JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 31 (2024) 116–123
Received: 10 April 2024
Accepted: 8 May 2024

Efficiency comparison of message brokers

Porównanie wydajności brokerów wiadomości

Sebastian Andrzej Dyjach*, Małgorzata Plechawska-Wójcik

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

Abstract
The aim of the article is to compare three main brokers used in the development of web applications: RabbitMQ, Apache Kafka and Apache Pulsar. To conduct the research, a custom application was created to compare two key metrics in the context of message queue performance. These metrics are: latency and number of processed messages per second. The conducted experiments showed that in scenarios requiring processing of backlogged messages by the broker and in cases of minimizing the impact of the SSL protocol on broker performance, Apache Pulsar proved to be the best solution. In the scenario examining message delivery delays, RabbitMQ turned out to be the best tool, while in the case of examining the stability of message processing in real-time, the best results were achieved with Apache Kafka.

Keywords: Apache Kafka; Apache Pulsar; RabbitMQ; message broker

Streszczenie
Celem artykułu jest porównanie trzech głównych brokerów używanych w rozwoju aplikacji internetowych: RabbitMQ, Apache Kafka i Apache Pulsar. Do przeprowadzenia badań została stworzona autorska aplikacja umożliwiająca porównanie dwóch kluczowych metryk w kontekście wydajności kolejek wiadomości. Są nimi opóźnienie oraz liczba wiadomości procesowanych na sekundę. Przeprowadzone eksperymenty wykazały, że w scenariuszach wymagających procesowania zaległych wiadomości przez broker oraz w przypadku minimalizacji wpływu protokołu SSL na wydajność brokera najlepszym rozwiązaniem jest Apache Pulsar. W scenariuszu badającym opóźnienia w dostarczaniu wiadomości najlepszym narzędziem okazał się RabbitMQ, natomiast w przypadku badania stabilności procesowania wiadomości w czasie rzeczywistym, najlepsze rezultaty osiągnęło narzędzie Apache Kafka.

Słowa kluczowe: Apache Kafka; Apache Pulsar; RabbitMQ; broker wiadomości

*Corresponding author
Email address: sebastian.dyjach@pollub.edu.pl (S.A Dyjach)

1. Wstęp

Współczesne aplikacje internetowe zawierają coraz bardziej zawiłe wymagania biznesowe. Klienci serwisów internetowych wymagają coraz bardziej spersonalizowanych usług, a właściciele platform chcą obserwować coraz więcej metryk związanych z zachowaniami klientów. Skuteczna realizacja wymagań biznesowych jest osiągalna poprzez przetwarzanie oraz analizę ogromnych ilości danych. Wymaga to nie tylko ich sprawnego przechowywania, lecz również szybkiego i efektywnego pozyskiwania danych.

W systemach, w których pozyskiwanie oraz przetwarzanie danych staje się kwestią kluczową, dużą popularnością cieszy się model *publish-subscribe*, którego implementacją są brokery wiadomości, pozwalając na efektywne przetwarzanie danych. W kontekście wszechobecności danych, wybór odpowiedniego brokera staje się nie tylko kwestią techniczną, lecz również strategiczną decyzją, wpływającą bezpośrednio na sukces organizacji. Skuteczność przetrzymywania i dostarczania danych może przekształcić się w konkurencyjną przewagę, podkreślając zatem konieczność starannego dobrania konfiguracji brokera do specyfiki i wymagań biznesowych. Niniejszy artykuł jest poświęcony analizie oraz porównaniu różnych brokerów wiadomości, mając na uwadze ich zdolność do obsługi rosnącej liczby danych, a także optymalizację procesów przetwarzania informacji.

Analizie zostały poddane trzy najbardziej popularne brokery wiadomości: Apache Kafka, Apache Pulsar oraz RabbitMQ. Celem artykułu jest wysnucie wniosków dotyczących doboru brokera w zależności od specyfiki systemu, w jakim ma zostać użyty.

2. Zakres badań

W celu przeprowadzenia eksperymentów pozwalających na porównanie poszczególnych narzędzi została wykonana aplikacja w szkielecie programistycznym Spring Boot [1] pozwalająca na badanie omawianych narzędzi w tym samym czasie. Umożliwia ona zbieranie danych pozwalających na bezpośrednie porównanie opóźnienia oraz przepustowości pomiędzy badanymi narzędziami. Badania obejmą dwa różne sposoby procesowania danych: procesowanie wcześniej nagromadzonych danych w brokerze oraz badań w czasie rzeczywistym, gdzie dane będą wysyłane do brokera tylko w przypadku dostępności konsumenta danych wiadomości.

Metryką, która pozwala na oszacowanie opóźnienia w dostarczeniu wiadomości do brokera jest czas pomiędzy wysłaniem wiadomości przez producenta do otrzymania danej wiadomości przez konsumenta. Porównanie przepustowości pomiędzy konkretnymi brokerami w przypadku procesowania danych wysłanych do brokera przed jego włączeniem jest możliwe dzięki pomiarowi czasu, jaki jest potrzebny na procesowanie danej

porcji danych. Jest on liczony od momentu wysłania badanej ilości danych do brokera, do średniego czasu w jakim dana wiadomość została przeprocesowana. W przypadkach, gdy była badana przepustowość procesowania w czasie rzeczywistym, jej definicja została określona jako liczba wiadomości procesowanych na sekundę, dla których wartości opóźnienia nie wzrastały wraz z upływem czasu. Aplikacja korzysta z platformy spring-cloud-binder [2], która oferuje mechanizmy partycjonowania ułatwiające budowanie aplikacji chmurowych, w związku z tym pojęcie partycji w badaniach przedstawionych w artykule określane jest jako partycja w elemencie binder narzędzia Spring Cloud.

3. Analiza literatury

W dostępnych źródłach istnieje szereg artykułów naukowych poruszających tematykę porównania właściwości brokerów wiadomości. Zajmują się one zarówno teoretycznym porównaniem budowy poszczególnych narzędzi, jak i przedstawiają eksperymenty badawcze, które pozwalają na bezpośrednie porównanie ich osiągów.

W pracy [3] zostało przedstawione porównanie wydajnościowe narzędzi RabbitMQ, Apache Kafka oraz Apache Pulsar w zależności od liczby partycji, stosunku jednoczesnych producentów do konsumentów oraz wielkości wiadomości. Autor przedstawia szereg badań, które porównują wydajność omawianych narzędzi. Z przedstawionych przez autora danych wynika, że przy dużej liczbie partycji wzrastają opóźnienia dla narzędzia Apache Kafka, w przeciwieństwie do Apache Puslar, dla którego zwiększenie liczby partycji przyniosło wzrost wydajnościowy w ujęciu procesowanych wiadomości na sekundę, bez ponoszenia kosztów w postaci większego opóźnienia. W porównaniu przepustowości w zależności od liczby partycji, średnio cztery razy większą przepustowość wykazywała Apache Kafka w porównaniu do pozostałych omawianych narzędzi, jednak wraz ze wzrostem liczby partycji jej przepustowość spadała. W badaniu wartością, dla której można było obserwować spadek przepustowości, była wartość stu partycji.

Zagadnienie zostało poruszone także w artykule [4]. Autorzy poruszają tam aspekty architektoniczne poszczególnych brokerów wiadomości, wpływających na konkretne aspekty ich funkcjonowania. Wskazują na podobieństwo w osiąganych wartościach opóźnień dla obu narzędzi, podkreślając jednak większy wzrost opóźnień wiadomości dla Apache Kafka w przypadku dużej intensywności otrzymywanych wiadomości, gdzie w RabbitMQ zwiększenie częstotliwości wysyłania nie wpływa bezpośrednio na opóźnienie. Według autora, Apache Kafka jest w stanie osiągać dużo lepsze wyniki procesowania dzięki sekwencyjnemu dyskowi zapisu oraz systemu cache stosowanego bezpośrednio na dysku. Podobne rozważania od strony teoretycznej są przeprowadzone w [5]. Z artykułu wynika, że Apache Kafka osiąga lepsze wyniki wydajnościowe od RabbitMQ dzięki dyskowi sekwencyjnemu przeznaczonemu do operacji wejścia-wyjścia. Autor porównuje także zużycie przestrzeni dyskowej obu narzędzi, wskazując na większe zużycie zasobów dla narzędzia RabbitMQ w badanej konfiguracji.

Istnieją publikacje naukowe, które podejmują analizę wpływu konkretnych parametrów narzędzi Apache Kafka oraz RabbitMQ w środowisku chmurowym. W [6] przedstawione są eksperymenty pozwalające na zbadanie faktycznego wpływu właściwości kolejek na ich wydajność. Przedstawiony jest wpływ liczby kolejek na wydajność RabbitMQ podczas używania tego narzędzia w trybie bez mechanizmu routingu, zwanego *fanout exchange*. W tej konfiguracji narzędzie wykazuje spadek wydajności powyżej dwudziestu utworzonych równolegle kolejek. Artykuł przedstawia także wpływ liczby utworzonych kolejek oraz używanego protokołu potwierdzania na zużycie procesora. Zauważalny wzrost zużycia *CPU* występował dla czterdziestu utworzonych kolejek w modelu dostarczania *at-least-once*, gdzie wiadomość ma zostać dostarczona jeden lub więcej razy. W omawianym artykule, dla brokera Apache Kafka przedstawiony jest wpływ parametru *batch-size* na wydajność kolejki. Broker uzyskuje największą wydajność dla wartości parametru batch-size równemu 80kB. Autor odnosi się także do wartości osiąganych dla poszczególnych brokerów wskazując, że większe zużycie procesora oraz wahania w alokowanych zasobach częściej występują w narzędziu Apache Kafka.

W pracy [7] autor zajmuje się porównaniem rozwiązywania kwestii przepustowości i opóźnień w narzędziach RabbitMQ oraz Apache Kafka. Badanie wydajności zostało przeprowadzone z użyciem narzędzia OpenMessaging Benchmark [8], które pozwala na przeprowadzenie testów poszczególnych narzędzi z użyciem skryptów Terraform bądź Helm Chartów. Według przedstawionych wyników, Apache Kafka osiąga zdecydowanie lepsze rezultaty zarówno pod względem przepustowości, która została zdefiniowana tak samo jak w pierwszej omawianej pozycji literatury, jak i wykazuje mniejsze opóźnienia w dostarczaniu wiadomości. Umożliwia również większą swobodę w skalowaniu horyzontalnym narzędzia, przez co lepiej sprawdza się dla systemów dużej wielkości.

Część publikacji naukowych została poświęcona analizie narzędzi typu Java Message Service (JMS). W publikacji [9] autorzy analizują architekturę wysyłania oraz otrzymywania wiadomości w JMS, następnie wskazując konkretne przypadki użycia dla tego typu kolejek. Z analizy wynika, iż kolejki takie jak np. RabbitMQ są kolejkami ogólnego przeznaczenia służącymi do procesowania wiadomości z niską częstotliwością wysyłanych komunikatów. W celu maksymalizacji wydajności oraz konieczności skalowania ruchu autorzy wskazują Apache Kafka jako narzędzie, które powinno być użyte zamiast kolejek typu JMS.

W pracy [10] autor zajmuje się porównaniem narzędzi RabbitMQ i Apache Kafka. Autor przedstawia wyniki badań, które są uzyskane na podstawie narzędzia OpenMessaging Benchmark [8]. Według badań Apache Kafka przewyższa RabbitMQ pod względem przepustowości w scenariuszu czterech producentów - czterech konsumentów. Mniejsze opóźnienie dostarczenia

wiadomości osiąga narzędzie RabbitMQ, jednak oba narzędzia uzyskują opóźnienia nie przekraczające kilku milisekund.

4. Charakterystyki brokerów wiadomości

W tym rozdziale zostanie wykonany przegląd brokerów wiadomości, które będą poddawanie badaniom, czyli Apache Kafka, Apache Pulsar oraz RabbitMQ. Zostały przedstawione ich główne właściwości w celu wyłonienia różnic, które następnie posłużą do opracowania scenariuszy badawczych.

4.1. Apache Kafka

Jednym z najpopularniejszych narzędzi typu publish-subscribe jest Apache Kafka [11]. Narzędzie zostało opracowane w języku Scala, pierwotnie stosowane jako rozwiązanie na platformie Linkedin [12], a następnie opublikowane jako narzędzie open-source pod koniec 2010 roku.

Jej powstanie wynikło z konieczności rozszerzenia systemu monitoringu w systemie, co prowadziło do znacznego skomplikowania architektury [3]. W budowie tego narzędzia kluczową rolę odgrywa broker, który zajmuje się obsługą całej komunikacji wysyłania jak i otrzymywania wiadomości. Dzięki brokerowi istnieje możliwość skalowania horyzontalnego w zależności od obciążenia, co ułatwia zarządzaniem w przypadku zmian natężenia ruchu w systemie. Jeden z brokerów jest liderem, który odpowiada za wszystkie operacje odczytu i zapisu danych, a pozostałe wykonują replikacje lidera. Broker jest bezstanowy, co oznacza, iż potrzebuje narzędzia, które będzie służyło dla niego jako źródło metadanych. Serwisami, które mogą służyć brokerowi do tego celu są zookeeper oraz Kafka Raft [13].

4.2. RabbitMQ

RabbitMQ [14] to otwarto-źródłowy broker wiadomości opracowany przez Rabbit Technologies i obecnie utrzymywany przez Pivotal Software. Jest implementacją protokołu Advanced Message Queuing Protocol (AMQP) w języku Erlang, który jest protokołem do komunikacji opartej na wiadomościach, która polega na kolejkowaniu wiadomości pochodzących z wymian, przechowywaniu ich i dostarczaniu do konsumentów.

Wiadomości są publikowane do wymian (ang. *exchange*), które następnie rozprowadzają kopie wiadomości do kolejek, korzystając z reguł zwanych powiązaniami (ang. *bindings*). Następnie broker dostarcza wiadomości do konsumentów zapisanych do kolejek, bądź konsumenci pobierają wiadomości z kolejek na żądanie. Ścieżka publikacji od wydawcy do konsumenta poprzez wymianę i kolejkę jest podobna do Apache Pulsar oraz Apache Kafki, jednak różni się mechanizmem wewnętrznym działania brokera, czyli wykonywaniem przekierowywań przez wymianę, a nie temat, a następnie rozsyłaniem danych za pomocą kolejki.

4.3. Apache Pulsar

Apache Pulsar [15] jest narzędziem open-source stworzonym przez Yahoo w roku 2013. Jego głównym celem było rozwiązanie problemów z wydajnością oraz skalowalnością, które występowały podczas korzystania z Java Messaging Service (JMS). W założeniu miał stanowić połączenie łączące najlepsze cechy kolejek wiadomości z funkcjonalnościami umożliwiającym przetwarzanie potoków danych.

Jedną z najbardziej widocznych różnic funkcjonalnych w stosunku do innych kolejek wiadomości jest ustrukturyzowanie danych w brokerze. W każdym brokerze należy stworzyć tenant, dla którego tworzone są konkretne przestrzenie, wewnątrz których gromadzone są tematy. Tenanty [16] to grupy użytkowników korzystające z tego samego widoku aplikacji. Widok ten obejmuje dane do których mają dostęp, konfiguracje poszczególnych komponentów oraz funkcjonalności. W wyniku takiego rozwiązania bardzo prosto można wdrożyć taki system do podejścia *multi-tenant*, gdzie aplikacja na jednym środowisku obsługuje wielu klientów. Ułatwia również wprowadzanie osobnych zabezpieczeń dla każdego klienta, a także narzuca spójne ustrukturyzowanie danych. Apache Pulsar, podobnie jak Apache Kafka wspiera również natywnie koncept partycji, co pozwala na skalowanie horyzontalne w obrębie danego tematu.

Struktura omawianego brokera składa się z dwóch dodatkowych elementów potrzebnych do ich prawidłowego funkcjonowania, którymi są *zookeeper* oraz *bookeeper*. Odpowiadają one odpowiednio za zarządzanie metadanymi, podobnie jak w narzędziu Apache Kafka oraz zarządzaniem danymi w brokerze. Dane przechowywane w brokerze są przechowywane w serwisach zwanymi *bookie,* których synchronizacja jest realizowana poprzez bookeepera. Taka struktura zapewnia wydajne skalowanie oraz szybki powrót ich dostępności w przypadku awarii.

5. Scenariusze badawcze

W celu przeprowadzenia porównania poszczególnych aspektów badanych narzędzi zostały stworzone scenariusze badawcze, które pozwolą na porównanie poszczególnych aspektów badanych narzędzi.

5.1. Badanie wydajności procesowania wcześniej zgromadzonych danych.

Przedstawiany scenariusz badawczy ma za zadanie zbadanie wydajności poszczególnych narzędzi w przypadku, gdy w brokerach zostały nagromadzone dane przed uruchomieniem jego konsumenta, które następnie będą przez niego procesowane. W badanym scenariuszu do brokera zostanie wysłanych 10000 wiadomości, których rozmiar jest stopniowo zwiększany: 100B, 10kB, 30kB, 100kB, aż do wielkości 300kB. W przedstawionym scenariuszu, w momencie wysyłania danych do brokera konsument odpowiadający za pobranie wiadomości jest wyłączony do czasu, kiedy ostatnia wiadomość zostanie wysłana do brokera. W tym momencie konsument zostaje włączony, a następnie mierzony jest średni czas od wysłania wiadomości przez producenta do otrzymania wiadomości przez konsumenta. Metryki zgromadzone w aplikacji są gromadzone w bazie danych *PostgreSQL*.

5.2. Badanie opóźnień dostarczania wiadomości w aplikacji

Niniejszy scenariusz badawczy ma za zadanie przedstawić wartości opóźnień dla każdego brokera wiadomości. W tym celu po poprawnym uruchomieniu producenta oraz konsumenta dla danego brokera, wysyłane były do niego wiadomości z określoną częstotliwością 40 wiadomości na sekundę. Następnie przy przetwarzaniu wiadomości przez konsumenta zapisywany był czas przetworzenia poszczególnej wiadomości, co pozwalało na zebranie metryk związanych z opóźnieniem oraz stabilnością brokera, takimi jak badanie skrajnych wartości opóźnień dla badanych narzędzi.

5.3. Badanie maksymalnej przepustowości narzędzia.

Kolejny ze scenariuszy ma za zadanie znaleźć maksymalną stabilną częstotliwość wysyłanych wiadomości w brokerze, dla której nie występował wzrost opóźnień w dostarczaniu wiadomości dla danego brokera wraz z upływem czasu. Badaniu zostały poddane dwie wielkości wiadomości – 100B oraz 10kB. Dla obu rozmiarów wiadomości zostały przyjęte inne kryteria jakości procesowania. Dla badania wiadomości o rozmiarze 100B opóźnienie w dostarczaniu wiadomości nie mogło przekroczyć 200ms, natomiast dla rozmiaru 10kB opóźnienie w dostarczaniu wiadomości nie mogło przekroczyć jednej sekundy.

5.4. Wpływ zastosowania protokołu szyfrowania na wydajność brokera

Scenariusz badający wpływ zastosowania protokołu szyfrowania na jakość procesowania pokrywał się ze scenariuszami przedstawionymi w punktach 5.1 oraz 5.2, jednak w tym wypadku broker działał ze skonfigurowanym protokołem szyfrowania. Dla każdego brokera została skonfigurowana metoda uwierzytelniania magazynami kluczy *keystore* oraz *truststore* z użyciem formatu JKS, który został utworzony przez firmę Oracle specjalnie na potrzeby języka Java [17]. Klucze, które zostały utworzone na potrzeby uwierzytelnienia w brokerze, zostały podpisane certyfikatem z podpisem własnym.

6. Wyniki przeprowadzonych scenariuszy badawczych

W niniejszym rozdziale zaprezentowano wyniki badań uzyskanych na podstawie przedstawionej metody badań dla scenariuszy badawczych.

6.1. Analiza wyników scenariusza badania wydajności zgromadzonych wcześniej danych

W Tabelach 1-3 zostały przedstawione wyniki dla scenariusza badania wydajności brokera w przypadku procesowania zaległych danych. W Tabeli 1 zostały przedstawione wyniki procesowania dla rozmiaru 100B. Najlepsze rezultaty procesowania dla tej wielkości wiadomości zostały odnotowane dla narzędzia Apache Pulsar. Dla pięćdziesięciu równolegle procesujących konsumentów uzyskał on czas procesowania 19 sekund, przy najlepszych rezultatach 25 sekund dla narzędzia RabbitMQ oraz 27 sekund dla narzędzia Apache Kafka.

Tabela 1: Wynik dla scenariusza badawczego 5.1 dla rozmiaru wiadomości 100B. Źródło: Opracowanie własne

Nazwa brokera	Liczba konsumentów (lp.)	Średnia (ms)
KAFKA	1	88,673.82
PULSAR	1	98,076.1
RABBIT	1	98,134.9
KAFKA	5	27,210.5
PULSAR	5	21,551.19
RABBIT	5	25,439.09
KAFKA	25	32,081.31
PULSAR	25	21,361.96
RABBIT	25	32,517.41
KAFKA	50	28,297.27
PULSAR	50	19,406.45
RABBIT	50	29,842.04
KAFKA	100	32,491.08
PULSAR	100	21,193.96
RABBIT	100	32,587.25

W Tabeli 2 zostały przedstawione wyniki eksperymentu dla rozmiaru wiadomości 10kB. Najlepszy rezultat procesowania został uzyskany przez Apache Pulsar i wyniósł 34 sekundy, przy 49 sekundach dla narzędzia Apache Kafka oraz 51 sekund dla narzędzia RabbitMQ.

Tabela 2: Wynik dla scenariusza badawczego 5.1 dla rozmiaru wiadomości 10kB. Źródło: Opracowanie własne

Nazwa brokera	Liczba konsumentów (lp.)	Średnia (ms)
KAFKA	1	157252.50
PULSAR	1	156557.17
RABBIT	1	159766.79
KAFKA	5	49260.76
PULSAR	5	40135.91
RABBIT	5	51974.70
KAFKA	25	50556.32
PULSAR	25	34611.28
RABBIT	25	61132.81
KAFKA	50	55535.96
PULSAR	50	37301.71

RABBIT	50	65082.18

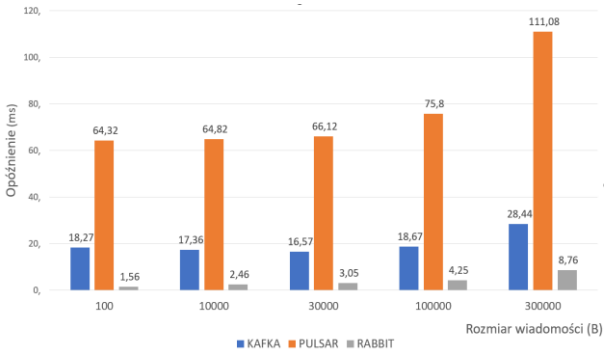
W Tabeli 3 zostały przedstawione rezultaty dla ostatniego badanego rozmiaru wiadomości, tj. 50kB. Ponownie najlepsze rezultaty osiągnął broker Apache Pulsar, osiągając czas procesowania 31 sekund dla 75 równocześnie przetwarzających wątków konsumenckich.

Tabela 3: Wynik dla scenariusza badawczego 5.1 dla rozmiaru wiadomości 50kB. Źródło: Opracowanie własne

Nazwa brokera	Liczba konsumentów (lp.)	Średnia (ms)
KAFKA	1	203077.76
PULSAR	1	191825.60
RABBIT	1	199929.27
KAFKA	5	75761.31
PULSAR	5	58928.77
RABBIT	5	73256.71
KAFKA	25	69158.70
PULSAR	25	37422.78
RABBIT	25	91072.26
KAFKA	50	61464.26
PULSAR	50	32776.03
RABBIT	50	86851.33
KAFKA	75	83750.76
PULSAR	75	31883.64
RABBIT	75	80850.79

6.2. Wpływ zastosowania protokołu szyfrowania na wydajność brokera

Na Rysunku 1 przedstawione są wartości opóźnień dla procesowania wiadomości o badanych rozmiarach. W badanym scenariuszu zdecydowanie najlepsze rezultaty osiągał broker RabbitMQ, którego średnie opóźnienie w procesowaniu wiadomości dla rozmiaru 100kB nie przekraczało 5ms, osiągając wartość 9ms dla rozmiaru 300kB. Nieco gorsze rezultaty zostały osiągnięte dla brokera Apache Kafka, gdzie średnie wartości opóźnień stabilizowały się na poziomie 20ms, z wyjątkiem największej badanej wielkości wiadomości 300kB, gdzie średnie opóźnienie w dostarczeniu wyniosło 30ms. Zdecydowanie najgorsze rezultaty w tym eksperymencie osiągał broker Apache Pulsar, który uzyskiwał opóźnienie rzędu 30ms dla wiadomości o rozmiarach do 30kB, następnie wzrastając do wartości 75ms dla wiadomości 50 kB, ostatecznie osiągając opóźnienie 111ms dla rozmiaru 100kB.

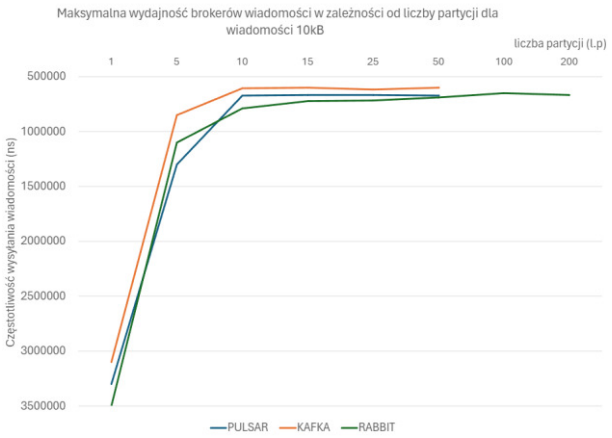


Rysunek 1: Wykres przedstawiający zależność wartości opóźnienia od rozmiaru procesowanej wiadomości dla poszczególnych brokerów. Źródło: Opracowanie własne.

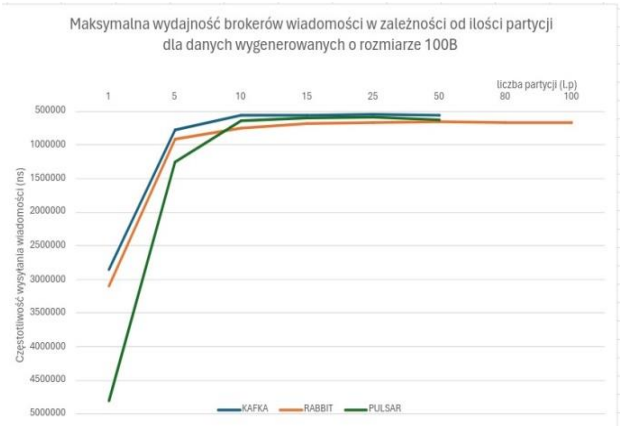
6.3. Rezultaty badania maksymalnej przepustowości narzędzia.

Na Rysunku 2 przedstawione są wyniki scenariusza badającego stabilną przepustowość danego narzędzia dla rozmiaru 10kB. Przedstawione są na w nim wartości częstotliwości wysyłania wiadomości do danego brokera, wyrażone w nanosekundach, dla których broker spełnia warunki stabilności określone dla eksperymentu. Ze zgromadzonych danych wynika, iż najlepszą przepustowość dla procesowania w czasie rzeczywistym posiada narzędzie Apache Kafka, którego maksymalna wydajność w tym scenariuszu wyniosła 1670 wiadomości na sekundę, w porównaniu do 1490 wiadomości dla Apache Pulsar oraz 1520 dla narzędzia RabbitMQ. Największa wydajność procesowania dla narzędzia Apache Kafka wystąpiła przy równoległym procesowaniu z piętnastu partycji, dla narzędzia Apache Pulsar dwudziestu pięciu partycji, natomiast najlepszy rezultat dla RabbitMQ wystąpił przy utworzeniu stu równoległych kolejek.

Podobne rezultaty zostały wykazane dla tego eksperymentu przeprowadzonego dla rozmiaru wiadomości 100B, którego wyniki przedstawione są na Rysunku 3. Dla mniejszej liczby partycji gorsze wyniki procesowania zostały uzyskane dla narzędzia Apache Pulsar. Wraz ze wzrostem partycji na danym temacie, tj. od dziesięciu partycji broker stabilizuje się na poziomie 1640 wiadomości na sekundę, co jest gorszym rezultatem od Apache Kafka (maksymalnie 1820/s), oraz lepszym od RabbitMQ (do 1560 wiadomości/s). W ujęciu procentowym, zwiększenie rozmiaru wiadomości spowodowało zmniejszenie wydajności maksymalnego stabilnego procesowania o około 10% dla Apache Kafka oraz Apache Pulsar, przy około 8% dla narzędzia RabbitMQ.



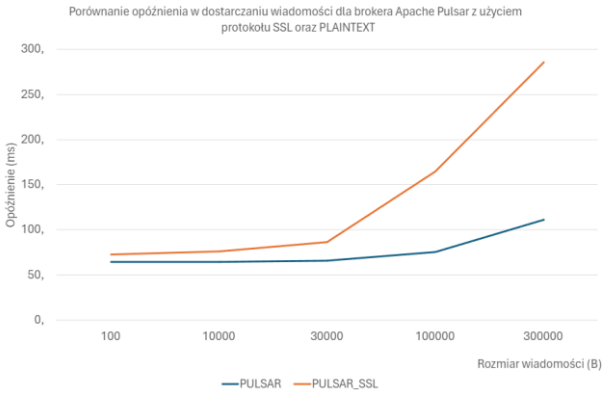
Rysunek *2*: Wykres przedstawiający maksymalne częstotliwości dla brokerów dla których procesowanie odbywało się stabilnie.
Źródło: Opracowanie własne.



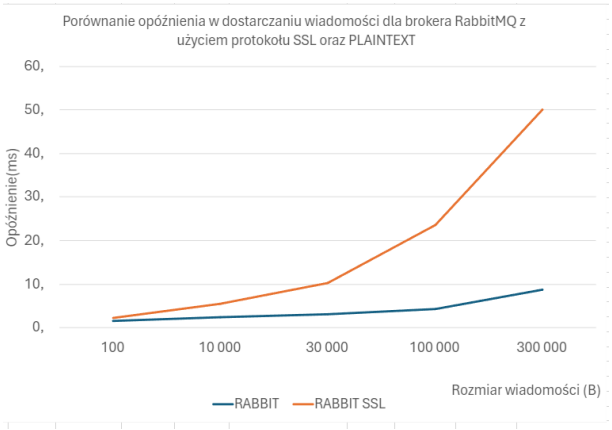
Rysunek 3: Wykres przedstawiający maksymalne częstotliwości dla brokerów dla których procesowanie odbywało się stabilnie.
Źródło: Opracowanie własne.

6.4. Rezultaty wpływu protokołu szyfrowania na parametry wydajności brokera.

Na Rysunkach 4-6 został przedstawiony wpływ zastosowania protokołu szyfrowania na opóźnienia osiągane w brokerach. W przypadku wszystkich omawianych narzędzi wraz ze wzrostem rozmiaru wiadomości zwiększa się różnica w opóźnieniu osiąganym przy zastosowania protokołu szyfrowania w stosunku do wyników uzyskiwanych protokołem *PLAINTEXT*. Największe różnice w wartościach opóźnień pomiędzy rezultatami uzyskiwanymi z użyciem protokołu szyfrowania, a standardowym przesyłek danych w brokerze zostały odnotowane dla narzędzia RabbitMQ, dla którego ponad dwukrotny wzrost opóźnienia jest zauważalny dla procesowania wiadomości o rozmiarze 30kB, dochodząc do ponad pięciokrotnej różnicy w wartościach opóźnienia dla wielkości 300kB, gdzie uzyskuje ono wartość rzędu 50 ms. Podobne maksymalne różnice zostały osiągnięte dla narzędzia Apache Kafka, jednak wystąpienie dwukrotnej różnicy w opóźnieniu wystąpiło w tym narzędziu dla wielkości 100kB. W przypadku wiadomości o rozmiarze 100kB wartości opóźnień sięgały 50ms, natomiast dla wartości 300kB dochodziły do 180 ms. Najmniejszy wpływ w ujęciu procentowym na procesowanie z użyciem protokołu SSL był zauważalny dla Apache Pulsar, dla którego wzrost opóźnienia przy zastosowaniu protokołu szyfrowania był niespełna 3-krotny, zbliżając się dla największego badanego rozmiaru wiadomości do granicy 300 ms.



Rysunek 4: Wykres przedstawiający wpływ użycia protokołu SSL na opóźnienia osiągane w brokerze Apache Pulsar. Źródło: Opracowanie własne.

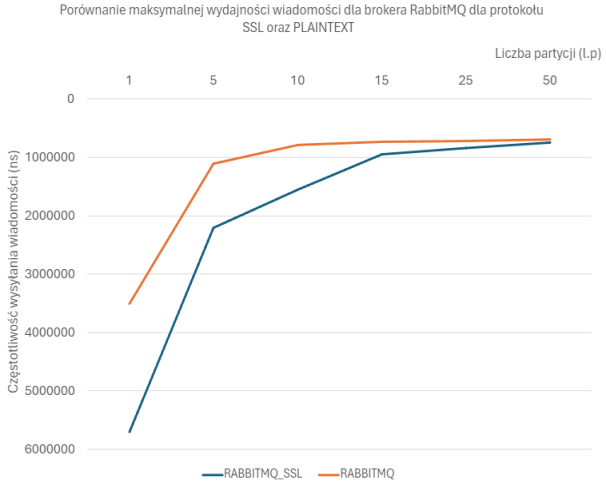


Rysunek 5: Wykres przedstawiający wpływ użycia protokołu SSL na opóźnienia osiągane w brokerze RabbitMQ. Źródło: Opracowanie własne.

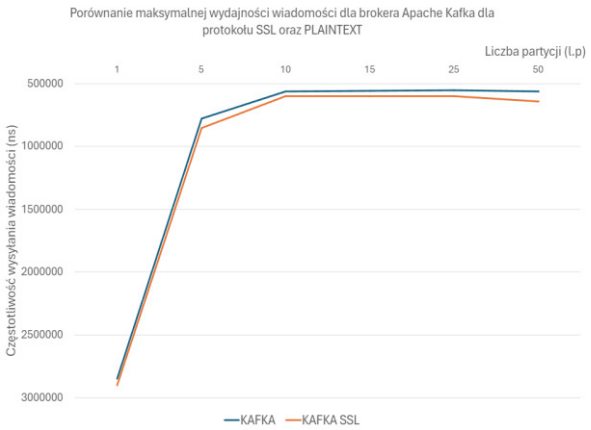
Na Rysunkach 7-9 zostały przedstawione częstotliwości wysyłania wiadomości, dla których konkretny broker zachowywał stabilność z użyciem protokołu *SSL*, w porównaniu do przesyłanych bez szyfrowania protokołem *PLAINTEXT* dla poszczególnych brokerów. Dla brokera Apache Kafka spadek wydajności dla maksymalnej wydajności wynosił 7-8% w stosunku do wiadomości wysyłanych protokołem SSL. Podobny wpływ na maksymalną wydajność procesowania przy zastosowaniu protokołu szyfrującego wystąpił w przypadku narzędzia Apache Pulsar, gdzie wpływ protokołu SSL na maksymalną stabilną częstotliwość procesowania wiadomości wynosił w granicach 5%. W przypadku RabbitMQ różnice dla mniejszej liczby partycji są zdecydowanie bardziej zauważalne. Wraz ze wzrostem liczby

równolegle procesujących konsumentów, dla których broker jest w stanie stabilnie procesować wiadomości różnica pomiędzy wynikami osiąganymi z użyciem protokołu SSL oraz bez niego zmniejsza się, w najwydajniejszym scenariuszu dla pięćdziesięciu partycji osiągając różnice w procesowaniu w granicach 7%.

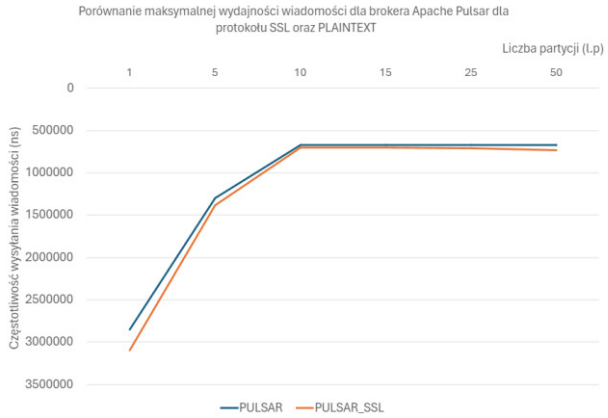
Rysunek 6: Wykres przedstawiający wpływ użycia protokołu SSL na opóźnienia osiągane w brokerze Apache Kafka. Źródło: Opracowanie własne.



Rysunek 7: Przedstawienie wpływu SSL na maksymalną wydajność brokera RabbitMQ. Źródło: Opracowanie własne.



Rysunek 8: Wpływu zastosowanie SSL na maksymalną wydajność brokera Apache Kafka. Źródło: Opracowanie własne.



Rysunek 9: Wpływu zastosowanie SSL na maksymalną wydajność brokera Apache Pulsar. Źródło: Opracowanie własne.

7. Wnioski

Na podstawie przeprowadzonych scenariuszy badawczych zauważalne są parametry, w których poszczególne brokery przeważają nad swoimi alternatywami. Pozwalają one na klasyfikacje poszczególnych brokerów w zależności od jego zastosowania i wyodrębnienie przypadków użycia, w których dany broker będzie najefektywniejszy. W przypadkach, gdzie zachodzi konieczność procesowania dużych wolumenów danych, a liczba otrzymywanych komunikatów w jednostce czasu może ulegać gwałtownej zmianie, istnieje ryzyko przekroczenia maksymalnej przepustowości danego brokera. W takich sytuacjach, zwłaszcza podczas przetwarzania wiadomości z systemów zewnętrznych, które mogą być częściowo przesyłane jako wsadowy strumień danych, najlepszym wyborem będzie Apache Pulsar. Świadczą o tym najlepsze wyniki uzyskiwane w eksperymencie, gdzie przetwarzane były zalegające dane do brokerów. Najbardziej optymalnym rozmiarem wiadomości dla brokera okazuje się być rozmiar rzędu kilku kilobajtów, na co wskazują gorsze rezultaty uzyskiwane dla mniejszej liczby partycji przy małym rozmiarze wiadomości. Narzędzie dobrze sprawdzi też się w przypadkach, gdy w systemie zachodzi potrzeba zastosowania szyfrowania

wysyłanych komunikatów, gdyż w tym brokerze badania wykazały najmniejszy wpływ jego zastosowania na końcową wydajność tego narzędzia.

W systemach, które używają brokerów wiadomości do implementacji serwisów o architekturze sterowanej zdarzeniami, najlepiej powinno sprawdzić się narzędzie Apache Kafka. Narzędzie oferuje najlepszą wydajność w procesowaniu wiadomości w czasie rzeczywistym, co zostało opisane w rozdziale 6.2, osiągając dużo mniejsze wartości opóźnień przetwarzanych komunikatów w stosunku do Apache Pulsar, co w przypadku zastosowania brokera w takiej architekturze może wpływać na jego stabilność. W przypadku analizowania różnic w wydajności procesowania czasu rzeczywistego dla tego narzędzia należy wziąć pod uwagę fakt, że liczba brokerów, które będą procesować dane komunikaty będzie skalowana, a więc przewaga w maksymalnej wydajności będzie rosła wraz ze wzrostem liczby instancji danego brokera. Dla mniejszych systemów, w których może zajść ograniczona ilość zasobów, najlepszym rozwiązaniem może być RabbitMQ. Cechuje się zdecydowanie najmniejszym opóźnieniem spośród badanych kolejek wiadomości, jednak wraz ze zwiększaniem rozmiaru procesowanych wiadomości, zauważalny jest spadek wydajności w procesowaniu dla tego narzędzia. Szczególnie widoczne jest to w przypadku stosowania protokołu szyfrującego, gdzie już dla mniejszych rozmiarów wiadomości, opóźnienie dostarczania tych komunikatów wzrosło wyraźniej niż w przypadku pozostałych omawianych narzędzi.

W przypadku wszystkich brokerów należy zwrócić uwagę na dużo dłuższy czas dostarczania danej wiadomości w przypadku stosowania protokołu szyfrującego dla pojedynczej wiadomości. W przypadku przetwarzania komunikatów z wysoką częstotliwością nie ma on wielkiego wpływu na stabilność brokera, natomiast czas dostarczenia pojedynczego komunikatu wzrasta o kilkadziesiąt procent już w przypadku przetwarzania wiadomości o rozmiarze 10kB. Biorąc pod uwagę ten fakt, należy projektować systemy używające protokoły szyfrowania w taki sposób, by miały możliwość wsadowego wysyłania takich informacji do brokera, co powinno zniwelować wpływ jego zastosowania na wydajność aplikacji.

Literatura

[1] Oficjalna dokumentacja narzędzia Spring Boot, https://docs.spring.io/spring-boot/docs/current/reference/html, [21.03.2024].

[2] Dokumentacja platformy spring-cloud-binder, https://cloud.spring.io/spring-cloud-stream/multi/multi_spring-cloud-stream-overview-binders.html, [21.03.2024].

[3] G. Fu, Y. Zhang, G. Yu, A Fair Comparison of Message Queuing Systems, IEEE Access 9 (2020) 421-431, https://dx.doi.org/10.1109/ACCESS.2020.3046503.

[4] Magnoni, Luca. Modern messaging for distributed systems. Journal of Physics: Conference Series., IOP Publishing (2015) 2-6.

[5] B. Singh., B. H. Chaitra. Comprehensive Review of Stream Processing Tools. International Research Journal of Engineering and Technology 7(5) (2020) 3537-3540.

[6] K. Sowmya, T. Sharvari, A study on Modern Messaging Systems - Kafka, RabbitMQ and NATS Streaming, CoRR abs/1912.03715 (2019) 2-5.

[7] M. Rokin, S. Hossain, M. Ashfakur, Benchmarking Message Queues, Department of Computer Science, Baylor University (2023) 298-312, https://doi.org/10.3390/telecom4020018.

[8] Oficjalna dokumentacja OpenMessaging Benchmark Framework, https://openmessaging.cloud/docs, [20.03.2024].

[9] P. Jeba, P. Marca, J. Arockia. Comparison of JMS Products, International Journal of Scientific Research in Computer Science (2018) 190-193, https://doi.org/10.32628/CSEIT183858.

[10] T. E. Pereira, R. de Araújo Souza, Performance analysis between Apache Kafka and RabbitMQ, UFCG (2020) 5-11.

[11] Oficjalna dokumentacja narzędzia Apache Kafka, https://kafka.apache.org/documentation, [22.03.2024].

[12] Platforma Linkedin, https://www.linkedin.com, [01.04.2024].

[13] V. E. Balas, L.C Jain, Replication in Raft vs Apache Zookeeper, Proceedings of the 9th International Workshop Soft Computing Applications (2020) 426-436, https://doi.org/10.1007/978-3-031-23636-5.

[14] Oficjalna strona narzędzia RabbitMQ, https://www.rabbitmq.com, [23.03.2024].

[15] Oficjalna dokumentacja narzędzia Apache Pulsar https://pulsar.apache.org/docs, [23.03.2024].

[16] A. Anjum, I. Odun-Ayo, Cloud multi-tenancy: Issues and developments, Proceedings of the 10th International Conference on Utility and Cloud Computing (2017) 209-214, https://doi.org/10.1145/3147234.3148095.

[17] Dokumentacja firmy Oracle dotycząca tworzenia *keystore* oraz *truststore* formatem JKS, https://docs.oracle.com/cd/E19509-01/820-3503/ggfen/index.html, [26.03.2024].