ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2017
Seria: ORGANIZACJA I ZARZĄDZANIE z. 113 Nr kol. 1992

Marcin STACHOWIAK, Bogumiła HNATKOWSKA
Politechnika Wrocławska
Wydział Informatyki i Zarządzania
marcin.stachowiak@pwr.edu.pl, bogumila.hnatkowska@pwr.edu.pl

MODEL WYDAJNEGO I SKALOWALNEGO SYSTEMU INTEGRACJI

Streszczenie. Niniejsza praca rozwiązuje problem projektu wydajnej i skalowalnej architektury systemu integracyjnego oraz doboru odpowiednich komponentów do jej praktycznej realizacji. Na podstawie przeprowadzonej analizy porównawczej różnych realizacji platform integracyjnych, bazujących na zaproponowanej architekturze wskazano, że w celu zastosowania modelu przesyłania komunikatów typu punkt-punkt najszybszym brokerem integracyjnym jest RabbitMQ, w połączeniu z protokołem AMQP. Dla modelu przesyłania komunikatów publikuj-subskrybuj, brokerem, który najszybciej przesłał określoną liczbę wiadomości jest Apache Kafka, z własnym dedykowanym protokołem.

Słowa kluczowe: integracja systemów, architektura broker integracyjny, protokół komunikacyjny wydajność, skalowalność, dostępność, niezawodność

MODEL OF EFFICIENT AND SCALABLE INTEGRATION SYSTEM

Abstract. This thesis solves the problem of designing an efficient and scalable architecture of the integration system and selecting the appropriate components for its practical implementation. Based on the research conducted on integration platforms consisting of specially selected components, it was pointed out that in order to implement the point-to-point messaging model, the fastest integration broker is RabbitMQ, in combination with the AMQP protocol. For the publish-subscribe messaging model the fastest broker is Apache Kafka, with its own dedicated protocol.

Keywords: system integration, architecture, integration broker, communication protocol, performance, scalability, availability, reliability

1. Wprowadzenie

Zwiększenie przychodów przedsiębiorstwa często wiąże się z modernizacją przebiegających w nim procesów biznesowych. Procesy mogą obejmować kilka działów przedsiębiorstwa oraz nawet kilka różnych firm, co wskazuje, że za realizację całego cyklu odpowiada kilka systemów. Umiejętne ich zintegrowanie jest kluczem do sukcesu firmy.

System integracyjny to zbiór różnego rodzaju oprogramowania i fizycznych maszyn, którego głównym celem umożliwienie aplikacjom na wzajemne korzystanie z oferowanych usług i zasobów. Warstwą pośredniczącą nazywa się grupę procesów, jak również wspierające je oprogramowanie i infrastrukturę, które mają za zadanie usprawnienie i jednoczesne ukrycie przed integrowanymi systemami aspektów technicznych związanych z nawiązywaniem i utrzymywaniem połączenia oraz transportem danych [2], [3].

Celem pracy jest propozycja oraz analiza wydajnościowa architektury warstwy pośredniczącej w komunikacji pomiędzy wieloma heterogenicznymi systemami. Zakłada się, iż warstwa ta będzie zbudowana zgodnie z paradygmatem MOA (ang. Message-oriented architecture) z komponentów dostępnych w ramach licencji Open Source.

Broker jest głównym składnikiem warstwy pośredniczącej, zapewniając połączenie pomiędzy różnymi systemami w celu umożliwienia automatycznej realizacji procesów biznesowych [1]. Komunikacja w architekturze zorientowanej na komunikaty odbywa się w trybie asynchronicznym, dlatego też broker często dostarcza funkcjonalność archiwizowania wiadomości oraz opcjonalnie wyszukiwania i podglądania jej zawartości [4].

Brokery integracyjne zaprojektowane są najczęściej wg. architektury gwiaździstej (ang. Hub-and-Spoke) lub szyny komunikatów (ang. Message Bus). W przypadku, gdy klientami warstwy pośredniczącej nie są aplikacje, lecz inne systemy pośredniczące, integrujący je broker może zostać oparty na architekturze wielogwiaździstej (ang. Multi Hub) [1].

Zaproponowana architektura została zrealizowana i przebadana w różnych wariantach, przy użyciu najpopularniejszych obecnie na rynku brokerów integracyjnych (tj. ApacheMQ, RabbitMQ i Apache Kafka). Pozwoliło to określić, które parametry systemu integracyjnego (w porównaniu do domyślnej konfiguracji) wpływają na poprawę jakości świadczonych usług.

Niestety, wśród dostępnej literatury przedmiotu trudno znaleźć prace porównujące realizacje alternatywnych architektur systemów integracyjnych pod względem interesujących cech. Jedną z niewielu jest [7], w której autor porównuje systemy pośredniczące RabbitMQ i Apache Kafka pod względem opóźnień w przesyłaniu wiadomości dla różnych rozmiarów komunikatów. Z przeprowadzonych badań wynika, że broker RabbitMQ dla wiadomości o rozmiarze 1 MB powoduje mniejsze opóźnienia niż Apache Kafka. Dla mniejszych rozmiarów komunikatów przesyłanie ich przez broker RabbitMQ skutkuje większymi opóźnieniami niż w przypadku wykonania tej samej czynności z brokerem Apache Kafka.

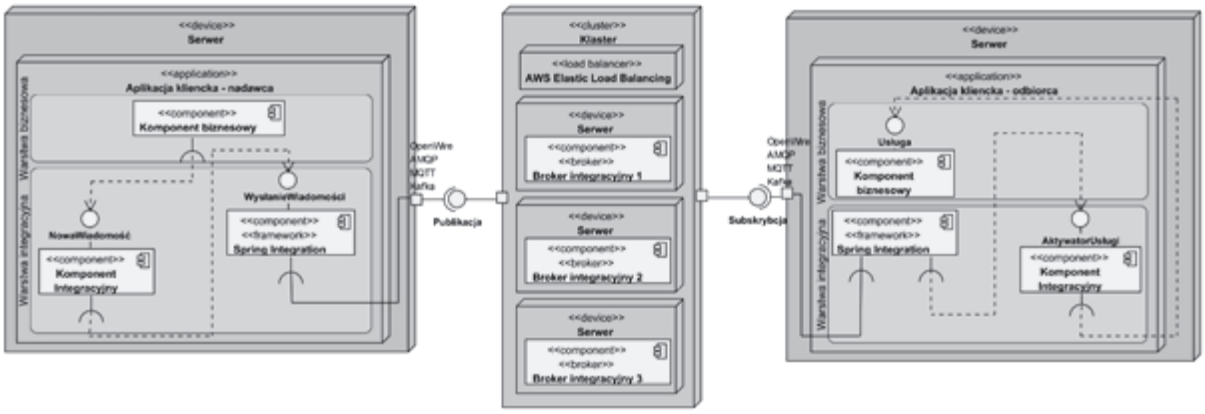
W [6] autorzy podjęli próbę wykonania testów wydajnościowych dla systemu pośredniczącego, wspomagającego proces pozyskiwania danych i ich ładowania do hurtowni danych. W publikacji została zasugerowana architektura dla procesów ETL, a wykorzystanym systemem pośredniczącym był Apache Kafka.

Dalsza część pracy jest zorganizowana następująco. Zaproponowaną w pracy architekturę platformy integracyjnej omówiono w rozdziale 2, a w rozdziale 3 – ocenę jej alternatywnych realizacji. Rozdział 4 zawiera podsumowanie pracy i wnioski.

2. Projekt architektury systemu integracyjnego

Celem pracy jest opracowanie architektury systemu integracyjnego cechującego się wysoką wydajnością, skalowalnością, niezawodnością i dostępnością przy następujących ograniczeniach: (a) Architektura systemu jest zgodna z paradygmatem MOA, komunikacja jest asynchroniczna i wykorzystuje standardowe protokoły komunikacyjne (b) Użyte komponenty dostępne są na licencji OpenSource. (c) Aplikacje integrujące są niezależnymi instancjami, a technologia w jakiej zostały wykonane nie przesądza o wyborze brokera integracji.

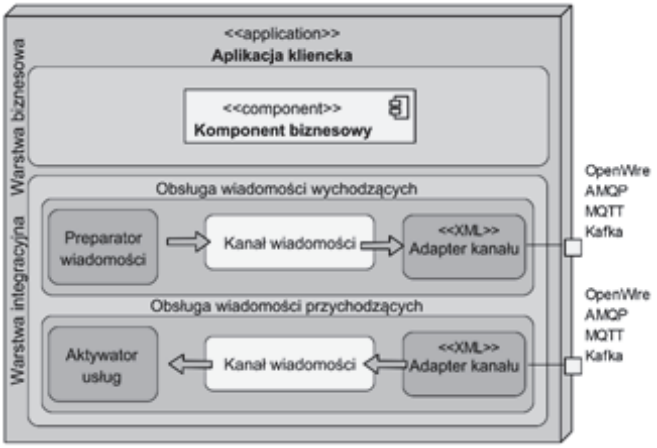
Proponowany system integracji ma architekturę gwiaździstą (patrz rys. 1). Aby zapewnić skalowalność i niezawodność rozwiązania zastosowano klaster obliczeniowy w warstwie pośredniczącej. Obsługą połączeń nawiązywanych pomiędzy pojedynczą instancją klastra, a aplikacją kliencką zajmuje się balanser obciążenia (ang. load balancer).



Rys. 1. Diagram wdrożeniowy zaprojektowanego systemu integracji
Źródło: Opracowanie własne.

Na rysunku 2 przedstawiono architekturę warstwy integracyjnej aplikacji klienckiej. Została ona zaprojektowana w sposób umożliwiający zorientowanie na usługi oraz połączenie z platformami integracyjnymi różnych producentów. Na rysunku można wyróżnić dwa typy adapterów: wyjściowy i wejściowy. Oba łączą wewnętrzny kanał komunikatów z platformą integracyjną. Adapter kanału wykorzystuje predefiniowane biblioteki, dostarczone przez

producenta brokera integracyjnego. Gdy preparator wiadomości opakuje przesyłane dane w obiekt reprezentujący komunikat, umieszcza go w wyjściowym kanale komunikacyjnym. Każdy kanał komunikacyjny wewnątrz aplikacji klienckiej dedykowany jest dla konkretnego tematu lub kolejki w systemie pośredniczącym. Adapter wyjściowy, widząc znajdującą się w kanale wiadomość, transportuje ją do brokera integracyjnego.



Rys. 2. Architektura warstwy integracyjnej aplikacji klienckiej, wysyłającej i odbierającej
Źródło: Opracowanie własne.

W plikach XML, wraz z definicją obiektów adapterów, określone zostają właściwości połączenia tj. nazwa i rodzaj punktu docelowego (kolejka, temat), sposób potwierdzenia odebrania wiadomości, sposób przechowywania wiadomości (trwały, nietrwały), liczba równoległych − konkurencyjnych konsumentów w ramach jednego adaptera wejściowego oraz liczba wiadomości pobieranych przez adapter wejściowy w ramach jednego zapytania.

3. Ocena wydajności architektury

W tym rozdziale opisano badania, przeprowadzone dla różnych implementacji zaproponowanej architektury systemu integracyjnego, zrealizowanych z wykorzystaniem najpopularniejszych brokerów integracyjnych (ActiveMQ, RabbitMQ, Apache Kafka) i protokołów komunikacyjnych (OpenWire, AMQP, MQTT, Kafka).

Do porównania alternatywnych implementacji proponowanej architektury zastosowano dwie grupy metryk. Pierwsza zawiera metryki, mające wpływ na jakość transmisji komunikatów (związane są z przebiegiem procesu biznesowego bezpośrednio). Są to:

- liczba wiadomości wysyłanych/odbieranych w czasie jednej sekundy przez aplikację kliencką,
- łączna liczba wiadomości wysyłanych/odbieranych w czasie jednej sekundy przez platformę integracyjną od wszystkich aplikacji klienckich,

- liczba utraconych wiadomości w czasie normalnej pracy i w momencie awarii jednego z brokerów integracyjnych,
- średnie opóźnienie dostarczenia wiadomości do aplikacji docelowej,
- liczba wiadomości dostarczonych do aplikacji docelowej w niepoprawnej kolejności.

Drugą grupę metryk stanowią parametry techniczne aplikacji klienckich i brokerów integracji wewnątrz klastra (od nich zależą wartości metryki z grupy pierwszej):

- zużywany transfer danych aplikacji klienckiej wysyłającej i odbierającej komunikaty,
- wykorzystanie mocy procesora przez brokery integracyjne,
- wykorzystanie pamięci RAM przez brokery integracyjne.

W trakcie badań przyjęto pewne założenia techniczne, dotyczące: (a) liczby nadawców – 2, (b) liczby odbiorców – 2, (c) liczby wiadomości produkowanych przez jednego nadawcę – 60000, (d) rozmiaru wiadomości – 15 KB. Dodatkowo, założono, iż komunikaty będą generowane ze zmiennym natężeniem.

W badaniach wykorzystano siedem identycznych i niezależnych maszyn wirtualnych dostępnych na serwerach firm Amazon zlokalizowanych w Irlandii. Trzy instancje zostały zarezerwowane na stworzenie klastra obliczeniowego, natomiast pozostałe cztery – dwóch nadawców i dwóch odbiorców. W tabeli 1 przedstawiono szczegóły dotyczące wynajętych instancji serwerów Amazon EC2.

Tabela 1

Parametry techniczne instancji Amazon EC2 typu t2.micro

System operacyjny	Ubuntu Server 16.04 LTS
Procesor	Intel Xeon Family
Liczba rdzeni/Taktowanie	1/2.5 GHz
Pamięć RAM	1GB
Rozmiar dysku twardego/Typ dysku twardego	8 GB/SSD

Źródło: Opracowanie własne

W tabeli 2 zebrano wyniki badań, które uzyskano dla różnych wariantów architektur systemów integracyjnych. Tabela opisuje możliwe kombinacje pomiędzy brokerami integracyjnymi, ich najpopularniejszą konfiguracją oraz protokołami komunikacyjnymi. Przekreślone komórki oznaczają brak możliwości realizacji danej konfiguracji (np. brak wsparcia przez framework Spring Integration dla protokołu AMQP i brokera ActiveMQ).

Oprócz wyboru rodzaju protokołu, system pośredniczący może zostać skonfigurowany za pomocą wielu parametrów. Do przeprowadzenia badań wybrano te, które mogą mieć istotny wpływ na wydajność, w tym: model przekazywania komunikatów (punkt-punkt, publikuj-subskrybuj), przechowywanie wiadomości (trwałe, nietrwałe), potwierdzenie dostarczenia wiadomości (bez potwierdzenia, automatyczne), liczba partycji (na ile partycji podzielono temat, na który są wysyłane wiadomości: 1, 3 – parametr brokera Apache Kafka).

Celem badania było wyłonienie takiego zestawu brokera integracyjnego z protokołem komunikacyjnym, który na podstawie trzykrotnie wykonanych pomiarów i uśrednionych wyników, najszybciej bezstratnie przesłał komunikaty do odbiorcy.

Najlepszym rozwiązaniem w modelu punkt-punkt okazał się system integracji oparty na brokerze RabbitMQ i protokole AMQP (liczba węzłów klastra: 3, utrwalanie wiadomości z potwierdzanie dostarczenia). Charakterystyki techniczne tego zestawu pokazano na rysunku 3. Proces wysyłania komunikatów do brokera integracyjnego RabbitMQ trwał średnio 59,9 s (rys. 3A). Przez większość tego czasu prędkość komunikatów dochodzących do brokera wynosiła ponad 2000 wiadomości na sekundę. Ostatni komunikat został dostarczony do aplikacji klienckiej po 132,8 s od rozpoczęcia badania. W trakcie, gdy system integracji odbierał komunikaty od producentów, prędkość ich wysyłania wynosiła średnio ok. 250 wiadomości na sekundę (rys. 3B). Było to spowodowane obciążeniem procesora i dużym zużyciem transferu danych (patrz rys. 3H, 3I i 3J). Po zakończeniu odbierania komunikatów wykorzystanie procesora i sieci zmalało, co pozwoliło na zwiększenie prędkości odbierania komunikatów z brokera do wartości 1250 wiadomości na sekundę.

Na rysunku 3C przedstawiono średnią liczbę wysyłanych wiadomości przez każdego z nadawców. System integracyjny nie faworyzował żadnego; prędkość wysyłania przez większość czasu trwania badania wynosiła około 1150 wiadomości na sekundę.

Na rysunku 3D porównano średnią liczbę komunikatów, które każdy z odbiorców pobierał z kolejki. Podczas badania nie została utracona żadna wiadomość (rys. 3E). Komunikaty wysłane jako ostatnie oczekiwały na odebranie ponad 60s – system najpierw zajmował się odbiorem i utrwaleniem wiadomości (rys. 3F). Większość wiadomości została dostarczona niezgodnie z kolejnością wysłania (rys. 3G).

Na rysunku 3H i 3I zaprezentowano średni transfer dla nadawcy i odbiorcy komunikatów. Wysyłanie danych w standardzie AMQP nie zwiększa zauważalnie rozmiaru wiadomości, ponieważ średnia prędkość danych wychodzących w początkowym etapie badania wynosiła 1700 KB/s, co odpowiada wysłaniu 1133 wiadomości o wadze 15 KB każda.

Parametry wydajnościowe klastra obliczeniowego (3 instancje brokerów), zostały przedstawione na rys. 3J i 3K. Na ich podstawie można sprawdzić poziom zapotrzebowania na moc obliczeniową i wewnętrzną pamięć RAM w poszczególnych etapach badania. W momencie napływania nowych komunikatów zajętość procesora każdej z maszyn wynosiła powyżej 95%. W drugiej fazie, gdy wiadomości z kolejki były tylko pobierane, zajętość procesora dwóch maszyn podążających za maszyną główną spadł poniżej 40%, podczas gdy procesor głównej maszyny, synchronizującej pracę pozostałych instancji, był zajęty w ponad 75% swojego czasu. W celu zwiększenia wydajności każdy z brokerów, oprócz zapisywania danych na dysk, przechowywał komunikaty przez pewien czas w pamięci RAM (wykorzystanie pamięci na granicy limitu fizycznego - rys. 3K).

Wyniki badań wskazały, iż najlepszym brokerem w modelu komunikacji publikuj-subskrybuj jest Apache Kafka z protokołem Kafka (liczba węzłów klastra: 3, utrwalanie wiadomości bez potwierdzania dostarczenia, liczba partycji dla tematu: 3).

Średnia prędkość odbierania wiadomości od nadawców przez warstwę pośredniczącą, wyniosła ponad 7500 wiadomości na sekundę (rys. 4A). Wynik ten jest kilka razy większy,

niż w przypadku ActiveMQ i RabbitMQ. Wysłanie wszystkich wiadomości zostało zakończone po 32,585 s od momentu rozpoczęcia badania. Na rysunku 4B można zaobserwować duży wzrost prędkości odczytywania wiadomości od momentu zakończenia procesu ich wysyłania. Jest to związane z zapisywaniem odbieranych danych i synchronizacją tej czynności w czasie rzeczywistym przez wszystkie systemy w klastrze. Wszystkie wiadomości dotarły do każdego z odbiorców (rys. 4E). Rysunek 4F przedstawia średnie opóźnienie dla wiadomości wysłanych w kolejnych odstępach czasu. Im później nadawca wysłał wiadomość, tym dłużej oczekiwała ona na dostarczenie w warstwie integracyjnej. Większość wiadomości została odebrania niezgodnie z kolejnością nadania (rys. 4G).

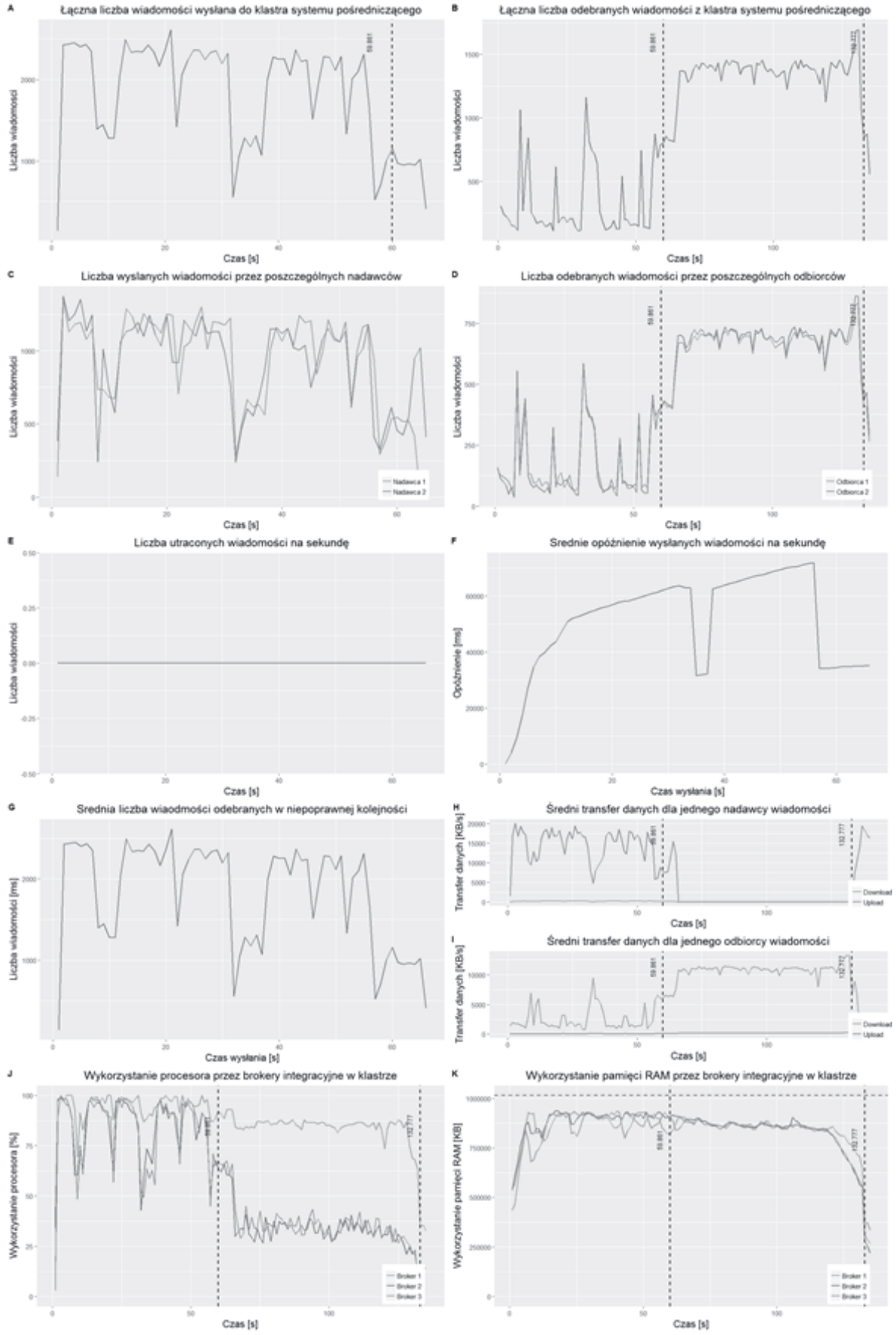
Rysunki 4H oraz 4I przedstawiają średni transfer dla nadawcy i odbiorcy. W końcowej fazie badania natężenie strumienia danych przychodzących do odbiorcy wynosiło około 40000 KB, co odpowiada 2667 wiadomościom o wadze 15 KB wysłanych w czasie jednej sekundy. Wartość ta pasuje do danych z rysunku 4D, z czego można wnioskować, że transfer wiadomości za pomocą protokołu Kafka nie powoduje dodatkowych danych obciążających łącze.

Pomimo dużego natężenia wiadomości, średnie użycie procesora w tym czasie dla każdej z maszyn systemu pośredniczącego nie przekroczyło 70% (rys. 4J). Dodatkowo, warto zwrócić uwagę na rysunku 4K, który przedstawia wysokie, dochodzące już do fizycznego limitu, zapotrzebowanie na wewnętrzną pamięć RAM dla każdego z węzłów.

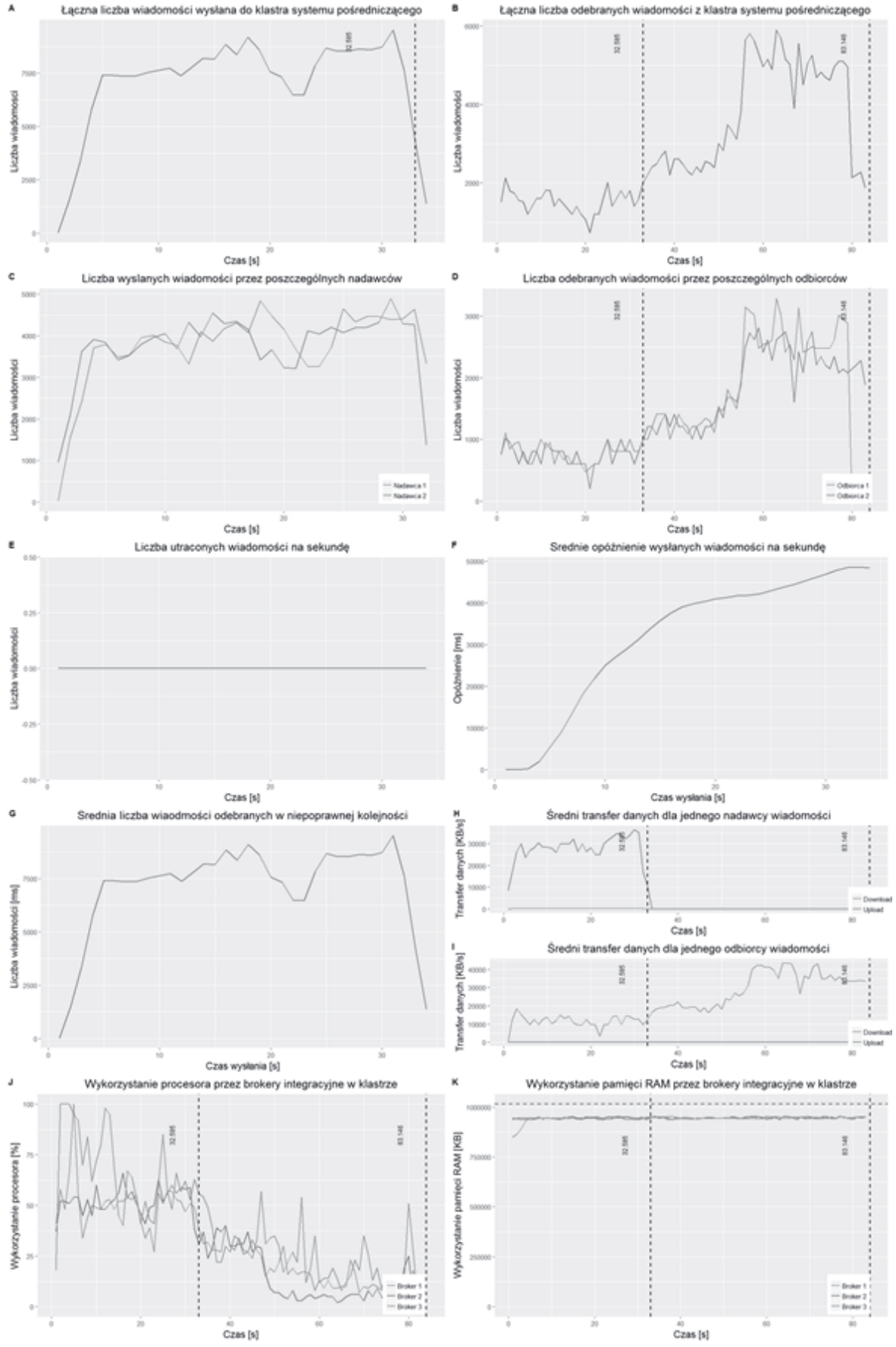
Wykorzystując model punkt-punkt najkrótszy czas przesłania wszystkich komunikatów uzyskała platforma integracyjna zbudowana z wykorzystaniem brokera RabbitMQ i protokołu AMQP. Nieco wolniejszy okazał się system wykorzystujący brokery integracyjne ActiveMQ i protokół OpenWire.

Dla modelu publikuj-subskrybuj bezkonkurencyjny okazał się system, oparty o broker Apache Kafka. Drugi najszybszy czas osiągnęła platforma zbudowana z brokerów RabbitMQ i wykorzystująca protokół AMQP. Broker ActiveMQ wraz z protokołem OpenWire zajął trzecie miejsce. Najgorsze rezultaty osiągnęły RabbitMQ i ActiveMQ w połączeniu z protokołem MQTT. Jest to prawdopodobnie związane z faktem, że protokół ten przeznaczony jest do transportu wiadomości o rozmiarze kilkadziesiąt razy mniejszym niż wykorzystane w testach komunikaty.

Opisane wyżej badania zostały wykonanie w środowisku produkcyjnym firmy Amazon. Wszystkie instancje serwerów były zlokalizowane w jednej, zamkniętej serwerowni w Irlandii. Wyniki przeprowadzonych testów mogą się różnić w przypadku wykonania ich w środowisku, w którym komunikujące się wzajemnie aplikacje oddzielone będą zaporą ogniową.



Rys. 3. Wyniki pomiarów brokera RabbitMQ, model komunikacji punkt-punkt
Źródło: Opracowanie własne



Rys. 4. Zestaw charakterystyk dla brokera Apache Kafka – model publikuj-subskrybuj
Źródło: Opracowanie własne

Aplikacje klienckie zostały napisane w języku Java, z wykorzystaniem frameworka Spring Integration. Wykorzystanie innej technologii również może wpłynąć na uzyskane wyniki.

W ramach eksperymentu pomiędzy aplikacjami przesyłano komunikat o rozmiarze 15 KB. Zmiana rozmiaru komunikatu wpłynie na szybkość przesyłania danych oraz pozostałe metryki wydajnościowe.

4. Podsumowanie

W pracy zaproponowano architekturę wydajnej i skalowalnej platformy integracyjnej, umożliwiającej niezależność i heterogeniczność aplikacji klienckich. W rozwiązaniu możliwa jest wymiana brokera integracyjnego, jak również zmiana protokołu komunikacyjnego bez potrzeby ponownej kompilacji aplikacji klienckich. Zaproponowana architektura umożliwia zmianę domyślnego trybu pracy każdego z brokerów na prace zespołową, w klastrze obliczeniowym.

W pracy przedstawiono również wyniki analizy porównawczej wydajności różnych implementacji zaproponowanej platformy integracji. Implementacje te dowodzą, iż za pomocą komponentów dostępnych w ramach licencji Open Source można zrealizować wydajny i skalowalny system integracji.

Najlepsze systemy integracyjne składają się z brokera integracyjnego RabbitMQ i protokołu AMQP − dla modelu przesyłania komunikatów typu punkt-punkt, oraz z brokera Apache Kafka z własnym, dedykowanym protokołem − dla modelu publikacji i subskrypcji, przy założeniu, ze wymieniane komunikaty są określonej wielkości. Ewentualna rezygnacja z funkcjonalności utrwalania wiadomości wewnątrz warstwy pośredniczącej oraz odbieranie komunikatów bez wysyłania potwierdzenia dostarczenia zwiększa wydajność całego systemu, pozwalając w rezultacie na obsługę większej liczby wiadomości. Decyzja ta jest jednocześnie punktem kompromisu, ponieważ skutkuje obniżeniem niezawodności systemu.

Możliwości konfiguracyjne oraz wzajemna niezależność i elastyczność komponentów zaproponowanej architektury czynią ją odporną na awarie oraz łatwą w dostosowaniu do istniejących procesów biznesowych przedsiębiorstwa, zapewniając przy tym najwyższą jakość interoperacyjności pomiędzy użytkownikami, urządzeniami mobilnymi, usługami w chmurze oraz innymi, szeroko pojętymi, zasobami informacji.

Osobną wartością są wygenerowane na podstawie badań grupy wykresów, które w kompletny sposób przedstawiają charakterystykę platform integracyjnych.

Przedstawioną w pracy architekturę systemu integracyjnego wraz z przykładem jej realizacji można w przyszłości rozszerzyć w kierunku automatycznej skalowalności, czyli

samoczynnego dodawania kolejnych fizycznych instancji brokerów komunikacyjnych w zależności od natężenia wiadomości i wykorzystywanych zasobów.

Bibliografia

1. Alonso G., Casati F., Kuno H., Machiraju V.: Web Services: Concepts, Architectures and Applications, Springer Publishing Company, Heidelberg 2010.
2. Chappell D.A.: Enterprise Service Bus., O'Reilly Media, Sebastopol 2004. p. 21−30, p. 43−56, p. 99−118.
3. Fowler M.: Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Boston 2002, p. 12−19, p. 63, p. 75−82, p. 528−529.
4. Kale V.: Guide to Cloud Computing for Business and Technology Managers: From Distributed Computing to Cloudware Applications. Chapman and Hall/CRC, 2014.
5. Kaur P. D., Kaur A., Kaur A.: Performance Analysis in Bigdata. "International Journal of Information Technology and Computer Science" (IJITCS), Vol. 7, 2015, p. 55–61.
6. Treat T.: Benchmarking message queue latency: http://bravenewgeek.com/benchmarking-message-queue-latency/. (access: 1.08.2017).
7. Verissimo, P., Rodrigues L.: Distributed Systems for System Architects. Norwell, MA, USA : Kluwer Academic Publishers 2001.