JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 14 (2020) 19-25
Wysłane: 2019-10-04
Przyjęte: 2019-10-15

Analiza funkcjonalna i wydajnościowa wybranych brokerów komunikatów w aplikacji rozproszonej

Tobiasz Kaciuczyk, Tomasz Korga*, Jakub Smołka
Politechnika Lubelska, Katedra Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono wyniki badań wydajności wybranych brokerów komunikatów: Apache ActiveMQ, RabbitMQ oraz Apache Kafka. Analizie został poddany czas przesłania wiadomości wyznaczony na podstawie czasu wysłania i odebrania komunikatu. Testy zostały przeprowadzone za pomocą autorskich aplikacji klienckich napisanych w języku Java. Badania uzupełniono opisem teoretycznym architektury każdego z narzędzi, w tym specyfikacji JMS i AMQP, oraz podstawowym opisem funkcjonalności brokerów.

Słowa kluczowe: broker komunikatów; mikrousługi; komunikacja asynchroniczna

*Autor do korespondencji.

Adres/adresy e-mail: jakub.smolka@pollub.pl, tobiasz.kaciuczyk@pollub.edu.pl, tomasz.korga@pollub.edu.pl

Functional and performance analysis of selected message brokers in a distributed application

Tobiasz Kaciuczyk, Tomasz Korga*, Jakub Smołka
Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. Article presents results of performance analysis of selected message brokers: Apache ActiveMQ, RabbitMQ and Apache Kafka. To analyze has been subjected time of messaging determined based by time of sending and receiving message. Tests were carried out by authorial client application, written in Java language. The research was supplemented with a theoretical description of each tools architecture, including JMS and AMQP specifications and a basic description of brokers functionality.

Keywords: message broker; microservices; asynchronous communication

*Corresponding author.

E-mail address/addresses: jakub.smolka@pollub.pl, tobiasz.kaciuczyk@pollub.edu.pl, tomasz.korga@pollub.edu.pl

1. Wstęp

Brokery komunikatów są odpowiedzią na wiele problemów, przed którymi stają architekci oprogramowania. Ich głównym zadaniem jest wyeliminowanie synchronicznej komunikacji pomiędzy odrębnymi modułami programu oraz zmniejszenie sprzężeń występujących między nimi. Funkcjonalności oferowane przez brokery komunikatów znajdują zastosowanie w systemach przetwarzających dużo informacji, które nie mogą być przetworzone w sposób synchroniczny. Dlatego są szczególnie przydatne w systemach rozproszonych, ale z powodzeniem znajdują zastosowanie również w systemach związanych z Internetem rzeczy, w systemach opartych na architekturze sterowanej zdarzeniami, czy w systemach analizujących duże ilości danych, na przykład w celu zasilania hurtowni danych.

2. Cel i przedmiot badań

Głównym celem przeprowadzonych badań było scharakteryzowanie popularnych brokerów komunikatów pod względem wydajności, mierzonej w czasie przetwarzania i przesyłania komunikatów. Badania zostały uzupełnione teoretycznymi opisami architektury brokerów oraz opisem podstawowych funkcjonalności, użytecznych z punktu widzenia użytkowników.

Do analizy zostały wybrane trzy narzędzia realizujące funkcję brokera komunikatów: Apache ActiveMQ, RabbitMQ oraz Apache Kafka.

3. Apache ActiveMQ

Apache ActiveMQ [1], [2] jest brokerem komunikatów opartym na specyfikacji JMS (Java Message Service) w wersji 1.1. Został napisany w języku Java, ale posiada wiele bibliotek klienckich dla innych języków. Jest rozwijany przez Apache Software Foundation jako otwarte oprogramowanie i został wydany na licencji Apache 2.0, która umożliwia każdemu dowolne używanie i modyfikowanie kodu źródłowego.

ActiveMQ jest narzędziem zaprojektowanym jako warstwa pośrednia zorientowana na wymianę komunikatów (ang. message-oriented-middleware). Jego podstawowym celem jest przekazywanie wiadomości i zdarzeń w aplikacji rozproszonej, gwarantując ich odebranie przez odpowiednie

komponenty. Żeby odpowiednio realizować tę funkcję broker powinien cechować się wysoką dostępnością, wydajnością i łatwą skalowalnością. Celem inżynierów pracujących nad ActiveMQ było dostarczenie narzędzia opartego na standardzie z funkcjonalnością integracji pomiędzy różnymi technologiami. ActiveMQ jest elastyczny w kwestii wykorzystywanego protokołu do realizacji komunikacji. Wspierane są między innymi rozwiązania takie jak AMQP, STOMP, MQTT, a także TCP, UDP, czy XMPP.

3.1. Architektura

ActiveMQ jest implementacją specyfikacji JMS, dlatego opis architektury tego narzędzia warto rozpocząć od opisania tego standardu.

Java Message Service został wydany w celu stworzenia standardowego API do wysyłania i odbierania komunikatów w języku Java. Dzięki tej specyfikacji programista może zaimplementować w swojej aplikacji obsługę wiadomości z brokera komunikatów bez specjalistycznej wiedzy o konkretnej implementacji brokera.

JMS definiuje w swojej specyfikacji następujące artefakty związane z architekturą komunikatów [1]:

- *JMS client* - klient kolejki, aplikacja do wysyłania i odbierania wiadomości napisana w języku Java, może odbierać wiadomości w sposób blokujący wątek programu lub nie;
- *Non-JMS client* - klient kolejki stworzony za pomocą dedykowanych rozwiązań dla języków programowania innych niż Java;
- *JMS producer* - producent, klient brokera, który tworzy i wysyła wiadomości;
- *JMS consumer* - konsument, klient brokera, który odbiera i przetwarza wiadomości;
- *JMS provider* - dostawca JMS, broker, kolejka, implementacja interfejsów JMS napisana w języku Java
- *JMS message* - wiadomość, komunikat, podstawowy artefakt specyfikacji JMS, wysyłany i odbierany przez klientów JMS;
- *JMS domains* - dziedziny, dwa modele przekazywania wiadomości: bezpośrednio, lub w architekturze publikacji/subskrypcji;
- *Administered objects* - obiekty konfiguracyjne, zawierają dane konfiguracyjne potrzebne klientom, dostępne za pomocą interfejsu JNDI (Java Naming and Directory Interface) języka Java;
- *Connection factory* - fabryka połączeń, używana przez klientów do tworzenia połączeń z brokerem (JMS provider);
- *Destination* - cel, obiekt do którego wiadomości są adresowane, lub od którego są odbierane.

Architektura JMS umożliwia skorzystanie z dwóch modeli dystrybucji wiadomości: tematów (ang. topic) topików i kolejek (ang. queue). Temat jest to model przekazywania komunikatów, w którym może być wiele subskrybentów danego kanału. Wiadomość przekazana do tematu, zostanie przesłana do każdego zarejestrowanego subskrybenta. W modelu kolejki wiadomość może zostać odebrana tylko raz przez jednego subskrybenta.

Wiadomość w JMS składa się z trzech elementów [2]:

- ciała wiadomości (ang. body) - może to być tekst, mapa, tablica bajtów, obiekt, lub strumień obiektów o typach prymitywnych;
- nagłówka - za jego pomocą programista może sterować sposobem, w jaki wiadomość jest przetwarzana przez broker;
- właściwości (ang. properties) - zawierają dodatkowe zmienne sterujące aplikacją, ale nie brokerem.

Specyfikacja JMS dostarcza wiele predefiniowanych nagłówków, pozwalających na sterowanie brokerem, niektóre z nich to [1]:

- *JMSDestination* - reprezentuje cel wiadomości: kolejkę lub temat
- *JMSDeliveryMode* - zawiera informację, czy wiadomość powinna zostać trwale zapisana;
- *JMSExpiration* - czas długości życia wiadomości wyrażony w milisekundach;
- *JMSPriority* - priorytet, wartość całkowita od 0 do 9;
- *JMSMessageID* - unikalny identyfikator wiadomości;
- *JMSTimestamp* - czas, w którym wiadomość została wysłana;
- *JMSReplyTo* - opcja umożliwiająca wyjątkowe sterowanie logiką aplikacji za pomocą komunikatu, reprezentuje ten sam obiekt co *JMSDestination*, programista może zdecydować do jakiej kolejki trafi odpowiedź na wysłany przez niego komunikat.

4. RabbitMQ

RabbitMQ [3-5] jest otwartym oprogramowaniem, został napisanym w języku Erlang, jego rozwijaniem zajmuje się Pivotal Software Inc. RabbitMQ został oparty na protokole AMQP (Advanced Message Queuing Protocol), który definiuje zarówno protokół komunikacji w sieci, ale również podstawowy model brokera.

4.1. Architektura AMQP

Specyfikacja AMQP 0-9-1 definiuje zarówno protokół komunikacji sieciowej jak również niektóre usługi i zachowania brokera. Model AMQ (Advanced Message Queuing) definiuje trzy główne komponenty brokera [4]:

- *exchange* (punkt wymiany) - odbiera komunikaty od wydawcy i kieruje je do kolejek;
- *queue* (kolejka) - odbiera komunikaty z punktu wymiany i wysyła je do odbiorców. Stanowi strukturę danych na dysku lub w pamięci podręcznej, miejsce przechowywania komunikatów;
- *binding* (wiązanie) - zasady wykorzystywane przez punkt wymiany w celu skierowania komunikatu do odpowiedniej kolejki, definiowane przez klientów.

Protokół AMQP 0-9-1 definiuje 4 rodzaje punktu wymiany [4]:

- *direct exchange* - dostarcza wiadomość do kolejki na podstawie klucza trasowania (ang. routing key) zawartego w wiadomości;
- *fanout exchange* - dostarcza wiadomość do wszystkich kolejek powiązanych z danym punktem wymiany, klucz trasowania jest ignorowany;
- *topic exchange* - dostarcza wiadomość do jednej lub wielu kolejek, na podstawie klucza trasowania;
- *headers exchange* - klucz trasowania jest ignorowany, zamiast niego o przekierowaniu wiadomości decydują opcje zawarte w jej nagłówku.

Kolejnym podstawowym elementem modelu AMQP jest kolejka. Każda kolejka musi posiadać unikalną nazwę. Może być trwała lub nietrwała: trwała zostanie przywrócona po restarcie aplikacji, nietrwała zostanie usunięta. Warto zaznaczyć, że w przypadku trwałej kolejki, przywrócone zostaną tylko te wiadomości, które zostały oznaczone flagą trwałości (persisted). Istnieje możliwość stworzenia kolejki wyłącznie dla konkretnego połączenia. Taka kolejka zostanie usunięta po przerwaniu danego połączenia. Kolejki mogą być też automatycznie usuwane gdy stracą wszystkich konsumentów wiadomości. Specyfikacja AMQP udostępnia również mechanizm kontrolowania dostarczenia wiadomości ACK (ang. acknowledgements). Wiadomość zostanie usunięta z kolejki dopiero po otrzymaniu informacji o jej poprawnym odebraniu przez konsumenta [4].

Wiadomość w AMQP posiada cechy nazywane atrybutami, na przykład [5]:

- *content type* - typ treści wiadomości;
- *content encoding* - sposób kodowania treści;
- *routing key* - klucz trasowania;
- *delivery mode* - sposób dostarczenia (trwały lub nie);
- *message priority* - priorytet wiadomości;
- *message publishing timestamp* - czas utworzenia wiadomości;
- *expiration period* - czas ważności wiadomości;
- *publisher application id* - identyfikator aplikacji publikującej wiadomość.

Treść wiadomości jest traktowana jako tablica bajtów. Broker nie modyfikuje ani nie czyta treści wiadomości. Atrybuty związane z treścią (*content-type*) są potrzebne dla zachowania konwencji przy używaniu formatów takich jak JSON. Wiadomości, które nie mogą zostać przetworzone, mogą być zwrócone do producentów, usunięte lub przekazane na inną kolejkę [3].

Duża elastyczność RabbitMQ wynika ze sposobu w jaki komunikaty mogą być przekazywane z punktu wymian do kolejek, oraz możliwości definiowania zasad przekazywania wiadomości podczas jej tworzenia. Wiązania pomiędzy punktem wymiany i kolejkami to podstawa architektury bazującej na komunikatach. RabbitMQ umożliwia łatwe przystosowanie aplikacji do zmieniających się wymagań [3].

5. Apache Kafka

Apache Kafka [6-8] powstała w odpowiedzi na potrzeby firmy LinkedIn w zakresie komunikacji wielu komponentów w systemie rozproszonym. Została udostępniona jako otwarte oprogramowanie.

Realizuje wzorzec wydawcy i subskrybenta (ang. publish/subscribe). Celem inżynierów pracujących nad Kafką było stworzenie narzędzia o następujących cechach: bardzo dużej wydajności, możliwości przetwarzania dużej liczby komunikatów jednocześnie, ich trwałym przechowywaniu, a także zachowaniem kolejności, w której komunikaty trafiły do brokera. Dodatkowo Kafka zapewnia mechanizmy zabezpieczeń przed awariami i wysoką skalowalność [6].

5.1. Architektura

Pojedyncza jednostka danych w Apache Kafka jest nazywana wiadomością, komunikatem, wierszem lub rekordem, w praktyce jest to tablica bajtów bez określonego typu. Komunikat może posiadać również identyfikator nazywany kluczem. Wiadomości mogą być grupowane w kolekcje nazwane partiami (ang. batch). Grupowanie wiadomości w partie wiąże się z dużo szybszym przetworzeniem całej partii, niż gdyby wiadomości byłyby wysyłane osobno. Ale czas przetworzenia pojedynczej wiadomości będzie oczywiście dłuższy (musi zostać przetworzona cała partia). Partie są zazwyczaj kompresowane co podnosi wydajność procesu [6].

Wiadomości są umieszczane w tematach. Temat w Apache Kafka jest analogicznym bytem do tabeli w bazie danych. Każdy temat musi zawierać przynajmniej jedną partycję. Wysłanie wiadomości polega na dopisaniu na koniec partycji. Odczytanie jest pobraniem wiadomości z zachowaniem kolejności, od pierwszej do ostatniej. Nie ma możliwości modyfikacji wiadomości, która znajduje się już w kolejce. Zachowanie kolejności jest możliwe tylko w obrębie pojedynczej partycji. Każda partycja może się znajdować fizycznie na innej maszynie, co znacząco ułatwia skalowanie już na poziomie tematów.

Pojedyncza instancja Apache Kafka jest nazywana brokerem. Wielu brokerów można łączyć w klastry. Broker jest właścicielem partycji. W przypadku gdy partycja należy do wielu brokerów, jeden broker staje się liderem partycji, a pozostałe służą jako mechanizm powielania danych na wypadek awarii.

Oprócz mechanizmów powielania Kafka posiada również mechanizmy retencji, czyli składowania danych, z wieloma możliwościami konfiguracyjnymi. W przypadku zatrzymania działania serwera dane nie zostaną utracone, ponieważ Apache Kafka domyślnie zapisuje wszystkie dane na dysku twardym.

6. Zestawienie funkcjonalności

W celu porównania funkcjonalności opisywanych brokerów, zostały one zestawione ze sobą w tabeli 1. Porównanie zostało opracowane na podstawie materiałów źródłowych [1-8], a także na podstawie praktycznych doświadczeń, zdobytych podczas pracy nad aplikacją badawczą wykorzystującą każdą z kolejek.

Tabela 1. Wydajność brokerów dla różnych rozmiarów wiadomości

Apache ActiveMQ	**RabbitMQ**	**Apache Kafka**
Protokół przesyłania wiadomości		
Wiele wspieranych protokołów, m. in.: Stomp, AMQP 1.0, MQTT, XMPP.	Wspierane protokoły: AMQP 0-9-1 z rozszerzeniami, Stomp, AMQP 1.0, MQTT.	Własna implementacja protokołu sieciowego opartego na protokole TCP.
Format wiadomości		
Według specyfikacji JMS: mapa, tablica bajtów, obiekt, strumień obiektów o typach prymitywnych.	Obiekt serializowany do tablicy bajtów. Możliwość własnej implementacji mechanizmu serializacji i deserializacji.	Obiekt serializowany do tablicy bajtów. Możliwość własnej implementacji mechanizmu serializacji i deserializacji.
Trwałość wiadomości		
Możliwość wyboru różnych sposobów persystencji: interfejs JDBC lub replikowane, plikowe bazy danych: KahaDB lub Replicated LevelDB Store.	Mechanizm persystencji wiadomości za pomocą rozproszonej bazy danych Mnesia (dane mogą być przechowywane w pamięci operacyjnej lub na dysku).	Każda wiadomość jest domyślnie zapisywana na dysku twardym.
Potwierdzenie dostarczenia wiadomości		
Tak, posiada konfigurowalny mechanizm.	Tak, posiada konfigurowalny mechanizm.	Brak.
Bezpieczeństwo		
Wspiera dołączane mechanizmy bezpieczeństwa. Domyślnie JAAS (Java SE Security) do uwierzytelniania oraz plik konfiguracyjny do autoryzacji.	Wspiera dołączane mechanizmy autoryzacji (np. LDAP) i uwierzytelniania (hasło lub certyfikaty), wspiera protokół TLS do szyfrowania komunikacji.	Wspiera dołączane mechanizmy autoryzacji i uwierzytelniania, wspiera protokół SSL/TLS do szyfrowania komunikacji.
Model subskrypcji brokera		
Kolejka (*JMSQueue*) lub temat (*JMSTopic*).	Oparty o mechanizm punktu wymiany, który ma wiele możliwości. Opisany w rozdziale 4.1.	Temat (ang. *topic*).
Środowisko rozproszone		
Wspiera tworzenie sieci brokerów (logicznie rozłącznych), klastra brokerów, i topologię *Master/Slave*.	Instancje mogą być łączone w klastry - tworząc jeden logiczny broker - lub federacje - brokerzy są logicznie oddzielni, mogą komunikować się ze sobą.	Został zaprojektowany dla systemów rozproszonych, używa narzędzia Zookeeper do zarządzania instancjami, łatwo skalowalna.
Zarządzanie brokerem		
Graficzny interfejs użytkownika dostępny przez przeglądarkę lub interfejs linii poleceń.	Graficzny interfejs użytkownika dostępny przez przeglądarkę lub interfejs linii poleceń.	Interfejs linii poleceń.
Interfejs programistyczny		
Biblioteka w języku Java, interfejs HTTP REST, interfejs WebSocket. Dodatkowe biblioteki klienckie dla większość popularnych języków programowania.	Oficjalne biblioteki klienckie w językach: Java, Erlang, .NET. Wiele dodatkowych bibliotek i rozszerzeń.	Oficjalne biblioteki klienckie w językach: Java, .NET. Wiele dodatkowych bibliotek i rozszerzeń.
Monitoring		
Monitoring możliwy przez JMX, interfejs graficzny, aplikację linii poleceń, dodatkowe narzędzia.	Metryki udostępnianie przez interfejs API, aplikację linii poleceń, interfejs graficzny.	Metryki domyślnie udostępniane przez JMX.
Inne		
Wsparcie standardu SOAP, integracja z Apache Camel, Spring, integracja z serwerami aplikacyjnymi poprzez wsparcie mechanizmu JNDI.	Integracja z narzędziem Spring.	Integracja z narzędziem Spring.

7. Badania wydajności

7.1. Środowisko badawcze

Badania zostały przeprowadzone za pomocą dwóch programów napisanych w języku Java, które pełniły rolę wydawcy oraz odbiorcy wiadomości dla każdego z brokerów.

Każda z kolejek została uruchomiona w osobnym kontenerze utworzonym za pomocą narzędzia Docker.

Testy wydajności zostały przeprowadzone na komputerze z parametrami technicznymi:

- procesor Intel Core i5-7200U CPU@2.5Ghz (4 rdzenie),
- 16GB pamięci RAM, dysk twardy typu SSD o pojemności 256GB,
- system operacyjny Ubuntu 18.04.

Programy zostały napisane w języku Java w wersji 1.11.0, dodatkowo zostały wykorzystane następujące narzędzia: serwer aplikacyjny Apache Tomcat 9.0.21, Spring Boot 2.1.6, Docker 18.06, Zookeeper 3.4.13.

Do badania wybrano stabilne wersje każdej z kolejek: Apache ActiveMQ 5.15.6, RabbitMQ 3.7.17, oraz Apache Kafka 2.12.

7.2. Metodyka badawcza

Do wykonania pomiarów czasu przesyłania wiadomości wykorzystano dwa programy. Pierwszy z nich przesyłał wiadomości do kolejek oraz odpowiadał za zapisanie czasu wysłania wiadomości. Drugi program miał za zadanie odebranie wiadomości oraz zapisanie czasu odebrania.

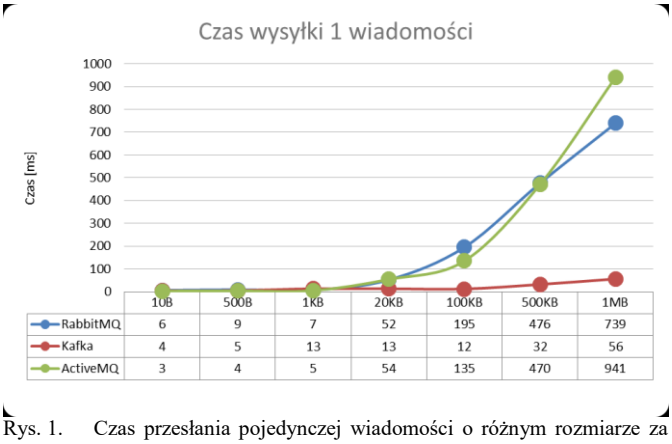
Podczas badań pozostawiono domyślną konfigurację każdego z brokerów. Wszystkie próby zostały przeprowadzone według wzorca: jeden wydawca - jeden subskrybent.

W ramach przypadków testowych dla każdego z testowanych brokerów stworzono siedem wiadomości o wielkościach: 10 B, 500 B, 1 KB, 20 KB, 100 KB, 500 KB oraz 1 MB. Każda z tych wiadomości w ramach pojedynczego przypadku testowego została przesłana: 1 raz, 100 razy, 1000 razy, 3000 razy, 5000 razy.

Różnica czasu pomiędzy odebraniem ostatniej, a wysłaniem pierwszej wiadomości pozwoliła na określenie czasu przesyłania wiadomości przez broker wiadomości, a dalsza obróbka danych umożliwiła określenie wydajności każdej z kolejek. Wszystkie testy odbyły się na jednym komputerze w jak najbardziej zbliżonych do siebie warunkach. Rezultaty badań pozwoliły na wygenerowanie wykresów widocznych w następnym punkcie.

7.3. Wyniki

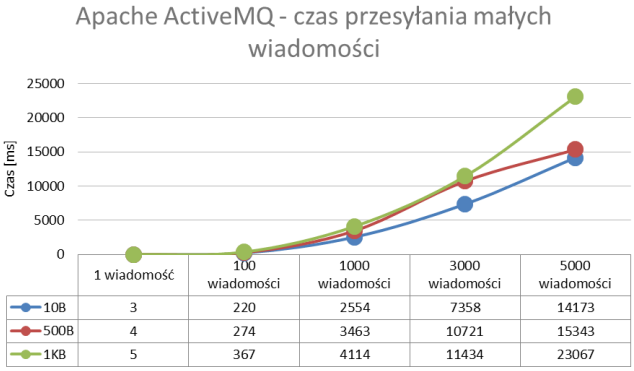
Pierwszym ze stworzonych testów badawczych było porównanie czasów wysyłki jednej wiadomości dla różnej jej długości w analizowanych kolejkach. Wyniki badania przedstawiono na Rys. 1.



	10B	500B	1KB	20KB	100KB	500KB	1MB
RabbitMQ	6	9	7	52	195	476	739
Kafka	4	5	13	13	12	32	56
ActiveMQ	3	4	5	54	135	470	941

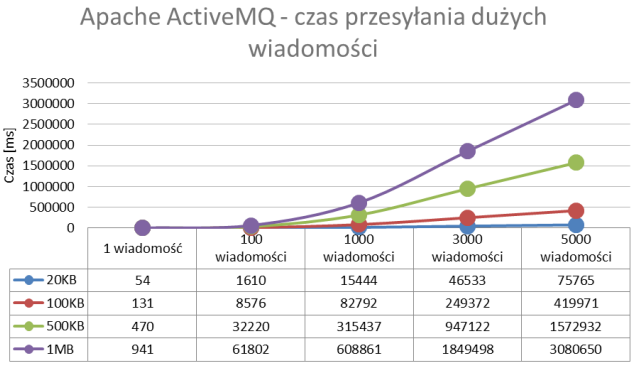
Rys. 1. Czas przesłania pojedynczej wiadomości o różnym rozmiarze za pomocą każdej z kolejek.

Kolejnymi testami były pomiary czasu przesłania wiadomości. Tutaj zostały one podzielone na dwie grupy: wiadomość mała (rozmiar poniżej 1 KB) oraz wiadomość duża (rozmiar powyżej 1 KB). Podział ten został wykonany z racji różnicy wielkości jednostki czasu i poprawy czytelności wyników.



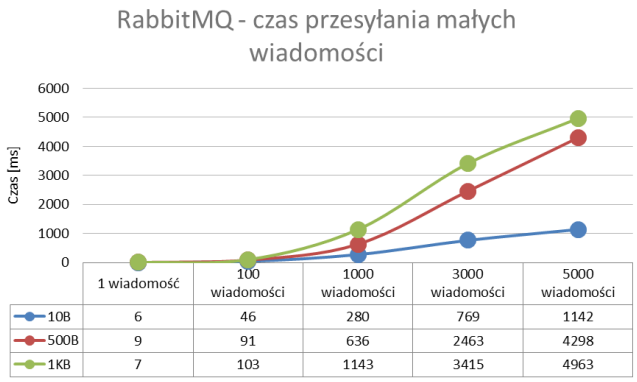
	1 wiadomość	100 wiadomości	1000 wiadomości	3000 wiadomości	5000 wiadomości
10B	3	220	2554	7358	14173
500B	4	274	3463	10721	15343
1KB	5	367	4114	11434	23067

Rys. 2. Czas przesłania różnych ilości wiadomości o rozmiarach 10 B, 500 B i 1 KB za pomocą Apache ActiveMQ.



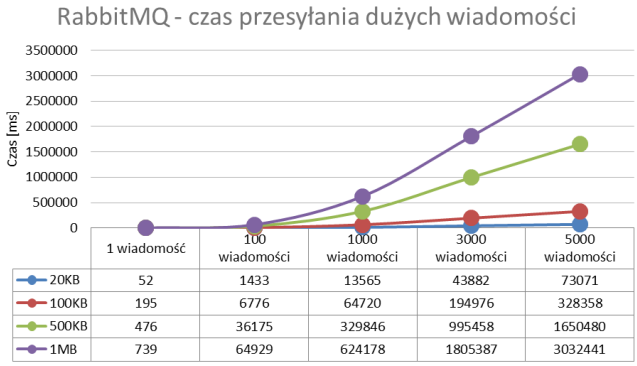
	1 wiadomość	100 wiadomości	1000 wiadomości	3000 wiadomości	5000 wiadomości
20KB	54	1610	15444	46533	75765
100KB	131	8576	82792	249372	419971
500KB	470	32220	315437	947122	1572932
1MB	941	61802	608861	1849498	3080650

Rys. 3. Czas przesłania różnych ilości wiadomości o rozmiarach 20 KB, 100 KB, 500 KB i 1 MB za pomocą Apache ActiveMQ.



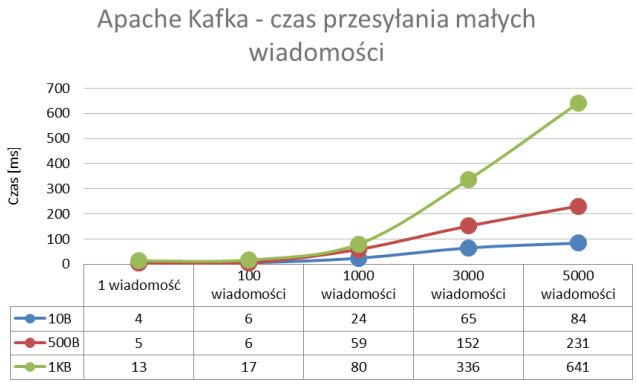
	1 wiadomość	100 wiadomości	1000 wiadomości	3000 wiadomości	5000 wiadomości
10B	6	46	280	769	1142
500B	9	91	636	2463	4298
1KB	7	103	1143	3415	4963

Rys. 4. Czas przesłania różnych ilości wiadomości o rozmiarach 10 B, 500 B i 1 KB za pomocą RabbitMQ.



	1 wiadomość	100 wiadomości	1000 wiadomości	3000 wiadomości	5000 wiadomości
20KB	52	1433	13565	43882	73071
100KB	195	6776	64720	194976	328358
500KB	476	36175	329846	995458	1650480
1MB	739	64929	624178	1805387	3032441

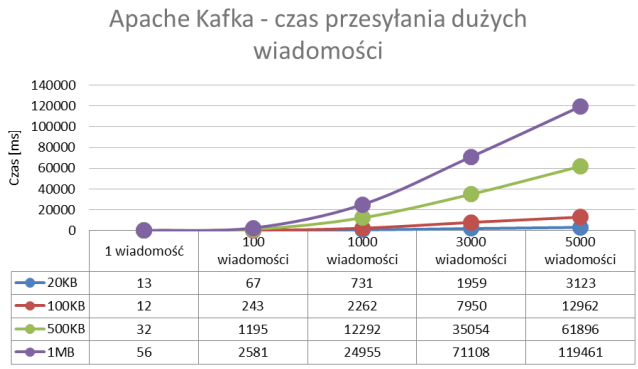
Rys. 5. Czas przesłania różnych ilości wiadomości o rozmiarach 20 KB, 100 KB, 500 KB i 1 MB za pomocą RabbitMQ.



	1 wiadomość	100 wiadomości	1000 wiadomości	3000 wiadomości	5000 wiadomości
10B	4	6	24	65	84
500B	5	6	59	152	231
1KB	13	17	80	336	641

Rys. 6. Czas przesłania różnych ilości wiadomości o rozmiarach 10 B, 500 B i 1 KB za pomocą Apache Kafka.

Dzięki przeprowadzonym testom możliwe było wyznaczenie wydajności kolejek dla różnych rozmiarów przesyłanych danych. W Tabeli 1. przedstawiono wydajność dla próbki 5000 wiadomości.



	1 wiadomość	100 wiadomości	1000 wiadomości	3000 wiadomości	5000 wiadomości
20KB	13	67	731	1959	3123
100KB	12	243	2262	7950	12962
500KB	32	1195	12292	35054	61896
1MB	56	2581	24955	71108	119461

Rys. 7. Czas przesłania różnych ilości wiadomości o rozmiarach 20 KB, 100 KB, 500 KB i 1 MB za pomocą Apache Kafka.

Tabela 2. Wydajność brokerów dla różnych rozmiarów wiadomości

Wydajność na sekundę			
	RabbitMQ	Kafka	ActiveMQ
10B	4 378,28	59 523,81	352,78
500B	1 163,33	21 645,02	325,88
1KB	1 007,46	7 800,31	216,76
20KB	68,43	1 601,02	65,99
100KB	15,23	385,74	11,91
500KB	3,03	80,78	3,18
1MB	1,65	41,85	1,62

8. Wnioski

Z przesyłaniem pojedynczej wiadomości (Rys. 1.) najlepiej poradził sobie Apache Kafka. Czas wysyłki pojedynczej wiadomości za pomocą tego brokera jest prawie niezmienny dla każdego z badanych rozmiarów. Dla pozostałych kolejek czas zaczyna znacząco rosnąć przy wiadomościach o rozmiarze powyżej 20 KB. Z tego badania wynika, że zarówno Apache ActiveMQ oraz RabbitMQ nie zostały zoptymalizowane pod względem przetwarzania wiadomości o dużych rozmiarach.

Wydajność poszczególnych brokerów najlepiej obrazuje Tabela 2. w której przedstawiono liczbę wysłanych wiadomości na sekundę. Zdecydowanie najbardziej wydajną kolejką jest Apache Kafka - dla każdego z badanych rozmiarów wiadomości Kafka osiągała wynik wielokrotnie lepszy od pozostałych brokerów. W porównaniu Apache ActiveMQ z RabbitMQ lepiej wypadł ten drugi, choć znaczące różnice wystąpiły tylko dla trzech najmniejszych badanych rozmiarów.

ActiveMQ w badaniu przesyłania małych wiadomości (Rys. 2.), wykazał podobne czasy dla każdego z badanych rozmiarów, wzrost czasu przesłania był spowodowany głównie zwiększeniem ilości przesyłanych komunikatów. W badaniu dużych wiadomości (Rys. 3), różnice w czasie przesłania pomiędzy poszczególnymi rozmiarami były już dużo większe.

RabbitMQ okazał się kolejką dużo szybszą od ActiveMQ w przypadku przesyłania małych wiadomości (Rys. 4.), co potwierdza wynik badania wydajności. W przypadku tego brokera jest jednak widoczna duża różnica pomiędzy najmniejszym rozmiarem komunikatu i pozostałymi, przy próbie wysłania 3000 i więcej wiadomości. W badaniu czasu przesyłania dużych wiadomości (Rys. 5.), wynik RabbitMQ okazał się niemal identyczny z wynikiem ActiveMQ, co również potwierdza wynik badania wydajności.

Wyniki przesyłania wiadomości o małych i dużych rozmiarach w przypadku Apache Kafka okazały się przewidywalne, czasy przesłania rosną proporcjonalnie z liczbą wysyłanych wiadomości oraz z ich rozmiarem (Rys. 6, Rys. 7.).

Z analizy funkcjonalnej wynika, że Apache Kafka oferuje najmniej dodatkowych funkcjonalności. Udostępnia tylko jeden model subskrypcji wiadomości (temat), brakuje mechanizmu kontroli dostarczania komunikatów. W przeciwieństwie do pozostałych badanych narzędzi, Kafka nie ma wbudowanego graficznego narzędzia administracyjnego. Mimo to posiada wszystkie podstawowe funkcjonalności, takie jak szyfrowanie komunikacji, interfejs linii poleceń, biblioteki klienckie, czy udostępnianie metryk.

Warto zwrócić uwagę, że Apache Kafka jest wyjątkowym narzędziem na tle pozostałych. W komunikacji nie zdecydowano się na wykorzystanie jednego z gotowych protokołów komunikacji, czy specyfikacji, ale zastosowano autorskie rozwiązania. Szczególny jest również sposób przechowywania wiadomości w kolejce, gdyż każdy komunikat jest automatycznie zapisywany na dysku twardym. Nie jest to dodatkowa opcja bezpieczeństwa, ale naturalny sposób na zarządzanie komunikatami w brokerze.

Powyższe cechy wskazują, że Apache Kafka jest narzędziem o bardzo dużej wydajności, która została osiągnięta częściowo przez podjęcie kompromisów i rezygnację z niektórych funkcjonalności

ActiveMQ oraz RabbitMQ pod względem funkcjonalności są również bardzo podobne do siebie. ActveMQ jest implementacją specyfikacji JMS, co sprawia, że jest bardzo proste w użyciu w systemach opartych o język Java. RabbitMQ wydaje się oferować większą uniwersalność. Za sprawą mechanizmu punktów wymiany, dostarcza wyjątkową elastyczność w konfiguracji narzędzia do własnych potrzeb.

Literatura

[1] Snyder, B., Bosanac, D., & Davies, R., Introduction to Apache Activemq. Active MQ in Action, 2017.

[2] Dokumentacja techniczna do Apache ActiveMQ https://activemq.apache.org/components/classic/documentation [22.09.2019]

[3] Roy G. M., RabbitMQ in Depth, Manning, 2017.

[4] Wprowadzenie do AMQP w RabbitMQ https://www.rabbitmq.com/tutorials/amqp-concepts.html [22.09.2019]

[5] Dokumentacja techniczna do RabbitMQ https://www.rabbitmq.com/documentation.html [22.09.2019]

[6] Narkhede N., Shapira G., Palino T., Kafka: The Definitive Guide. Real-Time Data and Stream Processing at Scale, O'Reilly, 2017.

[7] John V., Liu, X., A survey of distributed message broker queues. arXiv preprint arXiv:1704.00411, 2017.

[8] Dokumentacja techniczna do Apache Kafka https://kafka.apache.org/documentation/ [22.09.2019]