JOURNAL
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:

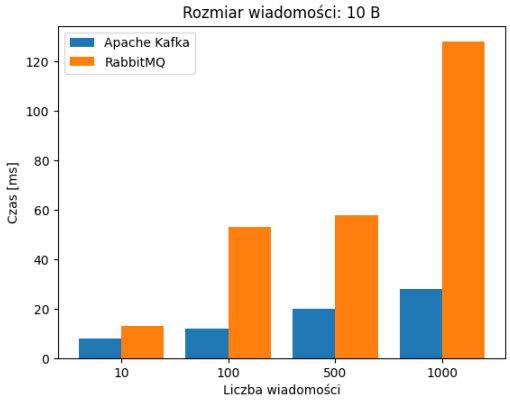
1. Aplikacja pełniąca funkcję wydawcy dla Apache Kafka.
2. Aplikacja pełniąca funkcję subskrybenta dla Apache Kafka.
3. Aplikacja pełniąca funkcję wydawcy dla RabbitMQ.
4. Aplikacja pełniąca funkcję subskrybenta dla RabbitMQ.

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

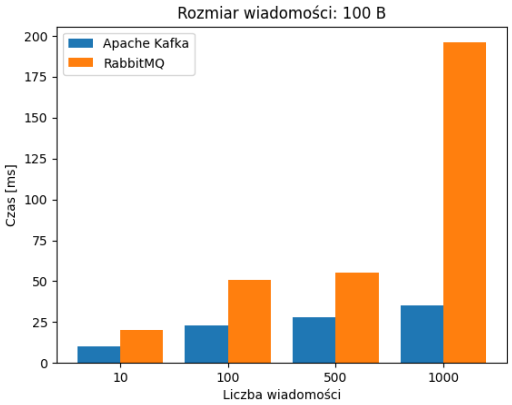
- Java JDK, wersja 11.0.8,
- Spring Boot, wersja: 2.5.

4.1. Badanie wydajności przy użyciu autorskiej aplikacji

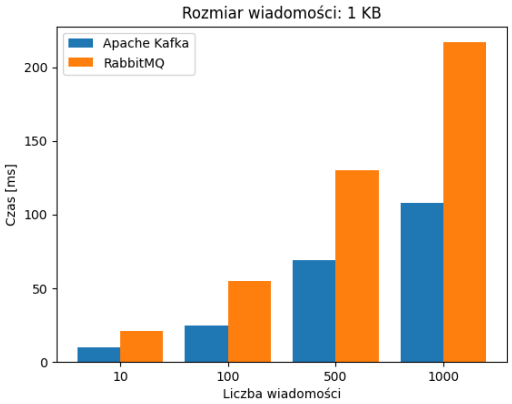
Do przeprowadzania badań wydajności brokerów Apache Kafka oraz RabbitMQ wykorzystano aplikacje producenta i subskrybenta. Badania przeprowadzono dla następujących rozmiarów wiadomości: 10B, 100B, 1KB, 10KB, 100KB, 1MB. Każdą z tych wiadomości przesłano 10 razy, 100 razy, 500 razy oraz 1000 razy. Przed 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.

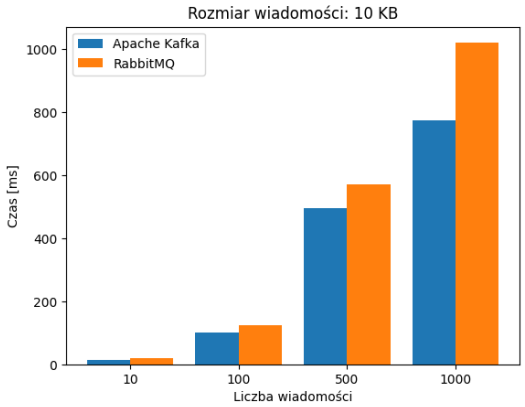


Rysunek 2: Porównanie czasu przesyłania wiadomość o rozmiarze 100B.

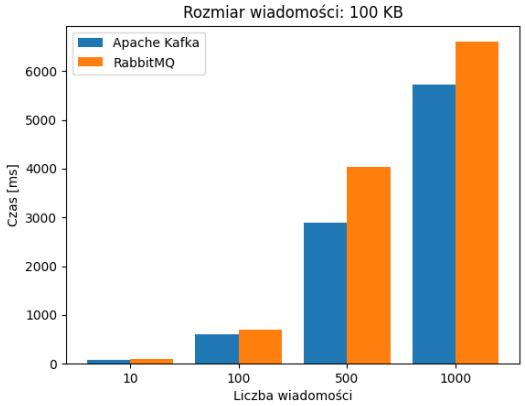


Rysunek 3: Porównanie czasu przesyłania wiadomość o rozmiarze 1KB.

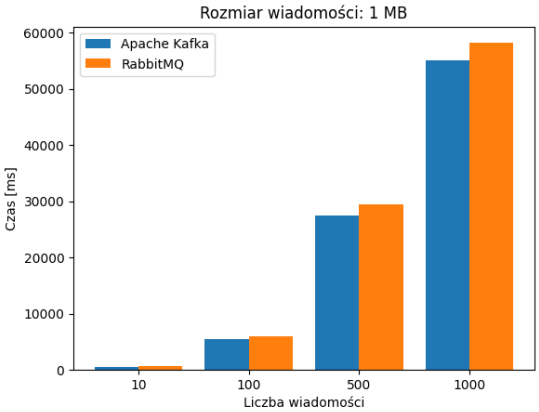
Dla każdej kombinacji rozmiaru oraz liczby przesyłanych wiadomości Apache Kafka posiada lepszą wydajność niż RabbitMQ. Najbardziej zbliżone wyniki (z nieznaczną korzyścią dla Apache Kafka) uzyskano w eksperymencie z plikiem o wielkości 1MB. Wydajność badanych brokerów (liczba przesłanych wiadomości na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

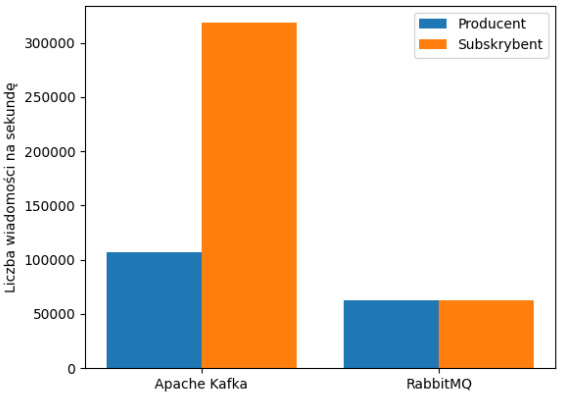
Odchylenie standardowe dla przesłanych wiadomości jest mniejsze dla Apache Kafka. Oznacza to, że wyniki badań wydajnościowych są bardziej zróżnicowane dla RabbitMQ.

4.2. Badanie wydajności przy użyciu pomocniczych narzędzi

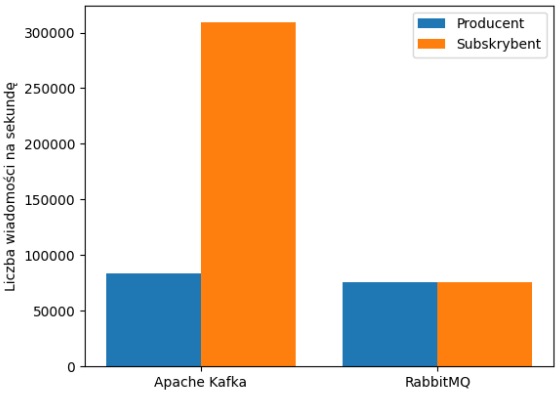
W podrozdziale 4.1 opisano eksperyment wydajnościowy z użyciem autorskich aplikacji. Scharakteryzowane zostały wykonane badania wydajnościowe z użyciem gotowych narzędzi pomocniczych dostępnych dla badanych technologii. Dla Apache Kafka są to skrypty kafka-producer-perf-test oraz kafka-producer-perf-test [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.

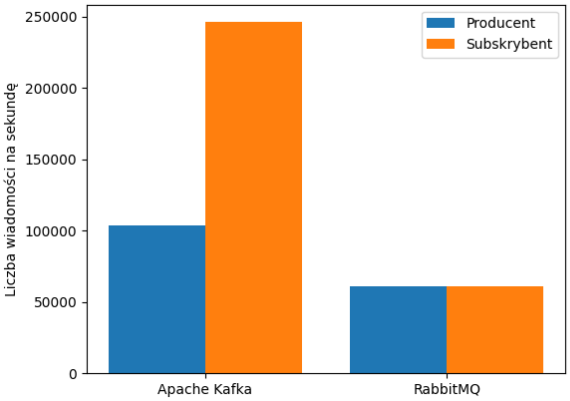


Rysunek 7: Model jeden producent, jeden subskrybent.

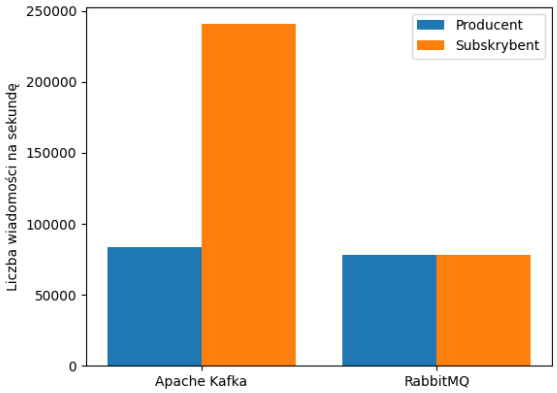


Rysunek 8: Model dwóch producentów, jeden subskrybent.

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].