical Communications Conference (WOCC2018) (2018) 1-5.

[9] P. Mahajan, A. Sachdeva, A Study of Encryption Algorithms AES, DES, and RSA for Security, Global Journal of Computer Science and Technology Network, Web & Security 13(15) (2013) 15-22.

[10] H.Y. Lin, S.T. Shen, W.G. Tzeng, B.S.P. Lin, Toward data confidentiality via integrating hybrid encryption schemes and Hadoop Distributed File System, in the Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA) (2012) 740-747.

[11] J. Cohen, S. Acharya, Towards a Trusted Hadoop Storage Platform: Design Considerations of an AES Based Encryption Scheme with TPM Rooted Key Protections," IEEE 10th International Conference on and Autonomic and Trusted Computing (UIC/ATC), Ubiquitous Intelligence and Computing (2013) 444-451.

[12] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, R. Buyya, Ensuring security and privacy preservation for cloud data services, ACM Computing Surveys 49(1) (2016) 1-39.

[13] W. Song, B. Wang, Q. Wang, Z. Peng, W. Lou, Y. Cui, A privacy preserved full-text retrieval algorithm over

encrypted data for cloud storage applications, Journal of Parallel and Distributed Computing 99 (2017) 14-27.

[14] P.V. Bharati, T. Sita Mahalakshmi, Data storage security in cloud using a functional encryption algorithm, In Emerging Research in Computing, Information, Communication and Applications, Springer Singapore (2016) 201-212.

[15] K.N. Prasetyo, Y. Purwanto, D. Darlis, An Implementation of data encryption for internet of things using blowfish algorithm on FPGA. International Conference on Information and Communication Technology (2014) 75-79.

[16] A. Abo-alian, N. L. Badr, M. F. Tolba, Data Storage Security Service in Cloud Computing: Challenges and Solutions, In Multimedia Forensics and Security, Springer International Publishing (2017) 25-57.

[17] H. Bagheri, A.A. Shaltooki, Big Data: Challenges, Opportunities and Cloud Based Solutions, International Journal of Electrical and Computer Engineering (IJECE) 5(2) (2015) 340-343.

[18] S.A. Thanekar, K. Subrahmanyam, A.B. Bagwan, A Study on MapReduce: Challenges and Trends, Indonesian Journal of Electrical Engineering and Computer Science (IJEECS) 4(1) (2016) 176-183. DOI: 10.11591/ijeecs.v4.i1.pp176-183.

[19] I. Bhargavi, D. Veeraiah, T.M. Padmaja, Securing BIG DATA: A Comparative Study Across RSA, AES, DES, EC and ECDH, In Computer Communication, Networking and Internet Security. Lecture Notes in Networks and Systems 5 (2017) 355-362.

[20] S. Aditham, N. Ranganathan, A System Architecture for the Detection of Insider Attacks in Big Data Systems. IEEE Transactions on Dependable and Secure Computing (2017) 1–1. doi:10.1109/tdsc.2017.2768533.

[21] Y. Reddy, Big Data Processing and Access Controls in Cloud Environment. 2018 IEEE 4th International Conference on Big Data Security on Cloud (Big Data Security), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS). doi:10.1109/bds/hpsc/ ids18.2018.00019.

[22] D. J. Bernstein, The Salsa20 family of stream ciphers, In New Stream Cipher Designs. Berlin, Germany: Springer, (2008) 84–97.

[23] S. Potteti, N. Parati, Secured Data Transfer For Cloud Using Blowfish, International Journal Of Advances In Computer Science And Cloud Computing 3(2) (2015) 17-22.

[24] P. Ghosh, V. Thakor, P. Bhathawala, Data Security and Privacy in Cloud Computing Using Different Encryption Algorithms, International Journal of Advanced Research in Computer Science and Software Engineering 7(5) (2017) 469-471.

[25] K.Sekar, M Padmavathamma, Comparative Study of Encryptio Algorithm over Big Data in Cloud Systems, International conference on Computing for Sustainable Global Development (INDIACom) (2016) 1571-1574.

[26] B.T. Reddy, K.B. Chowdappa, S.R. Reddy, Cloud Security using Blowfish and Key Management Encryption Algorithm, International Journal of Engineering and Applied Sciences (IJEAS) 2(6) (2015) 59-62.

[27] G. Saini, N. Sharma, Triple security of data in cloud computing, International Journal of Computer Science and Information Technologies 5(4) (2014) 5825-5827.

[28] D.S. Elminaam, H.M. Abdual-Kader, M.M. Hadhoud, Evaluating the performance of symmetric encryption algorithms, Int. J. Netw. Secur. 10(3) (2010) 216-222.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 145–151
Received: 10 April 2022
Accepted: 20 April 2022

Comparison of LeNet-5, AlexNet and GoogLeNet models in handwriting recognition

Porównanie modeli LeNet-5, AlexNet i GoogLeNet w rozpoznawaniu pisma ręcznego

Bartosz Michalski*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The aim of the study was to compare the accuracy of handwriting recognition and the time needed to classify data from the test sets. The Lenet-5, AlexNet and GoogLeNet architectures were used for the research. All selected architectures are models of convolutional neural networks. The research was carried out with the use of image databases, handwritten digits MNIST and handwritten letters EMNIST. After the tests, it was found that the GoogLeNet model showed the highest accuracy, and the LeNet-5 the lowest. However, the LeNet-5 model needed the least time to complete the task, and GoogLeNet the most. On the basis of the obtained results, it was found that increasing the complexity of the model positively influences the accuracy of object classification, but significantly increases the demand for computer resources.

Keywords: convolutional neural networks; handwriting classification

Streszczenie

Celem badania było porównanie dokładności rozpoznawania pisma odręcznego oraz czasu potrzebnego na klasyfikację danych ze zbiorów testowych. Do badań wykorzystano architektury Lenet-5, AlexNet i GoogLeNet. Wszystkie wybrane architektury są modelami konwolucyjnych sieci neuronowych. Badania przeprowadzono z wykorzystaniem baz obrazów odręczenie pisanych cyfr MNIST i odręcznie pisanych liter EMNIST. Po wykonaniu badań stwierdzono, że największą dokładnością wykazał się model GoogLeNet, a najmniejszą LeNet-5. Natomiast najmniej czasu na wykonanie zadania potrzebował model LeNet-5, a najwięcej GoogLeNet. Na podstawie otrzymanych wyników stwierdzono, że zwiększanie złożoności modelu wpływa pozytywnie na dokładność klasyfikacji obiektów, ale znacznie zwiększa zapotrzebowanie na zasoby komputera.

Słowa kluczowe: Konwolucyjne sieci neuronowe; klasyfikacja pisma odręcznego

*Corresponding author

Email address: **bartosz.michalski@pollub.edu.pl** (B. Michalski)

1. Wstęp

Historia sztucznych sieci neuronowych (SSN) sięga lat 40. XX wieku, kiedy to McCulloch i Pitts (1943) [18] przedstawili model matematyczny aktywności neuronowej mózgu, a Hebb (1949) [1] stworzył mechanizm uczenia się oparty na wzmocnieniu, aby wyjaśnić uczenie się w ludzkim mózgu. Rosenblatt (1958) [2] stworzył następnie model obliczeniowy elementów przetwarzających mózg, zwanych perceptronami.

Jedną z pierwszych udanych prób użycia głębokiego uczenia było stworzenie w 1989 roku architektury LeNet-5 [3]. W 1998 roku architektura ta wykazała się największą dokładnością w klasyfikacji odręcznie pisanych cyfr wśród obecnych wtedy na rynku rozwiązań. To napędziło dalszy rozwój użycia głębokiego uczenia w sztucznych sieciach neuronowych.

Kolejnym przełomowym momentem było użycie mocy obliczeniowej kart graficznych w trenowaniu sieci i późniejszej klasyfikacji danych. Pierwszym wyraźnym sukcesem było stworzenie architektury AlexNet, która wykazała się o ponad 10% mniejszym poziomem błędów w konkursie ImageNet w 2012 roku względem następnego wyniku w rankingu [4].

Dziś popularność SSN stale rośnie. Szerokie zastosowanie SSN jest możliwe dzięki sposobie działania i budowie, które mają odwzorowywać biologiczny system nerwowy człowieka. SSN są obecne w wielu miejscach, wykorzystywane do codziennej pracy wymagającej analizy danych, klasyfikacji czy sterowania. Dobrymi przykładami mogą być kontrola bagażów na lotnisku [5], rozpoznawanie twarzy [6], rozpoznawanie pisma odręcznego [7], prognozy giełdowe [8], a nawet pogody [9].

Ciągle powstają kolejne architektury cechujące się coraz większą dokładnością wykonywanej pracy. Od roku 2010 do 2017 odbywały się coroczne zawody klasyfikacji obiektów 2D z bazy ImageNet [10]. Aby wygrać SSN musiała osiągnąć możliwe najmniejszy poziom błędu podczas klasyfikacji danych testowych. Omawiane w pracy architektury AlexNet i GoogLeNet zwyciężały odpowiednio w roku 2012 i 2014 [4].

Istnieje wiele narzędzi ułatwiających tworzenie modeli sieci neuronowych takich jak biblioteka *Keras* zintegrowana z *TensorFlow* czy biblioteka *PyTorch*, dzięki którym nawet niedoświadczone osoby mogą szybko zacząć swoją przygodę z głębokim uczeniem

maszynowym i sieciami neuronowymi. Stworzone modele SSN można w prosty sposób wytrenować i przetestować korzystając z wielu dostępnych baz danych m.in. *MNIST* [11], *EMNIST* [12] *lub ImageNet* [10], zawierających nie tylko proste obiekty, jak zdjęcia obiektów czy pisma ręcznego, ale nawet próbek biologicznych pobranych od pacjentów.

W pracy porównano wspomniane wyżej architektury, czyli Lenet-5, AlexNet i GoogLeNet pod kątem dokładności rozpoznawania odręczenie pisanych liter i cyfr oraz czas potrzebny na klasyfikacje danych testowych. W kolejnych rozdziałach opisane zostały konwolucyjne sieci neuronowe, którymi te architektury są oraz same architektury. Następnie przedstawiono sposób realizacji badań, postawione hipotezy badawcze, implementacje modeli oraz opisano wybrane do badań bazy danych. Na koniec zamieszczone zostały uzyskane wyniki, ich opis oraz wnioski.

2. Konwolucyjne sieci neuronowe

Głównym obszarem, w jakim wykorzystywane są konwolucyjne sieci neuronowe (*CNN*) jest rozpoznawanie wzorców na obrazach. To pozwala zakodować w architekturze cechy charakterystyczne dla obrazu, dzięki czemu sieć lepiej nadaje się do zadań związanych z obrazem - przy jednoczesnym dalszym zmniejszeniu parametrów wymaganych do skonfigurowania modelu [16]. Różne wersje i konfiguracje CNN, potrafią osiągać dokładność ponad 99% podczas klasyfikacji odręczenie pisanych cyfr oraz ponad 95% podczas klasyfikacji odręczenie pisanych liter łacińskich [19].

Na CNN składają się trzy rodzaje warstw. Są to warstwy splotowe (konwolucyjne), warstwy łączące (ang. *Pooling Layers*) i warstwy w pełni połączone (ang. *Fully-Connected Layers*).

Warstwy splotowe, jak nazwa wskazuje, są podstawą CNN. Zawierają w sobie filtry (ang. *kernels*), których zadaniem jest wyodrębnienie cech odróżniających od siebie obrazy. Filtr to dwuwymiarowa tablica wag, która reprezentuje część obrazu. Jest nakładany na obszar obrazu, a następnie obliczany jest iloczyn skalarny między pikselami wejściowymi, a filtrem. Wynik jest podawany do macierzy wyjściowej. Filtr przesuwa się krok po kroku, powtarzając proces, aż przesunie się przez cały obraz. Ostateczne dane wyjściowe są znane, jako mapa funkcji lub mapa aktywacji.

Warstwy łączące mają za zadanie zmniejszenie obrazu. To oznacza mniejszą liczbę parametrów do wyuczenia przez sieć. Dzięki temu model CNN jest prostszy i wydajniejszy. Warstwa łącząca działa na każdej mapie aktywacji na wejściu i skaluje jej wymiarowość za pomocą funkcji „MAX". W większości CNN warstwy łączące mają postać warstw z maksymalnym łączeniem, z filtrami o wymiarowości 2×2, nakładanymi z krokiem 2 wzdłuż przestrzennych wymiarów danych wejściowych. To skaluje mapę aktywacji do 25% oryginalnego rozmiaru [16].

W warstwie w pełni połączonej, połączenia między neuronami wyglądają, że każdy neuron jest połączony z neuronem warstwy poprzedniej. Warstwa ta próbuje wygenerować wyniki klas z mapy aktywacji, które zostaną użyte do klasyfikacji.

3. Wykorzystane Architektury

3.1. LeNet-5

LeNet-5 to architektura, którą Yann LeCun i in. zaprezentowali w 1989 roku [3]. Była jedną z pierwszych CNN. Twórcy wykorzystywali stworzony program do rozpoznawania odręcznie zapisanych kodów pocztowych na listach. Dokładność klasyfikacji wynosiła 90%, co pokazywało, że SSN mogą mieć praktyczne zastosowania. Model składa się z siedmiu warstw, nie licząc warstwy wejścia. Wszystkie warstwy zawierają parametry, które można trenować (wagi). Wejście to obraz o wymiarach 32 x 32 piksele [13]. LeNet-5 definiuje podstawy CNN, jednak w momencie pojawienia się na rynku nie była popularna z powodu braku odpowiedniego sprzętu, a zwłaszcza kart graficznych.

3.2. AlexNet

AlexNet to model zaprezentowany przez Alexa Krizhevskiego i in. w 2012 roku [14]. Architektura ta konkurowała w zawodach ImageNet Large Scale Visual Recognition Challenge (ILSVRC) w tym samym roku, wygrywając je [4]. Współczynnik błędów top-5 modelu wynosił 15,3% i był niższy o 10,8% niż kolejny wynik w konkursie. AlexNet składa się z pięciu warstw splotowych, z czego po niektórych znajdują się warstwy łączące, oraz z trzech w pełni połączonych warstw [14]. Zastosowanie większej liczby warstw było kluczowe dla wydajność, ale potrzebowało wiele zasobów. Było to jednak możliwe dzięki wykorzystaniu kart graficznych w procesie uczenia sieci. Praca naukowa o AlexNet [14] ma duży wpływ na rozwój CNN. Według strony Google Scholar praca była cytowana ponad 90000 razy (styczeń 2022).

3.3. GoogLeNet

GoogLeNet to 22-warstwowa architektura, znana także pod nazwą Inception v1. GoogLeNet zostało zaprezentowana w 2014 roku przez firmę Google [15]. W tym samym roku zwyciężyła konkurs ImageNet [4] z poziomem błędów wynoszącym 6,7%. Największą nowością zaprezentowaną wraz z tą architekturą były moduły incepcji. Moduły te polegają na równoległym działaniu warstw konwolucyjnych, każda z różną dokładnością działania (różne wielkości filtrów). Takie rozłożenie warstw ma pomagać w klasyfikacji obiektów o różnych skalach. Wersja naiwna takiego modułu wykonuje splot na wejściu za pomocą 3 różnych rozmiarów filtrów (1x1, 3x3, 5x5). Dodatkowo wykonywany jest proces maksymalnego łączenia (ang. *max pooling*). Następnie wyjścia są łączone i wysyłane dalej. W celu ograniczenia zapotrzebowania na zasoby twórcy zaproponowali także wersję modułu incepcji z redukcją wymiarowości [15]. Wersja ta polega na ograniczeniu kanałów wejściowych przez dodanie warstw splotowych o rozmiarze 1x1 przed warstwami 3x3 i 5x5. Ar-

chitektura GoogLeNet wykorzystuje dziewięć modułów incepcji z redukcją wymiarowości.

4. Realizacja badań

4.1. Scenariusze Badawcze

Celem badania było porównanie modeli LeNet-5, AlexNet i GoogLeNet pod względem dokładności klasyfikacji odręczenie pisanych cyfr oraz liter alfabetu łacińskiego. Dodatkowo porównane zostały czasy potrzebne na klasyfikacje danych ze zbiorów testowych. W tym celu sformułowano przedstawione poniżej scenariusze badawcze.

Scenariusz 1:

1. Wytrenowanie modeli LeNet-5, AlexNet i GoogLeNet pod kątem klasyfikacji odręcznie pisanych cyfr.
2. Porównanie dokładności (metryka *accuracy*) każdego modelu.
3. Porównanie czasu potrzebnego na klasyfikacje danych ze zbioru testowego dla każdego modelu.

Każdy z modeli przeszedł 20 razy proces uczenia od zera. Do każdego procesu uczenia były wykorzystywane te same parametry dla poszczególnych warstw. Każdy taki proces trwał 20 epok, tak, aby upewnić się, że każdy z modeli zbliżył się do maksimum swojej dokładności. Do badania wykorzystany został zbiór danych MNIST [11], czyli 60000 obrazów uczących i 10000 obrazów testowych.

Scenariusz 2:

1. Wytrenowanie modeli LeNet-5, AlexNet i GoogLeNet pod kątem klasyfikacji odręcznie pisanych liter.
2. Ewaluacja skuteczności przez porównanie dokładności (ang. *accuracy*) każdego modelu.
3. Porównanie czasu potrzebnego na klasyfikacje danych ze zbioru testowego dla każdego modelu.

Badanie zostało przeprowadzone tak jak w scenariuszu pierwszym. Różnicą był zbiór danych. Wykorzystany został zbiór odręczenie pisanych liter ze zbioru EMNIST [12]. Zbiór składa się z 88800 obrazów uczących i 14800 obrazów testowych.

Głównym celem pracy jest sprawdzenie poprawności następujących hipotez:

1. Model oparty o architekturę GoogLeNet wykaże się większą dokładnością w klasyfikacji odręcznie pisanych cyfr.
2. Model oparty o architekturę GoogLeNet wykaże się większą dokładnością w klasyfikacji odręcznie pisanych liter.
3. Model oparty o architekturę LeNet-5 będzie potrzebował mniej czasu na klasyfikację zbioru odręcznie pisanych cyfr.
4. Model oparty o architekturę LeNet-5 będzie potrzebował mniej czasu na klasyfikację odręczenie pisanych liter.

4.2. Implementacja modeli

Modele zostały zaimplementowane w języku Python z użyciem bibliotek Tensorflow i Keras. Implementacje modeli zostały przedstawione kolejno na listingach 1, 2 oraz 3. Dodatkowo dla modelu GoogLeNet zaimplementowana została warstwa incepcji przedstawiona na listingu 4.

Listing 1: Implementacja modelu GoogLeNet

```
input_layer = layers.Input(shape=(32, 32, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224,
    interpolation="bilinear", input_shape=x_train.shape[1:])(input_layer)
temp = layers.Conv2D(64, 7, strides=2, padding='same',
    activation='relu')(input_tensor)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = layers.Conv2D(64, 1, strides=1, padding='same',
    activation='relu')(temp)
temp = layers.Conv2D(192, 3, strides=1, padding='same',
    activation='relu')(temp)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = inception_layer(temp, filters_1x1=64, filters_3x3_reduce=96,
    filters_3x3=128, filters_5x5_reduce=16, filters_5x5=32, filters_pool=32)
temp = inception_layer(temp, filters_1x1=128, filters_3x3_reduce=128,
    filters_3x3=192, filters_5x5_reduce=32, filters_5x5=96, filters_pool=64)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = inception_layer(temp, filters_1x1=192, filters_3x3_reduce=96,
    filters_3x3=208, filters_5x5_reduce=16, filters_5x5=48, filters_pool=64)
auxiliaryOutput1 = layers.AveragePooling2D((5, 5), strides=3)(temp)
auxiliaryOutput1 = layers.Conv2D(128, 1, padding='same',
    activation='relu')(auxiliaryOutput1)
auxiliaryOutput1 = layers.Flatten()(auxiliaryOutput1)
auxiliaryOutput1 = layers.Dense(1024, activation='relu')(auxiliaryOutput1)
auxiliaryOutput1 = layers.Dropout(0.7)(auxiliaryOutput1)
auxiliaryOutput1 = layers.Dense(10, activation='softmax')(auxiliaryOutput1)
temp = inception_layer(temp, filters_1x1=160, filters_3x3_reduce=112,
    filters_3x3=224, filters_5x5_reduce=24, filters_5x5=64, filters_pool=64)
temp = inception_layer(temp, filters_1x1=128, filters_3x3_reduce=128,
    filters_3x3=256, filters_5x5_reduce=24, filters_5x5=64, filters_pool=64)
temp = inception_layer(temp, filters_1x1=112, filters_3x3_reduce=144,
    filters_3x3=288, filters_5x5_reduce=32, filters_5x5=64, filters_pool=64)
auxiliaryOutput2 = layers.AveragePooling2D((5, 5), strides=3)(temp)
auxiliaryOutput2 = layers.Conv2D(128, 1, padding='same',
    activation='relu')(auxiliaryOutput2)
auxiliaryOutput2 = layers.Flatten()(auxiliaryOutput2)
auxiliaryOutput2 = layers.Dense(1024, activation='relu')(auxiliaryOutput2)
auxiliaryOutput2 = layers.Dropout(0.7)(auxiliaryOutput2)
auxiliaryOutput2 = layers.Dense(10, activation='softmax')(auxiliaryOutput2)
temp = inception_layer(temp, filters_1x1=256, filters_3x3_reduce=160,
    filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, filters_pool=128)
temp = layers.MaxPooling2D(3, strides=2)(temp)
temp = inception_layer(temp, filters_1x1=256, filters_3x3_reduce=160,
    filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, filters_pool=128)
temp = inception_layer(temp, filters_1x1=384, filters_3x3_reduce=192,
    filters_3x3=384, filters_5x5_reduce=48, filters_5x5=128, filters_pool=128)
temp = layers.GlobalAveragePooling2D()(temp)
temp = layers.Dropout(0.4)(temp)
output = layers.Dense(10, activation='softmax')(temp)
model = Model(inputs=input_layer, outputs=[output,
    auxiliaryOutput1, auxiliaryOutput2])
```

Listing 2: Implementacja modelu LeNet-5

```
from tensorflow import keras
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(6, kernel_size=5, strides=1,
    activation='tanh', input_shape=train_x[0].shape, padding='same'))
model.add(keras.layers.AveragePooling2D())
model.add(keras.layers.Conv2D(16, kernel_size=5, strides=1,
    activation='tanh', padding='valid'))
model.add(keras.layers.AveragePooling2D())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(120, activation='tanh'))
model.add(keras.layers.Dense(84, activation='tanh'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

Listing 3: Implementacja modelu AlexNet

```
model = keras.models.Sequential()
model.add(keras.layers.experimental.preprocessing.Resizing(224, 224,
    interpolation="bilinear", input_shape=x_train.shape[1:]))
model.add(keras.layers.Conv2D(96, 11, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPooling2D(3, strides=2))
model.add(keras.layers.Conv2D(256, 5, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPooling2D(3, strides=2))
model.add(keras.layers.Conv2D(384, 3, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Conv2D(384, 3, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Conv2D(256, 3, strides=4, padding='same'))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(4096, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(4096, activation='relu'))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(10, activation='softmax'))
```

Listing 4: Implementacja warstwy incepcji z redukcją wymiarowości

```
def inception_layer(temp,
                    filters_1x1,
                    filters_3x3_reduce,
                    filters_3x3,
                    filters_5x5_reduce,
                    filters_5x5,
                    filters_pool):
path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same',
    activation='relu')(temp)
path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same',
    activation='relu')(temp)
path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same',
    activation='relu')(path2)
path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same',
    activation='relu')(temp)
path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same',
    activation='relu')(path3)
path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(temp)
path4 = layers.Conv2D(filters_pool, (1, 1), padding='same',
    activation='relu')(path4)
return tf.concat([path1, path2, path3, path4], axis=3)
```

4.3. Bazy danych

4.3.1. MNIST

Baza obrazów odręcznie pisanych cyfr MNIST [11] zawiera zestaw uczący 60000 obiektów oraz zestaw testowy 10000 obiektów. Baza ta jest często wykorzystywana do testowania stworzonych modeli sztucznych sieci neuronowych. MNIST została zbudowana na podstawie Specjalnej Bazy Danych 3 NIST i Specjalnej Bazy Danych 1, które zawierają binarne obrazy odręcznie pisanych cyfr [2]. Oryginalne czarno-białe (dwupoziomowe) obrazy z NIST zostały znormalizowane pod względem rozmiaru, aby zmieściły się w polu 20x20 pikseli, zachowując ich proporcje. Uzyskane obrazy zawierają poziomy szarości uzyskane w wyniku zastosowania techniki antyaliasingu, używanej przez algorytm normalizacji. Obrazy zostały wyśrodkowane na obrazie 28x28 przez obliczenie środka masy pikseli i przemieszczenie obrazu tak, aby umieścić ten punkt w środku pola 28x28 [11].

4.3.2. EMNIST

EMNIST [12] (Extended MNIST) to zestaw sześciu baz danych mający na celu zapewnienie trudniejszej do klasyfikacji alternatywy dla zestawu danych MNIST [11]. Znaki Specjalnej Bazy Danych 19 NIST [17] zostały przekonwertowane do formatu, który pasuje do zbioru danych MNIST [11], czyniąc go kompatybilnym z każdą siecią zdolną do pracy z oryginalnym zbiorem danych [12]. Do badań wykorzystano bazę danych liter. Baza ta składa się z 88800 obiektów treningowych oraz 14800 obiektów testowych.

5. Wyniki

Wszystkie testy opisanych wcześniej modeli zostały przeprowadzone na komputerze osobistym wyposażonym w procesor Intel Core i7-6700 oraz 16 GB pamięci RAM. Na urządzeniu zainstalowany jest system operacyjny Windows 10 Pro w wersji 21H1. Do implementacji i wykonania badań na modelach użyty został interpreter Python w wersji 3.9 oraz biblioteka Tensorflow w wersji 2.6.0.

Na podstawie wcześniejszych opisów wybranych modeli można łatwo stwierdzić duże różnice pod względem skomplikowania ich budowy oraz zapotrzebowania na zasoby urządzenia, na którym pracują. Przedstawione poniżej wyniki powinny pokazać, czy używanie nowych i bardziej skomplikowanych rozwiązań może być opłacalne w pracy przy prostszych zadaniach takich jak klasyfikacja odręcznego pisma.

5.1. Dokładność klasyfikacji odręczenie pisanych cyfr

Wszystkie otrzymane wyniki klasyfikacji zbioru testowego MNIST [11] znajdują się w tabeli 1. Najlepszym średnim wynikiem dokładności wykazał się GoogLeNet osiągając 99,04%, a najgorszym, czyli 98,63%, LeNet-5. AlexNet uplasował się pomiędzy nimi z wynikiem 98,78%. Jak widać, różnice w otrzymanych wynikach są małe. Pomiędzy modelem, który wypadł najgorzej, a tym z najlepszym wynikiem, różnica ta, to zaledwie 0,39%. Z drugiej jednak strony pole do poprawy było niewielkie, ponieważ już LeNet-5 osiągnął wysoki poziom dokładności. Wykres pudełkowy przedstawiony na rysunku 1 pokazuje, że wraz ze wzrostem średniej dokładności klasyfikacji, poszerzał się też przedział otrzymanych wyników. Najgorszy wynik dla teoretycznie najlepszego modelu, był gorszy niż ten najniższy modelu LeNet-5.

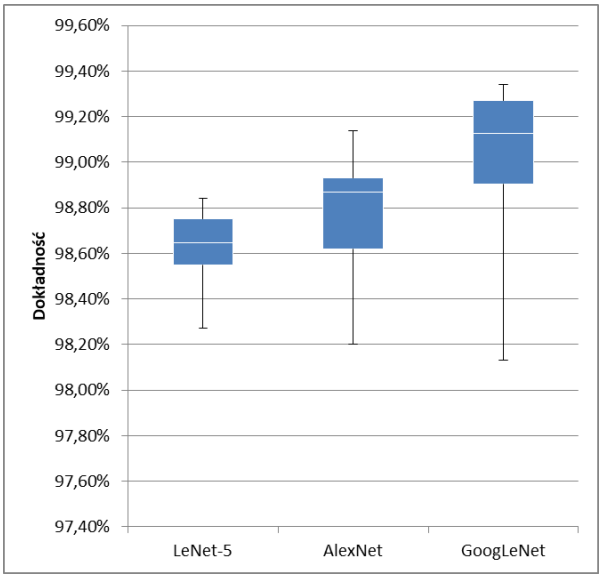
Tabela 1: Wyniki dokładności klasyfikacji odręczenie pisanych cyfr

	Dokładność		
	LeNet-5	AlexNet	GoogLeNet
1	98,84%	98,91%	99,23%
2	98,49%	98,90%	98,73%
3	98,75%	98,70%	98,13%
4	98,60%	99,05%	99,14%
5	98,55%	98,99%	99,14%
6	98,61%	98,86%	99,34%
7	98,34%	98,64%	99,03%
8	98,47%	99,00%	98,93%
9	98,63%	98,56%	98,83%
10	98,73%	98,20%	99,30%
11	98,76%	98,90%	99,27%
12	98,76%	98,56%	99,26%
13	98,75%	98,78%	99,11%
14	98,82%	98,88%	98,83%
15	98,60%	99,02%	98,95%
16	98,77%	99,14%	99,28%
17	98,55%	98,78%	98,57%
18	98,27%	98,56%	99,30%
19	98,66%	98,90%	99,31%
20	98,74%	98,20%	99,11%

5.2. Dokładność klasyfikacji odręczenie pisanych liter

W przypadku klasyfikacji zbioru odręczenie pisanych liter różnice pomiędzy modelami stają się znacznie wyraźniejsze. Średnio GoogLeNet i LeNet-5 osiągnęły kolejno 94,31% i 92,52%, a AlexNet 93,10%. W tym

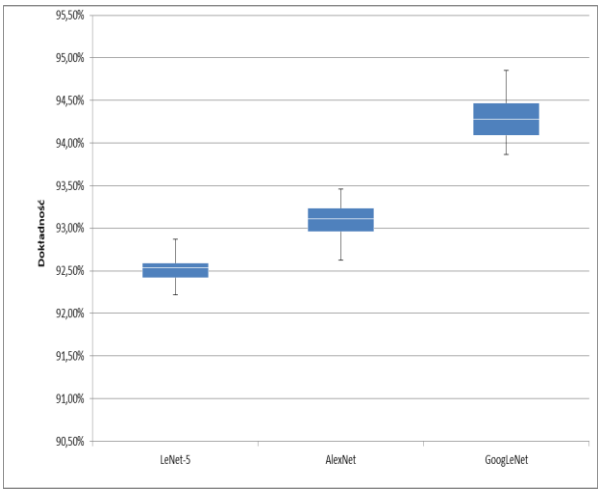
przypadku najlepszy wynik także uzyskał model GoogLeNet, a model LeNet-5 wyprzedził o 1,79%. Przewagę dobrze obrazuje wykres pudełkowy z rysunku 2. Widać na nim, że przedział otrzymanych wyników jest węższy niż w przypadku poprzedniego zbioru danych. Dodatkowo wykres dla modelu GoogLeNet nie pokrywa się z wykresem LeNet-5. Szczegółowe wyniki zostały zamieszone w tabeli 2.



Rysunek 1: Wykres pudełkowy wyników klasyfikacji cyfr.

Tabela 2: Wyniki dokładności klasyfikacji odręczenie pisanych liter

	Dokładność		
	LeNet-5	AlexNet	GoogLeNet
1	92,55%	93,10%	94,85%
2	92,62%	93,12%	94,04%
3	92,58%	93,18%	94,66%
4	92,35%	93,28%	94,04%
5	92,39%	92,98%	94,22%
6	92,52%	92,89%	94,09%
7	92,57%	93,04%	94,13%
8	92,39%	92,88%	94,79%
9	92,60%	92,76%	94,39%
10	92,45%	93,32%	94,57%
11	92,60%	93,24%	94,25%
12	92,43%	93,01%	93,87%
13	92,38%	93,46%	94,31%
14	92,87%	92,62%	94,65%
15	92,57%	93,18%	94,34%
16	92,22%	93,20%	94,43%
17	92,59%	93,43%	94,08%
18	92,68%	93,10%	93,93%
19	92,51%	93,25%	94,17%
20	92,49%	92,85%	94,44%



Rysunek 2: Wykres pudełkowy otrzymanych wyników klasyfikacji liter.

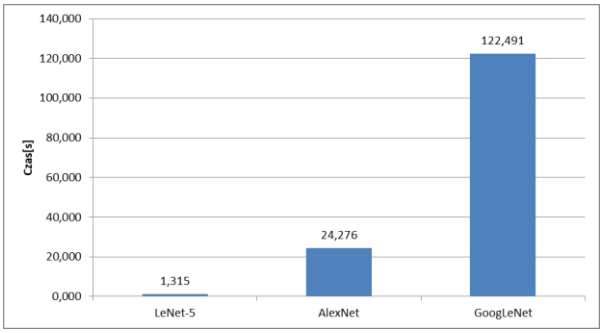
5.3. Czas potrzebny na klasyfikacje zbioru testowego cyfr

Wyniki czasów potrzebnych na klasyfikację zbioru testowego przedstawionych na rysunku 3 pokazują ogromne różnice w wymaganiach, co do urządzenia, na którym pracują poszczególne modele. Model LeNet-5 na klasyfikację zbioru testowego składającego się z 10000 obrazów potrzebował niewiele ponad sekundę, model AlexNet pracował na tym samym zbiorem już około 24 sekund, a GoogLeNet zajęło to ponad dwie minuty. Na podstawie wyników zamieszczonych w tabeli 3 widać, że czas potrzebny dla każdego modelu, w każdej próbie był podobny.

Tabela 3: Wyniki czasu potrzebnego na klasyfikacje zbioru testowego odręczenie pisanych cyfr

	Czas[s]		
	LeNet-5	AlexNet	GoogLeNet
1	1,277	24,932	122,920
2	1,321	24,268	122,400
3	1,347	24,117	122,929
4	1,301	24,528	122,834
5	1,326	24,188	122,795
6	1,300	24,072	122,161
7	1,299	24,231	122,089
8	1,319	24,145	122,082
9	1,328	24,177	122,830
10	1,304	24,077	122,240
11	1,385	24,135	122,586
12	1,309	24,244	122,257
13	1,305	24,180	123,006
14	1,319	24,900	122,167
15	1,315	24,447	122,272
16	1,318	24,119	122,612
17	1,275	24,126	122,410

18	1,348	24,232	122,332
19	1,305	24,200	122,331
20	1,296	24,209	122,559



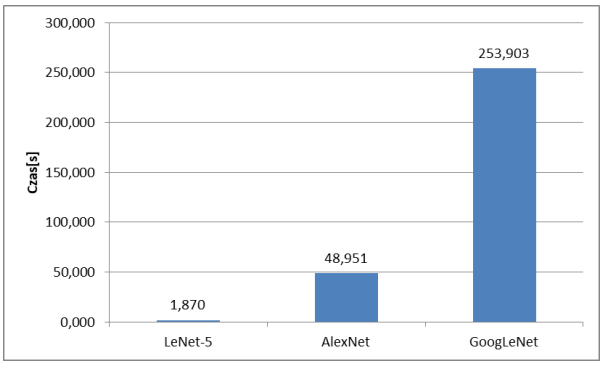
Rysunek 3: Porównanie średnich wartości czasu potrzebnego do klasyfikacji zbioru testowego cyfr.

5.4. Czas potrzebny na klasyfikacje zbioru testowego liter

Zwiększenie możliwych klas do sklasyfikowania z 10 do 26 oraz zbioru testowego z 10000 do 14800 spowodowała podwojenie czasu pracy modeli AlexNet i GoogLeNet, natomiast dla modelu LeNet-5 oznaczało to tylko około pół sekundy dłużej. Na rysunku 4 zamieszczone zostały średnie wyniki czasów potrzebnych na klasyfikację zbioru testowego, a w tabeli 4 zebrane są wszystkie otrzymane czasy.

Tabela 4: Wyniki czasu potrzebnego na klasyfikacje zbioru testowego odręczenie pisanych liter

	Czas[s]		
	LeNet-5	AlexNet	GoogLeNet
1	1,889	49,420	254,971
2	1,843	48,841	253,889
3	1,902	48,851	253,868
4	1,849	49,174	253,994
5	1,874	48,797	254,179
6	1,810	48,719	254,173
7	1,828	48,853	254,017
8	1,872	48,789	253,249
9	1,789	49,536	253,561
10	1,875	48,845	253,135
11	1,889	49,089	254,298
12	1,790	48,888	254,829
13	1,901	48,751	253,322
14	1,916	49,050	254,086
15	1,868	48,861	253,221
16	1,862	48,668	253,878
17	1,856	48,944	253,597
18	1,828	49,173	254,100
19	1,910	48,761	253,839
20	2,050	49,017	253,849



Rysunek 4: Porównanie średnich wartości czasu potrzebnego do klasyfikacji zbioru testowego liter.

6. Wnioski

Przed rozpoczęciem badań postawione zostały tezy mówiące o tym, że model GoogLeNet będzie dokładniejszy w klasyfikacji odręczenie pisanych liter jak i cyfr od modeli AlexNet i LeNet-5. Dodatkowo sformułowano tezy o tym, że LeNet-5 będzie szybszy od pozostałych dwóch modeli podczas klasyfikacji zarówno zbioru odręczeni pisanych cyfr jak i zbioru odręczenie pisanych liter. Na podstawie uzyskanych wyników można stwierdzić, że wszystkie postawione tezy są prawdziwe, ponieważ model GoogLeNet w obu przypadkach wykazał się większą dokładnością od pozostałych modeli, a LeNet-5 był ze wszystkich zdecydowanie najszybszy. Można też stwierdzić stosunkowo małe korzyści pod względem dokładności klasyfikacji wybranych zbiorów. W najlepszym przypadku uzyskano około 1,8% dla zbioru liter. Przychodzi to jednak wielkim kosztem użycia zasobów komputera. Na klasyfikacje zbioru testowego cyfr, model GoogLeNet potrzebował sto razy więcej czasu niż dużo prostszy model LeNet-5, a w przypadku zbioru liter jest nawet gorzej. Trzeba jednak wziąć pod uwagę fakt, że dwa nowsze z tego zestawienia modele (AlexNet i GoogLeNet), były projektowane z myślą o klasyfikacji dużo bardziej skomplikowanych obiektów znajdujących się na zdjęciach, z którymi model LeNet-5 mógłby mieć duży problem. Można bezpiecznie powiedzieć, że w przypadku klasyfikacji odręcznego pisma, korzyści uzyskane z użycia bardziej skomplikowanych rozwiązań są za małe względem kosztów jakie trzeba ponieść, aby ich użycie było opłacalne.

Literatura

[1] D. O. Hebb, The organisation of behaviour: a neuropsychological theory. New York: Science Editions (*1949*).

[2] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65(6) (1958) 386.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard., W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition. Neural computation 1(4) (1989) 541-551.

[4] O. Russakovsky, J. Deng, H. Su, et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput

Vis 115 (2015) 211–252. https://doi.org/10.1007/s11263-015-0816-y

[5] Ü. Budak, A. Şengür, U. Halici, Deep convolutional neural networks for airport detection in remote sensing images. 26th Signal Processing and Communications Applications Conference (SIU) (2018) 1-4, doi: 10.1109/SIU.2018.8404195.

[6] M. J. Aitkenhead, A. J. S. McDonald. A neural network face recognition system. Engineering Applications of Artificial Intelligence 16(3) (2003) 167-176.

[7] D. S. Maitra, U. Bhattacharya, S. K. Parui, CNN based common approach to handwritten character recognition of multiple scripts. 13th International Conference on Document Analysis and Recognition (ICDAR) (2015) 1021-1025, doi: 10.1109/ICDAR.2015.7333916.

[8] K. Nygren, Stock prediction–a neural network approach. Royal Instiute of Technology (2004) 1-34.

[9] S. S. Baboo, I. K. Shereef, An efficient weather forecasting system using artificial neural network. International journal of environmental science and development 1(4) (2010) 321.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (2009) 248–255.

[11] Y. LeCun, C. Cortes, The MNIST database of handwritten digits (2005).

[12] G. Cohen, S. Afshar, J. Tapson, A. van Schaik, EMNIST: an extension of MNIST to handwritten letters (2017) arXiv:1702.05373.

[13] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition in Proceedings of the IEEE 86(11) (1998) 2278-2324, doi: 10.1109/5.726791.

[14] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, K. Weinberger, eds., Advances in Neural Information Processing Systems 25. Curran Associates (2012) 1097–1105. arXiv:1803.01164

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going Deeper with Convolutions (2014) arXiv:1409.4842.

[16] K. O'Shea, R. Nash, An introduction to convolutional neural networks (2015) arXiv preprint arXiv:1511.08458.

[17] Grother, P. J, NIST special database 19. Handprinted forms and characters database, National Institute of Standards and Technology (1995).

[18] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics *5*(4) (1943) 115-133.

[19] E. Lukasik, M. Charytanowicz, M. Milosz, M. Tokovarov, M. Kaczorowska, D. Czerwinski, T. Zientarski, Recognition of handwritten Latin characters with diacritics using CNN. Bulletin of the Polish Academy of Sciences. Technical Sciences *69*(1) (2021).

JOURNAL COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 152–164
Received: 13 April 2022
Accepted: 10 May 2022

Comparative study of scaling parameters and research output of selected highly- and moderately-cited individual authors

Badanie porównawczych parametrów skalowania i dorobku badawczego wybranych wysoko i umiarkowanie cytowanych autorów

Keshra Sangwal*
Department of Applied Physics, Lublin University of Technology, ul. Nadbystrzycka 38, 20-618 Lublin, Poland

Abstract
The real data of cumulative citations l_n of selected nth paper of individual N papers published by some highly- and moderately-cited individual authors are analyzed to compare Hirsch and Hirsch-type indices h, h_1, h_f and h_m, and citation radii R and R_f from consideration of: (1) the number A_n of coauthors of the paper, (2) the normalization of citations l_n and cumulative fraction l_{nf} of citation of the nth paper by mean and median citations of the citations l_n of all N_c cited papers, and (3) the determination of effective rank n_{eff} of the l_{nf} citations. Analysis of the $l_n(n)$, $l_{nf}(n)$ and $l_{nf}(n_{eff})$ data was also carried out by using a Langmuir-type function $l = l_0[1-\alpha Kn/(1+Kn)]$, where l denotes the citations l_n and l_{nf} of all cited N_c papers arranged in the decreasing order, α is an effectiveness parameter, K is the so-called Langmuir constant, n denotes the rank n or n_{eff} of citations and l_0 is the value of l when n or n_{eff} approaches zero. For a comparison of the publication output of different authors it was found that the h_m index is more consistent than other indices, and it can be normalized to account for the publication career of different authors. However, Langmuir-type function is not adequate for comparison of the publication output of different authors because it describes the rank-order distribution patterns satisfactorily in terms of two parameters. To compare the publication output of different authors independent of their career length t, it is suggested to use scaling parameters h/t, h_f/t and h_m/t.

Keywords: Citation analysis; Citation distribution; Coauthorship; h-type indices; Langmuir-type function

Streszczenie
Przeanalizowano dane liczby cytowań kumulacyjnych l_n wybranego n-tego artykułu spośród N indywidualnych artykułów opublikowanych przez niektórych wysoko i umiarkowanie cytowanych pojedynczych autorów. Do porównania użyto wskaźników Hirscha h i Hirscha-podobnych h_1, h_f i h_m, oraz promienia cytowań R i R_f pod względem: 1) liczby A_n współautorów artykułów, 2) normalizacji liczby cytowań l_n i ułamka kumulacyjnego l_{nf} cytowania n-tego artykułu średnimi i środkowymi cytowaniami l_n wszystkich cytowań N_c artykułów, 3) określenia efektywnego rzędu n_{eff} cytowań l_{nf}. Analiza danych $l_n(n)$, $l_{nf}(n)$ i $l_{nf}(n_{eff})$ była prowadzona również wedle funkcji typu Langmuira $l = l_0[1-\alpha Kn/(1+Kn)]$, gdzie: l oznacza l_n i l_{nf} cytowania wszystkich cytowanych N_c artykułów umieszczonych w malejącym porządku, α parametr efektywności, K to tak zwana stała Langmuira, n oznacza rząd n lub n_{eff} cytowań oraz l_0 to wartość l, gdy n lub n_{eff} dąży do zera. Do porównania dorobku publikacyjnego różnych autorów stwierdzono, że wskaźnik h_m jest bardziej miarodajny od pozostałych oraz, że można je znormalizować w celu uwzględnienia kariery publikacyjnej różnych autorów. Funkcja typu Langmuira nie jest właściwa jednak do porównania dorobku publikacyjnego różnych autorów ponieważ opisuje ona w sposób zadowalający rozkład kolejności rzędu ich artykułów przy pomocy dwóch parametrów. Do porównania dorobku publikacyjnego różnych autorów, niezależnego od długości ich kariery t, zaproponowano stosowanie parametrów skalowania h/t, h_f/t and h_m/t.

Słowa kluczowe: Analiza cytowań; Rozkład cytowań; Współautorstwo; Współczynniki typu h; Funkcja typu Langmuira

*Corresponding author
Email address: k.sangwal@pollub.pl (K. Sangwal)

1. Introduction

For promotion/recruitment of faculty/research positions and award of research grants it is desired to compare the research output of candidates working in a scientific field. A commonly used measure for this purpose is the h index, proposed by Hirsch [1], which is defined as the number of papers of nth rank with citations $l_n \geq h$. A convenient way to determine the h index of an author is to look for the value of the rank n of the paper when $n \leq l_n$ from the plot of the decreasing number l_n of citations received by the nth paper against all of his/her N_c cited papers (rank–size distribution plots). The main advantages of the h index are that it is a single number characterizing publications and citations of an author and it is insensitive to uncited or relatively poorly cited papers. However, the citation distribution of an author is frequently skewed either with a few highly cited papers or a large number of papers with few citations. Consequently, since an n-ranked paper enters in the counting of the h index of an author, citations outside the h-core (i.e. citations $l_n > h$ received by papers $n < h$ and $l_n < h$ received by papers $n > h \geq N_c$) are not used anywhere.

In order to overcome the above disadvantage of the h index its several variants, simultaneously retaining its advantages, have been widely proposed and discussed (for example, see: refs. [2-13]).

It is well known that the Hirsch index h of an author increases with his/her publication duration t [1,5,14-17], typical h indices of researchers publishing in different research fields are different [1], and researchers publishing in large collaborations usually have high values of their h index [1,18,19]. In fact, these factors make the comparison of scientific research output of authors of different academic career length working across various research disciplines difficult. In order to compare the research output of researchers publishing a constant number of papers per year of similar quality during their publication duration t, Hirsch [1] proposed the parameter $m = h/t$ as a useful measure. From an analysis of the behavior of h index as a function of academic age t of about 1400 Italian physicists, Mannela and Rossi [16] found that a time-scaled index $h/t^{1/2}$ is related to the h index.

All citation-related measures, including the h index and its different variants, for the publication output of an author assume that all papers receiving citations are written by him/her alone. However, in the case of multiple-authored papers, it is unfair to award full credit to each author because this method in reality penalizes authors who publish alone. Therefore, devising of a fair method of counting of contributions of individual authors in multi-authored papers has drawn considerable attention for over four decades (for example, see: refs. [18,20-26]). The problem is also complicated because single-authored papers published in different journals with high impact factors usually earn the lowest number of citations and the number L of cumulative citations increases with increasing number A of coauthors, following approximately the relation: $L = L_0(0.2A)^{1/3}$ [27]. In this relation L_0 is a normalizing factor related to the journal.

The culture of authorship of papers in different disciplines is not the same. The following authorship patterns are usually observed [24,28]: (1) junior researchers are the first authors and group leaders are the last authors, (2) senior researchers are the first authors followed by junior researchers, (3) authors in multi-authored papers are listed in the order of decreasing contribution of coauthors involved their publication, and (4) all authors are arranged alphabetically, especially in large collaborations as in nuclear physics experiments. In case (2) it is not always possible to establish the contributions of the first, senior, corresponding authors. However, in general, contributions of different authors in multi-authored papers frequently remain unknown and the information that one usually has to determine the credit of authors in different multi-authored papers is the authors' list in the papers. Therefore, assigning due credit to the coauthors of papers in different disciplines is a problem in citation analysis.

For publications involving a large number of coauthors, Hirsch [1] suggested to normalize individual authors to normalize their h indices by a factor reflecting the average number of authors. Counting of contributions according to the order of the authors has also been proposed [21,27,29-33]. In this approach the contribution of the first author is always dominant in a multi-authored paper but equal contributions of the first and last (corresponding) author can also be considered [31].

It is well known that the average number of citations per paper differs among various scientific disciplines due to their citation behavior (for example, see: refs. [1,4,34-38]). Using the total number of citations [35,36] or distributions of citations to the papers published in different fields [4,34,37-39] different scaling parameters have been proposed to compare the research output of researchers in various fields. The average number of citations in some studies [1,4,34,35,37,38] whereas median or geometric mean of citations in others [36] has also been proposed as an effective scaling parameter in different fields.

To account for the effect of multiple coauthorship through the h index, Hirsch [40] proposed $\hbar$ (hbar) index as the number of an individual author's papers that have citations greater than or equal to the $\hbar$ index of all coauthors of each paper. However, this approach considers papers instead of authors. Following the idea advanced by Hirsch [1] of normalizing individual authors to normalize their h indices by a factor reflecting the average number of authors, Batista et al. [18] proposed to divide the h index of an individual author by the number of authors of the paper in the h-core. The resulting so-called h_1 index is defined as: $h_1 = h^2/A_h$, where A_h is the total number of coauthors of a particular author in the h-core. The main problem with this h_1 index is that its value is enormously changed for authors with some papers with a large number of coauthors. Egghe [41] proposed to count the contribution of multiple authors considering either fractions l_{nf} of citations (i.e. $l_{nf} = l_n/A_n$, where A_n is the number of authors of a paper) or fractions n_{eff} of rankings of all n-ranked papers (i.e. $n_{eff} = n/A_n$). Arrangement of the fractions l_{nf} of citations to all n-ranked papers of an author in decreasing order gives his/her new Hirsch-type index, namely the h_f index, which is again an integer as the h index. Similarly, counting of the rank n of each paper fractionally as inverse of the number A_n of its authors results in the so-called h_m index, which is a noninteger number [42,43].

Performance of multi-authorship indices for the ranking of authors has been discussed in the literature [26,44-46]. Abramo et al. [44] recommended to use indicators or evaluation methods that take into consideration authors' contributions. Sahoo [26] proposed I-index from consideration of an author's percentage share in the total citations received by his/her papers. He observed that equi-distribution of credit among the coauthors of a paper would give the most probable value of his I-index (with an associated small standard deviation which decreases with increasing h-index). Dunaiski et al. [45] reported best results with equal contribution of co-authors to a paper's score, independent of impact indicator used to compute paper scores.

The h index of authors is an empirical measure of their publication output. Since the h index of an author is defined in terms of the highly cited papers, its prediction is essentially associated with the description of the rank-order distribution of citations of his/her papers. This approach has been used by Egghe [47], and Egghe and Rousseau [48,49], who used Lotka's law for the rank-order distribution of citations. However, Sangwal [50] observed that Lotka-type function poorly describes real rank-order distributions of citations of different authors. From an examination of real citation distributions of authors Burrell [51] also arrived at a similar conclusion.

Following the concepts of processes of adsorption of additives during crystal growth, Sangwal [50] proposed four novel functions to describe rank-order distributions of citations of different authors. He observed that, among these functions, a Langmuir-type function satisfactorily describes the $l_n(n)$ data for different authors and that the Langmuir constant K of this function characterizes the citation behavior of an author. This function was later employed to explain rank-order distributions of journals published in different countries [52] and rank-order distributions of cumulative citations l_n and fractions l_{nf} of citations of papers recived by multi-authored papers published by moderately-cited authors working in different scientific fields [53]. Since the h, h_f and h_m indices of an author are related to the three distributions $l_n(n)$, $l_{nf}(n)$ and $l_{nf}(n_{eff})$, respectively, constructed from the bibliometric data of his/her publication output, it is interesting to analyze the research output of some highly- and moderately-cited authors using the proposed Langmuir-type function.

The aim of the present paper is fourfold: (1) to verify whether normalization of distribution of cumulative citations l of individual papers with total citations L and the cumulative fractions L_f of citations of the papers of an author, based on equal contribution of different authors by their mean and median citations, yields a universal citation distribution, (2) to analyze the distribution of normalized citations l and l_f using Langmuir-type function, (3) to compare the research output of selected authors using different citation indices proposed in the literature, and (4) to propose an objective measure for characterization (and comparison) of the publication output of different authors from distribution of cumulative citations l_n of his/her N_c cited papers published during their entire publication career.

2. The Langmuir-type function

Langmuir-type function of rank-order distribution of items is based on the concepts of adsorption processes involved during crystal growth. The basic concepts used in the derivation of this function have been described earlier [50,52,53]. In the context of citation distribution the Langmuir function is based on the following postulates:

(1) The N papers of an author have the same number s_{max} of possible active citation sites, and these possible sites for the papers can be represented in the form of a two-dimensional set of successive papers arranged at equal distance along the x axis such that s_{max} equi-spaced points, denoted as sites l_n, stacked along the y axis represent citations received (filled circles) and unreceived (open circles) by each of the N papers. Figure 1 schematically illustrates this arrangement of N papers in the form of decreasing citations

(2) The maximum number of cumulative l_0 citations belongs to the paper of rank $n = 0$ represented by s_{max} possible active sites, no citations are received by the papers of rank $n > N_c$ represented by s_{ad} sites, whereas $0 < l_n < l_0$ citations are received by the papers $0 < n < N_c$ represented by $(s_{max}–s_{ad})$ sites. If we define the ratio of the number of s_{ad} (covered) sites to the total number s_{max} of possible active sites as the coverage θ of active sites (i.e. $\theta = s_{ad}/s_{max}$), the coverage θ may be considered as a measure of unreceived citations by the nth paper.

(3) The rank n of a paper is a measure of probability of receiving citations by the paper. This assumption means that the number of l_n citations received by the nth rank paper decreases with increasing value of n whereas l_0 citations are produced by the paper of rank $n = 0$.

(4) The process of unreceiving (inhibition or blocking) of citations may be described by the usual isotherms applied in adsorption of adsorbant atoms, ions or molecules on active adsorption sites on the surface of a solid.

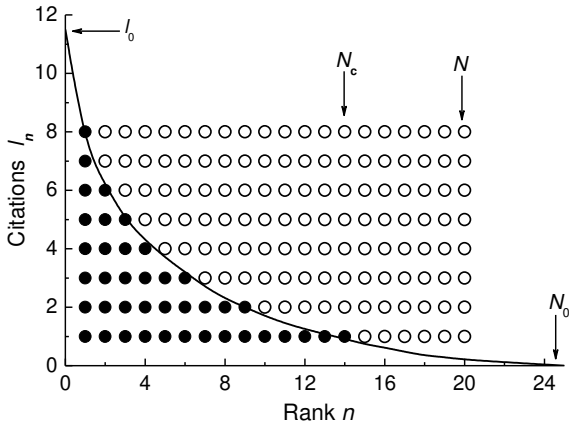


Figure 1: Graphical illustration of the number l_n of citations received by a set of n papers. Filled and open circles denote received (blocked) and unreceived (uncovered) citations, respectively. Dependence of l_n citations on rank n of papers is shown by the curve starting from citations l_0 at rank $n = 0$ and terminating at rank $n = N_c$. Note that N_c is the maximum number of papers that received citations ($N_c = 14$ here).

With the above assumptions the relationship between the number l of citations received by a paper and the coverage θ of blocked citation sites may be given by

$$l = l_0(1 - \alpha\theta) , \quad (1)$$

where l_0 is the maximum number of citations received when $\theta = 0$ and α is an effectiveness parameter for unblocking of citation sites (see below). When the cover-

age θ of citation sites follows Langmuir adsorption isotherm

$$\theta = \frac{Kn}{1+Kn}, \quad (2)$$

one obtains from Eq. (1) the usual Langmuir-type (LT) function relating the number l of citations with the rank n of the paper in the form

$$l = l_0\left[1-\alpha\left(\frac{Kn}{1+Kn}\right)\right]. \quad (3)$$

In Eqs. (2) and (3) the parameter K, usually known as the Langmuir constant, characterizes the behavior of citations generated by the papers whereas n is the rank of the paper. The units of K are inverse of paper rank (i.e. paper-rank^{-1}) and are determined by the way the paper rank n is expressed. If one defines a dimensionless paper rank $x = n/N_c$, Eq. (3) may be expressed as

$$l = l_0\left[1-\alpha\left(\frac{K'x}{1+K'x}\right)\right], \quad (4)$$

where the new dimensionless Langmuir constant $K' = KN_c$, with N_c as the number of papers which receive citations ($N_c < N$). The dimensionless Langmuir constant K' for citations of different paper–citation systems is related to their corresponding dimensionless differential energy Q by

$$Q = \ln K' = \ln(KN_c). \quad (5)$$

The effectiveness parameter α may be given by (cf. Eq. (1))

$$\alpha = \frac{1}{\theta}\left(1-\frac{l_i}{l_0}\right), \quad (6)$$

where l_i is the number of unreceived citations by a paper. In Eq. (6) when the active citation sites are blocked completely, i.e. $l_i = 0$, depending on the value of coverage θ the following situations are possible regarding the values of α: (1) when the coverge $0 \le \theta \le 1$, $\infty \ge \alpha \ge 1$, and (2) when $\theta \ge 1$, $0 \le \alpha \le 1$. In the latter case, the highest value of unity for α is achieved when $\theta = 1$ whereas $\alpha \to 0$ when $\theta \to \infty$.

When $l/l_0 = 0$, Eq. (4) gives

$$K' = KN_c = (\alpha-1)^{-1}. \quad (7)$$

According to relation (7) the dimensionless Langmuir constant $K' = KN_c$ is inversely proportional to $(\alpha-1)$ with the proportionality constant $(KN_c)_0 = 1$ such that $\ln(KN_c)_0 = 0$. Since K and K' are always positive, Eq. (7) implies that $\alpha \ge 1$. This inference is consistent with our earlier findings [50,52,53].

3. Bibliometric data and their analysis

For the analysis we used examples of bibliometric data on the cumulative citations received by papers published by 11 selected authors. Of these, six authors are from those who published first single-authored original papers which received more than 20000 citations each up to 2013. These authors were selected from the authors of first single-authors of top-cited papers, which showed mainly an initial increase and then, after going through a maximum value, a slow decrease in the number of their yearly citations in successive years [54]. The other five authors are from those who were nominated professors in chemistry by the President of Poland in 2013 and received citations lower than 2000 each up to 2012 [53]. Hereafter these two groups of authors are referred to as highly- and moderately-cited authors, respectively.

The Thomson Reuters' Web of Knowledge database was employed to collect the basic bibliometric data. The data represent papers published till 2013 and 2012 by the highly- and moderately-cited authors, respectively. The data were collected during 21-24 October 2014 and 1-15 April 2014, respectively. An author's authorship of each paper was identified unambiguously following his/her affiliation, research area and names of frequent coauthors. Despite the fact that these bibliometric data were collected some years ago, more recent data are not included in the analysis in view of long publication careers, lying between 32 and 68 years, of most of the authors (see Table 1).

The basic bibliometric data for an author comprised the number N of cumulative papers, the number L of cumulative citations received by N_c papers (such that $N_c < N$) and the publication duration t since the publication of the first paper. In addition to the above data, the citations l_n received by individual nth paper published by an author and the number A_n of its coauthors were collected from the Web of Knowledge database. From these lists of the values of the number l_n of citations received by each of his/her nth paper and the number A_n of its coauthors, the author's cumulative citations $L = \sum l_n$, his/her cumulative fractions L_f of citations (i.e. $L_f = \Sigma l_{nf}$, where the fraction l_{nf} of l_n citations of a paper is: $l_{nf} = l_n/A_n$), and the effective rank n_{eff} of the papers (i.e. $n_{eff}(n) = \Sigma 1/A_n$, where A_n is the number of authors of the nth paper), were calculated. In view of different authorship patterns used in the lists of authors of the papers considered in this study and enormously high number of coauthors in many cases, l_{nf} and n_{eff} were calculated on the assumption of equal contributions of A_n coauthors of the nth-ranked paper fetching l_n citations. The values of the cumulative papers N published by different authors, the cumulative number N_c of their papers receiving cumulative citations L, cumulative fraction L_f of citations, and their publication duration t are given in Table 1. In columns 2 and 4 of this table N_c^a denotes the cumulative number of papers without the so-called outliers fetching exceptionally high citations whereas L^a is the cumulative number of citations received by N_c^a papers.

The real distribution of $l_n(n)$, $l_{nf}(n)$ and $l_{nf}(n_{eff})$ data of different papers published by the selected authors were analyzed using Eq. (3). Nonlinear least-squares fitting, involving chi-square residual, of the citation data

Table 1: Bibliometric data and various calculated indices for different authors

Author	N (N_c/N_c[a])	t	L (L[a])	L_f	h (A_h)	h_1	h_f	h_m (N_m)	R[b]	R_f
Laemmli UK	93 (84/81)	45	236007 (13478)	5345.3	58 (172)	19.56	40	21.43 (55)	(65.50)	41.24
Becke AD	84 (70/67)	36	90924 (15603)	11953.3	48 (86)	26.79	40	31.42 (44)	(70.47)	61.68
Southern EM	127 (110/109)	56	38598 (6612)	3460.4	41 (120)	14.01	28	19.11 (36)	(45.88)	33.19
Felsenstein J	113 (99/94)	48	46478 (12312)	9513.5	51 (74)	35.15	30	41.92 (51)	(62.60)	55.03
Shannon RD	171 (152/150)	45	43179 (9080)	4240.0	45 (141)	14.36	28	18.30 (39)	(53.76)	36.74
Scatchard G	84 (73/72)	68	29077 (5655)	2800.6	33 (87)	12.52	25	16.03 (32)	(42.43)	29.86
MBa	103 (70/69)	15	1331 (582)	237.01	17 (572)	0.51	7	5.24 (21)	20.58 (13.61)	8.69
MCy	80 (68)	35	1117	310.06	18 (85)	3.81	9	8.06 (27)	18.85	9.93
TJe	164 (92)	15	852	390.01	18 (46)	7.04	10	8.06 (15)	16.56	11.14
BBo	80 (74)	41	963	393.35	17 (48)	6.02	10	8.34 (19)	17.51	11.19
BPa	56 (44/43)	32	688 (593)	426.33	16 (32)	8.0	11	10.32 (15)	14.80 (13.74)	11.64

[a] Values of N_c for cumulative citations L[a]. [b] Values of R calculated from citations, excluding outliers as mentioned in Table 2, are given in the parentheses.

Table 2: Mean and median values of citations of different authors

Author	Citations L (with coauthors)			Citations L_f (without coauthors)	
	N range	Mean	Median	Mean	Median
Laemmli UK	≥4	166.40	87.0	65.99	9.5
Becke D	≥4	232.88	91.0	178.41	46.0
Southern EM	≥2	60.66	24.0	31.75	9.75
Felsenstein J	≥6	130.98	49.0	101.21	39.0
Shannon RD	≥6	41.66	23.0	15.98	6.0
Scatchard G	≥2	78.54	26.5	38.90	11.75
MBa	All (≥2)	22.71 (12.19)	6.0	3.39	1.5
MCy	All	16.43	6.5	4.56	2.83
TJe	All	9.17	5.0	4.24	1.75
BBo	All	13.01	7.0	5.32	3.0
BPa	All (≥2)	15.64 (13.79)	6.5	9.69	4.0

was carried out with commercially available "Origin 9.1" package. This package yields values of the fitting parameters of an equation, their standard deviations and the corresponding goodness-of-the-fit parameter R^2. In the case of Eq. (3) the best-fit parametrs, among others, are effectiveness parameters α, α_f and α_f^{ef}, and Langmuir constants K, K_f and K_f^{ef} for the $l^*(n)$, $l_f^*(n)$ and $l_f^*(n_{eff})$ data. Here l^* and l_f^* are cumulative relative citations normalized by dividing l_n and l_{nf} citations by the average (mean) or median number of citations $<L>$ and $<L_f>$, respectively, obtained from the values of l_n and l_{nf} citations of N_c cited papers of an author (see Table 2). As seen from Table 2, in the case of some of the authors, especially the highly cited authors, their highly cited papers (i.e. outliers) were excluded while calculating their mean and median citations for the papers. The above normalization procedure was used to verify whether distributions of total citations l_n of nth papers of different authors converge to a single curve.

It should be mentioned that the fractions l_{nf} of citations of individual nth paper for the various authors are distinguished from those for the cumulative citations l_n of multi-authored papers by the subscript "f". Similarly, the average value $<L_f>$ of his/her contributed citations L_f is distinguished from the average value $<L>$ of cumulative citations L of an author by the subscript "f". However, while denoting relative citations l_n^* and l_{nf}^* the subscript n for the rank n has been omitted in different figures and tables, and the best-fit parameters of Eq. (3) for the contributed citations l_f^* of individual papers for the various professors are distinguished from those for the cumulative citations l^* of multi-authored papers by the subscript "f".

4. Results and discussion

4.1. Normalized citations of individual papers of authors and their dimensionless rank

Figures 2 and 3 show the real data of relative citations l^* and l_f^* of papers published by different highly- and moderately-cited authors, respectively, as a function of dimensionless rank n/N_c of their papers. In these figures the relative citations l^* and l_f^* of each paper represent cumulative citations l_n of each nth paper and their cumulative fractional citations l_{nf} normalized by dividing them by their mean $<L>$ and $<L_f>$ citations, respectively. Figure 4 shows, as an example, relative citations l^* and l_f^* of papers published by moderately-cited authors as a function of dimensionless rank n/N_c of their papers. Here the l_n and l_{nf} citations are normalized by their median $<L>$ and $<L_f>$ citations, respectively.

It may be seen that the $l^*(n/N_c)$ and $l_f^*(n/N_c)$ data, normalized by both mean as well as median $<L>$ and $<L_f>$ citations, for different authors result in broad single curves and that the spead or dispersion of the data in the y direction is relatively large in the entire n/N_c range. As indicated by the spread of the data, normalization of citations of individual papers by mean values $<L>$ and $<L_f>$ of citations leads to a better convergence in the entire n/N_c range for all authors (see Figures 2 and 3), but no universality of citation distributions for the authors, similar to that reported by Radicchi et al. [37] for

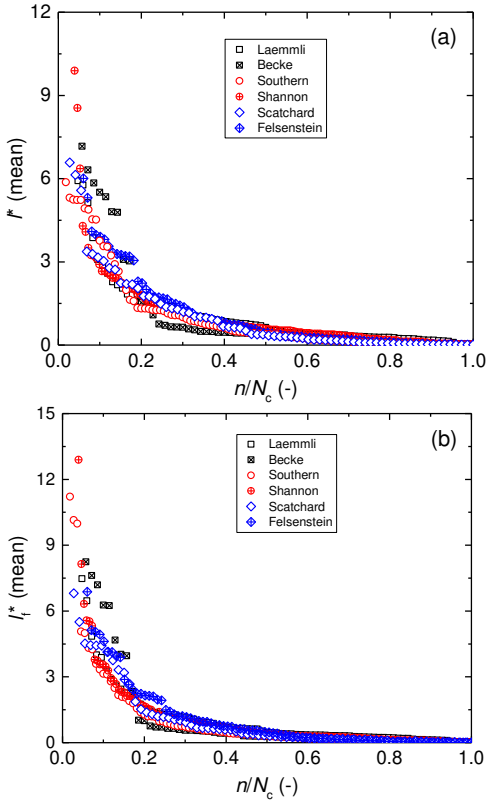


Figure 2: Normalized citations (a) l^* and (b) l_f^* of papers published by highly-cited authors as a function of dimensionless rank n/N_c of their papers. Normalization of citations was carried by the mean $\langle L_n \rangle$ and $\langle L_{fn} \rangle$ citations. See text for details.

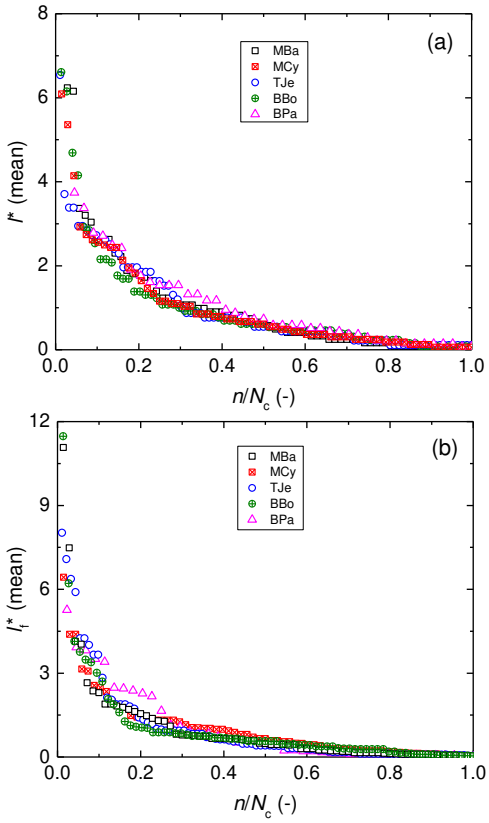


Figure 3: Normalized citations (a) l^* and (b) l_f^* of papers published by moderately-cited authors as a function of dimensionless rank n/N_c of their papers. Normalization of citations was carried by the mean $\langle L_n \rangle$ and $\langle L_{fn} \rangle$ citations. See text for details.

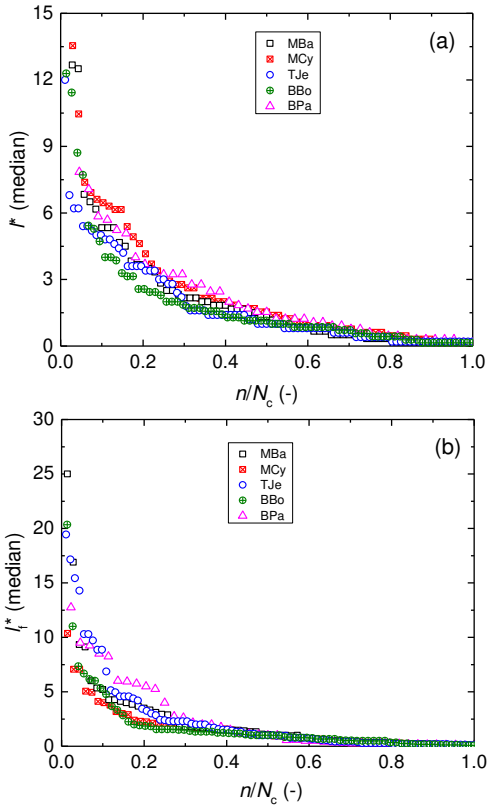


Figure 4: Normalized citations (a) l^* and (b) l_f^* of papers published by moderately-cited authors as a function of dimensionless rank n/N_c of their papers. Citations L_n and L_{fn} are normalized by their median $\langle L_n \rangle$ and $\langle L_{fn} \rangle$ citations.

the distribution of citations within different disciplines, is observed. However, large spread in the relative citations of different papers at low n/N_c is a general feature of these citation patterns. These dispersions are associated with different dependence of l^* and l_f^* on n/N_c for each author. This means that the distribution patterns of l^* and l_f^* are characteristic for each author, and the values of the parameters of any function such as Langmuir-type function (3) capable of describing them are representative of the author.

4.2. Rank-order distribution of normalized citations of different authors

Figures 5 and 6 show the data of relative citations l^* and l_f^* of papers, normalized by the mean values of $\langle L \rangle$ and $\langle L_f \rangle$ citations, of different highly- and moderately-cited authors, respectively, as a function of the rank n of their papers. Solid and dashed curves in the figures present best-fit plots for l^* and l_f^* with the values of the parameters of Eq. (3) listed in Table 3.

The distribution patterns of relative citations l^* and l_f^* normalized by the mean citations of the papers for different moderately-cited authors in Figure 6a,b are somewhat similar and frequently overlap each other. This feature may be noted from Figure 6a. However, when all papers of an author are included in the normalization of citations, the overlapping of the two citation distribution appears to improve when the first highly-cited papers are excluded from the normalization

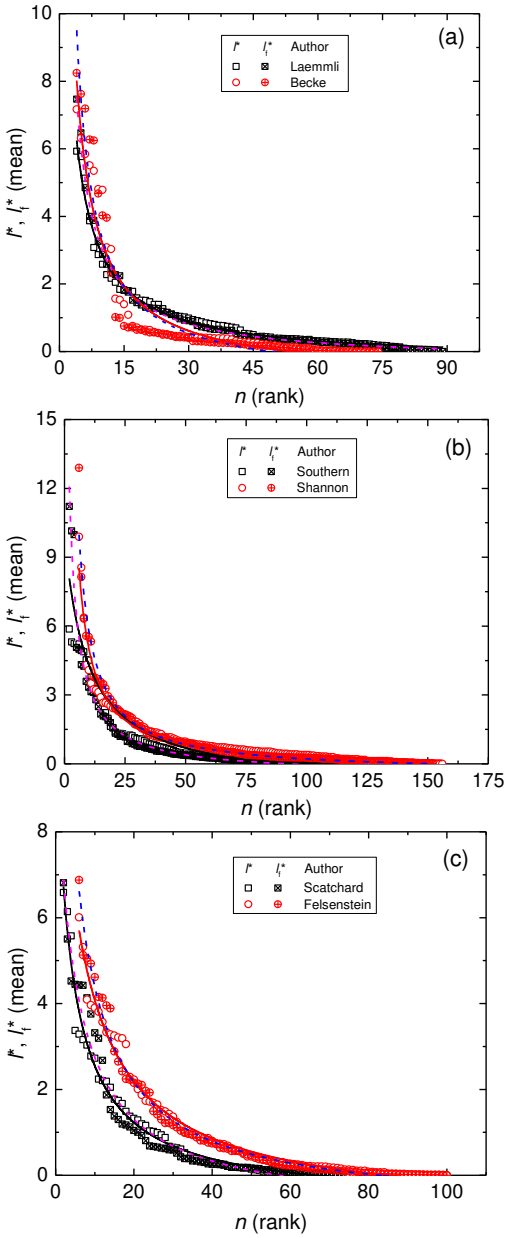


Figure 5: Plots of l^* and l_f^* of papers of different highly-cited authors against their rank n: (a) Laemmli and Becke, (b) Southern and Shannon, and (c) Scatchard and Felsenstein. Solid and dashed curves are drawn for l^* and l_f^* with the best-fit values of the parameters of Eq. (3) listed in Table 2. Cumulative citations L_n and cumulative fractionalized citations L_{fn} are normalized by mean $\langle L_n \rangle$ and $\langle L_{fn} \rangle$ citations.

(see Figure 6b).

From the trends of the distributions of relative citations l^* and l_f^* of papers published by different highly- as well as moderately-cited authors, the following inferences can be made:

(1) Mean citations of individual papers of different authors is better for comparison of their publication output.

(2) Rank-order distribution patterns of relative citations l^* and l_f^* normalized by the mean citations of the papers for different authors are different and are characterized by the two sets of three parameters: l_0^*, α and K for the $l^*(n)$ data and l_{f0}^*, α_f and K_f for the $l_f^*(n)$ data.

(3) Inclusion of citations l_n and l_{nf} of individual highly-cited papers during the analysis leads to large differences in the distributions of normalized citations.

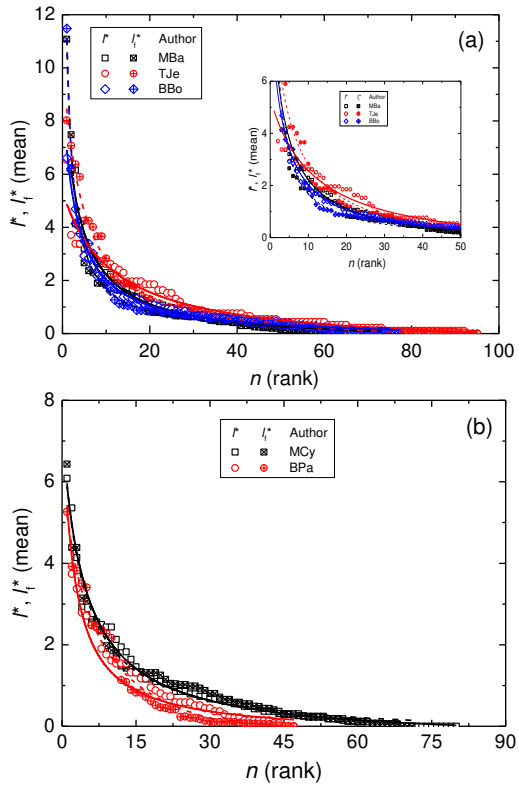


Figure 6: Plots of l^* and l_f^* of papers of different moderately-cited Polish authors against their rank n: (a) MBa, TJe BBo, and (b) MCy and BPa. Solid and dashed curves are drawn for l^* and l_f^* with the best-fit values of the parameters of Eq. (3) listed in Table 3. Inset in (a) shows magnified part of the plots close to the origin. Normalization of citations by mean citations.

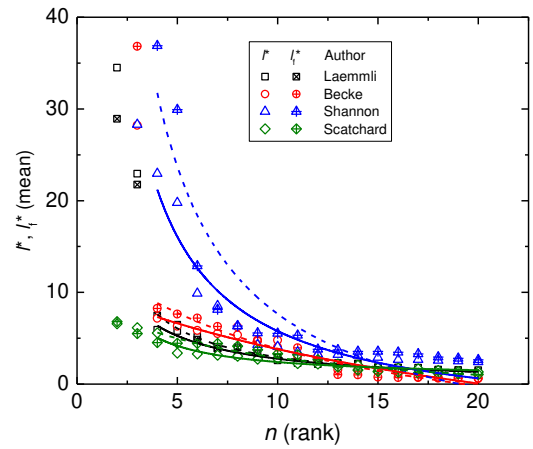


Figure 7: Plots of normalized citations l^* and l_f^* of papers of four highly-cited authors against their rank n in the range of $n < 20$. Solid and dashed curves are drawn for l^* and l_f^* in the range $4 \le n \le 20$ with the best-fit parameters of Eq. (3) listed in Table 4. Normalization of l^* and l_f^* citations by mean citations.

Figure 7 shows plots of relative citations l^* and l_f^* of papers of four highly-cited authors against their rank n in the range of $2 < n < 20$. Solid and dashed curves are

Table 3: Values of parameters of Langmuir-type function (3) for $l^*(n)$ and $l_f^*(n)$ data of different authors

Author	Citations l^* (with coauthors)				Citations l_f^* (without coauthors)			
	l_0^*	α	K	R^2	l_{f0}^*	α_f	K_f	R^2
Laemmli UK	23.40	1.0115	0.658	0.9841	111.7	1.0027	3.325	0.9946
Becke AD	44.81	1.0178	1.043	0.9292	1162	1.0007	27.82	0.9178
Southern EM	10.0	1.1083	0.105	0.8951	27.79	1.0174	0.624	0.9755
Felsenstein J	12.05	1.0730	0.161	0.9849	19.97	1.0404	0.30	0.9872
Shannon RD	523804	1.0	9997	0.9616	623514	1.0	9999	0.9679
Scatchard G	10.31	1.0617	0.244	0.9828	10.0	1.0874	0.20	0.9655
MBa	10.17	1.0362	0.307	0.9690	26.87	1.0041	1.152	0.9840
MCy	7.25	1.0550	0.214	0.9828	7.57	1.0342	0.258	0.9796
TJe	5.35	1.1083	0.088	0.9492	10.57	1.0505	0.233	0.9892
BBo	9.16	1.0255	0.315	0.9887	28.46	1.0014	1.551	0.9820
BPa	7.78	1.0294	0.415	0.9593	6.27	1.1571	0.171	0.9761
	10.0[a]	1.1857[a]	0.113[a]	0.9936[a]				

[a] Best-fit values for data without first point.

Table 4: Values of parameters of Langmuir-type function (3) for $l^*(n)$ data in the rank range $4 \le n \le 20$

Author	Citations l^* (with coauthors)				Citations l_f^* (without coauthors)			
	l_0^*	α	K	R^2	l_{f0}^*	α_f	K_f	R^2
Laemmli UK	33.57	1.0065	1.033	0.9591	177.8	1.002	5.383	0.9881
Becke AD	10.58	2.2264	0.040	0.9535	15.08	1.592	0.088	0.9438
Shannon RD	368560	1.0	3581	0.8991	640567	1.0	3987.4	0.8393
Scatchard G	8361	0.9999	472	0.9125	10	1.218	0.140	0.8760

Table 5: Values of parameters of Langmuir-type function (3) for $l_f^*(n_{eff})$ data

Author	N_c^{ef}	l_{f0}^{*ef}	α_f^{ef}	K_f^{ef}	R^2
Laemmli UK	32.83	165474	1.0	9993	0.9886
Becke AD	48.92	47087	1.00002	1248	0.8894
Southern EM	59.59	24632	1.00003	939.4	0.9556
Felsenstein J	75.31	26.05	1.03429	0.465	0.9853
Shannon RD	58.54	271559	1.0	7898	0.9144
Scatchard G	40.89	10.0	1.10088	0.340	0.9562
MBa	17.91	11.216	1.02460	1.727	0.9890
MCy	22.87	10.0	1.00177	1.341	0.8380
TJe	35.08	10.61	1.08863	0.369	0.9786
BBo	31.31	13.06	1.00222	1.718	0.9675
BPa	21.69	10.0	1.03760	0.747	0.7520

drawn for l^* and l_f^* in the range $4 \le n \le 20$ with the best-fit parameters of Eq. (3) listed in Table 4. From these plots the following features may be noted:

(1) The values of both l^* and l_f^* for all of the four authors show relatively poor fit in view of their irregular decreasing trends with increasing rank n.
(2) Except in the case of Scatchard, for low n the values of l^* and l_f^* steeply decreases with increasing n.
(3) In the case of Becke and Shannon, after an initial steep decrease the values of l^* and l_f^* decrease relatively poorly with increasing n.

These deviations of the data from the best-fit plots are associated with the difference in the number s_{max} of the active citation sites for the papers of an author (see assumption 1 in Section 2).

The distribution of relative citations l_f^* normalized by mean citations $\langle L_f \rangle$ of N_c cited papers of different authors can also be analyzed as a function of their effective rank n_{eff}. Figure 8a and b shows the plots of relative citations l_f^* of papers of the 6 highly-cited and 5 moderately-cited authors, respectively, as a function of their effective rank n_{eff}. The curves in the figures are drawn according to Eq. (3) with the best-fit values of the new parameters l_{f0}^{*ef}, α_f^{ef} and K_f^{ef} of Eq. (3) listed in Table 5. Table 5 also includes the values of the effective number N_c^{ef} cited papers of different authors. It may be noted that the data for all authors are characterized by individual sets of the new parameters l_{f0}^{*ef}, α_f^{ef} and K_f^{ef} of Eq. (3) but the mutual dispersion among them is smaller when the values of l_f^* are normalized by the mean $\langle L_f \rangle$ of cumulative fractions l_{nf} of citations of their papers. The values of the goodness-of-the fit parameter R^2 for the plots of the $l_f^*(n_{eff})$ data are also comparable with those for the plots of their $l_f^*(n)$ data (see Tables 3 and 4).

4.3. Parameters characterizing citation distributions of different authors

The above analysis of the publication output of different authors by Langmuir-type function (3) shows that real rank-order distributions of citations l_n and fractions l_{nf} of citations of papers of an author can be described by the two sets of three parameters: l_0, α and K, and l_{f0}, α_f and K_f, respectively. The parameters l_0 and l_{f0} are related to the parameters l_0^* and l_{f0}^* by the relations: $l_0 = l_0^* \langle L \rangle$ and $l_{f0} = l_{f0}^* \langle L_f \rangle$, where l_0^* and l_{f0}^* are given in Table 3

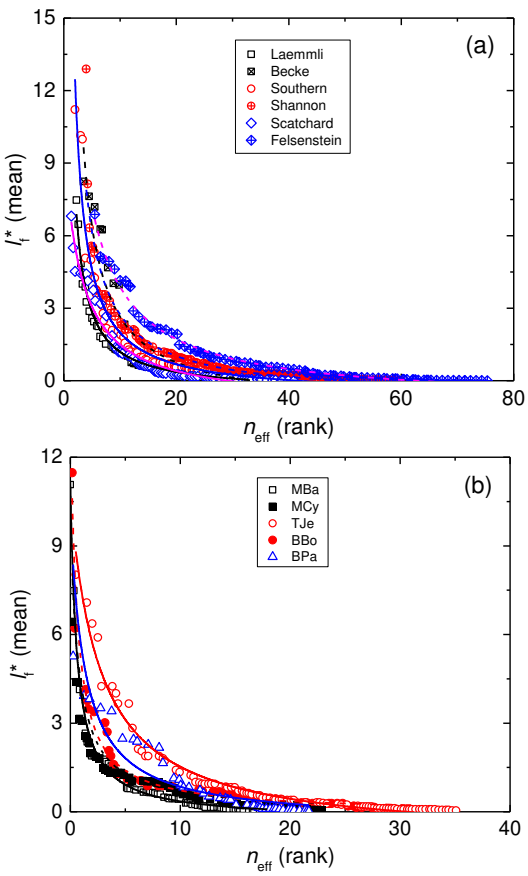


Figure 8: Plots of relatively citations l_f* of papers of (a) highly-cited and (b) moderately-cited authors against their effective rank n_{eff}. Curves are drawn with the best-fit values of the parameters of Eq. (3) listed in Table 4. Normalization of l_f* citations by mean citations.

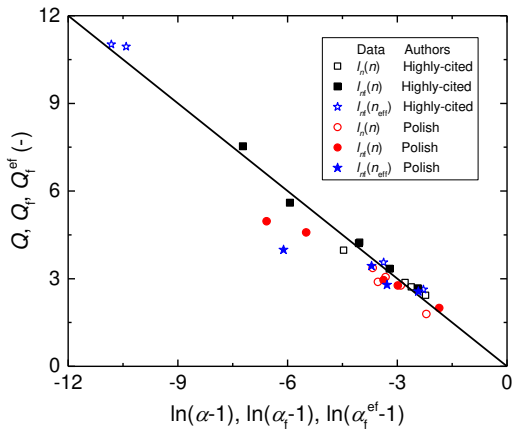


Figure 9: Plots of Q = ln$K$' = ln(KN_c) against ln(α–1), Q_f = lnK_f' = ln(K_fN_c) against ln(α_f –1) and Q_f^{ef} = ln$K_f'^{ef}$ = ln($K_f^{ef}N_c^{ef}$) against ln(α_f^{ef}–1) for different authors. Linear plot represents the dependence expected from Eq. (10) with slope –1.

whereas <L> and <L_f> are listed in Table 2. Similarly, as in the case of the $l_{nf}(n)$ distributions, the $l_{nf}(n_{eff})$ distribution may be represented by a new set of three parameters: l_{f0}^{ef}, α_f^{ef} and K_f^{ef}, with l_{f0}^{ef} = l_{f0}^{*ef}<L_f>. The values of the new parameters l_{f0}^{*ef}, α_f^{ef} and K_f^{ef} are listed in Table 5 and those of <L_f> are given in Table 2.

The best-fit values of the parameters l_0, l_{f0} and l_{f0}^{ef} obtained from the $l_n(n)$, $l_{nf}(n)$ and $l_{nf}(n_{eff})$ distributions are usually very high and comparable (cf. Tables 2, 3 and 5). However, the mean citations <L> and <L_f> are parameters characteristic for an author. Similarly, the effectiveness parameters α, α_f and α_f^{ef}, the Langmuir constants K, K_f and K_f^{ef}, and the number N_c and the effective number N_c^{ef} of citable papers of an author are characteristic parameters of his/her publication output.

Eq. (7) may be rewritten in the form (cf. Eq. (5))

$$q = q_0 - \ln(\alpha - 1)\ , \qquad (8)$$

where q = ln$k$, q_0 = lnk_0, k denotes dimensionless Langmuir constant K' = KN_c, K_f' = K_fN_c and K_f'* = K_f*N_c* as above, k_0 denotes the values of K', K_f' or K_f'* when (α–1) = 0, and α denotes the values of α, α_f and α_f* corresponding to K', K_f' and K_f'*, respectively. Now the parameter q represents the dimensionless differential energy Q, Q_f and Q_f* corresponding to KN_c, K_fN_c and K_f*N_c*, respectively. According to Eq. (8), q_0 = lnk_0 = 0 because the proportionality constant $(kN_c)_0$ = k_0 = 1 (see Section 3).

Figure 9 shows the data of Q = ln$K$' = ln(KN_c), Q_f = lnK_f' = ln(K_fN_c) and Q_f^{ef} = ln$K_f'^{ef}$ = ln($K_f^{ef}N_c^{ef}$) against ln(α–1), ln(α_f–1) and ln(α_f^{ef}–1), respectively, for different authors according to Eq. (8). The values of different quantities were taken from Tables 1, 3 and 5. As expected from Eq. (8), one observes from Figure 9 a practically linearly decreasing plot of the data of Q, Q_f and Q_f*, with a slope of unity and an intercept of zero, against increasing ln(α–1), ln(α_f–1) and ln(α_f*–1), respectively, for the distribution of citations to the papers published by different authors. Since the effectiveness parameters α, α_f and α_f^{ef} are related to KN_c, K_fN_c and $K_f^{ef}N_c^{ef}$. by Eq. (8), it can be argued that the latter parameters, and the related dimensionless differential energies Q, Q_f and Q_f^{ef}, also characterize the publication output of an author.

From the above discussion it may be concluded that pairs of an author's mean citations <L> and dimensionless differential energy Q (where Q = ln(KN_c)) and <L_f> and Q_f (where Q_f = ln(K_fN_c)) are possible measures of characterization (and comparison) of his/her publication output with that of other authors from the distribution of cumulative citations l_n of their N_c cited papers published during their entire publication career. When contributions of coauthors to the citation l_n of the papers of an author are counted as l_{nf} and their ranking is arranged as effective rank n_{eff}, the relevant parameters obtained from the distribution of l_{nf} of their N_c^{ef} papers are: <L_f> and Q_f^{ef} = ln($K_f^{ef}N_c^{ef}$).

The main problem with the above analysis is that the pair of parameters cannot be combined into a single index for the comparison of the publication output of different authors.

4.4. Comparison of various single citation indices

Based on his stochastic model [55], Burrell [56] proposed a simple geometric distribution involving the number N of papers receiving L citations for an author to calculate his/her so-called quasi h index (h'). Using

the published data of the number of papers N and citations L of 15 authors, Burrell [56] showed that, if one or more outliers in the upper tail of the citation distribution of an author are omitted, his approach gives good estimate of their h index. It may be noted that the values of this h' index, estimated by Burrell, of the 15 authors are comparable with the values of their citation radius $R = (L/\pi)^{1/2}$ (cf. ref. [57]).

From the distributions of citations $l_n(n)$, $l_{nf}(n)$ and $l_{nf}(n_{eff})$ data of the different authors their Hirsch index h, the new index h_f and the Hirsch-type index h_m, respectively, were determined using the procedures mentioned above. Following the proposal of Batista et al. [18], the so-called h_1 index was calculated from the relation: $h_1 = h^2/A_h$, where A_h is the total number of coauthors of a particular author in the h-core. Excluding the citations of the outliers, the citation radii R and R_f of the authors were also calculated from their cumulative citations L and L_f (cf. ref. [57]). The calculated values of the h, h_f, h_m and h_1 indices and the citation radii R and R_f of the authors are included in Table 1. The values of the number A_h of authors in the h-core, the number of cumulative citations L of the authors without their outliers, the number N_c of papers without outliers, and the number N_m of the papers involved in the h_m index of the authors are given in Table 1 in the parentheses.

From Table 1 it may be noted that the value of the h index of different authors is related directly neither to their cumulative citations L nor to the number of papers published by them. For example, Felsenstein, with L = 46478 citations, has an h index equal to 51 whereas Becke, with citations twice the citations of Felsenstein, has h equal to 48 only. Another example is that of TJe and MCy having h index equal to 18 but the latter's papers received citations about 40% higher than that of the former. Similarly, Becke with 70 cited papers has a higher h index than that of Southern and Shannon with 110 and 152 cited papers, respectively.

As in the case of the h index of different highly-cited authors, their h_f index is also related neither to their cumulative citations L_f nor to the number of papers published by them (Table 1). For example, Laemmli and Becke have h_f = 40 but Becke has L_f twice higher than that of Laemmli. Similarly, although Shannon has about 40% higher cited papers than Southern, both have the same h_f = 28. In the case of moderately-cited Polish professors the above trends are not so clear. However, as judged from the number A_h of coauthors in the h-core and the number N_c of cited papers of an author, the value of his/her h_f index is determined by the number of coauthors of the high-ranked (i.e. more cited) papers and the number of his/her published papers.

As seen from Table 1, the estimated citation radius R for different moderately-cited Polish authors are comparable with their h index. Similarly, the values of R_f for these authors are also comparable with their h_f index. In contrast to this, the estimated values of R for the highly-cited authors are much higher than their h index even when citations from outliers are excluded from cumulative citations. However, the values of their R_f are close to their h_f in some cases or somewhat higher than their h_f in other cases. These observations suggest that $R \approx h$ and $R_f \approx h_f$ when the distribution of citations l_n and l_{nf} of succeeding papers in the range of more cited papers (i.e. $n < h$) is a smoothly, rather than a steeply, decreasing function of lowering rank n of the papers. This is the situation in the case of moderately-cited Polish professors. However, the values of R and R_f are higher than h and h_f, respectively, when the papers of rank $n < h$ in the citation distributions receive relatively high citations l_n and l_{nf}. This is the situation in the case of highly-cited authors (see Table 1). While calculating his h' index of different authors from their cumulative citations L and papers N, Burrell [56] also attributed the estimated high values of the h' index of some of the authors than those of their h index to the citations of highly cited papers.

Table 1 reveals that the h_1 index is not a suitable citation-based measure of comparison of the research output of different authors. For example, Laemmli, with citations five times higher than that of Felsenstein, has h_1 roughly one-half of the h_1 of Felsenstein. Similarly, although MBa's papers received more citations than MCy's papers, MBa has h_1 lower than that of MCy by a factor of 7.5. This anomalous behavior of the h_1 index is associated with the total number A_h of coauthors considered in calculating it from the h index. A similar behavior was reported by Schreiber [42].

It is interesting to note the difference in the number of papers included in the calculation of citation radii R and R_f and Hirsch and Hirsch-type indices h, h_f and h_m of different authors. While the citations received by all cited papers N_c of an author are taken into account in the calculation of his/her citation radii R and R_f, counting of h index is based on cited papers $n_c \leq h < N_c$ and that of h_f index is based on even a lower number of cited papers (see Table 1). For example, in the case of Laemmli, the counting of his h index involves 58 papers but that of h_f index based on fractions of citations of his papers with coauthors is just 40. The deficiencies of the h and h_f indices of different authors mentioned above are overcome by the h_m index, which is determined by fractions n_{eff} of the rank of a higher number of their papers. The determination of this h_m index of an author involves the number of papers comparable with that involved in determining his/her h index (see Table 1). In the case of Laemmli for example, the number N_m of the cited papers involved in the calculation of h_m index is 55 in comparison with n_c = 58 cited papers in determining the h index. These observations suggest that h_m index is superior to h and h_f for comparison of the publication output of different authors because it takes into account the number of coauthors of their papers as well as the number of cited papers comparable with those involved in the calculation of the classical h index.

From the data of cumulative fraction L_f of citations of the authors given in Table 1 one finds empirically that, with the exception of Felsenstein and MBa, $h_m \approx (2L_f)^{1/3}$. The estimated value of h_m of 26.70 is much lower than the calculated 41.92 for Felsenstein whereas

Table 6: Comparison of different scaling parameters for citations analysis of various authors

Author	t (yr)	N_c	$<L>$	$<L_f>$	b	b_f	b_m
Laemmli UK	45	84	166.40 (2)	65.99 (3)	1.29 (2)	0.89 (2)	0.476 (4)
Becke AD	36	70	232.88 (1)	178.41 (1)	1.33 (1)	1.11 (1)	0.8728 (2)
Southern EM	56	110	60.66 (5)	31.75 (4)	0.73 (7)	0.50 (6)	0.341 (7)
Felsenstein J	48	99	130.98 (3)	101.21 (2)	1.06 (5)	0.625 (4)	0.8733 (1)
Shannon RD	45	152	41.66 (6)	15.98 (6)	1.00 (6)	0.622 (5)	0.407 (5)
Scatchard G	68	73	78.54 (4)	38.90 (5)	0.485 (10)	0.368 (8)	0.236 (9)
MBa	15	70	12.19 (10)	3.39 (11)	1.133 (4)	0.467 (7)	0.349 (6)
MCy	35	68	16.43 (7)	4.56 (9)	0.514 (8)	0.257 (10)	0.230 (10)
TJe	15	92	9.17 (11)	4.24 (10)	1.20 (3)	0.667 (3)	0.537 (3)
BBo	41	74	13.01 (9)	5.32 (8)	0.415 (11)	0.244 (11)	0.203 (11)
BPa	32	44	13.79 (8)	9.69 (7)	0.50 (9)	0.344 (9)	0.3225 (8)

that of h_m of 7.80 is higher than the calculated 5.24 for MBa. As seen from Table 1, these anomalies are associated with differences in the number of coauthors of the top-cited papers. The number of coauthors per paper in the former case is relatively low in comparison with those in the latter case.

Finally, it should be noted from Tables 1 and 2 that the h, h_f and h_m indices of different authors are, in general, related to their mean citations $<L>$ and $<L_f>$. Consequently, all highly-cited authors with high $<L>$ and $<L_f>$ are in the top six positions whereas the moderately-cited authors with the low of $<L>$ and $<L_f>$ occupy positions lower than those of the highly-cited authors.

4.5. Scaling parameters for research output of different authors

The h, h_f and h_m indices and the pairs of characteristic parameters $<L>$ and Q, and $<L_f>$ and Q_f of an author, discussed above, are related to the distributions $l_n(n)$, $l_{nf}(n)$ and $l_{nf}(n_{eff})$, respectively, constructed from his/her bibliometric data. The inherent disadvantage with all of these measures of the publication output of an author is that their values do not take into account the length of publication career of the author. It is difficult to calculate career-length independent characteristic parameters from Langmuir-type function for individual authors, but this factor can be considered in the h, h_f and h_m indices using the ideas used before [58].

The relationship between cumulative citations $L(t)$ and Hirsch index h is given by [1]

$$L(t) = Ah^2(t), \tag{9}$$

whereas according to progressive nucleation mechanism [57,58] and stochastic model [55]

$$L(t) = at^2. \tag{10}$$

In the above equations A is an empirical constant lying between 3 and 5, and a is the so-called citation acceleration related, among others, to the average number of papers published per year by an author. From the above relations one obtains

$$h = bt, \tag{11}$$

where the proportionality constant $b = (a/A)^{1/2}$, with units year^{-1}.

Relation (11) means that the average h of an author increases linearly with publication career t. Since other Hirsch-related indices like h_f and h_m are based on similar concept, it can be argued that these indices also increase linearly with t. The calculated values of the parameter b, denoted as b, b_f and b_m corresponding to h, h_f and h_m, respectively, are given in Table 6.

From Table 6 it may be noted that the parameters b, b_f and b_m for different authors lead to a major change in their ranking. When the contribution of coauthors is taken into account for a comparison of the publication of different authors, the ranking of highly-cited authors is drastically changed, and highly-cited authors like Southern EM and Scatchard G are shifted downward but moderately-cited authors like TJe and MBa of short publication career are shifted upward to 3rd and 6th ranks, respectively.

5. Summary and conclusions

The real data of cumulative citations l_n of nth paper of individual N papers published by selected authors are analyzed without and with consideration of the number A_n of coauthors of the paper, the normalization of citations l_n and cumulative fraction l_{nf} of citation of the nth paper by mean and median citations of the citations l_n of all N_c cited papers and the determination of cumulative fraction n_{eff} of the rank of the l_{nf} citations. In view of different authorship patterns used in the lists of authors of the papers considered in this study and enormously high number of coauthors in many cases, l_{nf} and n_{eff} were calculated on the assumption of equal contributions of A_n coauthors of the nth-ranked paper fetching l_n citations.

It was found that Langmuir-type function (3) is not adequate for comparison of the publication output of different authors because it describes the rank-order distribution patterns satisfactorily in terms of two parameters. However, when equal contribution of coauthors is taken into account for the comparison of their publication output, the h_m index is more consistent than other Hirsch-type indices. To account for the length of their publication career t, it is suggested to use scaling parameters h/t, h_f/t and h_m/t for a comparison of the publication output of different authors

Acknowledgement

The author expresses his gratitude to Dr Kazimierz Wójcik for his assistance with collection of the bibliometric data used in this work.

References

[1] J.E. Hirsch, An index to quantify an individual's scientific research output, Proceedings of the National academy of Sciences of the United States of America 102(46) (2005) 16569-16572.

[2] M. Kosmulski, A new Hirsch-type index saves time and works equally well as the original h-index, ISSI Newsletter 2(3) (2006) 4-6.

[3] B. Jin, L. Liang, R. Rousseau, L. Egghe, The R- and AR-indices: complementing the h-index, Chinese Science Bulletin 52(6) (2007) 855-863.

[4] S. Alonso, F. Cabrerizo, E. Herrera-Viedma, F. Herrera, h-index: A review focused in its variants, computation and standardization for different scientific fields, Journal of Informetrics 3(4) (2009) 273-280.

[5] T.R. Anderson, R.K.S. Hankin, P.D. Killworth, Beyond the Durfee square: Enhancing the h-index to score total publication output, Scientometrics 76(3) (2008) 577-588.

[6] Q.L. Burrell, On Hirsch's h, Egghe's g and Kosmulski's h(2). Scientometrics 79(1) (2009) 79-91.

[7] L. Egghe, Characteristic scores and scales based on h-type indices. Journal of Informetrics 4(1) (2010) 14–22.

[8] F. Franceschini, D. Maisano, The Hirsch spectrum: a novel tool for analyzing scientific journals, Journal of Informetrics 4(1) (2010) 64-73.

[9] F. Franceschini, D. Maisano, The citation triad: an overview of a scientist's publication output based on Ferrers diagrams, Journal of Informetrics 4(4) (2010) 503-511.

[10] W. Glänzel, A. Schubert, Hirsch-type characteristics of the tail of distributions. The generalized h-index, Journal of Informetrics 4(1) (2010) 118-123.

[11] A. Tietze, P. Hofmann, The h-index and multi-author h_m-index for individual researchers in condensed matter physics, Scientometrics 119 (2019) 171-185.

[12] M. Ameer, M.T. Afzal, Evaluation of h-index and its qualitative and quantitative variants in Neuroscience, Scientometrics 121 (2019) 653-673.

[13] S. Li, H. Shen, P. Bao, X, Cheng, h_u-index: a unified index to quantify individuals across disciplines, Scientometrics 126 (2021) 3209-3226.

[14] M. Kosmulski, New seniority-independent Hirsh-type index, Journal of Informetrics 3(4) (2009) 341-347.

[15] H.A. Abt, A publication index that is independent of age, Scientometrics 91(3) (2011) 1053-1058.

[16] R. Mannella, P. Rossi, On the time dependence of the h-index, Journal of Informetrics 7(1) (2013) 176-182.

[17] D. Pradhan, P.S. Paul, U. Maheswari, S. Nandi, T. Chakraborty, C3-index: a PageRank based multi-faceted metric for authors' performance measurement, Scientometrics 110 (2017) 253–273.

[18] P.D. Batista, M.G. Campiteli, O. Kinouchi, & A.S. Martinez, Is it possible to compare researchers with different scientific interests?, Scientometrics 68(1) (2006) 179-189.

[19] M. Ausloos, A simple law about co-authors and their ranking: The co-author core, Scientometrics 95(3) (2013) 895-909

[20] D.D.S. Price, Multiple authorship, Science 212(4498) (1981) 986.

[21] S.E. Hodge, D.A. Greenberg, Publication credit, Science 212(4498) (1981) 950.

[22] P. Vinkler, Research contribution, authorship and team cooperativeness, Scientometrics 26(1) (1993) 213-230.

[23] A.F. Pereira de Araújo, Increasing discrepancy between absolute and effective indexes of research output in a Brazilian academic department, Scientometrics 74(3) (2008) 425-437.

[24] N. Assimakis, M. Adam, A new author's productivity index: p-index, Scientometrics 85(2) (2010) 415-427.

[25] R.S.J. Tol, Credit where credit's due: accounting for co-authorship in citation counts, Scientometrics 89(1) (2011) 291-299.

[26] S. Sahoo, Analyzing research performance: proposition of a new complementary index, Scientometrics 108(2) (2016) 489-504.

[27] J.-W. Hsu, D.-W. Huang, Correlation between impact and collaboration, Scientometrics 86(1) (2011) 317-324.

[28] M. Kosmulski, The order in the lists of authors in multi-author papers revisited, Journal of Informetrics 6(4) (2012) 639-644.

[29] C.H. Sekercioglu, Quantifying coauthor contributions, Science 322(5900) (2008) 371.

[30] N.T. Hagen, Credit for coauthors, Science 323(5914) (2009) 583.

[31] N.T. Hagen, Harmonic publication and citation counting: sharing authorship credit equitably – not equally, geometrically or arithmetically, Scientometrics 84(3) (2010) 785-793.

[32] N.T. Hagen, Harmonic coauthor credit: A parsimonious quantification of the byline hierarchy, Journal of Informetrics 7(3) (2013) 784-793.

[33] C.T. Zhang, A proposal for calculating weighted citations based on author rank, EMBO Reports 10(5) (2009) 416-417.

[34] I. Podlubny, Comparison of scientific impact expressed by the number of citations in different fields of science, Scientometrics 64(1) (2005) 95-99.

[35] J.E. Iglesias, C. Pecharroman, Scaling the h-index for different scientific ISI fields, Scientometrics 73(3) (2007) 303-320.

[36] J. Lundberg, Lifting the crown – Citation z-score, Journal of Informetrics 1(2) (2007) 145-154.

[37] F. Radicchi, S., Fortunado, C. Castellano, Universality of citation distributions: Towards an objective measure of scientific impact, Proceedings of the National academy of Sciences of the United States of America 105(45) (2008) 17268–17272.

[38] G. Abramo, T. Cicero, C.A. D'Angelo, Revisiting the scaling of citations for research assessment, Journal of Informetrics 6(3) (2012) 420-479.

[39] L. Bornmann, H.-D. Daniel, Universality of citation distribution – a validation of Radicchi et al.'s relative indicator $c_f = c/c_0$ at the micro level using data from chemistry, Journal of the American Society for Information Science and Technology 60(8) (2009) 1664-1670.

[40] J.E. Hirsch, An index to quantify an individual's scientific research output that takes into account the effect of multiple coauthorship, Scientometrics 85(3) (2010) 741-754.

[41] L. Egghe, Mathematical theory of the h- and g-index in case of fractional counting of authorship, Journal of the American Society for Information Science and Technology 59(10) (2008) 1608-1616.

[42] M. Schreiber, A modification of the h-index: The h_m-index accounts for multi-authored manuscripts, Journal of Informetrics 2(3) (2008) 211-216.

[43] M. Schreiber, A case study of the modified Hirsch index h_m accounting for multiple co-authors, Journal of the

American Society for Information Science and Technology 60(6) (2009) 1274-1282.

[44] G. Abramo, C.A. D'Angelo, F. Rosati, The importance of accounting for the number of co-authors and their order when assessing research performance at the individual level in the life sciences, Journal of Informetrics 7 (2013) 198-208.

[45] M. Dunaiski, J. Geldenhuys, W. Visser, Author ranking evaluation at scale, Journal of Informetrics 12 (2018) 679-702.

[46] M. Salman, M.M. Ahmed, M.T. Afzal, Assessment of author ranking indices based on multi-authorship, Scientometrics 126 (2021) 4153-4172.

[47] L. Egghe, A rationale for the Hirsch-index rank-order distrubtion and a comparison with the impact factor rank-order distribution, Journal of the American Society for Information Science and Technology 60(10) (2009) 2142-2144.

[48] L. Egghe, R. Rousseau, An informetric model for the Hirsch-index, Scientometrics 69(1), (2006) 121-129.

[49] L. Egghe, R. Rousseau, The Hirsch index of a shifted Lotka function and its relation with the impact factor, Journal of the American Society for Information Science and Technology 63(5) (2012) 1048-1053.

[50] K. Sangwal, Comparison of different mathematical functions for the analysis of citation distribution of papers of individual authors, Journal of Informetrics 7(1) (2013) 36-49.

[51] Q.L. Burrell, Formulae for the h-index: A lack of robustness in Lotkaian informetrics?, Journal of the American Society for Information Science and Technology 64(7) (2013) 1507-1514.

[52] K. Sangwal, Citation and impact factor distributions of scientific journals published in individual countries, Journal of Informetrics 7(3) (2013) 487-504.

[53] K. Sangwal, Distributions of citation of papers of individual authors publishing in different scientific disciplines: Application of Langmuir-type function, Journal of Informetrics 8(4) (2014) 972-984.

[54] K.Y. Chuang, Y.S. Ho, Bibliometric profile of top-cited single-author articles in the Science Citation Index Expanded, Journal of Informetrics 8(4) (2014) 951-962.

[55] Q.L. Burrell, Hirsch's h-index: A stochastic model, Journal of Informetrics 1(1) (2007) 16-25.

[56] Q.L. Burrell, The h-index: A case of the tail wagging the dog?, Journal of Informetrics 7(3) (2013) 774-783.

[57] K. Sangwal, On the relationship between citations of publication output and Hirsch index h of authors: conceptualization of tapered Hirsch index h_T, circular citation area radius R and citation acceleration a, Scientometrics 93(3) (2012) 987-1004.

[58] K. Sangwal, On the age-independent publication index, Scientometrics 91(3) (2012) 1053-1058.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 165–171
Received: 21 April 2022
Accepted: 27 April 2022

Analysis of the performance of iOS applications developed using native and cross-platform technology

Analiza wydajności aplikacji iOS stworzonych przy użyciu technologii natywnej i crossplatformowej

Marcin Michałowski* , Maria Skublewska-Paszkowska

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

Study presented in this paper concerns the comparative analysis of the performance of iOS applications developed using native and cross-platform technologies. For the purpose of the research, two iOS applications were implemented: the first one was created using the Swift programming language, while the second one using the Flutter technology. For both applications, a set of research scenarios was defined, which assumed the examination of the time of execution and CPU consumption during the execution of operations, such as: sorting integers, writing and reading string from a file or writing and reading records from the SQLite database. The conducted analysis showed that it is not possible to clearly state which application is more efficient in terms of execution time and CPU consumption, because they obtained divergent results for different research scenarios. The native application performed better for file and database operations, while the cross-platform one obtained lower time and CPU consumption when sorting numbers.

Keywords: iOS; Flutter; crossplatform applications; native applications

Streszczenie

Badania przedstawione w niniejszym artykule dotyczą analizy porównawczej wydajności aplikacji iOS stworzonych przy użyciu technologii natywnej i crossplatformowej. Na potrzeby badań zostały utworzone dwie aplikacje iOS: pierwsza zaimplementowana przy użyciu języka Swift, natomiast druga przy użyciu technologii Flutter. Dla obu aplikacji określono zestaw scenariuszy badawczych, które zakładały zbadanie czasu wykonania oraz zużycia jednostki obliczeniowej w czasie wykonywania poszczególnych operacji takich jak: sortowanie liczb całkowitych, zapis i odczyt ciągu znaków z pliku oraz zapis i odczyt rekordów z bazy danych SQLite. Przeprowadzona analiza wykazała, że nie da się jednoznacznie stwierdzić, która aplikacja jest bardziej wydajna pod względem czasowym i zużycia procesora, ponieważ uzyskiwały one rozbieżne wyniki dla różnych scenariuszy badawczych. Aplikacja natywna uzyskała lepsze rezultaty w przypadku operacji na plikach i operacji na bazie danych, natomiast aplikacja crossplatformowa wykazała się niższym czasem i zużyciem procesora podczas sortowania liczb.

Słowa kluczowe: iOS; Flutter; aplikacje wieloplatformowe; aplikacje natywne

*Corresponding author

Email address: marcin.michalowski@pollub.edu.pl (M. Michałowski)

1. Wstęp

Na przestrzeni ostatnich kilkunastu lat obserwuje się bardzo dynamiczny rozwój urządzeń i technologii mobilnych [1]. Gdy telefony komórkowe pojawiły się na rynku, na ich zakup mogli sobie pozwolić nieliczni, a ich możliwości były ograniczone jedynie do wykonywania połączeń. Z czasem urządzenia zaczęły oferować możliwość wysyłania wiadomości tekstowych, czy robienia zdjęć. W dzisiejszych czasach telefony przerodziły się w smartfony, które towarzyszą ludziom na co dzień, posiadają dużo większe możliwości i ułatwiają życie ich użytkownikom. Smartfony pełnią funkcję nawigacji, odtwarzaczy audio i wideo, aparatu, umożliwiają dostęp do Internetu oraz wiele innych, co jest możliwe dzięki aplikacjom, które są instalowane na tych urządzeniach.

Rynek urządzeń i aplikacji mobilnych to ogromne korzyści finansowe dla producentów. Ze względu na to, zarówno firmy produkujące urządzenia, jak i programiści kładą duży nacisk na to, aby urządzenia były coraz wydajniejsze, a aplikacje coraz lepiej zoptymalizowane pod względem zużycia zasobów urządzenia.

W rzeczywistości rynek urządzeń mobilnych jest podzielony pomiędzy dwa systemy – Android i iOS, gdzie Android stanowi 69% rynku, iOS 30%, a pozostały 1% dzieli się pomiędzy mniej istotne, najczęściej stare i nierozwijane już systemy [2]. Jeżeli firma tworząca aplikacje chce trafić do możliwie największej liczby użytkowników, musi rozwijać równocześnie dwie aplikacje – jedną dla systemu Android, a drugą na urządzenia z systemem iOS. Takie podejście wiąże się z posiadaniem dwóch zespołów do stworzenia tych aplikacji, a następnie do utrzymania ich i dalszego rozwoju, co oznacza wysokie koszty.

Aktualnie główną technologią do tworzenia aplikacji na platformę iOS jest język programowania Swift, jednak mając na uwadze wysokie koszty produkcji i utrzymania dwóch osobnych aplikacji, pojawił się pomysł programowania crossplatformowego, które umożliwia stworzenie jednej aplikacji, którą będzie można uru-

chomić zarówno na systemie Android, jak i iOS. Na rynku jest kilka technologii, które umożliwiają programowanie międzyplatformowe jak np. React Native, czy Xamarin, jednak w ostatnich latach najbardziej popularnym rozwiązaniem stał się Flutter [3].

Flutter jest technologią crossplatformową stworzoną przez Google i przedstawioną podczas Dart Developer Summit w 2015 roku. Aktualnie jest on prężnie rozwijany i daje możliwość stworzenia aplikacji nie tylko na platformy mobilne, ale także na systemy Windows, Linux, Mac, FuchsiaOS oraz jako aplikacje internetowe, które można uruchomić w przeglądarce [1]. Ponadto, w systemie FuchsiaOS, który ma być następcą systemu Android, cały interfejs i aplikacje tworzone są we Flutterze, co pokazuje, że Google będzie rozwijać tę technologię [4].

Pomimo wszystkich wymienionych korzyści płynących ze stosowania Fluttera, zastanawiające jest, czy ta technologia jest warta używania ze względu na wydajność, porównując ją z technologiami natywnymi. Poniższa praca podejmuje problem porównania dwóch identycznych pod względem funkcjonalności aplikacji stworzonych przy użyciu technologii natywnej iOS jaką jest Swift oraz technologii crossplatformowej Flutter pod względem szybkości wykonywania zadań, a także zużycia procesora.

2. Przegląd literatury

Przed przystąpieniem do analizy został wykonany przegląd literatury naukowej dotyczącej aplikacji mobilnych stworzonych przy użyciu narzędzia Flutter i technologii natywnych pod względem ich wydajności. Ze względu na fakt, że technologia Flutter jest dosyć nowa, przegląd wykazał, że jej temat nie występuje dosyć często w publikacjach naukowych, aczkolwiek bazy z literaturą naukową zawierają wiele artykułów porównujących aplikacje stworzone przy użyciu technologii natywnych z aplikacjami stworzonymi przy użyciu technologii crossplatformowych innych niż Flutter, co jest znaczące w kontekście powyższego tematu pracy.

M. Olsson w swojej pracy [5] porównuje aplikacje stworzone przy użyciu technologii natywnych – odpowiednio Kotlin dla Androida i Swift dla iOS, z aplikacją stworzoną przy użyciu narzędzia Flutter. Aplikacje zostały porównane pod względem zużycia procesora, liczby linijek kodu potrzebnych do ich stworzenia oraz wyglądu i ogólnych odczuć z ich używania przez użytkowników. Badania wykazały, że Flutter nie odbiega wydajnością pod względem zużycia procesora od technologii natywnych, a do stworzenia aplikacji w tej technologii potrzeba było prawie 3-krotnie mniej linijek kodu niż do stworzenia aplikacji w języku Swift oraz 2-krotnie mniej niż przy aplikacji napisanej przy użyciu języka Kotlin. Większość użytkowników testujących aplikacje nie zauważyło różnicy pomiędzy nimi, a jedynym mankamentem technologii Flutter okazały się animacje przewijania list, które nie były tak płynne, jak w przypadku natywnych aplikacji.

D. Gałan, K. Fisz i P. Kopniak w swoim artykule [6] porównują dwie aplikacje na platformę Android stworzone przy użyciu technologii natywnej Kotlin oraz crossplatformowej Flutter pod względem szybkości wykonywania konkretnych operacji np. sortowania liczb czy operacji na bazie danych. Autorzy poddali również analizie liczbę linijek kodu źródłowego obu aplikacji, dostępności bibliotek oraz wsparcia społeczności w obu technologiach. Badania wykazały, że aplikacja zaimplementowana za pomocą natywnej technologii jest w większej liczbie przypadków wydajniejsza, a natywna technologia jaką jest Android SDK (Kotlin) ma większą liczbę dostępnych bibliotek i większe wsparcie społeczności niż technologia Flutter.

Kolejną publikacją podejmującą tematykę porównania aplikacji natywnych i crossplatformowych pod względem ich wydajności jest artykuł P. Kotarskiego, K. Śledzia i J. Smołki [7], w którym porównane zostały aplikacje na platformę Android stworzone przy użyciu narzędzi Android SDK (Java), Android NDK, Apache Cordova i Xamarin. Badania polegały na zmierzeniu czasów sortowania liczb w tablicach, zapisu pliku o rozmiarze 10MB do pamięci urządzenia oraz odczytu pliku tekstowego o rozmiarze 10MB. Przeprowadzone testy wykazały, że Apache Cordova jest wyraźnie gorsza pod względem wydajności niż pozostałe technologie. Ponadto autorzy stwierdzili, że nie można jednoznacznie stwierdzić, która technologia jest najwydajniejsza, ponieważ wszystkie technologie oprócz Apache Cordova uzyskały bardzo zbliżone wyniki.

P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik i J. Smołka w swoim artykule [8] dokonali analizy wydajności mobilnej aplikacji crossplatformowej utworzonej za pomocą technologii Xamarin oraz dwóch natywnych aplikacji dla platformy Android i iOS. Analiza dotyczyła czasu obliczania liczby π z dokładnością do 10000 miejsc po przecinku, zapisu i odczytu z pliku tekstowego obliczonej liczby π, pobierania grafiki o rozmiarze 7MB oraz ustalenia pozycji GPS. Wyniki pokazały, że prawie we wszystkich przypadkach technologie natywne są wydajniejsze niż technologia Xamarin. Autorzy zauważyli ponadto, że Xamarin.Forms pozwala znacznie skrócić czas implementacji aplikacji, co wpływa na zmniejszenie kosztów stworzenia aplikacji mobilnej.

W kolejnej publikacji [9], autorzy D. Dobrzański i W. Zabierowski, porównali ze sobą pod względem wydajności dwie aplikacje stworzone za pomocą technologii natywnych oraz aplikację crossplatformową zaimplementowaną przy użyciu technologii Xamarin. W swoim artykule przeprowadzili badania, dotyczące operacji na bazie danych SQLite, nakładania efektu na grafikę określoną liczbę razy i odświeżania danych w interfejsie użytkownika. Na podstawie wyników stwierdzono, że Xamarin jest wolniejszy niż technologie natywne, jednak nie są to duże różnice czasowe.

O. Axelsson i F. Carlström w swojej pracy [10] porównali aplikacje stworzone przy użyciu technologii natywnych napisanych w języku Swift i Java oraz aplikacji crossplatformowej stworzonej za pomocą technologii React Native. Na podstawie przeprowadzonych badań autorzy stwierdzili, że narzędzie React Native jest mniej

wydajne od technologii natywnych pod względem zużycia procesora i pamięci RAM. Ponadto badania wykazały, że przy użyciu tej technologii da się stworzyć identyczny interfejs użytkownika, jednak nie jest on tak płynny i responsywny jak w przypadku natywnych technologii, co szczególnie widać przy animacjach.
Kolejną publikacją, która porusza podobny problem jest praca M. Rodríguez-Sánchez Guerra [11], w którym porównane zostały ze sobą aplikacje crossplatformowe stworzone przy użyciu następujących technologii: React Native, Ionic i Flutter. Badania polegały na przeprowadzaniu prostych operacji CRUD (ang. Create, Read, Update, Delete) na liście elementów wyświetlanej na ekranie i zmierzenie czasu poszczególnych operacji. Wyniki badań pokazały wyraźną przewagę Fluttera nad pozostałymi technologiami we wszystkich testowanych przypadkach i na obu platformach systemowych.
W ramach pracy postawiono trzy hipotezy badawcze w celu zweryfikowania ich poprawności:

1. Aplikacja stworzona przy użyciu natywnej technologii jest bardziej wydajna pod względem czasowym oraz zużycia procesora w przypadku operacji na plikach.
2. Aplikacja stworzona przy użyciu natywnej technologii jest bardziej wydajna pod względem czasowym oraz zużycia procesora w przypadku operacji na bazie danych.
3. Aplikacja stworzona przy użyciu crossplatformowej technologii jest bardziej wydajna pod względem czasowym oraz zużycia procesora w przypadku operacji sortowania liczb.

3. Metoda badawcza

Celem przeprowadzenia badań zaprojektowano i stworzono dwie aplikacje mobilne przeznaczone dla systemu iOS o identycznych funkcjonalnościach przy użyciu technologii Swift i Flutter. Aby porównać wydajność tych aplikacji zdefiniowano poniższe scenariusze badawcze:

- sortowanie 10000, 100000 i 1000000 losowo wygenerowanych liczb całkowitych z zakresu <0,1000> algorytmem Quicksort,
- zapis 10000, 100000 i 1000000 losowo wygenerowanego łańcucha znaków do pliku tekstowego,
- odczyt łańcucha znaków o długości 10000, 100000 i 1000000 z pliku tekstowego,
- zapis 1000, 10000, 100000 rekordów do bazy danych SQLite.
- odczyt 1000, 10000, 100000 rekordów z bazy danych SQLite.

Każdy scenariusz został wykonany 20 razy.

3.1. Środowisko badawcze

Do zbadania czasu wykonania scenariuszy badawczych wykorzystane zostały dwie aplikacje na platformę iOS o tych samych funkcjonalnościach. Aplikacje rejestrowały i wyświetlały czasy wykonania poszczególnych operacji. Ponadto do zbadania zużycia procesora podczas wykonywania operacji użyto narzędzia Activity Monitor. Urządzeniem testowym, na którym zostały przeprowadzone badania był smartfon Apple iPhone 7. Szczegółowa specyfikacja urządzenia została przedstawiona w Tabeli 1.

Tabela 1: Parametry urządzenia mobilnego, na którym przeprowadzono badania

Model	Apple iPhone 7
System operacyjny	iOS 12
Pamięć RAM	2GB
Procesor	Apple A10 Fusion
Liczba rdzeni	4
Taktowanie procesora	2,34GHz
Pamięć wbudowana	32GB

Wyżej opisane urządzenie zostało podłączone kablem typu Lightning do komputera Apple MacBook Air M1 i za jego pomocą zostały zainstalowane poszczególne aplikacje. Połączenie z komputerem zostało również wykorzystane do użycia narzędzia Activity Monitor w celu zmierzenia zużycia procesora podczas wykonywania poszczególnych operacji. W Tabeli 2 zostały przedstawione szczegółowe parametry komputera.

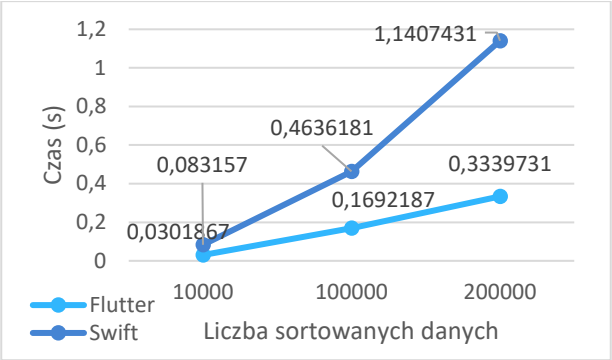
Tabela 2: Parametry komputera wykorzystanego do badań

Model	Apple MacBook Air M1
System operacyjny	macOS Monterey 12.1
Pamięć RAM	16GB
Procesor	Apple M1
Liczba rdzeni	8
Taktowanie procesora	3,2GHz
Pamięć wbudowana	256GB

4. Analiza wyników

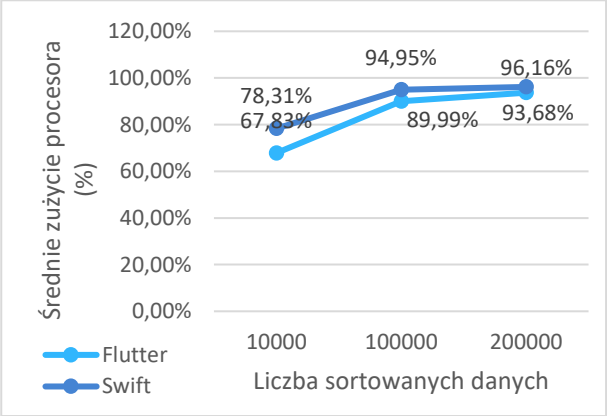
4.1. Sortowanie liczb całkowitych

Na Rysunku 1 przedstawiono średni czas wykonania sortowania w zależności od liczby danych. Można zauważyć, że aplikacja stworzona przy użyciu technologii Flutter wykonała scenariusz sortowania szybciej niż aplikacja stworzona przy użyciu języka Swift. Najmniejszą różnicę czasową uzyskano przy sortowaniu 10000 liczb i wyniosła ona ok. 0,05s, co daje ok. 275% na korzyść aplikacji crossplatformowej, natomiast największa różnica została uzyskana podczas sortowania 200000 liczb i wyniosła 0,8s, co oznacza, że aplikacja natywna była wolniejsza o ok. 341% w porównaniu do aplikacji stworzonej przy użyciu technologii Flutter. Można zauważyć, że różnica czasowa rośnie wraz z liczbą sortowanych liczb.



Rysunek 1: Średni czas sortowania liczb.

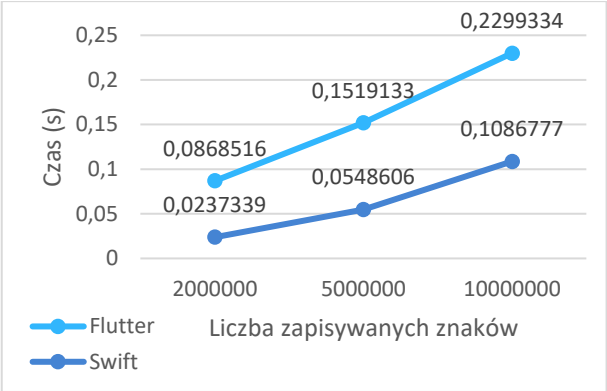
Na Rysunku 2 przedstawiono średnie zużycie procesora podczas sortowania w zależności od ilości liczb. Można zauważyć, że aplikacja crossplatformowa uzyskała niższe procentowe obciążenie jednostki obliczeniowej niż aplikacja natywna. Różnice wyniosły odpowiednio ok. 10,5% w przypadku 10000 liczb, ok. 5% w przypadku 100000 liczb i ok. 2,5% w przypadku 200000 liczb. Można zatem zauważyć, że różnice w zużyciu między aplikacjami maleją wraz ze wzrostem ilości sortowanych liczb.



Rysunek 2: Średnie zużycie procesora podczas sortowania liczb.

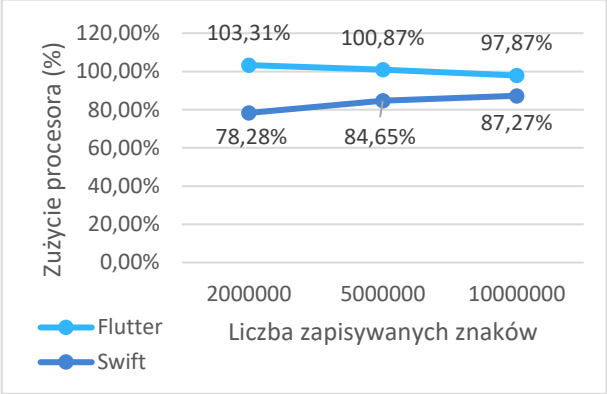
4.2. Zapis znaków do pliku

Na Rysunku 3 przedstawiono średni czas zapisu ciągu znaków do pliku z rozszerzeniem „.txt". Można zauważyć, że aplikacja stworzona przy użyciu technologii natywnej szybciej wykonała ten scenariusz badawczy. Najmniejszą różnicę czasową uzyskano podczas zapisu 2000000 znaków i wyniosła ona ok. 0,043s, co daje ok. 365% na korzyść aplikacji natywnej, natomiast największa różnica czasowa została uzyskana w przypadku zapisu 10000000 znaków i wyniosła ona 0,12s, co oznacza, że aplikacja crossplatformowa była wolniejsza o ok. 211% w porównaniu do aplikacji stworzonej przy użyciu języka programowania Swift. Można zauważyć, że różnice w czasie wykonania pomiędzy aplikacjami rosną wraz ilością zapisywanych znaków.



Rysunek 3: Średni czas zapisu znaków do pliku.

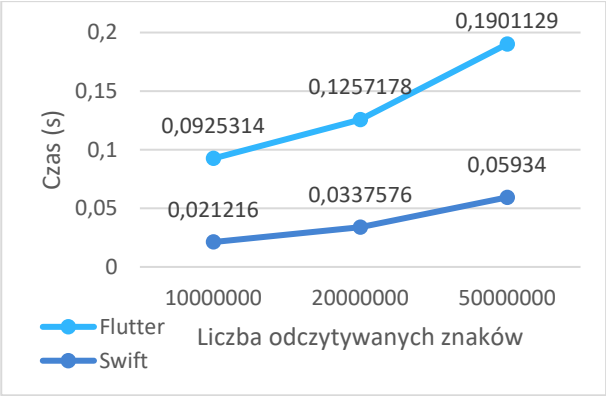
Na Rysunku 4 przedstawiono średnie zużycie procesora podczas zapisu ciągu znaków do pliku. Można zaobserwować, że aplikacja stworzona przy użyciu języka programowania Swift potrzebowała mniej czasu procesora i uzyskała niższe wyniki procentowe obciążenia jednostki obliczeniowej. Różnice w wynikach pomiędzy aplikacjami zmniejszają się wraz ze wzrostem liczby zapisywanych znaków, a największa różnica była przy 2000000 znaków i wyniosła 25%. Warto również podkreślić, że aplikacja crossplatformowa do wykonania zapisu wykorzystała drugi rdzeń procesora o czym świadczy średnie zużycie przekraczające 100%.



Rysunek 4: Średnie zużycie procesora podczas zapisywania znaków do pliku.

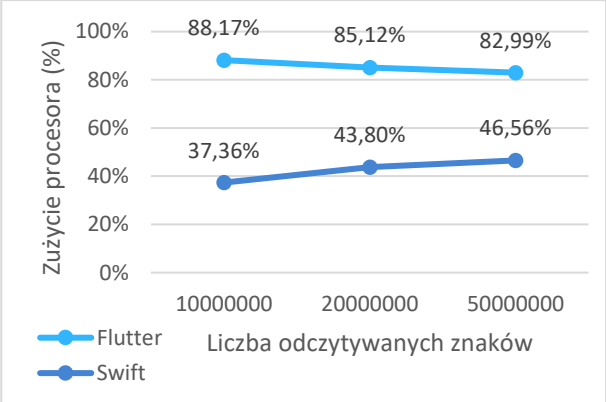
4.3. Odczyt znaków z pliku

Na Rysunku 5 przedstawiono średni czasu odczytu ciągu znaków z pliku o rozszerzeniu „.txt". Można zauważyć, że aplikacja wytworzona przy użyciu technologii natywnej szybciej wykonała odczyt z pliku. Najmniejszą różnicę czasową uzyskano podczas odczytu 10000000 znaków i wyniosła ona ok. 0,07s, co daje ok. 436% na korzyść aplikacji natywnej, natomiast największa różnica czasowa została uzyskana w przypadku 50000000 znaków i wyniosła ok. 0,13s, co oznacza, że aplikacja crossplatformowa była wolniejsza o ok. 320% w porównaniu do aplikacji natywnej. Można zauważyć, że różnice między aplikacjami procentowo maleją wraz ze wzrostem liczby odczytywanych znaków.



Rysunek 5: Średni czasu wykonania odczytu znaków z pliku.

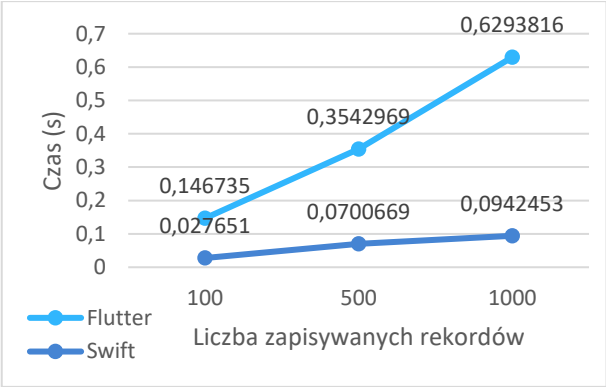
Na Rysunku 6 przedstawiono wykres średniego zużycia procesora podczas odczytu znaków z pliku w zależności od liczby odczytywanych znaków. Można zaobserwować, że aplikacja natywna uzyskała znacznie niższe wyniki zużycia jednostki obliczeniowej. Największą różnicę widać w przypadku odczytu 10000000 znaków i wyniosła ona aż 50%. Dysproporcje w zużyciu procesora między aplikacjami zmniejszają się wraz ze wzrostem liczby odczytywanych znaków.



Rysunek 6: Średnie zużycie procesora podczas odczytu znaków z pliku.

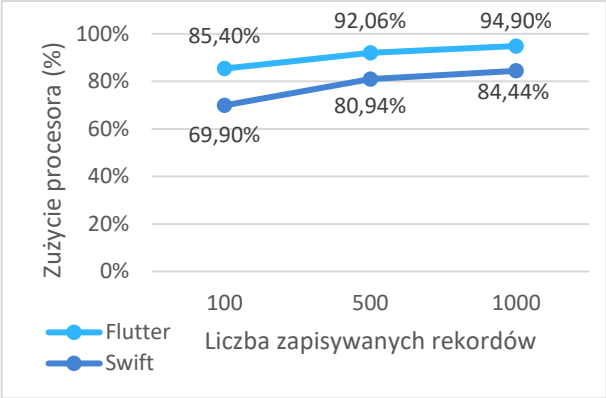
4.4. Zapis rekordów do bazy danych

Na Rysunku 7 przedstawiono średni czas zapisu rekordów do bazy danych SQLite w zależności od ich liczby. Można zaobserwować, że aplikacja zaimplementowana przy użyciu języka programowania Swift szybciej wykonała zapis rekordów do bazy danych. Najmniejszą różnicę czasową uzyskano w przypadku zapisu 100 rekordów i wyniosła ona ok. 0,12s, co daje ok 530% na korzyść aplikacji natywnej, natomiast największa różnica została uzyskana w przypadku 1000 rekordów i wyniosła ok. 0,53s, co oznacza, że aplikacja crossplatformowa była wolniejsza o ok. 667%. Można zauważyć, że różnice w wynikach między aplikacjami rosną wraz ze wzrostem liczby zapisywanych rekordów.



Rysunek 7: Średni czas wykonania zapisu rekordów do bazy danych.

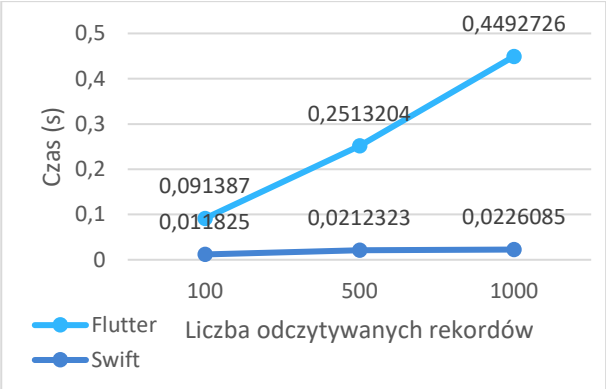
Na Rysunku 8 przedstawiono średnie zużycie procesora podczas zapisu rekordów do bazy danych w zależności od ich liczby. Można zaobserwować, że aplikacja stworzona przy użyciu technologii natywnej uzyskała niższe zużycie procesora dla wszystkich zestawów danych. Największą różnicę widać przy zapisie 100 rekordów i wynosi ona ok. 15%, natomiast najmniejszą różnicę uzyskano w przypadku 1000 rekordów i wyniosła ona ok. 10,5%. Można zauważyć, że dysproporcje wyników pomiędzy aplikacjami zmniejszają się wraz ze wzrostem liczby zapisywanych rekordów.



Rysunek 8: Średnie zużycie procesora podczas zapisu rekordów do bazy danych.

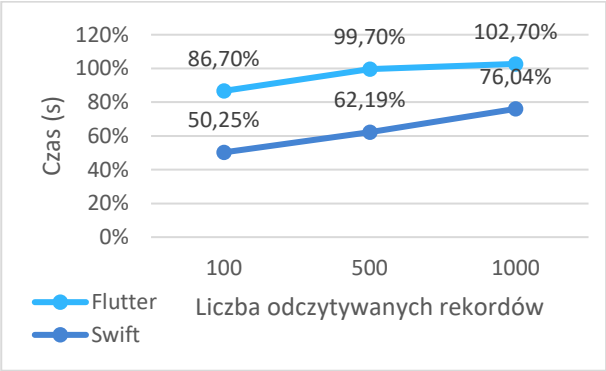
4.5. Odczyt rekordów z bazy danych

Na Rysunku 9 przedstawiono średni czas wykonania odczytu rekordów z bazy danych SQLite w zależności od ich liczby. Można na nim zaobserwować, że aplikacja stworzona przy użyciu technologii natywnej szybciej wykonała ten scenariusz badawczy dla wszystkich zestawów danych. Największą dysproporcję widać podczas odczytu 1000 rekordów i wynosi ona ok. 0,42s, czyli prawie 20 razy tyle ile wyniósł czas odczytu w przypadku aplikacji natywnej. Najmniejszą różnicę czasową uzyskano w przypadku odczytu 100 rekordów i wyniosła ona ok. 0,08s, co oznacza, że aplikacja crossplatformowa była o ok. 772% wolniejsza w porównaniu do aplikacji stworzonej przy użyciu języka programowania Swift. Można zauważyć, że różnice w czasie wykonania rosną wraz ze wzrostem liczby odczytywanych rekordów.



Rysunek 9: Średni czas wykonania odczytu rekordów z bazy danych.

Na Rysunku 10 przedstawiono wykres średniego obciążenia procesora podczas odczytu rekordów z bazy danych SQLite w zależności od ich liczby. Można na nim zauważyć, że aplikacja zaimplementowana przy użyciu języka Swift uzyskała niższe zużycie procesora dla wszystkich zestawów danych. Najmniejszą różnicę uzyskano w przypadku 1000 rekordów i wyniosła ona ok. 27%, natomiast największa różnica została uzyskana w przypadku 500 rekordów i wyniosła ona ok. 37,5%. Można zauważyć, że dysproporcje w wynikach między aplikacjami zmniejszają się wraz z liczbą odczytywanych rekordów.



Rysunek 10: Średnie zużycie procesora podczas odczytu rekordów z bazy danych.

5. Wnioski

Celem niniejszej pracy była analiza porównawcza aplikacji iOS stworzonych przy użyciu technologii natywnej, jaką jest język programowani Swift oraz technologii crossplatformowej, jaką jest Flutter. Aby zrealizować założony cel przeprowadzono badania w ramach przygotowanych wcześniej scenariuszy badawczych, dla których zmierzono czas realizacji oraz zużycie procesora podczas ich wykonywania.

W pierwszym scenariuszu zbadano operację sortowania tablicy liczb całkowitych z zakresu <1, 1000> o określonym rozmiarze. Badanie wykazało, że aplikacja crossplatformowa znacznie szybciej wykonuje operacje sortowania niezależnie od rozmiaru tablicy sortowanych liczb, a przy tym mniej obciąża jednostkę obliczeniową.

Kolejnym badanym scenariuszem był zapis ciągu znaków alfanumerycznych do pliku z rozszerzeniem „txt”. Z przeprowadzonych badań wynikło, że aplikacja natywna dużo szybciej wykonuje zapis do pliku na dysku, a przy tym znacznie mniej obciąża procesor. W przypadku aplikacji stworzonej przy użyciu technologii Flutter został zaangażowany drugi rdzeń procesora, a pomimo tego czas wykonania operacji był większy niż w aplikacji natywnej.

Aplikacja stworzona przy użyciu języka Swift okazała się szybsza i bardziej wydajna również przy kolejnym scenariuszu, jakim był odczyt ciągu znaków alfanumerycznych z pliku o rozszerzeniu „.txt”. Po przeanalizowaniu wyników stwierdzono, że technologia natywna radzi sobie lepiej z operacjami na plikach zarówno pod względem czasu wykonania i obciążenia procesora.

Ostatnie dwa scenariusze dotyczyły zapisu i odczytu rekordów z bazy danych SQLite. Z wykonanych badań wynikło, że aplikacja natywna szybciej wykonuje operacje na bazie danych, a przy tym mniej obciąża jednostkę obliczeniową.

Na podstawie przeprowadzonych badań udało się potwierdzić wszystkie postawione hipotezy. Aplikacja natywna zaimplementowana przy użyciu języka programowania Swift okazała się bardziej wydajna pod względem szybkości wykonania oraz zużycia procesora zarówno w przypadku operacji na plikach, jak i operacji na bazie danych, natomiast aplikacja crossplatformowa stworzona przy użyciu technologii Flutter uzyskała lepsze wyniki czasowe oraz mniej obciążała procesor w przypadku operacji sortowania liczb całkowitych.

Z przeprowadzonych badań wynika, że zastosowanie odpowiedniej technologii powinno być dostosowane do funkcjonalności implementowanej aplikacji.

Literatura

[1] M. Napoli, Beginning Flutter: A Hands On Guide to App Development, John Wiley & Sons, 2019

[2] Mobile & Tablet Operating System Market Share Worldwide https://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#daily-20220107-20220107-bar [dostęp: 08.01.2022]

[3] Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021 "https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/" [dostęp: 10.01.2022]

[4] Fuchsia OS Official Site https://fuchsia.dev/ [dostęp: 08.01.2022]

[5] M. Olsson, A Comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development, Blekinge Institute of Technology, 2020.

[6] D. Gałan, K. Fisz, P. Kopniak, A multi-criteria comparison of mobile applications built with the use of Android and Flutter Software Development Kits, Journal of Computer Sciences Institute 19 (2021) 107-113. https://doi.org/10.35784/jcsi.2614

[7] P. Kotarski, K. Śledź, J. Smołka, Analysis of the impact of development tools used on the performance of the mobile application, Journal of Computer Sciences Institute 6 (2018) 68-72. https://doi.org/10.35784/jcsi.642

[8] P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik, J. Smołka, Performance analysis of native and cross-platform mobile applications, Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska 7(2) (2017) 50-53. https://doi.org/10.5604/01.3001.0010.4838

[9] D. Dobrzański, W. Zabierowski, The comparison of native apps performance on iOS (Swift) and Android with cross-platform application – Xamarin: student project, International Journal of Microelectronics and Computer Science 8 (2018) 112-116

[10] O. Axelsson, F. Carlström, Evaluation Targeting React Native in Comparison to Native Mobile Development. Ergonomics and Aerosol Technology, LUP Student Papers (2016) 105 http://lup.lub.lu.se/student-papers/search/publication/8886469

[11] M. Rodríguez-Sánchez Guerra, Cross-platform development frameworks for the development of hybrid mobile applications: Implementations and comparative analysis. Escuela superior de ingenieria grado en ingenieria informatika (2018) 86. https://rodin.uca.es/handle/10498/20951