Zeszyty Naukowe WWSI, No 31, Vol. 18, 2024, s. 7-40

DOI: 10.26348/znwwsi.31.7

Analiza porównawcza możliwości i ograniczeń wykorzystania silników Apache Flink, Apache Spark oraz Apache Storm w strumieniowym przetwarzaniu danych

Tomasz Waksmundzki[*], Dariusz Chaładyniak[**]
Warszawska Wyższa Szkoła Informatyki, Polska

Streszczenie

Artykuł przedstawia analizę porównawczą trzech silników do przetwarzania strumieniowego danych: Apache Flink, Apache Spark i Apache Storm. Obejmuje ona kryteria porównawcze, takie jak: architektura, interfejsy, tryby przetwarzania, tryby uruchomieniowe, niezawodność, skalowalność, wydajność, źródła i ujścia danych. W ramach badań przeprowadzono serię eksperymentów, w których każdy silnik był testowany w czasie rzeczywistym podczas realizacji zadania detekcji anomalii w pomiarach parametrów środowiskowych. Eksperymenty obejmowały analizę wpływu liczby czujników, rozmiaru okna agregującego oraz obciążenia systemu na opóźnienia w generowaniu ostrzeżeń, zużycie zasobów (CPU, pamięć RAM) i liczbę wygenerowanych ostrzeżeń przez każdy z silników. Wnioski z badań dostarczają informacji na temat efektywności i przydatności każdego z analizowanych silników w kontekście przetwarzania strumieniowego danych, zwłaszcza w zastosowaniach związanych z detekcją anomalii środowiskowych.

Słowa kluczowe **–** przetwarzanie strumieniowe, Apache Flink, Apache Spark, Apache Storm

[*] E-mail: tomasz.waksmundzki@outlook.com

[**] E-mail: dchalad@ms.wwsi.edu.pl

Zgłoszono do druku 25 września 2024 r.

1. Wstęp

Dynamiczny rozwój nauk informatyczno-technicznych powoduje lawinowy wzrost złożoności i ilości danych, co stanowi nowe wyzwanie zarówno dla środowisk badawczych, jak i dla firm technologicznych. Ilość generowanych danych w czasie rzeczywistym jest ogromna i już teraz w Internecie przekracza ona 2,5 EB dziennie [1]. Ze względu na charakter strumieni danych, przetwarzanie wsadowe, czyli przetwarzanie danych w skończonych, zmagazynowanych partiach, w czasie dalekim od czasu rzeczywistego, może okazać się niewystarczające, co prowadzi do wykorzystania paradygmatu jakim jest przetwarzanie strumieniowe danych.

Przetwarzanie strumieniowe, nazywane również przetwarzaniem w czasie rzeczywistym, polega na ciągłej analizie strumienia danych bez potrzeby ich gromadzenia w celu wykrycia występujących prawidłowości czy wzorców zgodnych z określonymi założeniami, które zostały uwzględnione podczas projektowania systemu przetwarzania danych strumieniowych [2]. Stosowanie przetwarzania strumieniowego jest obligatoryjne nie tylko w przypadkach, gdy przechowywanie danych do analizy jest niemożliwe, ale także gdy potencjał wzrostu ilości danych w przyszłości rośnie w danym systemie (np.: dołączanie kolejnych urządzeń IoT) lub w przypadku, gdy zdarzenia w sposób naturalny będą napływać w sposób nieskończony (np.: dane meteorologiczne, finansowe, telekomunikacyjne). Specyfika przetwarzania strumieniowego pozwala na jego szerokie stosowanie w wielu dziedzinach i zastosowaniach takich jak np.: wykrywanie oszustw oraz nieprawidłowości, wykrywanie anomalii oraz odchyleń, analiza sieci społecznościowych, analiza danych medycznych oraz analiza łańcuchów dostaw oraz logistyki.

Przetwarzanie strumieniowe danych wykorzystuje różnorodne narzędzia i technologie, które umożliwiają skuteczne przetwarzanie danych w czasie rzeczywistym. W niniejszym artykule skupiono się na trzech, popularnych, otwarto-źródłowych silnikach przetwarzania strumieniowego:

- Apache Flink – rozproszony silnik przetwarzania danych, umożliwiający obliczenia stanowe nad nieograniczonymi i ograniczonymi strumieniami danych, działający w różnych środowiskach, oferujący wydajne obliczenia w pamięci podręcznej na dowolną skalę, wykorzystywany m.in. przez Alibaba, AWS, Ericsson, eBay, Pinterest, ResearchGate, Uber, Zalando [3].

- Apache Spark – zintegrowany silnik analizy danych do przetwarzania dużych zbiorów danych, oferujący wysokopoziomowe API oraz zoptymalizowany silnik, umożliwiający także przetwarzanie strumieniowe, wykorzystywany przez m. in. Amazon, AutoDesk, DataBricks, eBay, NASA [4].
- Apache Storm – rozproszony system do wykonywania obliczeń w czasie rzeczywistym, umożliwiający niezawodne przetwarzanie strumieni danych bez ograniczeń, wykorzystywany przez m. in. Yahoo, Twitter, Spotify, Alibaba [5].

2. Analiza porównawcza

2.1. Architektura

Klient (ang. *Client*), menadżer pracy (ang. *JobManager*) oraz menadżer zadań (ang. *TaskManager*) to komponenty architektury Apache Flink [6]. Zadaniem klienta jest przekształcenie kodu programu w graf przepływu danych, którego wykonywanie jest zarządzane w sposób rozproszony przez JobManagera. Głównym zadaniem JobManagera jest koordynacja i dystrybucja zadań do poszczególnych TaskManagerów w taki sposób, aby zapewnić skalowalność i wydajność podczas przetwarzania danych w ramach dostępnych zasobów, a ponadto odpowiada za zarządzanie pamięcią. Kontroluje także proces realizacji zadań, wykrywa i reaguje na wystąpienie błędów. W ramach swoich zadań, JobManager zarządza procesem tworzenia punktów kontrolnych (ang. checkpointing). TaskManager jest elementem środowiska Apache Flink odpowiedzialnym za wykonywanie konkretnych zadań zleconych przez JobManagera. W ramach jednego TaskManagera mogą być uruchomione zadania, których liczba zależy od poziomu zrównoleglenia procesu przetwarzania. Poza realizacją zadań, TaskManager komunikuję się z innymi węzłami w celu realizacji zadań w ramach przetwarzania rozproszonego.

Platforma Apache Spark w swojej architekturze zawiera takie składowe jak: SparkContext, Cluster Manager oraz Worker Node [7]. Kontekst Sparka (ang. *SparkContext*) jest centralnym elementem koordynującym działanie aplikacji Spark i jest obiektem sterownika (ang. *Driver program*) wykorzystywanym w całym cyklu życia aplikacji. W przypadku wykorzystywania Sparka w klastrze, SparkContext współpracuje z menedżerem klastra (ang. Cluster Manager), którego zadaniem jest alokacja zasobów zgodnie z zastosowaną konfiguracją. Węzeł roboczy (ang. *Worker Node*) jest węzłem, który uruchamia procesy (ang. *Executors*) i realizuje faktyczne

zadania przetwarzania danych, a także ich przechowywania. Węzeł roboczy łączy się z programem sterującym oraz menedżerem klastra, aby koordynować przetwarzanie danych i negocjować dostęp do zasobów [8].

Architektura Apache Storm opiera się na kilku elementach tj. Nimbus, Supervisor, Worker Process oraz Zookeeper [9]. Węzeł główny (Nimbus) jest kluczowym komponentem systemu Apache Storm i odpowiada za koordynację i delegowanie zadań do Supervisors. Nimbus jest odpowiedzialny za zarządzanie i monitorowanie topologii, która opisuje, w jaki sposób dane powinny być przetwarzane w systemie. Węzły robocze to proces uruchamiany na każdym węźle w klastrze Storm. Supervisor odpowiada za uruchamianie i monitorowanie procesów roboczych (Worker Process), które wykonują konkretne czynności w oparciu o zadanie przydzielone przez Nimbusa. Worker Process jest odpowiedzialny za wykonywanie pojedynczych, niezależnych operacji na danych [9]. Bezstanowość węzłów Storma powoduje potrzebę wykorzystania Zookeepera.

Analizując powierzchniowo architektury omawianych platform można dojść do wniosku, że są one bardzo podobne do siebie ze względu na występujący podział komponentów na dwie główne grupy: komponent zarządzający (JobManager, Spark Driver, Nimbus) i komponenty robocze (TaskManager, Executor, Supervisor). Mimo podobieństwa należy pamiętać, że każdy z omawianych silników ma indywidualne podejście do rozwiązywania problemów przetwarzania strumieniowego opracowane na etapie ich projektowania.

2.2. Interfejsy

Apache Flink, Apache Spark oraz Apache Storm udostępniają szeroki zakres interfejsów w różnych językach programowania. Wszystkie platformy wspierają tworzenie aplikacji w językach Java, Scala oraz Python. Ponadto Apache Spark wyróżnia się poprzez udostepnienie interfejsów dla języka R, natomiast Apache Storm oferuje wielojęzykowy protokół, który pozwala na wykorzystywanie topologii Storma w językach takich jak JavaScript, Ruby czy C#.

Pod kątem liczby dostępnych interfejsów, twórcy Apache Flink dążyli do ograniczenia ich liczby, argumentując to ułatwieniem wyboru od strony użytkownika. Równie istotna była chęć realizacji pełnej unifikacji między przetwarzaniem danych strumieniowych jak i danych wsadowych. W przypadku Apache Spark dostępna jest

szeroka gama różnych interfejsów co z jednej strony pozwala świadomym użytkownikom wybrać potencjalnie najefektywniejsze rozwiązania, ale z drugiej strony nowi użytkownicy mogą być przytłoczeni liczbą dostępnych opcji. Apache Storm udostępnia cztery interfejsy, w tym dwa oznaczone jako eksperymentalne, co oznacza, że mogą wystąpić problemy z kompatybilnością wsteczną.

Każdy z silników udostępnia interfejs, który gwarantuje, że dane zdarzenie zostanie przetworzone dokładnie raz (ang. exactly once). Wyszczególnić można następujące gwarancje przetwarzania:

- co najmniej raz (ang. *at least once*) – zdarzenie jest przetwarzane jeden lub więcej razy;
- dokładnie raz (ang. *exactly once*) – zdarzenie jest przetwarzane dokładnie raz;
- najwyżej raz (ang. *at most once*) – zdarzenie jest przetwarzane nie więcej niż jeden raz (może także zostać nieprzetworzone).

W zależności od realizowanego zadania, uniknięcie duplikatów jak i utraty zdarzeń może być warunkiem obligatoryjnym poprawnego działania systemu przetwarzania strumieniowego. W przypadku Apache Flink zachowanie semantyki przetwarzania dokładnie raz leży u podstaw działania silnika i jest ściśle powiązana z mechanizmami odporności na błędy zastosowane w architekturze Apache Flink. Podobnie jest w przypadku Apache Spark, który również wspiera tę semantykę [8]. Twórcy Apache Storm wprowadzili gwarancje przetworzenia zdarzenia dokładnie raz w interfejsie Trident, co jest jego podstawową przewagą nad podstawowym interfejsem Storma.

Istotnym aspektem zauważalnym u wszystkich omawianych platform jest udostępnienie interfejsu, który umożliwia posługiwanie się zapytania SQL w celu rozwiązania problemów przetwarzania strumieniowego. Apache Flink oferuje SQL/Table API, czyli interfejs wysokiego poziomu, który zalecany jest w przypadku zadań niewymagających jawnej kontroli nad grafem wykonawczym, stanami czy operacjami. Apache Spark oferuje zaawansowany interfejs Structured Streaming zoptymalizowany pod kątem analizy danych strumieniowych. W przypadku Apache Storm dostępny jest interfejs Storm SQL, który na ten moment oznaczony jako eksperymentalny, zawiera sporo ograniczeń, co zawęża zakres zadań, w których może zostać użyty.

Decyzja dotycząca wyboru interfejsu w przypadku omawianych silników przetwarzania danych strumieniowych powinna być podejmowana na podstawie szczegółowej analizy specyfiki zadań, które system ma realizować. Interfejsy Apache Flink, Apache Spark oraz Apache Storm mają różne cechy, możliwości i ograniczenia, dlatego analiza jest ważnym elementem opracowywania systemu. Dobranie odpowiedniego interfejsu pozwoli na zbudowanie systemu w sposób optymalny.

2.3. Tryby przetwarzania

Podczas planowania architektury systemu przetwarzania danych ważne jest uwzględnienie trybu, w którym dane będą procesowane tj. [10]:

- tryb przetwarzanie danych w czasie rzeczywistym (ang. *continuous streaming*);
- tryb przetwarzanie danych w czasie zbliżonym do rzeczywistego (ang. *micro-batching*);
- tryb przetwarzanie danych w trybie batch (ang. *batch processing*) w czasie późniejszym.

Przetwarzanie danych w czasie rzeczywistym wymagane jest w przypadku, gdy na podstawie zdarzeń oczekiwany jest natychmiastowy rezultat oraz brak opóźnień. Przetwarzanie w czasie prawie rzeczywistym, znane także jako micro-batching, polega na przetwarzaniu danych w mniejszych porcjach, niż w przypadku tradycyjnego przetwarzania wsadowego. Dzięki temu umożliwia ono uzyskanie danych w czasie zbliżonym do rzeczywistego, z nieznacznym opóźnieniem. Ten sposób przetwarzania jest często aktywowany w określonych interwałach czasowych, np. co dwie minuty lub na podstawie innych kryteriów, takich jak osiągnięcie określonego rozmiaru partii danych. W praktyce, różnica między przetwarzaniem w czasie rzeczywistym i prawie rzeczywistym jest niewielka, co prowadzi do używania tych pojęć zamiennie w opisach architektury systemów przetwarzania danych oraz oprogramowania. Przetwarzanie wsadowe polega na gromadzeniu danych nieprzetworzonych w pamięci oraz ich przetwarzaniu w pewnych odstępach czasu. Ten tryb przetwarzania wiąże się z opóźnieniami między otrzymaniem zdarzenia a rezultatem jego analizy.

Warto dodać, że Apache Flink udostępnia ujednolicone interfejsy API do obsługi nie tylko przetwarzania strumieniowego, ale także przetwarzania wsadowego, co umożliwia elastyczne przejście między różnymi trybami, w zależności od charakteru danych. Apache Spark podobnie jak Apache Flink pozwala na przetwarzanie strumieniowe i wsadowe danych dzięki wykorzystaniu abstrakcji dyskretnego strumienia

(DStream). W przeciwieństwo do poprzednich dwóch omawianych silników, silnik Apache Storm projektowany był do realizacji zadania przetwarzania danych w czasie rzeczywistym [8, 11].

2.4. Niezawodność

Niezawodność (ang. *reliability*) w kontekście przetwarzania strumieniowego można zdefiniować jako zdolność systemu do ciągłego i spójnego przetwarzania danych strumieniowych. Niezawodność odgrywa kluczową rolę w systemach analizujących dane w czasie rzeczywistym. Jej utrata, nawet chwilowa, może prowadzić do nieprzewidywalnych skutków, wśród których może pojawić się utrata danych lub zatrzymanie procesu przetwarzania. W przypadku dużych systemów przetwarzania strumieniowego, taka sytuacja może spowodować zatrzymanie głównych ścieżek przetwarzania oraz negatywnie wpłynąć na inne komponenty infrastruktury.

W koncepcji przetwarzania strumieniowego można wyszczególnić dwa podejścia: podejście stanowe (ang. *stateful*) oraz podejście bezstanowe (ang. *stateless*) [12]. Operator, czyli komponent realizujący funkcje strumieniowego przetwarzania danych, pełniący role o charakterze bezstanowym, przetwarza kolejne rekordy wejściowe a wynik zdefiniowanych operacji w ramach zadań operatora określa w oparciu o bieżący, aktualnie analizowany rekord. Operator stanowy określa wynik zdefiniowanych w jego obrębie operacji na podstawie aktualnie analizowanego rekordu, ale bierze również pod uwagę poprzednio przetwarzane rekordy oraz stan całego systemu. W przetwarzaniu stanowym kluczowe jest monitorowanie zapisanego stanu podczas analizy danych.

Niezawodność w rozwiązaniach przetwarzania strumieniowego, takich jak Apache Flink, Apache Spark i Apache Storm, jest kluczowym aspektem dla utrzymania stabilności i ciągłości działania aplikacji przetwarzania strumieniowego. Wszystkie te platformy oferują zaawansowane mechanizmy niezawodności, aby zapewnić spójność przetwarzania nawet w przypadku wystąpienia awarii czy błędów. Punkty kontrolne stanowią kluczowy mechanizm w każdym z tych silników, umożliwiając przywrócenie stanu systemu do momentu sprzed awarii. Choć ich wewnętrzne implementacje mogą się różnić, mają one wspólny cel: zapewnienie ciągłości przetwarzania danych w obliczu potencjalnych awarii czy błędów. Każdy z tych silników udostępnia także mechanizmy wysokiej dostępności węzła głównego, aby zapewnić

ciągłość działania klastra w przypadku awarii lidera. Wysoka dostępność realizowana poprzez replikacje i odpowiednie zarządzanie instancjami przez Apache Zookeeper'a. Ze względu na zaawansowane mechanizmy niezawodności i wysokiej dostępności, Apache Flink, Apache Spark i Apache Storm są zdolne do efektywnego przetwarzania strumieniowego danych w środowiskach produkcyjnych wymagających nieprzerwanego działania.

2.5. Tryby uruchomieniowe

Platformy Apache Flink, Apache Spark i Apache Storm oferują dedykowane tryby zarówno dla środowisk testowych, jak i produkcyjnych. W trybach deweloperskich lub testowych umożliwiają szybkie uruchamianie aplikacji w kontrolowanym środowisku, eliminując konieczność korzystania z rzeczywistych danych i zasobów produkcyjnych. To pozwala na efektywne testowanie aplikacji w różnych konfiguracjach, bez potrzeby uruchamiania oddzielnego klastra. W trybach dedykowanych środowiskom produkcyjnym silniki pozwalają na równoczesne uruchamianie kilku aplikacji w jednym klastrze, ale wymaga to zbadania wymagań każdej aplikacji oraz dostępności zasobów klastra, aby osiągnąć optymalną wydajność. W tabeli 1 kompleksowo zestawiono tryby uruchomieniowe dla każdej z platform w zależności o typu środowiska.

Tabela 1. Zestawienie trybów uruchomieniowych z podziałem na środowiska

	Apache Flink	**Apache Spark**	**Apache Storm**
Środowisko testowe	Tryb aplikacji	Tryb klienta	Tryb lokalny
Środowisko produkcyjne	Tryb sesji	Tryb klastowy	Tryb zdalny

Wdrożenie aplikacji każdego z silników wymaga skonfigurowania klastra co może odbywać na różne sposoby w zależności od postawionych wymagań i założeń. Apache Flink, Apache Spark oraz Apache Storm umożliwiają integrację z takimi narzędziami do zarządzania zasobami jak: Docker, Kubernetes oraz Apache Hadoop YARN [1, 8].

W dobie dynamicznego rozwoju usług chmurowych, popularnym rozwiązaniem jest wdrażanie aplikacji przetwarzania strumieniowego bez potrzeby konfiguracji

własnego środowiska wykonawczego na maszynie wirtualnej. Zamiast tego udostępniane są usługi zoptymalizowane pod kątem zadań obliczeniowych skupionych na przetwarzaniu danych strumieniowych. W tabeli 2 przedstawiono zestawienie wybranych usług chmurowych wspierających wdrożenie aplikacji Apache Flink, Apache Spark oraz Apache Storm.

Tabela 2. Zestawienie wybranych usług chmurowych wspierających wdrażanie aplikacji Apache Flink, Apache Spark oraz Apache Storm [3, 4]

Dostawcy	Usługi	Apache Flink	Apache Spark	Apache Storm
AWS	Amazon EMR	✓	✓	-
	Amazon Kinesis Data Analytics	✓	-	-
Azure	HDInsight	-	✓	-
	Databricks	-	✓	-
GCP	DataProc	✓	✓	-
Alibaba Cloud	Realtime Compute for Apache Flink	✓	-	-
Cloudera	Cloudera Streaming Analytics	✓	-	-

Zestawienie wybranych usług oferowanych przez dostawców rozwiązań chmurowych pokazuje różnice w poziomie wsparcia pomiędzy Apache Storm a Apache Flink czy Apache Spark. Apache Storm cechuje się najgorszym wsparciem, jeżeli chodzi o gotowe, chmurowe rozwiązania pozwalające na wdrożenie aplikacji strumieniowych.

2.6. Źródła i ujścia danych

Źródła (ang. *source*) i ujścia (ang. *sink*) danych są podstawowymi komponentami uzupełniającymi system przetwarzania strumieniowego. Ich dobór jest kluczowy i może diametralnie wpływać na wydajność analizy danych. Źródło danych to miejsce, z którego system przetwarzania strumieniowego może pobrać dane do analizy. Głównym celem źródeł danych jest dostarczenie nowych danych do strumienia przetwarzania. Źródła danych mogą być generatorami nowych danych (np. bezpośredni odczyt pomiarów z czujników IoT), ale także pełnić funkcje magazynu lub agregatu, w którym dane z rozproszonych pierwotnych źródeł są zbierane, gromadzone i utrzymywane w spójny sposób. Ta zdolność do gromadzenia danych z wielu źródeł i utrzymania spójności jest szczególnie ważna w przypadku przetwarzania strumieniowego, gdzie dane napływają w czasie rzeczywistym, często w nieregularnych interwałach. Ujście danych to miejsce, do których system przetwarzania strumieniowego przesyła wyniki analizy lub przetworzone dane. Ich głównym celem jest przyjmowanie danych z przetwarzania strumieniowego i przechowywanie, wizualizacja lub przekierowywanie ich do innych systemów lub aplikacji. W dalszej części tego rozdziału zostały omówione najpopularniejsze źródła oraz ujścia danych wykorzystywane w przetwarzaniu strumieniowym danych w podziale na następujące grupy: systemy i formaty danych, bazy i magazyny danych, brokerzy wiadomości oraz inne technologie i narzędzia.

HDFS (ang. *Hadoop Distributed File System*) to rozproszony system plików będący integralną częścią projektu Apache Hadoop cechujący się wysoką odpornością na awarie, wysoką przepustowością dostępu do danych oraz możliwością magazynowania dużych zbiorów danych. Obiektowa pamięć masowa to technologia pozwalająca przechowywać i zarządzać danymi w postaci obiektów. Jest to nowoczesny rodzaj pamięci masowej, który różni się od tradycyjnych systemów plików, które organizują dane w hierarchiczne struktury folderów i plików. W obiektowej pamięci masowej dane są przechowywane jako niepodzielne jednostki zwane obiektami, które zawierają zarówno same dane, jak i metadane. Jedną z podstawowych zalet obiektowej pamięci masowej jest bardzo wysoka trwałość oraz odporność danych, a także wysoki poziom skalowalności co przyczynia się do jej szerokiego wykorzystywania w rozwiązaniach chmurowych. Apache Flink oraz Apache Spark cechują się dużą elastycznością wykorzystania obiektowej pamięci podręcznej jako źródła albo ujścia danych [3, 4, 5], natomiast w dokumentacji Apache Storm brak jest

informacji na temat możliwości wykorzystania chmurowych rozwiązań w postaci obiektowej pamięci masowej.

Wszystkie omawiane silniki przetwarzania strumieniowego wspierają integracje z takimi formatami danych jak format tekstowy, JSON, CSV oraz Avro, a dodatkowo Apache Flink oraz Apache Spark współpracują także z Apache Orc oraz Apache Parquet [3, 4, 5]. Apache Flink, Apache Spark i Apache Storm umożliwiają szeroką integrację z wieloma bazami danych, zarówno relacyjnymi (np. MySQl, Postgres, Microsoft SQL Server, Oracle), jak i nierelacyjnymi (np. MongoDb, Apache Cassandra), oraz z popularnymi magazynami danych (np. Redis, Apache HBase) w kontekście przetwarzania strumieniowego [3, 4, 5].

Brokerzy wiadomości to systemy umożliwiający wymianę wiadomości między aplikacjami, co realizowane jest poprzez konwersje protokołów przesyłania komunikatów nadawcy oraz odbiory. są szeroko wykorzystywane w zadaniach przetwarzania strumieniowego danych ze względu na np.: możliwość asynchronicznej komunikacji, skalowalność oraz zapewnienie integralności danych. Wszystkie omawiane platformy przetwarzania strumieniowego integrują się z wieloma narzędziami realizującymi zadania brokera wiadomości takimi jak: Apache Kafka, RabbitMQ, Apache Pulsar, ActiveMQ [3, 4, 5].

Oprócz systemów plików, baz danych i brokerów wiadomości wyróżnić można szereg narzędzi i technologii takich jak: narzędzia do analizy danych (np. Elasticsearch), hurtowanie danych (np. Apache Hive) czy usługi przetwarzające dane (np. Azure Event Hubs, Amazon Kinesis) które z powodzeniem mogą być integrowane z systemami przetwarzania strumieniowego opartymi na Apache Flink, Apache Spark i Apache Storm [3, 4, 5].

2.7. Skalowalność i wydajność

Skalowalność jest pojęciem opisującym zdolność systemu do dostosowania się do zmiennego obciążenia. Apache Flink umożliwia ręczną konfiguracje skalowalności poprzez dobór odpowiedniej liczby menadżerów zadań (TaskManager) poprzez określenie liczby slotów na zadania w ramach pojedynczej instancji TaskManagera [3]. Konfiguracja równoległości realizowania zadań oraz dostępność zasobów wprost wpływa na wydajność całego systemu przetwarzania. Ponadto Apache Flink udostępnia mechanizm elastycznej skalowalności nazywany trybem reaktywnym,

który konfiguruje w taki sposób zadanie, aby zawsze korzystało z wszystkich dostępnych zasobów w klastrze i dąży do jak najwyższej równoległości w ramach dostępnych zasobów. Po automatycznym przeskalowaniu zadań, następuje wznowienie przetwarzania poprzez wykorzystanie punktów kontrolnych. Dodatkowo, tryb ten pozwala na implementowane zaawansowanych scenariuszy w których ma dojść do automatycznego przeskalowania systemu na podstawie metryk systemowych [3].

Elastyczne skalowanie jest eksperymentalną funkcją obsługiwaną tylko w kilku trybach wdrażania aplikacji, ale w przyszłości planowane jest usunięcie tych ograniczeń. Apache Spark został zaprojektowany w taki sposób, aby mógł łatwo skalować się w poziomie, co oznacza, że może być rozszerzany poprzez dodawanie kolejnych węzłów do klastra. Dzięki temu możliwe jest zwiększenie mocy obliczeniowej i pojemności pamięci masowej, co umożliwia przetwarzanie coraz większych ilości danych. Apache Spark charakteryzuje się dynamicznym mechanizmem planowania co pozwala na obsługę dużej liczb zadań jednocześnie przy zachowaniu małego opóźnienia [13]. Apache Storm realizuje zadania przetwarzania strumieniowego w sposób rozproszony. W celu osiągnięcia maksymalnej wydajności i mocy obliczeniowej stosuje zrównoleglenie realizacji zadań poprzez wykorzystanie skalowania poziomego oraz pionowego. Skalowanie poziome odbywa się poprzez zwiększenie liczby węzłów obliczeniowych w klastrze co pozwala zwiększyć moc obliczeniową w razie potrzeby. Skalowanie pionowe realizowane jest w ramach pojedynczego węzła przetwarzania i polega na wykorzystaniu kilku JVM, a także wykorzystywanie wielu wątków w ramach pojedynczej JVM [14].

W artykule [15] opisano rezultaty eksperymentu porównującego wydajność analizy reprezentatywnych zbiorów danych przy użyciu Apache Spark i Apache Flink, uwzględniając czas przetwarzania oraz wykorzystanie zasobów. Autorzy pokazali, że Spark jest około 1,7 razy szybszy, niż Flink w przypadku przetwarzania dużych zbiorów danych (rozmiar 1,2 TB), natomiast Flink sprawdza się lepiej przy przetwarzaniu małych grafów (rozmiar 13,7 GB). Przyczyną uzyskania takich wyników są różnice projektowe obu platform takie jak: podejście do optymalizacji czy zarządzania pamięcią.

W tabeli 3 przedstawiono agregacje rezultatów badań, który porównywały m. in. Apache Flink, Apache Spark oraz Apache Storm biorąc pod uwagę takie kategorie jak czas przetwarzania, zużycie pamięci obliczeniowej, opóźnienie podczas przetwarzania danych, wydajność i skalowalność.

Tabela 3. Zestawienie porównawcze silników Apache Flink, Apache Spark i Apache Storm pod kątem wydajności [6, 16, 17]

Kategoria	Studium przypadku	Apache Flink	Apache Spark	Apache Storm
Czas przetwarzania	Duży zestaw danych	długi	krótki	-
	Mały zestaw danych	krótki	długi	-
	Rozmiar klastra	długi	krótki	-
	Wysyłanie tweeta o rozmiarze 100KB w jednej wiadomości	krótki	długi	krótki
	Wysyłanie 5 tweetów o rozmiarze 500KB w jednej wiadomości	krótki	długi	długi
	Zbiór danych w formacie JSON	długi	krótki	-
	Przetwarzanie dużych grafów	duży	mały	-
	Ogromny zbiór danych	mały	duży	-
	Agregacja danych w oknach czasowych	krótki	długi	średni
	Agregacja danych w oknach czasowych (duża fluktuacja danych wejściowych)	krótki	krótki	średni
Zużycie CPU	Przetwarzanie strumieniowe – badanie 1	najmniejsze	średnie	największe
Zużycie CPU	Przetwarzanie strumieniowe – badanie 2	najmniejsze	duże	duże
Opóźnienie	Wykorzystanie różnych zestawów danych	-	małe	duże
Wydajność	Analiza tekstu: szukanie wzorca, zliczanie słów w oknie czasowym	większa	mniejsza	-
Skalowalność	Analiza tekstu: szukanie wzorca, zliczanie słów w oknie czasowym	lepsza	gorsza	

Na podstawie opublikowanych wyników przeprowadzonych eksperymentów przez różne grupy naukowe można wyciągnąć wnioski, które mogą okazać się pomocne w nakierowaniu zespołu projektowego na konkretny silnika przetwarzania strumieniowego:

- Apache Flink charakteryzuje się najmniejszym zużyciem pamięci obliczeniowej niż pozostałe omawiane platformy co świadczy o bardzo dobrej optymalizacji pod kątem realizacji analizy strumieniowej;
- Apache Spark lepiej radzi sobie z analizą dużych zbiorów danych niż Apache Flink, a w przypadku małych zestawów danych sytuacja ulega odwróceniu;
- Apache Flink osiąga wyższą wydajność i lepszą skalowalność niż Apache Spark w przypadku zadań opartych o analizę tekstów;
- czas przetwarzania zadania przez każdy silnik zależy od charakterystyki zestawu danych;
- Apache Flink radzi sobie lepiej z agregacją danych w oknach czasowych niż Apache Spark i Apache Storm.

Warto zaznaczyć, że ciężko jest jednoznacznie określić, który z silników jest najbardziej wydajny bez dokładnego zbadania danego przypadku. Czas przetwarzania zadania może się różnić w zależności od charakterystyki danych, takich jak rozmiar, złożoność, rodzaj i struktura. Ważne jest, aby wziąć pod uwagę specyfikę zadania i dostosować wybór platformy do jego wymagań. Należy przeprowadzić szczegółową analizę skupioną na takich parametrach jak: złożoność danych, wymagany maksymalny czas przetwarzania, dostępne zasoby obliczeniowe oraz skalowalność, aby dokonać najbardziej optymalnego wyboru.

3. Studium przypadku

Monitorowanie i kontrola parametrów środowiskowych, takich jak temperatura, wilgotność i ciśnienie w laboratoriach prowadzących eksperymenty fizyczne, chemiczne i biologiczne, są kluczowe pod kilkoma względami. Bezpieczeństwo pracowników stanowi priorytetową kwestię. Głównym celem tego systemu jest zapewnienie bezpieczeństwa pracowników laboratorium. Praca z niebezpiecznymi substancjami i urządzeniami może stwarzać ryzyko, dlatego ciągłe monitorowanie parametrów atmosferycznych pozwala na natychmiastową reakcję na wszelkie nieprawidłowości, które mogą zagrażać zdrowiu i życiu pracowników.

Kolejnym istotnym aspektem jest zachowanie stabilnych warunków eksperymentalnych. W kontekście badań fizycznych, chemicznych i biologicznych, stałe warunki atmosferyczne są niezbędne do uzyskania wiarygodnych i powtarzalnych wyników. Nawet niewielkie fluktuacje w temperaturze, wilgotności lub ciśnieniu mogą wpłynąć

na wyniki eksperymentów. Dlatego kontrola i monitorowanie tych parametrów pozwala utrzymać stałe warunki eksperymentalne.

Zgodność z normami i przepisami regulującymi pracę laboratoriów to kolejny istotny aspekt. W laboratoriach często obowiązują rygorystyczne przepisy i normy dotyczące warunków atmosferycznych w zależności od rodzaju badań prowadzonych w laboratorium. System monitorowania pomaga w spełnieniu tych wymogów, co ma kluczowe znaczenie zarówno dla zapewnienia bezpieczeństwa, jak i wiarygodności badań. Optymalizacja procesów badawczych jest dodatkowym atutem systemu. Analiza zebranych danych pozwala na identyfikację trendów i wzorców, co może prowadzić do usprawnienia eksperymentów, zwiększenia efektywności oraz oszczędności zasobów. Szybkie reagowanie na awarie jest kluczowe w przypadku nieprawidłowości w parametrach atmosferycznych. System monitorowania jest w stanie wykryć awarie lub nieprawidłowości i natychmiast powiadomić personel laboratorium, umożliwiając szybką reakcję i minimalizację strat.

3.1. Koncepcja

W ramach studium przypadku opracowano system wykrywający anomalie na podstawie pomiarów parametrów środowiskowych w laboratorium. W rzeczywistych systemach tego rodzaju dane pomiarowe pochodzą z czujników umieszczonych w monitorowanych pomieszczeniach. W symulowanym środowisku, jakie wykorzystano w tym przypadku, dane pomiarowe zostały generowane symulacyjnie, zgodnie z określonymi regułami. Symulowanie czujników pozwala na skalowanie w zakresie dostępnych zasobów, co zostało wykorzystane w przeprowadzonych eksperymentach.

Kluczowym aspektem jest określenie, jakie dane oraz w jakim formacie, czujniki powinny przekazywać do systemu monitorującego. Na rysunku 1 przedstawiono pojedynczą ramkę danych wejściowych (zdarzenie – obiekt Event) w formacie JSON z przykładowymi danymi.

Głównym zadaniem systemu jest weryfikacja, czy żaden z parametrów środowiskowych nie przekroczył założonych zakresów dopuszczalnych w środowisku laboratoryjnym. Na potrzebę studium przypadku założono następujące wartości graniczne:

- temperatura – wartość minimalna: 24°C, wartość maksymalna: 26°C;
- wilgotność – wartość minimalna: 60%, wartość maksymalna: 80%;
- ciśnienie – wartość minimalna: 1010hPa, wartość maksymalna: 1015hPa.

```
{
  "uuid": "e933fdaa-4a82-11ee-be56-0242ac120002",
  "sensorUuid": "4df90d0d-0a73-4d02-b084-476751c6ba68",
  "timestamp": ":1693071673",
  "eventType": "CONTROL",
  "measurement": {
    "temperature": 26.0,
    "humidity": 74.0,
    "pressure": 101200.0
  }
}
```


Rysunek 1. Zdarzenie wejściowe – obiekt Event w formacie JSON

Aby uniknąć alarmów ze strony systemu monitoringu w przypadku pojedynczych przekroczeń dopuszczalnych wartości, postanowiono agregować zbiór zdarzeń pochodzących z jednego czujnika i podejmować decyzje o wygenerowaniu alarmu na podstawie średnich wartości parametrów środowiskowych. W przypadku przekroczenia przez co najmniej jeden parametr (po uśrednieniu) założonych, dopuszczalnych wartości, system generuje dane wyjściowe (obiekt Alert) w formacie JSON, jak przedstawiono na rysunku 2.


```
{
  "uuid": "4a3ba942-4a81-11ee-be56-0242ac120002",
  "sensorUuid": "4df90d0d-0a73-4d02-b084-476751c6ba68",
  "timestampBegin": "1693071673",
  "timestampEnd": "1693072168",
  "measurementAvg": {
    "temperature": 26.1,
    "humidity": 71.0,
    "pressure": 101244.0
  },
  "events": {
    "sensorUuid": "4df90d0d-0a73-4d02-b084-476751c6ba68",
    "eventsList": [ ···
    ]
  }
}
```


Rysunek 2. Zdarzenie wyjściowy – obiekt Alert w formacie JSON

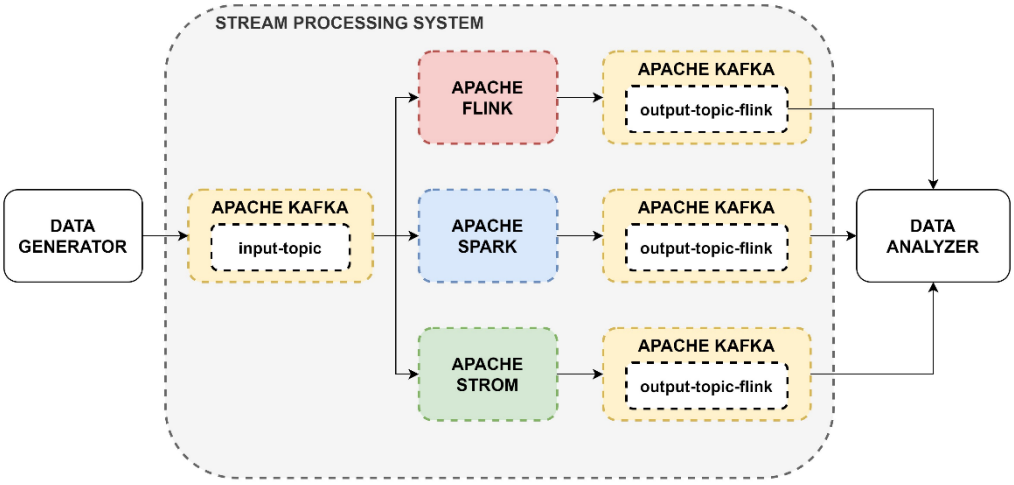
Opisany wyżej proces od pobrania danych wejściowych, poprzez ich analizę, aż do ewentualnego zwrócenia danych wyjściowych realizowany jest przez trzy, niezależne aplikacje oparte kolejno na silnikach: Apache Flink, Apache Spark oraz Apache Storm.

Takie podejście, dzięki zastosowaniu dokładnie takiego samego strumienia danych wejściowych, pozwala na ocenę każdego silnika pod kątem opóźnień, zużycia zasobów oraz poprawności realizacji zadania przetwarzania strumieniowego.

3.2. Zasada działania opracowanego systemu przetwarzania danych strumieniowych

Zaprojektowany system przetwarzania danych strumieniowych zbudowany jest z czterech komponentów: aplikacji Apache Flink, Apache Spark, Apache Storm oraz brokera wiadomości Apache Kafka.

Ponadto w celu symulowania danych wejściowych oraz na potrzebę analizy danych przetworzonych opracowano dwie aplikacje: generator danych oraz analizator danych, będące komponentami pomocniczymi. Na rysunku 3 przedstawiono diagram przepływu danych między komponentami systemu przetwarzania danych strumieniowych oraz komponentami pomocniczymi.



Rysunek 3. Diagram przepływu danych między komponentami systemu przetwarzania danych strumieniowych oraz komponentami pomocniczych

Apache Kafka pełni rolę źródła oraz ujścia danych dla każdej z aplikacji przetwarzania strumieniowego. Wybór Apache Kafka jako jednego z komponentów całego systemu jest motywowany nie tylko jej kompatybilnością z różnymi silnikami prze-

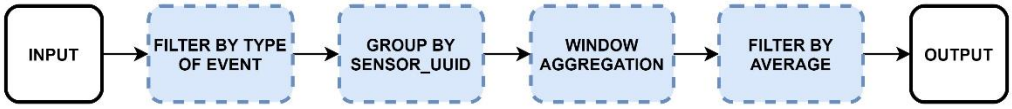
twarzania strumieniowego, ale także jej powszechnym zastosowaniem w rzeczywistych systemach przetwarzania danych tego typu. Ponadto, charakter danych w ramach tego eksperymentu idealnie wpisuje się w zalety, jakie Apache Kafka oferuje tzn.: mały rozmiar danych, wymagana wysoka przepustowość danych. W ramach Apache Kafka zostały utworzone topici:

- input-topic – topic wejściowy, wspólny dla wszystkich aplikacji;
- output-topic-flink – topic wyjściowy, przeznaczony tylko dla aplikacji Apache Flink;
- output-topic-spark – topic wyjściowy, przeznaczony tylko dla aplikacji Apache Spark;
- output-topic-storm – topic wyjściowy, przeznaczony tylko dla aplikacji Apache Storm.

Ze względu na niestosowanie architektury rozproszonej w omawianym studium przypadku, zdecydowano o użyciu pojedynczych partycji w ramach każdego topicu kafkowego, co pozwoliło na uzyskanie prostej konfiguracji. Ze względu na badawczy charakter systemu nie zapewniono mechanizmów minimalizujących utratę danych w przypadku awarii brokera Kafki.

Wszystkie aplikacje przetwarzania strumieniowego zostały stworzone w języku Java (wersja 11) przy użyciu środowiska IntelliJ IDEA. Aplikacje zostały opracowane w oparciu o następujące wersję silników: Apache Flink – 1.16.0, Apache Spark – 3.3.1, Apache Storm – 2.4.0. Jednym z kluczowych wymagań wynikający z przyjętego studium przypadku było zachowanie semantyki „przetwarzania dokładnie raz”. W tym kontekście, interfejsy DataStream API oraz Trident były optymalnym wyborem, ponieważ zapewniały właśnie taką semantykę. Jednak, w przypadku interfejsu Spark Streaming, zdecydowano się na użycie interfejsu Map, który pozwolił na wykrywanie zduplikowanych zdarzeń i ich odrzucanie. Kolejnym istotnym wymaganiem było dostarczenie operatorów i funkcji wykorzystywanych w przetwarzaniu strumieniowym, takich jak filtrowanie, grupowanie oraz agregacja okienkowa. Tylko interfejs DataStream API dostarczał wbudowany operator agregacji okienkowej oparty na liczbie zdarzeń, natomiast w przypadku interfejsów Trident i Spark Streaming opracowano własną implementacje tego operatora.

Aplikacje Apache Flink, Apache Spark oraz Apache Storm realizuje zadanie przetwarzania danych strumieniowych zgodnie z algorytmem przedstawionym na rysunku 4. Proces przetwarzania rozpoczyna się od odczytania zdarzenia z topicu wejściowego i jego konwersji do obiektu Event (rysunek 1). Następnym krokiem jest filtracja, w którym odrzucane są wszystkie zdarzenia o typie zdarzenia różnym od CONTROL, co realizowane jest na podstawie wartości pola *eventType*. Tak przefiltrowany strumień danych jest następnie grupowany według unikalnych identyfikatorów czujników, określanych przez pole *sensorUuid*. To prowadzi do dalszej analizy, która jest prowadzona niezależnie dla każdego czujnika. W kolejnym etapie przetwarzania, zdarzenia są agregowane w skokowym oknie licznikowym, co oznacza buforowanie określonej przez rozmiar okna liczby zdarzeń. Rozmiar tego okna jest określany na podstawie parametru przekazywanego do aplikacji podczas jej uruchamiania (domyśla wartość – 100). Po każdorazowym osiągnięciu tej ustalonej liczby zdarzeń w oknie, na podstawie zagregowanych danych, tworzony jest obiekt Alert (rysunek 2). Ostatnim etapem przetwarzania w opracowanym systemie jest filtracja końcowa, której zadaniem jest sprawdzenie, czy którykolwiek z parametrów środowiskowych (wartości uśrednione) przekroczył dopuszczalne wartości. W przypadku wykrycia takie anomalii, obiekt Alert przekazywany jest do odpowiedniego topicu w Apache Kafka.



Rysunek 4. Algorytm przetwarzania danych strumieniowych w systemie w kontekście studium przypadku

Aplikacje Apache Flink, Apache Spark i Apache Storm wykorzystywały domyślne konfiguracje. W przypadku aplikacji Apache Storm zdecydowano się na zmianę wartości parametru TOPOLOGY_TRIDENT_BATCH_EMIT_INTERVAL_MILLIS z 500 na 5. Zadaniem tego parametru jest ograniczenie częstotliwości emitowanie partii danych do kolejnych etapów przetwarzania. Jest to przydatny mechanizm w przypadku dużych strumieni danych, który pozwala zwiększyć przepustowość kosztem większego opóźnienie. W przypadku omawiane studium przypadku domyślna war-

tości tego parametru zwiększała znacząco opóźnienie przetwarzania mimo wystarczających zasobów i osiąganiu wysokiej przepustowości. Aby uzyskać najprostszą konfigurację wszystkich aplikacji przetwarzania strumieniowego, stopień przetwarzania równoległego ustawiono na wartość 1.

Generator danych został opracowany w języku Python jako narzędzie konfigurowalne, mające na celu pełne odwzorowanie rzeczywistego zachowania systemu oraz zachowanie wymaganego obciążenia danymi. Jego zadaniem jest pełna symulacja zachowań grupy czujników obecnych w rzeczywistych systemach tego typu. Symulacja zachowań grupy czujników polega na generowaniu oraz emisji danych o ustalonym formacie (rysunek 1) na topic wejściowy Kafki, w oparciu o dane wejściowe:

- NS – liczba czujników;
- F – częstotliwość generowania zdarzeń (liczba zdarzeń na 1 minutę dla jednego czujnika);
- NE – liczba generowanych zdarzeń dla pojedynczego czujnika.

Po uruchomieniu programu następuje generowanie $NS \times NE$ zdarzeń dla NS czujników, a dodatkowo tworzonych jest NE zdarzeń o typie zdarzenia *NONE*, w celu ich późniejszego odfiltrowania na etapie przetwarzania strumieniowego. Wyznaczanie wartości temperatury, wilgotności oraz ciśnienia w ramach pojedynczego czujnika oparto na funkcji trygonometrycznej sinus, opisanej następującym wzorem (1):

$$A * \sin\left(\frac{T}{P} + B\right) + \frac{V_{max} - V_{min}}{2} + R \tag{1}$$

gdzie:

T – numer porządkowy zdarzeń w kontekście jednego czujnika,

P – parametr cyklicznie inkrementowany lub dekrementowany,

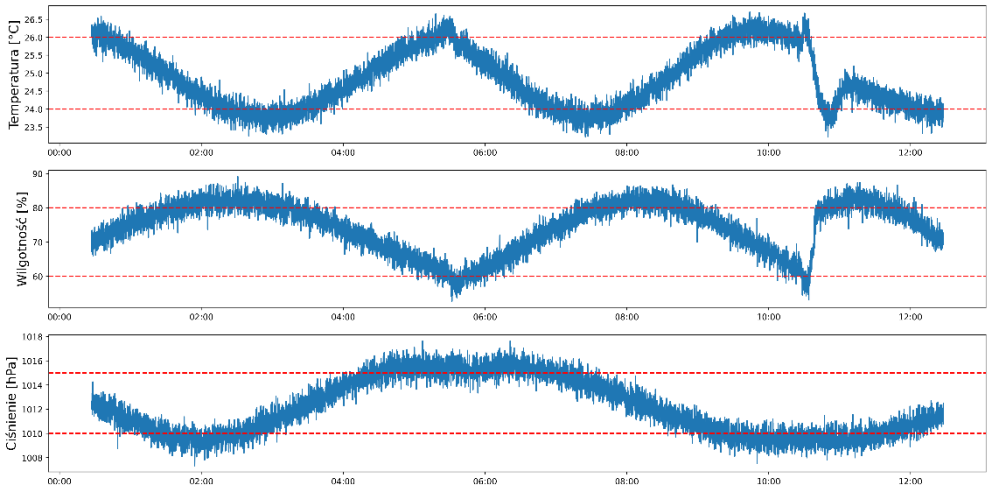
A, B – stałe liczby całkowite w obrębie wybranego parametru środowiskowego,

V_{max}, V_{max} – zdefiniowane zakresy dopuszczalnej wartości danego parametru środowiskowego,

R – liczba losowa z rozkładu normalnego.

Zastosowanie funkcji trygonometrycznej sinus oraz licznych, zmiennych parametrów miało na celu uzyskanie oryginalnego, nieprzewidywalnego zbioru wartości. Wprowadzenie nieliniowej funkcji pozwala na stworzenie bardziej złożonych i zróżnicowanych wzorców w danych dla każdego parametru powietrza oraz dla każdego

czujnika. Na rysunku 5 przedstawiono przykładowy rozkład wartości dla wybranego czujnika w okresie około 12 godzin.

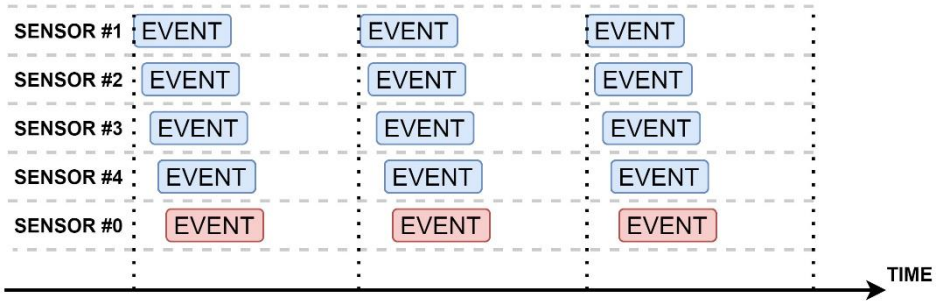


Rysunek 5. Przykładowy, 12-godzinny rozkład parametrów środowiskowych: temperatury, wilgotności i ciśnienia

Po utworzenie wszystkich zestawów danych następuje emisja zdarzeń z ustaloną częstotliwości F dla każdego z czujników. Generator danych został skonstruowany w taki sposób, aby zagwarantować, że wszystkie zdarzenia w ramach jednego cyklu zostaną przesłane przed rozpoczęciem się kolejnego cyklu. Zrealizowano to poprzez optymalizację polegającą między innymi na generowaniu zestawu danych przed samym procesem wysyłki. Dzięki takiemu rozwiązaniu zapewniono uporządkowaną sekwencje zdarzeń w ramach pojedynczego czujnika, a ponadto uniknięto wpływu spadku przepustowości wynikającej z zakolejkowania zdarzeń po stronie generatora. Na rysunku 6 przedstawiono sposób wysyłania zdarzeń przez generator dla czterech czujników na topik wejściowy Kafki. Jak można zauważyć, generator nie emituje zdarzeń równolegle dla wszystkich czujników, lecz robi to w sposób sekwencyjny. Czujnik #0 oznacza serie zdarzeń o typie zdarzenia równym NONE, które zostaną odfiltrowane na początkowym etapie przetwarzania.

Analizator danych to narzędzie umożliwiające przeprowadzanie analizy danych na dowolnym etapie procesu przetwarzania strumieniowego, co jest możliwe dzięki przechowywaniu przetworzonych zdarzeń w topicach Apache Kafka. Analizator danych

został opracowany w języku Python i na podstawie przetworzonych danych umożliwia przeprowadzenie analizy opóźnień oraz wizualizacji wybranych parametrów.



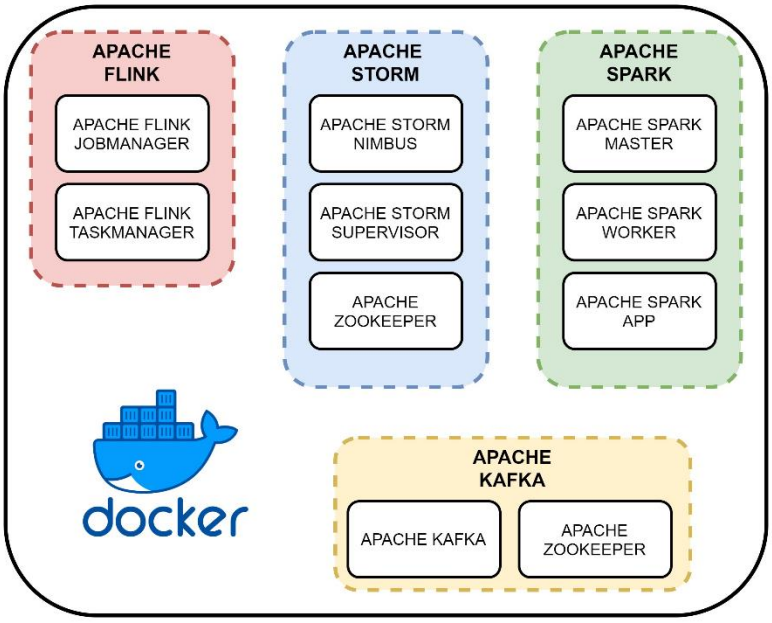
Rysunek 6. Sposób emitowania danych przez generator danych na przykładzie symulacji 4 czujników

3.3. Architektura opracowanego systemu przetwarzania danych strumieniowych

Wybór środowiska uruchomieniowego jest jednym z kluczowych aspektów podczas projektowania systemów informatycznych w tym systemów przetwarzania strumieniowego. W przypadku omawianego eksperymentu zdecydowano się na rozwiązanie oparte o środowisko dockerowe, które jest wspierane jako środowisko uruchomieniowe dla omawianych silników przetwarzania strumieniowego. Docker jest wiodącym narzędziem do konteneryzacji aplikacji, co wiąże się z wieloma korzyściami wykorzystywanymi w zaimplementowanym rozwiązaniu. Konteneryzacja pozwala na izolację ich zasobów, co przekłada się na stabilność i niezawodność pracy oraz pozwala uniknąć konfliktów zasobowych, ponadto gwarantuje, że każde narzędzie ma dostęp do potrzebnych zasobów w sposób kontrolowany.

W niniejszym rozwiązaniu zdecydowano się na ograniczenie pamięci RAM do 2GB dla każdego z kontenerów, ze względu na parametry wykorzystanej maszyny wirtualnej. Dodatkowo, Docker cechuje się łatwością zarządzania kontenerami, taką jak tworzenie, uruchamianie, zatrzymywanie i usuwanie. To cecha niezwykle przydatna jest w omawianym przypadku, ze względu na wymóg częstej zmiany parametrów konfiguracyjnych aplikacji. Dokładny scenariusz przeprowadzonych eksperymentów opisano w rozdziale 3.4.

Na potrzeby omawianego studium przypadku opracowano cztery skrypty oparte o narzędzie Docker Compose, pozwalające na zarządzenie zestawem aplikacji Apache Flink, Apache Spark, Apache Storm oraz brokera Apache Kafka. Na rysunku 7 przedstawiono grafikę obrazującą strukturę środowiska dockerowego w ramach tego eksperymentu.



Rysunek 7. Struktura środowiska uruchomieniowego opracowanego systemu przetwarzania strumieniowego

W ramach omawianego studium przypadku nie zdecydowano się na architekturę rozproszoną, dlatego eksperyment został wykonany na pojedynczej maszynie wirtualnej, która była uruchamiana na infrastrukturze dostawcy usług chmurowych Oracle Cloud. Maszyna wirtualna była skonfigurowana z następującymi parametrami:

- CPU: 3.0 GHz procesor Ampere® Altra™;
- pamięć RAM: 24 GB;
- ilość rdzeni: 4;
- system operacyjny: Ubuntu w wersji 22.04.2.

3.4. Scenariusz opracowanych eksperymentów

W kontekście analizowanego studium przypadku przeprowadzono eksperymenty, których celem była ocena opóźnień przetwarzania występujących w zaprojektowanym systemie przetwarzania strumieniowego w różnych konfiguracjach systemu. Ponadto wykonano testy związane z monitorowanie zużycia zasobów (CPU i pamięć RAM) w sytuacjach zmiennego obciążenia systemu. W ramach studium przypadku przeprowadzono szereg eksperymentów, które skupiły się na różnych aspektach zaprojektowanego systemu. W tabeli 4 zestawiono zrealizowane eksperymenty wraz ze szczegółami konfiguracji.

Eksperymenty I i II polegały na przeprowadzeniu serii testów, natomiast eksperyment III został zrealizowany w ramach jednego testu, jednak w jego trakcie zastosowano dynamiczną zmianę liczby czujników. Każdy test był inicjowany za pomocą skryptu, który tworzył czystą instancję systemu przetwarzania danych strumieniowych z odpowiednią dla niego konfiguracją. Natomiast zakończenie każdego testu obejmowało uruchomienie skryptu, którego celem było pełne usunięcie instancji systemu przetwarzania strumieniowego.

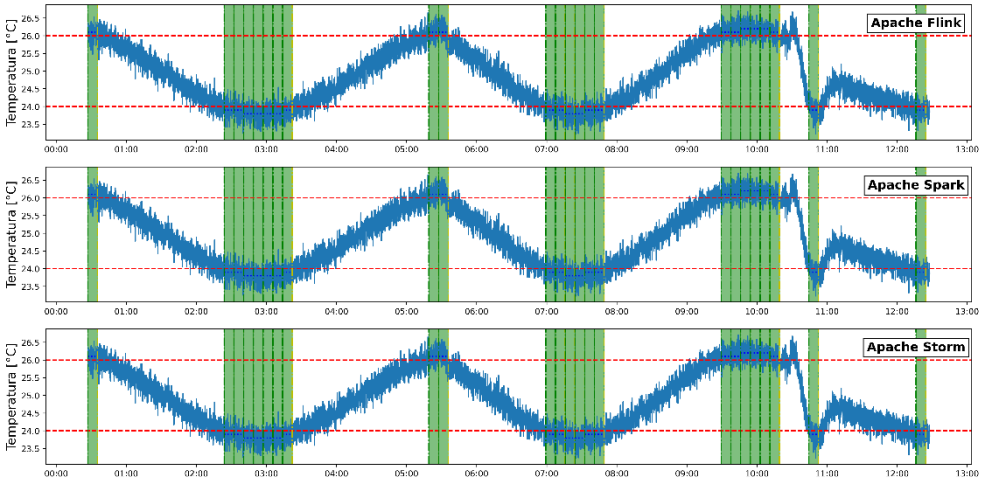
Tabela 4. Zestawienie wykonanych eksperymentów w ramach studium przypadku

LP	Opis	Parametr		Czas trwania pojedynczego testu / liczba testów
		Nazwa	Wartość	
I	Wpływ liczby czujników na opóźnienie przetwarzania	Liczba czujników	1 5 10 50 100 200 400 750 1000 3000 5000 10000	2 godziny / 12
		Rozmiar okna agregującego	100	
		Częstotliwość emitowania zdarzeń z jednego czujnika	12	

II	Wpływ rozmiaru okna agregującego na opóźnienie przetwarzania	Rozmiar okna agregującego	5 10 20 50 100 200 500 1000	2 godziny / 8
		Liczba czujników	500	
		Częstotliwość emitowania zdarzeń z jednego czujnika	12	
III	Wpływ dynamicznego obciążenia na zużycie zasobów: CPU oraz pamięci RAM	Liczba czujników	0 100 500 1000 2000 0 5000 0 10000	~12 godzin / 1
		Rozmiar okna agregującego	100	
		Częstotliwość emitowania zdarzeń z jednego czujnika	12	

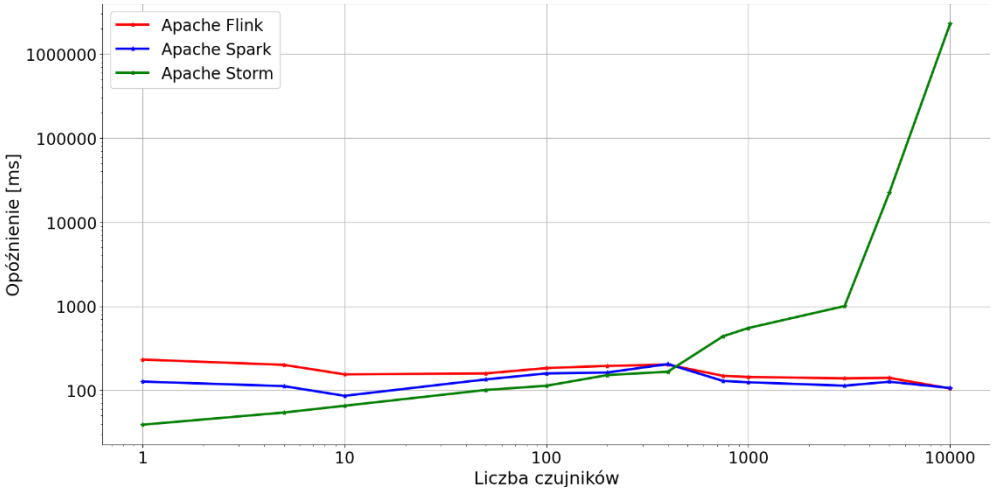
3.5. Wyniki oraz wnioski

Zgodne z założoną formą studium przypadku, wszystkie zdarzenia wyjściowe wygenerowane przez każdą aplikację przetwarzania danych strumieniowych powinny być identyczne pod kątem ich ilości oraz zawartości. Podczas wszystkich eksperymentów, w przypadku każdego testu analizowano oraz weryfikowano poprawność otrzymywanych danych wyjściowych, przy użyciu opracowanego analizatora danych. Na rysunku 8 przedstawiono przykładowy rozkład wygenerowanych danych wyjściowych na tle danych wejściowych pochodzących z jednego czujnika. Czerwonymi, przerywanymi liniami oznaczono granice zakresu dozwolonych wartości, natomiast zielone paski przedstawiają poszczególne okna agregacyjne w ramach, których wykryto przekroczenie wartości dopuszczalnych przez dany parametr środowiskowy, co oznaczało wygenerowanie zdarzenia wyjściowego (obiekt Alert).



Rysunek 8. Rozkład wygenerowanych zdarzeń wyjściowych przez każdą z aplikacji na tle zdarzeń wejściowych na przykładzie temperatury

W ramach eksperymentu I zbadano wpływ liczby czujników, a tym samym ilości zdarzeń wejściowych na opóźnienie przetwarzania w trzech aplikacjach przetwarzania strumieniowego. Wizualizacje wyników przedstawiono na rysunku 9. Warto zwrócić uwagę na zastosowanie skali logarytmicznej na obu osiach, co poprawia czytelność wykresu.

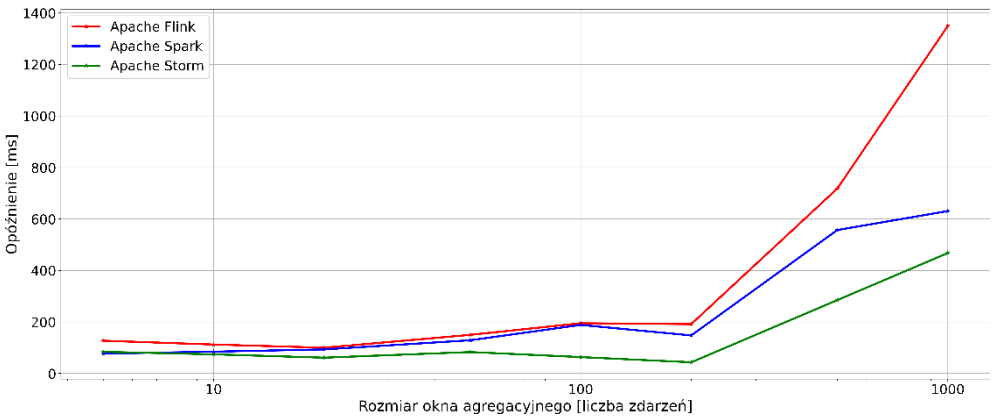


Rysunek 9. Wpływ liczby czujników na opóźnienie przetwarzania w każdej z aplikacji przetwarzania strumieniowego

Średnie opóźnienie w przetwarzaniu dla Apache Flink i Apache Spark wahało się między 86 ms a 232 ms we wszystkich przeprowadzonych testach. Te wyniki wskazują na utrzymanie stałej przepustowości przez obie aplikacje, co oznacza, że mogą obsłużyć znacznie więcej niż 10 000 czujników z akceptowalnym opóźnieniem. Ograniczenie liczby czujników wynikało głównie z wydajności generatora danych, aby uniknąć spadku wydajności samego generatora. Aplikacja Apache Storm osiągała najniższe opóźnienia przetwarzania przy niewielkiej liczbie czujników, ale wraz ze wzrostem liczby czujników do 750 i więcej, zaobserwowano stopniowy, a potem gwałtowny spadek przepustowości przetwarzania, co przejawiało się bardzo długimi opóźnieniami przetwarzania, mierzonymi w minutach.

Na podstawie tego eksperymentu można wywnioskować, że w badanej konfiguracji, najniższą przepustowością charakteryzowała się aplikacja Apache Storm. W przypadku pozostałych dwóch aplikacji nie zaobserwowano punktu krytycznego, który wskazywałby na spadek ich przepustowości. Warto zauważyć, że w każdym teście tego eksperymentu każda aplikacja była konfigurowana identycznie, co mogło wpłynąć negatywnie na przepustowość aplikacji Apache Storm w przypadku dużej liczby czujników. Apache Storm posiada mechanizmy umożliwiające zwiększenie przepustowości kosztem wzrostu opóźnienia. Niebagatelny wpływ na wydajność poszczególnych aplikacji miał fakt, że każda z nich działała tylko na jednym węźle roboczym.

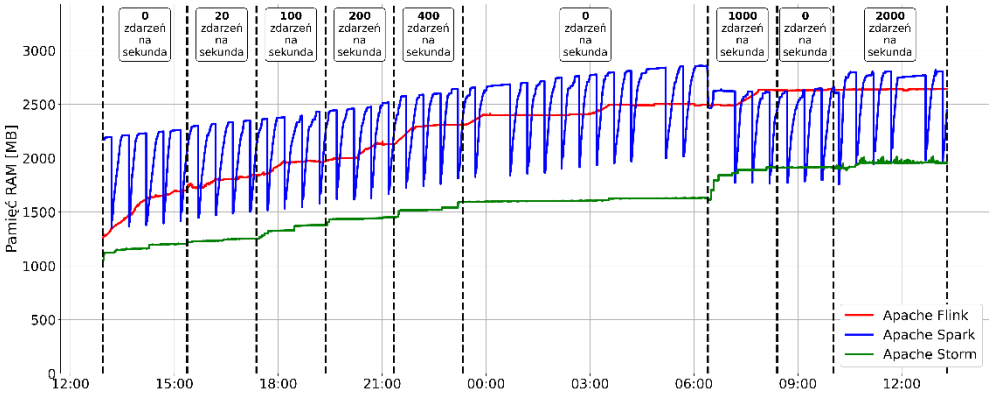
Eksperyment II skupiony był na zbadaniu zależności między rozmiarem okna agregacyjnego, a poziomem opóźnienia przetwarzania. Wyniki serii testów przedstawiono graficznie na rysunku 10.



Rysunek 10. Wpływ rozmiaru okna agregacyjnego na opóźnienie przetwarzania w każdej z aplikacji przetwarzania strumieniowego

Dla okien agregacyjnych o rozmiarze mniejszym niż 200 zdarzeń, średni czas przetwarzania we wszystkich aplikacjach przetwarzania strumieniowego wynosił mniej niż 200 ms przy stałym obciążeniu wejściowym (500 czujników). Powyżej tej wielkości okna zauważono wzrost opóźnień w przetwarzaniu w każdym przypadku, choć z różną dynamiką. Największy wzrost opóźnienia zaobserwowano w przypadku Apache Flink, podczas gdy najmniejszy wystąpił w przypadku Apache Storm. Różnice w średnich opóźnieniach między aplikacjami, pomimo korzystania z identycznego rozmiaru okna agregacyjnego i stałego obciążenia, mogą wynikać z kilku czynników. Jednym z kluczowych czynników jest zróżnicowana architektura każdego z tych silników przetwarzania strumieniowego. Inne przyczyny takich różnic mogą być związane z wybranymi strategiami zarządzania zasobami oraz zastosowanymi mechanizmami optymalizacyjnymi.

Eksperyment III zorientowany był na zbadaniu wpływu zmiennego obciążenia danymi wejściowymi na zużycie zasobów takich jak pamięć RAM, czy CPU. Na rysunku 11 przedstawiono rezultaty pomiaru zużycia pamięci RAM podczas zaplanowanych zmian liczby czujników symulowanych przez generator danych.



Rysunek 11. Wpływ liczby przetwarzanych zdarzeń na poziomu zużycia pamięci RAM przez każdą z aplikacji przetwarzania strumieniowego

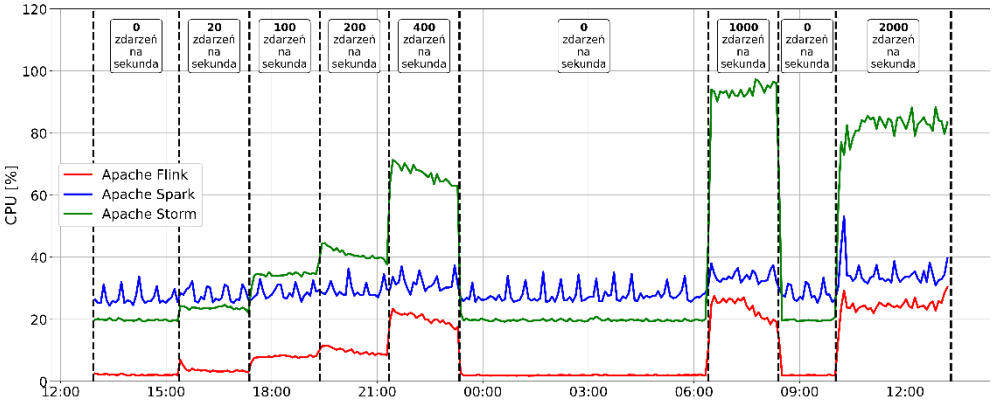
Podczas testu, początkowa faza charakteryzowała się brakiem obciążenia danymi wejściowymi. Mimo to, w każdym przypadku zauważalny był wzrost poziomu zużycia pamięci RAM. W przypadku aplikacji Apache Spark, można było obserwować

okresowe uwalnianie pamięci w cyklach co 30 minut. To zjawisko wynikało z automatycznego uruchamiania procesu czyszczenia pamięci (GC). Warto podkreślić, że proces ten nie jest obligatoryjny, co można zauważyć, analizując dane okresie od 0:00 do 6:00. Stopniowy wzrost obciążenia od godziny 15:15 do 23:15 skutkował zwiększeniem zużycia pamięci RAM. W przypadku Apache Spark wzrost ten miał charakter liniowy (z wyjątkiem okresowego uwalniania pamięci) i utrzymywał się na najwyższym poziomie w porównaniu do pozostałych aplikacji.

Jeśli chodzi o Apache Storm, można zaobserwować schodkowy wzrost zużycia pamięci, który następował nie tylko zaraz po zwiększeniu obciążenia. Od godziny 23:15, przez 7 godzin, generator danych został wyłączony. To spowodowało spadek dynamiki wzrostu zużycia pamięci RAM w przypadku Apache Flink i Apache Storm. Jednak zużycie pamięci przez Apache Spark utrzymywało charakter liniowy. Po okresie przestoju w przetwarzaniu, generator danych został uruchomiony, a ilość zdarzeń osiągnęła poziom 1000 zdarzeń na sekundę, co odpowiada 5000 czujnikom generującym dane co 5 sekund. To spowodowało gwałtowny wzrost zużycia pamięci przez Apache Storm, podczas gdy wzrost zużycia pamięci w przypadku Apache Flink wystąpił dopiero po ponad godzinie od uruchomienia generatora.

W przypadku Apache Spark zauważono zaskakujący spadek zużycia pamięci, co skutkowało utrzymaniem się zużycia pamięci RAM na stałym poziomie z pominięć procesu uwalniania pamięci RAM. Kolejne wyłączenie generatora danych spowodowało utrzymanie się zużycia pamięci na tym samym poziomie przez wszystkie silniki. Ostatnim etapem była emisja 2000 zdarzeń na sekundę. W przypadku Apache Flink poziom używanej pamięci RAM pozostał na stałym poziomie, w Apache Spark zaś nastąpił wzrost zużycia, po którym nastąpiła jego stabilizacja. W przypadku Apache Storm można zaobserwować minimalny wzrost oraz pewne fluktuacje w ilości używanej pamięci RAM. To zjawisko może być związane ze spadkiem przepustowości aplikacji Apache Storm i znacznym wzrostem opóźnienia przetwarzania.
Jednocześnie, wraz z pomiarem zużycia pamięci RAM, monitorowano także zużycie zasobu jaki jest CPU maszyny wirtualnej. Na rysunku 12 można zaobserwować charakter zmian poziomu zużycia pamięci procesora na przestrzeni około 12 godzin.

Podczas całego testu, Apache Flink charakteryzował się niższym zużyciem CPU niż pozostałe aplikacje do przetwarzania strumieniowego. W przypadku gwałtownego wzrostu obciążenia, odnotowany został wzrost zużycia CPU, który stopniowo malał w czasie, nawet przy utrzymaniu stałego obciążenia przez generator danych.



Rysunek 12. Wpływ liczby przetwarzanych zdarzeń na poziomu zużycia CPU przez każdą z aplikacji przetwarzania strumieniowego

W przypadku Apache Spark zaobserwowano charakterystyczne, cykliczne szczyty zużycia CPU, wynikające z operacji czyszczenia pamięci. Największy wzrost zużycia CPU nastąpił podczas ostatniego etapu testu, kiedy system był najbardziej obciążony danymi wejściowymi. Wyłączając fluktuacje związane z procesami czyszczenia pamięci, zużycie CPU przez Apache Spark oscylowało między 25% a 35% jednego rdzenia, z niewielkimi spadkami podczas bezczynności generatora danych oraz niewielkimi wzrostami przy dużych obciążeniach. To stanowiło unikalną cechę w porównaniu do pozostałych aplikacji. Trend zmiany poziomu zużycia CPU podczas dynamicznej zmiany obciążenia w Apache Storm przypominał ten, który zaobserwowano w przypadku Apache Flink. Przy obciążeniu przekraczającym 100 zdarzeń na sekundę, Apache Storm wykazywał największe zużycie CPU spośród wszystkich aplikacji przetwarzania strumieniowego.

4. Podsumowanie

W ramach niniejszego artykułu dokonano porównawczej analizy możliwości i ograniczeń wykorzystania silników Apache Flink, Apache Spark i Apache Storm w strumieniowym przetwarzaniu danych oraz zrealizowano serie eksperymentów w ramach opracowanego studium przypadku.

Podczas analizy porównawczej skupiono się na różnych kluczowych kryteriach związanych z przetwarzaniem strumieniowym, takich jak: architektura, interfejsy, tryby przetwarzania, tryby uruchomieniowe, niezawodność, skalowalność i wydajność, źródła i ujścia danych oraz wykorzystanie uczenia maszynowego. Jednakże, ostateczny wybór silnika do przetwarzania strumieniowego nie może być jednoznacznie określony wyłącznie na podstawie analizy porównawczej. Decyzja ta jest zależna nie tylko od samej specyfikacji silnika, ale także od charakteru danych, które będą przetwarzane, a także od indywidualnych potrzeb i wymagań projektu lub organizacji. Każdy z omówionych silników może być bardziej lub mniej odpowiedni w zależności od konkretnego scenariusza i kontekstu zastosowania. Dlatego analiza porównawcza pełni głównie rolę narzędzia wspomagającego proces podejmowania decyzji, dostarczając informacji na temat różnic i podobieństw między analizowanymi rozwiązaniami. Ostateczny wybór powinien być starannie przemyślany, uwzględniając wszystkie istotne czynniki oraz aktualne potrzeby i cele projektu.

Głównym celem przygotowania i realizacji studium przypadku było przeprowadzenie badań porównawczych w celu oceny skuteczności aplikacji Apache Flink, Apache Spark i Apache Storm w kontekście wykrywania anomalii na podstawie strumieniowych zdarzeń wejściowych. Studium przypadku miało również na celu dostarczenie realnego przykładu wykorzystania produkcyjnie narzędzia do analizy i monitorowania parametrów środowiskowych w celu wykrywania przekroczeń określonych norm. W ramach tego badania przeprowadzono następujące eksperymenty:

- wpływ ilości zdarzeń wejściowych na opóźnienie przetwarzania;
- wpływ rozmiaru okna agregującego na opóźnienie przetwarzania;
- wpływ zmiennego obciążenia systemu na zużycie jego zasobów (CPU, pamięć RAM).

Wyniki tych badań miały na celu zapewnić wgląd w to, który z tych narzędzi najlepiej sprawdza się w zadaniu wykrywania anomalii w danym kontekście. Studium przypadku stanowiło również praktyczny przykład wykorzystania narzędzi analitycznych w rzeczywistym środowisku produkcyjnym, co może pomóc w zrozumieniu, jak te narzędzia mogą być użyteczne w innych dziedzinach, które wymagają analizy strumieniowych danych w czasie rzeczywistym.

Literatura

[1] Ullah, S., Awan, M.D., Sikander Hayat Khiyal, M. *Big Data in Cloud Computing: A Resource Management Perspective*. Sci. Program. 2018, 2018, https://doi.org/10.1155/2018/5418679.

[2] Perera, S. *A Gentle Introduction to Stream Processing*. https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97.

[3] Apache Software Foundation Apache Flink – Documentation Available online: https://nightlies.apache.org/flink/flink-docs-release-1.16/.

[4] Apache Software Foundation Apache Spark – Documentation Available online: https://spark.apache.org/docs/latest/index.html.

[5] Apache Software Foundation Apache Storm – Documentation Available online: https://storm.apache.org/releases/2.4.0/.

[6] Carbone, P., Katsifodimos, A., Kth, †, Sweden, S., Ewen, S., Markl, V., Haridi, S., Tzoumas, K. *Apache FlinkTM: Stream and Batch Processing in a Single Engine*. 2015, 28-38, https://www.researchgate.net/publication/308993790_Apache_Flink_Stream_and_Batch_Processing_in_a_Single_Engine.

[7] Asif Abbasi, M. *Learning Apache Spark 2*. 2017, Packt Publishing, Birmingham.

[8] Inoubli, W.; Aridhi, S.; Mezni, H.; Maddouri, M.; Mephu Nguifo, E. *An Experimental Survey on Big Data Frameworks*. Futur. Gener. Comput. Syst. 2018, Vol. 86, 546-564, doi:10.1016/j.future.2018.04.032.

[9] Ficco, M., Pietrantuono, R., Russo, S. *Aging-Related Performance Anomalies in the Apache Storm Stream Processing System*. Futur. Gener. Comput. Syst. 2018, Vol. 86, Issue C, 975-994, doi:10.1016/j.future.2017.08.051.

[10] Tomar, A. *Micro-Batching vs Streaming*. https://www.linkedin.com/pulse/micro-batching-vs-streaming-amit-tomar/.

[11] Gorasiya, D.V. *Comparison of Open-Source Data Stream Processing Engines: Spark Streaming*, Flink and Storm. Tech. Rep. 2019, doi:10.13140/RG.2.2.16747.49440.

[12] Liu, P., Xu, H., Da Silva, D., Wang, Q., Ahmed, S.T., Hu, L. *FP4S: Fragment-Based Parallel State Recovery for Stateful Stream Applications*. Proc. 2020 IEEE 34th Int. Parallel Distrib. Process. Symp. IPDPS 2020, 2020, 1102-1111, doi:10.1109/IPDPS47924.2020.00116.

[13] Novikov, I. *How Do You Compare the Scalability and Fault Tolerance of Hadoop and Spark Clusters?* https://www.linkedin.com/advice/3/how-do-you-compare-scalability-fault-tolerance.

[14] Van Der Veen, J.S., Van Der Waaij, B., Lazovik, E., Wijbrandi, W., Meijer, R.J. *Dynamically Scaling Apache Storm for the Analysis of Streaming Data*. Proc. - 2015 IEEE 1st Int. Conf. Big Data Comput. Serv. Appl. BigDataService 2015, 2015, 154-161, doi:10.1109/BigDataService.2015.56.

[15] Marcu, O.C., Costan, A., Antoniu, G., Pérez-Hernández, M.S. *Spark versus Flink: Understanding Performance in Big Data Analytics Frameworks*. Proc. – IEEE Int. Conf. Clust. Comput. ICCC 2016, 433-442, doi:10.1109/CLUSTER.2016.22.

[16] Neubauer, T. *Flink vs Spark: Benchmarking Stream Processing Client Libraries*. 2021, https://quix.io/blog/compare-client-libraries-spark-flink-quix.

[17] Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V. *Benchmarking Distributed Stream Data Processing Systems*. Proc. – IEEE 34th Int. Conf. Data Eng. ICDE 2018, 2018, 1519-1530, doi:10.1109/ICDE.2018.00169.

Comparative Analysis of Capabilities and Limitations in The Utilization of Apache Flink, Apache Spark, and Apache Storm in Stream Processing of Data

Abstract

This article focuses on conducting a comparative analysis of three engines for stream processing of data: Apache Flink, Apache Spark, and Apache Storm. The analyses encompass various comparative criteria, such as architecture, interfaces, processing modes, execution modes, reliability, scalability, performance, data sources and sinks, as well as the utilization of machine learning. As part of the research, a series of experiments were conducted, wherein each engine was tested in real-time while performing the task of anomaly detection in environmental parameter measurements. The experiments included an analysis of the impact of the number of sensors, the size of the aggregating window, and system load on delays in generating warnings, resource consumption (CPU, RAM), and the number of warnings generated by each engine. The conclusions of the study can provide valuable information regarding the effectiveness and usefulness of each analyzed engine in the context of stream processing of data, especially in applications related to environmental anomaly detection.

Keywords: *stream processing, Apache Flink, Apache Spark, Apache Storm*