Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa magisterska

# WYKRYWANIE ANOMALII W STRUMIENIACH DANYCH

Łukasz Kiszka

Promotor
dr inż. Marek Wojciechowski

Poznań, 2016 r.

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

## Streszczenie

Przedmiotem niniejszej pracy magisterskiej jest przedstawienie problemu wyszukiwania anomalii w danych strumieniowych. Praca przybliża pojęcie anomalii oraz opisuje, czym są dane strumieniowe i jakie problemy napotykamy podczas ich przetwarzania. Autor poświęca uwagę algorytmom uczenia maszynowego dla danych strumieniowych, które z powodzeniem mogą zostać użyte do wykrywania nieprawidłowości. Przytoczone zostają potencjalne obszary, gdzie można zastosować wykrywanie anomalii. Zaprezentowano platformy oraz biblioteki wspomagające przetwarzanie danych strumieniowych bądź wykorzystywane do budowy systemu zorientowanego na detekcję nieprawidłowości w danych. Następnie omówiono architekturę systemu wykrywającego anomalie wraz z przykładowymi komponentami, które mogą zostać wykorzystane do jej implementacji. Na końcu autor prezentuje próbę zbudowania części architektury opartej o Apache Kafka, Apache Storm oraz Apache Samoa. Szczególną uwagę poświęcono bibliotece Apache Samoa – w niej przetestowano uczenie maszynowe, a dokładniej klasyfikację z wykorzystaniem algorytmu Vertical Hoeffding Tree. Dokonano również analizy biblioteki pod kątem wykorzystania jej w systemie zorientowanym na wykrywanie anomalii w danych strumieniowych.

## Abstract

In this master's thesis Author presented a problem of detecting anomalies in streaming data. The thesis brought closer a definition of anomaly and described what streaming data are or what problems could be found within their processing. The Author focused on a machine learning algorithm for streaming data which could be used for detecting anomalies successfully. There were mentioned specific examples of areas where detecting anomalies might be used. The Author presented systems and libraries which supported streaming data processing or are used to build anomaly detection system. These works were the starting point for further considerations – presenting an architecture of a detecting anomalies system with exemplary components used for their implementation. In the end the Author presented an attempt of building the architecture based on Apache Kafka, Apache Storm and Apache Samoa. A particular attention was paid to Apache Samoa library. Additionally, the Author tested machine learning – a Vertical Hoeffding Tree algorithm classification specifically. The library was analyzed for being prepared for the implementation with the use of streaming data anomalies detecting method.

# Spis treści

# Rozdział 1

# Wprowadzenie

## 1.1 Geneza pracy

W dzisiejszym świecie w każdej sekundzie ma miejsce napływ dużej ilości danych. IBM w swoim artykule [1] wskazuje, że codziennie przybywa 2,5 tryliona bajtów danych. Dane te pochodzą z różnych źródeł: od aktywności użytkowników w internecie przez pomiary pochodzące z różnego rodzaju czujników czy telefonów komórkowych po informacje gromadzone przez instytucje publiczne oraz prywatne. Wiadomości te przyjmują różną postać. Mogą to być dane ustrukturyzowane, jak chociażby udostępniane poprzez publiczne API, lub dane bez zdefiniowanej struktury np. pochodzące z serwisów społecznościowych. Powyższe cechy danych określa się terminem *"Big Data"*. Termin Big Data oznacza duży, różnorodny oraz zmienny zbiór danych, często określany jako 3V *(od angielskich słów volume, velocity, variety)*, jednakże Albert Bifet rozszerzył tę definicję do 6V, dodając do niej *value, variability oraz veracity*, czyli wartość danych, ich zmienność oraz wiarygodność. Zauważono, że dane, które są gromadzone, mogą dostarczyć ich posiadaczom wiedzy. Nastąpił rozwój narzędzi oraz nauki pod kątem odkrywania wiedzy w danych (*ang. data mining*). Do wydobywania wiedzy z danych stosuje się często techniki oparte o zagadnienia statystyczne czy związane ze sztuczną inteligencją. Analiza dużych zbiorów danych wymaga również dużych mocy obliczeniowych komputerów oraz rozwoju narzędzi, które potrafią być skalowalne na wiele maszyn w celu zrównoleglenia obliczeń w tzw. klastry obliczeniowe. Zastosowanie klastrów obliczeniowych pociągnęło za sobą opracowanie nowych paradygmatów przetwarzania danych np. map-reduce oraz algorytmów, które będą potrafiły wykrzesać siłę z klastrów, by szybciej uzyskać porządany wynik. Duże wolumeny nieustrukturyzowanych danych wraz z połączeniem z rozproszonymi systemami wymusiły także opracowanie nowych technik przechowywania danych. Dostrzeżono, że relacyjne bazy danych nie są idealnym rozwiązaniem do przechowywania i analizy wszystkich rodzajów danych. Zaczął rozwijać się nurt baz danych NoSQL (skrót od *ang. not only SQL*, czyli nie tylko SQL). Ideą NoSQL nie jest wypchnięcie relacyjnych baz danych, lecz uzupełnienie ich w tych dziedzinach, gdzie relacyjne bazy danych nie są idealnym rozwiązaniem np. przechowywanie danych grafowych czy bazy danych klucz-wartość, które są dobrym mechanizmem pamięci podręcznej (*ang. cache*). Przetwarzanie danych zgromadzonych w bazach danych nazywamy przetwarzaniem wsadowym (*ang. batch*). Wydobycie wiedzy tkwiącej w danych wzmacnia konkurencyjność firm oraz pozwala wyciągnąć informacje, trendy czy wzorce, których dotychczas nie byliśmy świadomi. Można stwierdzić, że w dzisiejszym świecie to dane i ich analiza odgrywają niezwykle ważną rolę zarówno w biznesie, jak i nauce. Istotna stała się także szybkość, z jaką jesteśmy w stanie dokonać analizy danych, czego konsekwencją jest stworzenie systemów analizy danych w czasie rzeczywistym (*ang. real-time computing*). Systemy czasu rzeczywistego posiadają podobne cechy jak systemy wsadowe z tą różnicą, że dokonują analizy danej wejściowej w momencie jej pojawienia się. Zauważono, że wiele krytycznych systemów informatycznych potrzebuje natychmiast wydobyć informacje z danych, by odpowiednio zareagować na powstały wynik. Ważnym elementem systemów krytycznych jest znajdowanie

ciągów danych odbiegających swoją charakterystyką od pozostałych danych uznanych za normę. Takie dane nazywamy anomaliami. Wykrycie zaburzenia pozwala zapobiec wielu konsekwencjom. Wyobraźmy sobie system bankowy, do którego trafia żądanie przelewu na dużą kwotę z zastrzeżeniem, że użytkownik proszący o taki przelew nigdy wcześniej nie podejmował podobnej akcji. System w tym miejscu powinien zasygnalizować anomalię w działaniu użytkownika i skierować takie żądanie przelewu do weryfikacji np. przez opiekuna klienta po stronie banku. Wykrywanie anomalii w świecie, gdzie rozkład danych nieustannie się zmienia, jest niezmiernie istotne. Niniejsza praca przygląda się dokładniej problemowi wykrywania anomalii w danych strumieniowych z wykorzystaniem technik uczenia maszynowego oraz zastosowania wspomnianego rozwiązania w praktyce.

## 1.2 Cel i zakres pracy

Celem pracy jest przegląd systemów przetwarzania danych strumieniowych oraz wybranie jednego, na którym zostanie oparta zaprojektowana platforma do wykrywania anomalii w danych. W ramach pracy zostaną także opisane techniki wykrywana anomalii oraz sposoby radzenia sobie ze zjawiskiem nazywanym *Concept Drift* stosowane przez systemy przetwarzania danych strumieniowych. Ponadto poruszony będzie temat przetwarzania danych strumieniowych w systemach klasy Big Data. W części praktycznej zostanie przedstawiona architektura przykładowego systemu przetwarzania danych strumieniowych ukierunkowanego na wykrywanie anomalii w danych wraz z pokazaniem przykładowych narzędzi, które mogą posłużyć do implementacji tego systemu.

Struktura pracy jest następująca:

Drugi rozdział dostarczy informacji o danych strumieniowych. Zwróci się uwagę na podział danych na wsadowe i strumieniowe, w kontekście którego zostaną przedstawione architektury systemów przetwarzania dużych wolumenów danych. Następnie zostanie omówiona specyfika danych strumieniowych oraz zjawisko zmiany charakterystyki danych, czyli concept drift. Autor opisze także strukturę uczenia maszynowego w strumieniach danych oraz przedstawi narzędzia przetwarzania danych strumieniowych. W kolejnym rozdziale zostanie wyjaśnione pojęcie anomalii w danych oraz zaprezentowane techniki wykrywania anomalii. Ponadto podane zostaną przykłady systemów, gdzie stosuje się wykrywanie anomalii. W rozdziale czwartym została opisana zaproponowana przez autora architektura platformy zorientowanej na wykrywanie anomalii oraz przedstawiono, jakie przykładowe technologie czy biblioteki mogą zostać użyte do zbudowania wspomnianej platformy. W rozdziale piątym zostanie omówiona część praktyczna pracy, na którą będzie składać się integracja części komponentów opisanych w architekturze systemu oraz zostanie bliżej przedstawiona biblioteka Apache SAMOA. Autor dokona oceny biblioteki oraz zaproponuje w niej zmiany, dzięki którym będzie bardziej dostosowana do wykorzystania w projektowanym systemie. Rozdział szósty stanowi podsumowanie pracy.

# Rozdział 2

# Przetwarzanie danych strumieniowych

## 2.1 Wprowadzenie

Wyróżniamy dwa tryby przetwarzania danych: w czasie rzeczywistym oraz wsadowe (przetwarzanie danych wcześniej zgromadzonych). Obydwa sposoby mają swoje zastosowanie w systemach przetwarzających duże wolumeny danych, jednakże każdy z trybów jest używany do innych celów.

| **Przetwarzanie wsadowe (*ang. batch*)** | **Przetwarzanie strumieniowe (*ang. streaming*)** |
|---|---|
| dane wejściowe statyczne obrazujące migawkę danych | dane wejściowe dynamiczne, cały czas się zmieniają |
| przetwarzanie całych lub znacznej większości wolumenu danych | przetwarzanie tylko danych, które nadeszły w danym czasie lub w wyznaczonym oknie np. dane z ostatnich 10 sekund |
| bardzo długi czas wykonywania obliczeń rzędu minut, godzin | szybki czas przetwarzania - wynik dostępny w przeciągu milisekund lub sekund |
| całościowa analiza danych | proste funkcje agregujące, rolujące, obliczające metryki |

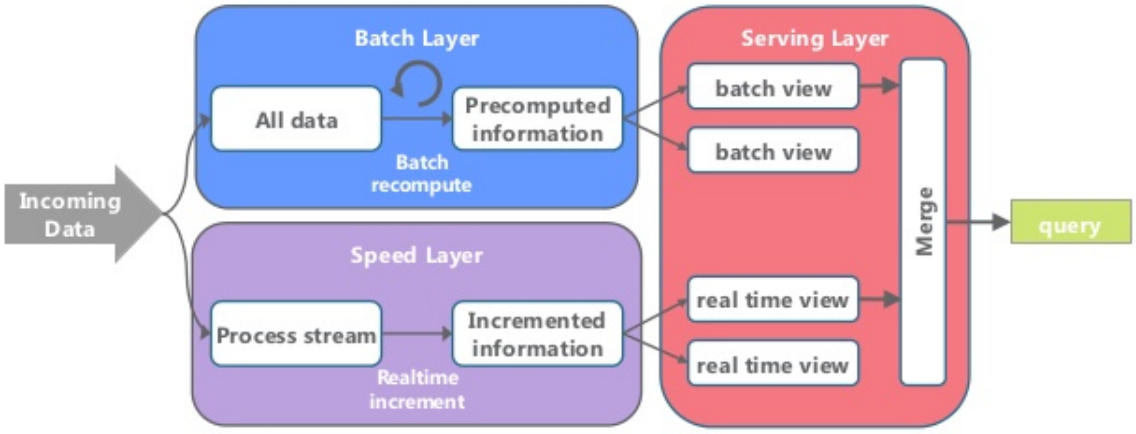
Tablica 2.1: Różnice między przetwarzaniem wsadowym oraz strumieniowym

W tablicy 2.1 zostały przedstawione różnice pomiędzy przetwarzaniem wsadowym a strumieniowym. Z tabeli wynika, że przetwarzanie wsadowe przetwarza całość lub fragment danych gromadzonych przez system, dzięki czemu jako wynik otrzymujemy pełną analizę danych. Dzieje sie to kosztem czasu, ponieważ przetwarzanie dużych wolumenów danych jest bardzo kosztowne w zależności od złożoności obliczeń. Przetwarzanie strumieniowe natomiast dostarcza nam wynik bardzo szybko, lecz bierze pod uwagę wyłącznie dane, które dotarły do systemu w konkretnym momencie. Istnieją także rozwiązania, gdzie przetwarzanie strumieniowe korzysta z danych tymczasowych gromadzonych w źródłach danych zorientowanych na szybki dostęp np. w bazach danych NoSQL przechowujących informacje w pamięci - eliminuje to opóźnienia wynikające z odczytów dyskowych - typu klucz-wartość np. Redis, czy opartych o dedykowane maszyny nastawione na szybki dostęp do danych w celu ich analizy (przechowujących dane również w pamięci operacyjnej).

Architekci oprogramowania w odpowiedzi na duży przyrost danych, jaki obserwujemy w dzisiejszych czasach, zaprojektowali dedykowane architekury przetwarzania dużych wolumenów danych, które dostarczają użytkownikom wyniki w jak najkrótszym czasie. Przykładem architektury, łączącej przetwarzanie strumieniowe i wsadowe, jest zaprojektowana przez Nathana Marza architektura lambda [2] opisana w książce *"Big Data. Najlepsze praktyki budowy skalowalnych systemów obsługi danych w czasie rzeczywistym"* [3] . Autor wyróżnia trzy warstwy: przetwarzania wsadowego, przetwarzania strumieniowego oraz warstwę udostępniania danych(*ang. serving layer*). Przychodząca nowa informacja trafia do repozytorium danych (np.

bazy danych, HDFS), z którego, wraz z danymi historycznymi, będzie przetwarzana przez warstwę wsadową. W tym samym czasie zostaje ona przetworzona przez warstwę strumieniową. Warstwa strumieniowa służy do obliczania wyników dla najnowszych danych, które nie zostały jeszcze uwzględnione w ostatnim ukończonym przetwarzaniu wsadowym. Można stwierdzić, że okno czasowe, z jakiego pochodzą dane w tej warstwie, jest ograniczone przez rozpoczęcie i zakończenie przetwarzania wsadowego, przez co przetwarza ona mniejszy wolumen danych. Warstwa udostępniania danych służy do przechowywania wyników obydwóch przetwarzań nazywanych widokami. Jest odpowiedzialna za połączenie wyników uzyskanych z długiego przetwarzania wsadowego, które zawsze przetwarza cały zgromadzony wolumen danych, oraz strumieniowego, które dostarcza wyników uwzględniających dane najnowsze. Wyniki są składowane w repozytoriach danych (np. Cassandra, HBase). Architektura lambda rozwiązuje problem wynikający z długich opóźnień dostarczania wyników obliczeń przez warstwę wsadową, wskutek czego użytkownik pobierający dane z wyników owego przetwarzania nie miał najświeższych informacji, a dane zaledwie do pewnego punktu w czasie, w którym rozpoczeły się obliczenia w tej warstwie.

System zorientowany na wykrywanie anomalii w danych powinien znaleźć się w module odpowiedzialnym za przetwarzanie strumieniowe, ponieważ kluczowy element stanowi tutaj czas wykrycia anomalii i reakcji na nią. Zastosowanie wykrywania nieporządanych cech w informacjach podczas przetwarzania wsadowego ma swoje uzasadnienie, gdy obliczenia poszukujące błędów są długotrwałe oraz czas otrzymania wyniku nie jest krytyczny. Przykładem anomalii wykrywanej w danych wsadowych może być odnalezienie zależności w danych historycznych wskazującej np., że klienci o podanych cechach najczęściej nie spłacają kredytu. Tak wyszukaną zależność możemy następnie zaaplikować do modelu w warstwie przetwarzania strumieniowego, by po wykryciu klienta o podanych cechach występował alert. Przykład ten pokazuje, w jaki sposób wykrywanie anomalii w danych wsadowych może być uzupełnieniem odnajdywania ich w strumieniach danych.




Rysunek 2.1: Schemat architektury lambda

Źródło: https://www.linkedin.com/pulse/lambda-architecture-what-buzz-abhishek-srivastava

Warstwa strumieniowa oblicza widoki z pojawiających się danych. Dzięki temu użytkownik końcowy, wykonując zapytanie, otrzymuje połączone widoki utworzone przez obydwa rodzaje przetwarzań, przez co zawsze ma pogląd na aktualne dane. Powstała także koncepcja architektury Kappa [4], której autorem jest Jay Kreps. Model kappa jest uproszczeniem architektury lambda - autor tej koncepcji zauważył, że rozbudowana architektura lambda jest trudna w utrzymaniu, ponieważ musimy przygotowywać dwa te same algorytmy dla warstwy strumieniowej i wsadowej działające na osobnych bibliotekach np. Storm w warstwie strumieniowej oraz Hadoop w warstwie wsadowej. W konsekwencji w architekturze kappa usunięto warstwę

wsadową, pozostawiając wyłącznie strumieniową i udostępniania danych. Dane wejściowe są przetwarzane przez warstwę strumieniową oraz składowane w systemach przechowywania danych, gdyby była potrzeba ponownego przeliczenia np. po zmianie algorytmów. Kappa jest dobrym rozwiązaniem do reorganizacji oraz migracji źródeł danych czy wszędzie tam, gdzie zbędne jest używanie algorytmów, które potrzebują przetwarzać cały zbiór danych do otrzymania wyniku.

Ważną częścią systemów przetwarzających dane są także aplikacje raportowe, analityczne bądź interaktywne pulpity (*ang. dashboard*), które prezentują wyniki analiz użytkownikom końcowym. Klienci takich aplikacji w oparciu o otrzymane dane mogą podejmować lepsze decyzje biznesowe lub poddawać te dane dalszej analizie, by wyciągnąć z nich jeszcze więcej informacji. Na szczególną uwagę w kontekście danych strumieniowych zasługują interaktywne pulpity, które są w stanie na bieżąco informować użytkowników o wynikach analizy informacji bez konieczności ciągłego odświeżania takiego widoku. W oparciu o te dane operator może natychmiast podejmować akcje usprawniające procesy z dziedziny, w której działa.

## 2.2 Dane strumieniowe

Według literatury strumień danych jest to nieograniczony zbiór elementów (s,t), gdzie s jest wierszem należącym do schematu strumienia, a t jest stemplem czasowym elementu. Inna definicja wspomina, że strumień danych to ciągła sekwencja elementów, gdzie każdy z nich jest odczytywany tylko raz [5]. Przetwarzaniem danych strumieniowych nazywamy analizę tych danych w czasie rzeczywistym przy użyciu ciągłych zapytań. Źródłem ich mogą być wszystkie systemy, które podczas swojej pracy cały czas produkują dane niezależnie od ingerencji użytkownika. Przykładem są systemy sczytujące dane z czujników czy każda aplikacja tworząca logi ze swojego działania. Takie dane mogą zostać następnie pobrane przez systemy obliczeń strumieniowych w celu dalszej analizy. Przetwarzanie danych strumieniowych ma zastosowanie dla danych w ciągłym ruchu, ale również w wielu przypadkach może być stosowane dla danych składowanych w repozytoriach danych. Do przetwarzania danych strumieniowych oprócz tworzenia bardziej wydajnych, rozproszonych, równoległych algorytmów wykorzystuje się także dedykowane urządzenia sprzętowe, które prócz procesorów wykorzystują często szybsze jednostki obliczeniowe, takie jak procesory graficzne czy programowalne macierze bramek. Systemy oparte o powyższe rozwiązania są najczęściej komercyjne i dedykowane do specjalnych zadań.

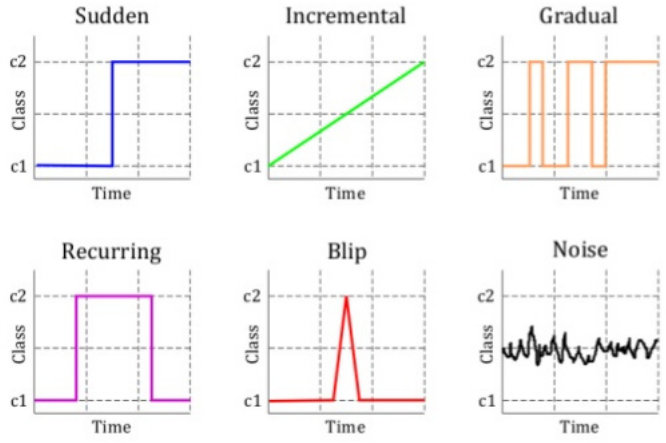
W systemach przetwarzających dane strumieniowe, oprócz zwykłego trybu przetwarzania danych w czasie rzeczywistym, wyróżniamy także przetwarzanie małych partii danych (*ang. micro-batching*). Można stwierdzić, że micro-barching jest przekrojem zbiorów reprezentujących przetwarzanie wsadowe i strumieniowe. Ideą tego przetwarzania jest grupowanie danych przychodzących w małych porcjach i przetwarzanie ich razem zamiast każdej przybyłej informacji osobno. Dzięki temu rozwiązaniu możemy tworzyć okna, gdzie wspólnie przetwarza się dane pochodzące z podanej porcji informacji. Możemy tworzyć okna ilościowe np. pięć ostatnich zdarzeń, lub czasowe - jak dane z ostatnich 10 sekund. Micro-batching był odpowiedzią na problemy wydajnościowe u kolebki tworzenia systemów przetwarzania strumieniowego. Na samym początku takie systemy miały kłopoty z efektywnym przetwarzaniem odpowiedzi oraz potwierdzania jej przetworzenia, w związku z czym malała przepustowość systemu. Twierdzono wtedy, że systemy strumieniowe nie zaistnieją samodzielnie, lecz będą jedynie uzupełnieniem przetwarzania wsadowego. Dzisiaj wiemy, iż wspomniana teza okazała się błędna, do czego w dużej mierze przyczynił się postęp informatyki, a w szczególności: opracowanie nowych algorytów, sposobów skalowalności systemów tudzież wzrost wydajności sprzętu oraz utworzenie nowych, szybszych rozwiązań pozwalających przechowywać dane i prędzej je pobierać.

W aplikacjach przetwarzania strumieni danych (oraz innych systemów informatycznych) ważną rzeczą jest odporność na awarie i gwarancja przetworzenia w całości

przybyłych danych. Pierwszą cechę pomaga nam zachować architektura rozproszona aplikacji. Wiele systemów przetwarzania danych, które wykorzystuje środowiska rozproszone dzieli się na serwery, które są nadzorcami *ang. supervisors* oraz zwykłe serwery robocze *ang. workers*, do których dedykowane są obliczenia. W przypadku, gdy wystąpi błąd w serwerze roboczym, nadzorca jest w stanie go zrestartować, a nawet oddelegować obliczenie do innej maszyny lub uruchomić nową instancję roboczą. Nadzorcy często są również projektowani jako bezstanowe aplikacje, by w razie problemów był możliwy ich restart bez wpływu na działanie całej platformy. Druga cecha polega na zagwarantowaniu klientowi, czy wysłana przez niego informacja zostanie przetworzona. Wyróżnamy tutaj kilka sposobów na zachowanie tej funkcjonalności. Pierwszym z nich jest *as least once* polegający na tym, że każde przetworzenie danej zostanie wykonane przynajmniej raz, jednak w przypadku awarii może być wykonane wielokrotnie aż do uzyskania wyniku. Wadą tego rozwiązania jest nie zawsze potrzebne powtarzanie obliczeń czy ponowne wysyłanie tej samej danej przez klienta. Druga metoda nazywa się *at most once*, wedle której przetwarzanie zostanie wykonane wyłącznie raz lub w ogóle. Tutaj podczas awarii aplikacji dane nie zostaną przetworzone i zgubimy informację o nich. Ostatnim sposobem jest *exactly once* oznaczający, że wiadomość zostanie przetworzona dokładnie raz. Jest to najbardziej oczekiwany sposób gwarancji przetworzenia wiadomości, ponieważ gwarantuje on klientowi, że raz wysłana wiadomość zostanie dokładnie raz przetworzona bez konieczności ponownej wysyłki w razie awarii. By uzyskać taką gwarancję w aplikacjach, stosuje się pamięci podręczne lub systemy kolejkowe, dzięki którym w przypadku awarii systemu wiadomości będą mogły zostać ponownie zaczytane bez konieczności kolejnej wysyłki ze strony klienta.

## 2.3 Concept Drift

Charakterystyka badanych przez nas danych może zmieniać się wraz z upływem czasu - mówimy wtedy o zmiennym środowisku, z którego pochodzą informacje. Wpływ na zmianę, w zależności od badanych danych, może mieć wiele czynników np. pora roku, dzień, aktualna pogoda czy odkrycie nowych złóż surowców naturalnych. Zmieniają się one wraz ze środowiskiem, z jakiego pochodzą analizowane dane. Zjawisko zmiany w nieprzewidywalny sposób właściwości klasy decyzyjnej, którą próbujemy przewidzieć, nazywamy concept drift. Innymi słowy, concept drift oznacza, że poprzedni wyuczony model - w wyniku zmiany charakterystyki danych - jest już nieaktualny i musimy zbudować go od nowa. Artykuły [6] [7] dostarczają informacji o concept drift oraz sposobach radzenia sobie z nim. Zjawisko to wpływa także na trafność oceny, czy dane przetwarzane w konkretnym czasie powinny być traktowane jako anomalie czy nie.




Rysunek 2.2: Rodzaje zmian w strumieniach danych

Źródło: MINING DATA STREAMS WITH CONCEPT DRIFT, Dariusz Brzeziński

Przykładem - pozwalającym na zrozumienie tego zjawiska - będzie wykrywanie anomalii pogodowych. W zimę temperatura -10 stopni w Polsce będzie uważana za normę i nikogo nie zdziwi. Gdyby ta sama wartość pojawiła się późną wiosną, powinna zostać zaklasyfikowana jako anomalia, ponieważ w naszym regionie o tej porze roku tak niskie temperatury nie występują. W zależności od pory roku zmienia się charakterystyka danych - odmienne pomiary są traktowane jako anomalie. System przetwarzający dane strumieniowe musi być odporny na zjawisko concept driftu i powinien umieć wykryć zmianę w danych możliwie najszybciej oraz poprawnie na nią zareagować.

Zmiany w danych mogą następować w różny sposób. Mogą być łagodnymi lub gwałtownymi przejściami z jednej charakterystyki w drugą. Na rysunku 2.2 zobrazowano rodzaje zmian występujących w strumieniach danych. Zmiany te mogą być nagłe (*sudden*), przyrostowe(*incremental*), zmieniające się stopniowo w czasie(*ang. gradual*) oraz powracające (*recurring*). Wymienione wyżej rodzaje zmian są naturalne i zależne od dziedziny, z której pochodzą. Nieporządanymi zmianami w danych są nagłe wyskoki(*ang. blip*) oraz szum(*ang. noise*). Systemy muszą być odporne na te zjawiska i powinny prawidłowo je wykrywać. Nagły, chwilowy wyskok nie może zostać zinterpretowany jako zmiana charaktetystyki danych. Szum jest zjawiskiem powstającym w przypadku, gdy w danych, które badamy, występują również przypadkowe pomiary zakłócające ogólną charakterystykę danych. Wykrycie fluktuacji nie jest proste, ponieważ należy zbudować mechanizm, który będzie potrafił odróżnić szum od prawdziwych zmian w danych. Musimy wtedy zastosować mechanizmy, które wyekstrahują tylko istotne z punktu analizy dane. W tym celu możemy zaimplementować moduł, który za pomocą zdefiniowanych reguł i/lub z użyciem uczenia maszynowego będzie filtrował strumień wejściowy pozbywając się szumu.

By dostosowywać model do zmian, powinniśmy zaimplementować mechanizmy odpowiedzialne za ich wykrycie - tzw. detektory zmian (*ang. triggers*). Sposoby dostosowania do przekształceń będą różniły się od ich rodzaju. Do wykrywania przemian w danych stosuje się klasyfikatory pojedyńcze oraz ich zbiory. Wyróżniamy wykrywanie zmian nagłe pod wpływem zewnętrznego czynnika lub detekcję na podstawie badania przybyłych do nas danych. Czynnikiem zewnętrznym może być informacja wydobyta podczas analizy danych statycznych bądź wynikająca z obserwacji i mówiąca np. o specyficznym zachowaniu użytkowników systemu w określone dni tygodnia, gdy pada deszcz. Istnieją techniki stosujące pojedyńcze klasyfikatory, które po wykryciu zmian zapominają o starych informacjach i przeprowadzają ponowne uczenie modelu dla nowych danych. Operują one na ruchomych oknach, w których wykrywane są zmiany np. za pomocą algorytmu ADWIN (*ADaptive Sliding WINdow*). W przypadku zespołów klasyfikatórów budujemy wiele modeli danych. W pierwszym przypadku zbudowane modele odpowiadają różnym charakterystykom danych. Gdy przybędzie nowa informacja, zostaje przeprowadzone głosowanie, na podstawe którego zostanie ona prawidłowo zaklasyfikowana. W drugim przypadku każdy zbudowany model odpowiada innej charakterystyce danych wynikającej z ich zmiany. Kiedy pojawia się nowa informacja, dokonujemy przełączania modelu danych w zależności, do którego została ona dopasowana. Metody oparte o zespoły klasyfikatorów częściej mają swoje zastosowanie w praktyce. Powszechnie stosuje się również zewnętrzne czynniki wspomagające detekcję concept driftu. Wracając do przykładu o wykrywaniu anomalii pogodowych - gdybyśmy dodali do niego element, który na podstawie dodatkowych danych kalendarzowych byłby w stanie przełączać model w zależności od pory roku czy nawet miesiąca.

Zjawisko concept driftu oraz techniki jego wykrywania mogą posłużyć do budowy systemu znajdującego anomalie w danych. Zmiany w charakterystyce przetwarzanych danych bądź zjawisko nagłego wyskoku można zaklasyfikować jako anomalie. System wykrywający nieprawidłowości z powodzeniem może zostać użyty jako detektor zmian w charakterystyce danych. Jako detektor anomalii można zastosować wspomniany wcześniej algorytm ADWIN. Może on zostać użyty w systemie wykrywającym anomalie jako czarna skrzynka wykrywająca zmiany w charakterystyce

danych. Tak wykryta zmiana może zostać od razu rozpoznana jako anomalia lub posłużyć jako sygnał dla dalszych algorytmów uczenia maszynowego, dzięki któremu będą one w stanie przełączyć model na bardziej odpowiedni do zaistniałej sytuacji. System korzystający z wyniku platformy wykrywającej anomalie powinien zareagować w odpowiedni sposób np. automatycznie zmieniając ustawienia maszyny lub powiadomić człowieka o danych, które nie są danymi uważanymi z jego punktu widzenia za normalne.

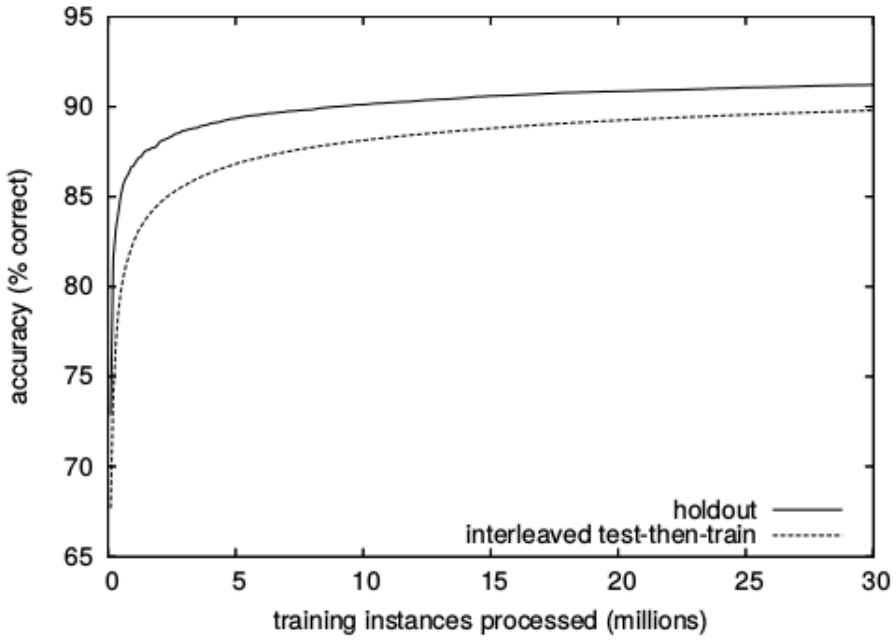
## 2.4 Uczenie maszynowe w danych strumieniowych

Uczenie maszynowe (*ang. machine learning*) jest dziedziną sztucznej inteligencji zajmującą się badaniem algorytmów i systemów usprawniających swoje działanie razem ze zdobywanym doświadczeniem. Usprawnienie działania systemu może polegać na podejmowaniu lepszych decyzji wraz z upływem czasu np. poprzez dostarczanie nam lepszych rekomendacji produktów po analizie naszych zakupów czy fraz wpisywanych w wyszukiwarki internetowe. Algorytmy te możemy podzielić na kategorie w zależności od przebiegu procesu uczenia. Uczenie nadzorowane (*ang. supervised learning*) polega na wcześniejszym uczeniu się algorytmu z przygotowanych wzorców. Przykładem algorytmu reprezentującego uczenie nadzorowane są algorytmy klasyfikacji np. drzewa decyzyjne czy naiwny klasyfikator Bayesa. Uczenie nienadzorowane (*ang. unsupervised learning*) polega na bazowaniu algorytmu na obserwacji danych, a w szczególności poszukuje związków przyczynowo-skutkowych czy korelacji między danymi. Przykładem wykorzystującym tę technikę jest analiza skupień, która może korzystać z algorytmu k-means czy algorytmu strumieniowego CluStream. Istnieje także uczenie pół-nadzorowane polegające na uczeniu ze zbioru danych, w którym jest mała ilość informacji oznaczonych oraz duża ilość danych bez etykiet.

Uczenie maszynowe w połączeniu z danymi strumieniowymi jest trudniejsze niż w zastosowaniach wsadowych. Powinniśmy uzyskać wynik obliczeń w możliwie najkrótszym czasie po pojawieniu się badanej informacji, mając do dyspozycji ograniczone zasoby pamięciowo-obliczeniowe. Musimy być także gotowi na nieustanną analizę informacji, a nie - jak w przypadku danych wsadowych - w określonym czasie, co oznacza, że zbudowany model musi być dostępny przez cały czas działania aplikacji strumieniowej. Ważnym problemem do rozwiązania podczas uczenia maszynowego na danych strumieniowych jest również wykrycie zjawiska concept drift, które zostało opisane w poprzednim podrozdziale.

Do oceny algorytmów uczenia nadzorowanego operujących na danych strumieniowych musimy użyć innych procedur. Nie znajdzie w tym miejscu zastosowania sprawdzian krzyżowy (*ang. cross-validation*) używany najczęściej przy uczeniu z danych wsadowych, ponieważ dane strumieniowe ciągle przybywają, mamy ograniczony czas na procedurę uczenia, co utrudnia powtarzanie tego kroku wiele razy. Musimy zastosować techniki budowy modelu danych, które zredukują proces uczenia i umożliwią przeprowadzenie go w jak najkrótszym czasie. W książce [8] w rozdziale *2.2 Evaluation Procedures for Data Streams* autorzy zaproponowali dwie procedury pozwalające na ocenę klasyfikatorów działających na danych strumieniowych. Pierwsza z nich polega na zastosowaniu pojedyńczego podziału na zbiór uczący i testujący (*ang. holdout set*), druga natomiast - na używaniu przybyłych danych najpierw do testowania modelu, a następnie do jego uczenia (*ang. Interleaved Test-Then-Train* lub inaczej *Prequential*). Holdout używamy, gdy mamy dostępny zbiór danych historycznych, dzięki któremu możemy zbudować model. W tym rozwiązaniu najczęściej dokonujemy podziału zbioru danych w proporcjach 2/3 dla uczenia, 1/3 dla testów. Ważne jest, by dane testujące nie były użyte do budowy klasyfikatora, co mogłoby spowodować błąd jego oceny. W tym przypadku możemy oceniać modele cyklicznie np. co milion przykładów, by nie powodować spadku wydajności procesu przetwarzania. Do budowy modelu można wykorzystać także zadania wsadowe czerpiące z danych historycznych. Zadanie takie może wy-

konywać korektę w utworzonym modelu np. przez analizę danych historycznych czy zmianę wprowadzoną przez czynnik zewnętrzny np. eksperta, który poprawia wyliczoną wartość przez algorytm. Aktualizacja modelu również odbywa się cyklicznie. Prequential, jak wcześniej wspomniano, polega na używaniu każdej przybyłej informacji najpierw do testowania modelu, a następnie do jego uczenia. Kolejność nie jest tutaj przypadkowa, ponieważ model jest testowany przez przykład, z którym nie miał wcześniej styczności. Zaletą tego rozwiązania jest brak podziału na zbiór uczący i testujący, przez co wszystkie dostępne dane są wykorzystywane do budowy modelu. Nie potrzebujemy także wydzielać osobnej pamięci potrzebnej do przechowywania zbioru testującego. Powyższa technika z powodzeniem może być stosowana zarówno do przetwarzania statycznych danych strumieniowych oraz zmieniających swoją charakterystykę.




Rysunek 2.3: Wyniki uczenia z użyciem techniki holdout oraz prequential.

Źródło: Albert Bifet, Richard Kirkby DATA STREAM MINING A Practical Approach (Figure 2.2)

Na rysunku 2.3 widzimy porównanie dokładności modeli predykcyjnych z użyciem techniki holdout oraz prequential. W przykładzie dla obydwóch metod zostały użyte te same zbiory danych. Oceny pracy klasyfikatorów dokonywano co sto tyś. przykładów. Na wykresie można zauważyć, że stabilizuje się on po przetworzeniu ok. miliona przykładów. Ponadto widać, że dokładność metody *prequential* rośnie wraz z wzrostem przetworzonych danych wolniej niż w metodzie *holdout*. Podsumowując, powyższe metody nie będą wydajne, gdy będziemy operować na małych zbiorach danych - tutaj lepszym wyborem jest technika walidacji krzyżowej. Metoda *Test-Then-Train* bardziej nadaje się do danych o zmiennej charakterystyce, ponieważ w tej technice model jest aktualizowany na bieżąco, co pozwala nam reagować na zmiany w danych. Istnieje jeszcze procedura przetwarzania nazwana *data chunks*. Jest ona bardzo podobna do metody *Test-Then-Train*. Polega na testowaniu i aktualizacji modelu dopiero, gdy zbierzemy pewien kawałek danych (*chunk*), którego rozmiar będzie z góry określony. Takie kawałki danych najpierw będą użyte do testowania istniejącego modelu, a następnie do jego aktualizacji. Zaletą tego rozwiązania jest ograniczenie czasu potrzebnego na trenowanie i testowanie, ponieważ nie wykonujemy tych akcji dla pojedyńczej instancji danych, a dla ich zbioru. Metodę tą można porównać z przetwarzaniem danych określanym jako micro-batching.

By spełnić wymagania stawiane uczeniu maszynowemu w danych strumieniowych

dokonano próby adaptacji algorytmów do nowych architektur systemów, które pozwalają na przyspieszenie obliczeń. Zaczęto wymyślać algorytmy mogące działać w środowiskach rozproszonych, gdzie praca algorytmu może w pewnym stopniu zostać zrównoleglona. Wraz z rozwojem systemów przetwarzania danych strumieniowych zaczęły powstawać przy nich biblioteki pozwalające na uczenie maszynowe. Przykładem jest platforma Apache Spark Streaming wraz z biblioteką MLib. Zauważono także potencjał, jaki tkwi w mocy obliczeniowej zawartej w procesorach graficznych (*Graphics Processing Unit [GPU]*), które okazują się być wydajniejsze oraz pozwalają na większe zrównoleglenie obliczeń niż procesory CPU. Zaczęto implementować algorytmy uczenia maszynowego przystosowane do wykonywania obliczeń za pomocą procesorów graficznych. Przykładem takiej platformy jest Nvidia CUDA [9]. Dostarcza ona wiele gotowych rozwiązań oraz interfejsy programistyczne dla języków programowania takich jak Python, C++ czy Matlab. Zaczęły także powstawać rozwiązania, które łączą przetwarzanie w platformach rozproszonych oraz przetwarzanie z użyciem kart graficznych - HeteroSpark utworzona w celu implementacji uczenia głębokiego. Autorzy rozwiązania postawili na elastyczność architektury. Jednostki GPU są jednostkami pomocniczymi, które mają za zadanie przyspieszyć obliczenia wykonywane na poszczególnych węzłach platformy. Mamy możliwość podłączania procesorów graficznych działających na zdalnych maszynach lub lokalnie. Możemy je również wyłączyć - wtedy Spark będzie wykonywał obliczenia tylko na swoim klastrze.

Oprócz próby adaptacji istniejących algorytmów powstały także nowe, z założenia przystosowane do działania na danych strumieniowych. Przykładem takiego algorytmu może być strumieniowa wersja algorytmu analizy skupień - CluStream. Jest to algorytm klastrowania bazujący na koncepcji mikroklastrów (obrazów danych z pewnego przedziału czasu), dzięki której łatwo możemy dokonywać aktualizacji oraz szybkiej analizy w czasie rzeczywistym. Mikroklastry są podstawową strukturą danych, która przechowuje zaktualizowany model. Algorytm posiada dwie fazy: online oraz offline. W fazie pierwszej zbiór mikroklastrów jest przechowywany w pamięci operacyjnej. Przychodzące dane zostają dołączone do istniejących lub tworzą nowe mikroklastry. Gdy algorytm zauważy, że wraz z pojawieniem się nowych danych zachodzi potrzeba rekonfiguracji klastrów, może dokonać próby połączenia dwóch najbliższych klastrów lub podjąć decyzję o usunięciu jednego z nich. Wskutek tej operacji powstaje miejsce na nowy klaster. W fazie drugiej, offline, zostaje użyty wagowy algorytm k-means obliczający końcowe klastry na podstawie mikroklastrów [10]. Drugi przykład algorytmu przystosowanego do analizy danych strumieniowych stanowi algorytm Very Fast Decision Tree (VFDT), inaczej nazywany Hoeffding Trees, o którym wspomina książka [8]. Jest to algorytm klasyfikacji oparty o drzewa decyzyjne, który przystosowano do pracy w środowiskach rozproszonych, ponieważ potrafi przeprowadzać obliczenia równolegle. Wyróżniamy zrównoleglenia horyzontalne oraz wertykalne. Pierwsze oznacza podział instancji danych, drugie podział danych ze względu na atrybuty. Algorytm z sukcesem może zostać wykorzystany wraz z techniką *prequential evaluation*. Implementacja tego algorytmu, a dokładniej Vertical Hoeffding Tree, który jest implementacją algorytmu VFDT oferującym pionowe zrównoleglenie, została dokonana na platformie Apache SAMOA. Platforma ta, jako procedury przetwarzania danych, wykorzystuje procedurę *Test-Then-Train*. Na szczególną uwagę zasługuje odmiana omawianego algorytmu nazwana Concept-adapting Very Fast Decision Tree (CVFDT), czyli rozszerzenie algorytmu VFDT potrafiące wykryć zmiany w danych, zaadapotować się do nich oraz zaktualizować model.

## 2.5 Przegląd narzędzi przetwarzania danych strumieniowych

Obecnie istnieje wiele narzędzi pozwalających przetwarzać strumienie danych. Są to systemy samodzielne, ale zaczynamy także zauważać trend tworzenia aplikacji przetwarzania danych strumieniowych na komercyjne systemy wchodzące w skład szeroko pojętych hurtowni danych (np. systemy ETL, BI). Widać też co-

raz więcej rozwiązań w postaci usług, gdzie użytkownik może wykupić dostęp do zdalnej platformy analitycznej bez potrzeby jej instalacji. Powstają również dedykowane rozwiązania sprzętowe do analizy danych oparte o szybkie nośniki danych oraz duże pamięci operacyjne. Platformy przetwarzające dane strumieniowe - szczególnie te z otwartym kodem źródłowym - mają kilka cech wspólnych. Jedną z nich jest udostępnianie przez platformę narzędzia webowego, dzięki któremu jesteśmy w stanie obserwować stan klastra obliczeniowego oraz interaktywnych konsol (*read-eval-print loop (REPL)*) pozwalających na pisanie kodu przeznaczonego na daną platformę, a zarazem natychmiastowo wyświetlających wynik wpisanej przez nas operacji. Wskutek tego łatwo możemy stawiać pierwsze kroki w danej technologii. Mają one także wsparcie dla wielu języków programowania. Kluczową rolę w dziedzinie przetwarzania danych odgrywają języki, które łączą w sobie cechy języków funkcyjnych i obiektowych - np. Scala, Python oraz Java. Ta ostatnia - dzięki wprowadzeniu od wersji ósmej wsparcia dla pewnych mechanizmów funkcyjnych - wydaje się mniej ograniczona niż wcześniej. Istnieje także język R, który jest dedykowanym językiem do obliczeń statystycznych, czyli dobrym narzędziem do analizy danych posiadającym rozszerzenia pozwalające na analizę danych strumieniowych. Większość systemów posiada także moduły odpowiedzialne za uczenie maszynowe, przetwarzanie danych grafowych czy dedykowane języki stworzone dla danej platformy, często składnią wzorujące się na języku SQL.

Poniżej zostaną opisane narzędzia, na które warto zwrócić uwagę, gdy rozważamy utworzenie systemu analizy danych strumieniowych. Lista to ledwie niewielki obraz tego, co obecnie jest dostępne na rynku, lecz narzędzia zostały dobrane tak, by zwrócić uwagę na różnorodną architekturę czy podejście w projektowaniu systemu. Oprócz systemów odpowiadających za analizę danych, opisano także wartą uwagi bibliotekę odpowiedzialną za rozproszone uczenie maszynowe. Przedstawiono również system, który może zostać wykorzystany jako baza danych dla danych strumieniowych oraz skalowalnego pośrednika wiadomości, mogącego z powodzeniem pośredniczyć w dostarczaniu danych do systemu przetwarzania danych, a także w odbiorze wyników z tego systemu.

- **Apache Storm** - rozproszony system stworzony do obliczeń na danych w czasie rzeczywistym stworzony przez zespół Backtype, a następnie rozwijany przez Twittera. Jego ideą jest tworzenie topologii, która przyjmuje postać skierowanego grafu bez cykli. Przez topologię przepływają strumienie danych. W grafie wyróżniamy kilka rodzajów węzłów. Wylewki (*ang. spouts*) są źródłem strumienia danych. Wylewkami mogą być systemy kolejkowe. Gromy (*ang. bolts*) wykonują obliczenia na strumieniach, mogą mieć wiele wejść i wyjść. Storm zapewnia przetwarzanie at-least-once. Architektura systemu sprawia, że tworzenie nowych topologii jest bardzo intuicyjne, lecz programista jest zmuszony tworzyć bardzo duże ilości kodu. By uprościć pisanie aplikacji na platformę, stworzono bibliotekę Trident. Biblioteka ta wprowadza abstrakcję, dzięki której wiele problemów możemy rozwiązać szybciej i mniejszym nakładem pracy. Trident wprowadza lepsze zarządzanie stanem topologii, a z powodu zastosowania abstrakcji nie jest ważne, czy do przechowywania stanu wykorzystujemy pamięć operacyjną czy bazę danych, ponieważ interfejs do każdego źródła danych jest taki sam. Trident poprawia gwarancję przetworzenia danych przez topologię na exactly-once. Powstało także rozszerzenie biblioteki o nazwie Trident-ML, które dostarcza algorytmy uczenia maszynowego. Niestety Trident wprowadza dodatkową logikę wpływającą na wydajność przetwarzania i tam, gdzie potrzebujemy bardzo dużej wydajności, musimy z niego zrezygnować. Storm jest wykorzystywany obecnie w wielu projektach oraz istnieje do niego duża ilość dokumentacji, przez co na pewno jest wartym uwagi systemem, którego działanie zostało potwierdzone w praktyce. Na platformę powstaje także wiele projektów, które ją rozszerzają o nowe funkcje i algorytmy. [11] [12]

- **Heron** - system Heron został stworzony przez Twittera, który nie był do końca zadowolony z wydajności Apache Storm. Heron czerpie bardzo wiele rzeczy ze

Storma. Również tworzymy tutaj topologię przetwarzania danych. Co ważne, topologie zbudowane na Apache Storm będą mogły zostać zmigrowane na platformę Heron bez ingerencji w kod. Heron względem Storma poprawia możliwości związane z debugowaniem topologii, poprawia konsolę zarządzania oraz skalowalność w budowie topologii, by zachować jak najmniejsze opóźnienia wewnątrz niej. Heron jest także wydajniejszy od Storma, co potwierdziły przeprowadzone przez twórców eksperymenty. Wykazały one, że przepustowość Herona jest od 10 do 14 razy szybsza względem Storma, a opóźnienia są mniejsze od 5 do 15 razy. Eksperyment wykazał również, że Heron zużywa mniej zasobów systemowych [13]

- **Apache Kafka** - systemy strumieniowe często zamiast wystawiać własny interfejs, przez który aplikacje klienckie dostarczałyby dane wejściowe, używają pośredników (*ang. brokers*), do których to trafiają komunikaty od klientów, a następnie system pobiera z nich dane. Apache Kafka jest to pośrednik wiadomości stworzony z myślą o obsłudze danych w czasie rzeczywistym przez Linkedin. Cechami systemu są odporność na błędy, duża przepustowość, małe opóźnienia w przetwarzaniu oraz skalowalność i działanie w środowisku rozproszonym. Platforma posiada podział wiadomości na tematy (*ang. topic*). Temat jest to kategoria wiadomości, które są w nim publikowane. Kafka obsługuje wielu producentów oraz konsumentów danego tematu. Każdy z konsumentów musi mieć zapisaną informację, ile danych do tej pory pobrał z pośrednika, by np. po jego restarcie nie pobierać wiadomości, które już zostały przez niego przetworzone. Taka informacja jest nazywana przesunięciem (*ang. offset*). Dzięki tym wszystkim cechom jest dobrym narzędziem do budowy przepływów danych strumieniowych, przez co często jest przedstawiana w połączeniu z silnikami przetwarzania danych rzeczywistych. Zapewnia gwarancję przetworzenia danych at-least-once. [14]

- **Apache Samza** - system stworzony przez Linkedin razem z Apache Kafka, dzięki czemu współpraca tych systemów jest już zaimplementowana przez twórców. Architekturę oparto o graf skierowany acykliczny. Elementami, które dokonują obliczeń, są tutaj zadania (*ang. jobs*), w których źródłem, jak i wynikiem, są dane strumieniowe. Zadanie na wejściu i wyjściu może mieć wiele strumieni. Taka architektura jest podyktowana bardzo dobrą współpracą Samzy z Kafką. Wszystkie dane są pobierane, a następnie wysyłane na Kafkę. Zadania są następnie dzielone na podzadania(*ang. task*). Liczba podzadań odpowiada stopniowi zrównoleglenia zadania. Każde podzadanie jest konsumentem jednego strumienia danych, czyli jednej partycji brokera Kafka. Liczba podzadań jest ustalona przez liczbę strumieni, jakie są na wejściu zadania. Podzadania zawsze mają przypisane do siebie konkretne partycje, dzięki czemu mamy pewność, że w przypadku wznowienia pracy po wystąpieniu awarii będziemy czytać te same dane. [15]

- **Apache Spark Streaming** - stworzony na uniwersytecie Berkley w Kalifornii. Spark Streaming jest częścią platformy Spark, która służy do rozproszonych obliczeń na dużej ilości danych. Platforma ta, oprócz rozszerzenia do analizy danych strumieniowych, posiada także biblioteki wspierające uczenie maszynowe oraz analizę danych grafowych. Sam projekt Spark został okrzyknięty następcą technologii Hadoop, ponieważ eliminował zapis częściowych wyników pomiędzy operacjami np. do systemu plików HDFS na rzecz przechowywania ich w pamięci operacyjnej, do której jest szybszy dostęp. Do tej cechy przyczynił się model danych użyty w architekturze Spark oparty o rozproszony, niezmienny model danych (*ang. Resilient Distributed Dataset [RDD]*). Dzięki zastosowaniu RDD dane mogą być wymieniane między wykonawcami programu rozproszonymi w klastrze. Obiekty RDD są bardzo elastyczne i mogą przechowywać różne typy danych, a nawet zbiory danych np. pary. RDD jest modelem wejściowym oraz wyjściowym z zadań platformy Spark. Sam moduł odpowiedzialny za przetwarzanie strumieniowe jest oparty o przetwarzanie małych porcji danych (micro-barch), a nie każdej przybyłej danej osobno, dlatego, de facto,

nie jest uważany za system przetwarzania strumieniowego. Do przetworzenia danych potrzebuje czasu - musi poczekać na zapełnienie się okna wyznaczonego przez projektanta algorytmu, dopiero wtedy przystępuje do obliczeń na komplecie zgromadzonych danych. Spark Streaming posiada zaimplementowane połączenia do popularnych źródeł danych, takich jak Kafka, Flume, HDFS czy Twitter. [16]

- **Apache Flink** - jest to platforma stworzona przez społeczność wspierającą wolne oprogramowanie. Flink wywodzi się z projektu Stratosphere, który został utworzony na berlińskich uczelniach wyższych, a pod nazwą Flink istnieje dopiero od 2014 roku. Architektonicznie przypomina platformę Spark, jednakże jest uważany za jej następcę. Głównym elementem platformy jest silnik nazwany *Distributed Streaming Dataflow*. Ideą jest zamiana danych w ciągły ich przepływ, na którym są wykonywane obliczenia, by zminimalizować zapis pośrednich wyników obliczeń. Silnik ten jest stosowany zarówno do przetwarzania danych strumieniowych - do tego został głównie stworzony, ale pozwala także obliczać informacje wsadowe. Strumień danych reprezentują obiekty DataStream, które są głównym modelem danych systemu. W przeciwieństwie do Sparka, wszystkie dane są przetwarzane na bieżąco - nie mamy tutaj do czynienia z przetwarzaniem typu micro-batch, co wpływa na poprawę przepustowości i redukcję opóźnień. Zoptymalizowano także operacje wykonywane na danych, takie jak join, map, reduce itd. System posiada zaimplementowane połączenia do wielu źródeł danych - np. Kafka, Flume czy Twitter. Posiada także dedykowane biblioteki do przetwarzania danych grafowych oraz uczenia maszynowego. Istotną zaletą Flinka jest kompatybilność z topologiami stworzonymi na platformę Storm. Możemy zmigrować cały graf bez zmian lub użyć stworzonego wcześniej kodu np. implementującego gromy wewnątrz operacji Flinka. [17]

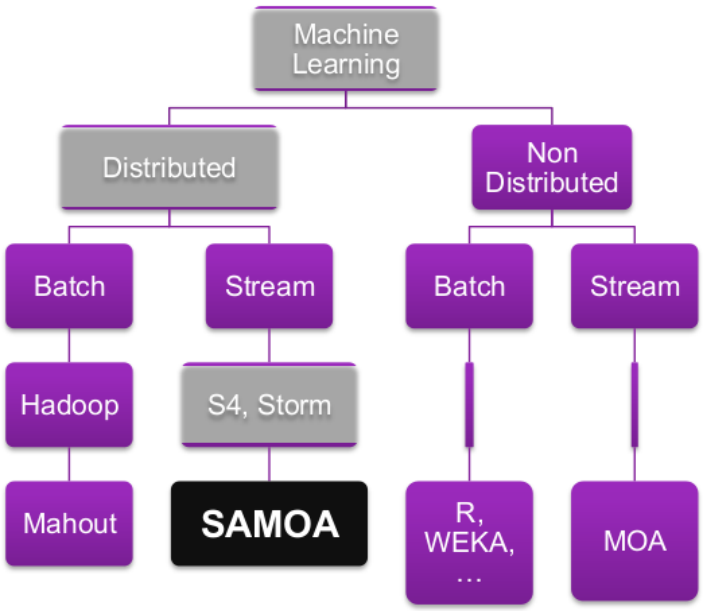
- **Amazon Kinesis** - to część architektury Amazon Web Services (AWS) stworzonej przez firmę Amazon. Wyróżnikiem owej platformy jest jej działanie w chmurze. Użytkownicy mogą z niej korzystać w modelu platformy jako usługi (*ang. Platform as a service (PaaS)*). Dzięki temu rozwiązaniu użytkownik skupia się wyłącznie na pisaniu kodu aplikacji bez myślenia o zapewnieniu zasobów sprzętowych pod samą platformę. Zaletą Amazon Kinesis jest także jego integracja z pozostałymi komponentami chmury AWS np. bazą danych Dynamo czy Amazon Kinesis Analytics. Jednakże należy do systemów komercyjnych. [18]

- **Druid** - to magazyn danych stworzony do wykonywania operacji analitycznych oraz nastawiony na małe opóźnienia przy zapisie danych. Oparty jest o architekturę rozproszoną i nastawiony na przechowywanie danych rzędu petabajtów oraz przyjmowaniu milionów informacji na sekundę. Dzięki tym cechom stanowi bardzo dobre narzędzie, które może uzupełniać systemy przetwarzania danych strumieniowych. Jest on rozwijany na zasadach otwartego oprogramowania, dzięki czemu w oparciu o tę platformę powstały projekty, które pozwalają na wizualizację informacji w niej przechowywanych.[19]

- **Massive Online Analysis (MOA)** - powstała na uniwersytecie Waikato w Nowej Zelandii. Jest to biblioteka używana do analizy danych strumieniowych. Zawiera dużą kolekcję algorytmów uczenia maszynowego, potrafi także obsługiwać zjawisko concept drift. MOA, oprócz intefejsu linii poleceń, udostępnia także graficzny interfejs użytkownika oraz API dla języka Java. Poważnym minusem platformy jest brak wsparcia do skalowalności i rozproszonej architektury, co wyklucza ją z zastosowań analizy danych, w której potrzeba większej liczby maszyn do wykonywania szybszych obliczeń. [20]

- **Apache SAMOA** - biblioteka stworzona przez Yahoo Labs, a następnie wydana pod licencją Apache. Samoa jest skrótem od Scalable Advanced Massive Online Analysis, co możemy tłumaczyć jako zaawansowana, skalowalna analiza danych strumieniowych. To biblioteka pozwalająca na implementację

rozproszonych algorytmów uczenia maszynowego. Jej taksonomię przedstawiono na rysunku 2.4. SAMOA, jako platforma, ma wiele cech wspólnych z MOA, ponieważ przy tworzeniu obu bibliotek brał udział Albert Bifet. W przeciwieństwie do MOA potrafi obsługiwać rozproszone środowiska i jest w pełni skalowalna. Architektura biblioteki pozwala na łączenie jej z popularnymi platformami przetwarzania danych strumieniowych, takimi jak Storm, S4, Samza czy Flink. Można także uruchomić ją w trybie lokalnym bez potrzeby instalacji na silnikach przetwarzających dane strumieniowe. Obecnie SAMOA ma zaimplementowane algorytmy Vertical Hoeffding Tree (klasyfikacja), CluStream (klastrowanie) oraz Adaptive Model Rules(regresja). Dużą zaletą jest udostępniony przez bibliotekę interfejs pozwalający na tworzenie własnych rozproszonych implementacji algorytmów. Głównym obiektem jest procesor realizujący obliczenia związane z algorytmem. Wiadomości przesyłane w platformie określa się jako ContentEvent. Zawierają one informacje, które będą przetwarzane przez procesor. Procesory można łączyć ze sobą za pomocą obiektów Stream. Połączone procesory, implementujące określony algorytm, nazywamy topologią, która jest zamknięta w obiekcie Task. Następnie, przy uruchamianiu konkretnego zadania, podajemy nazwę zadania, które chcemy uruchomić. Istnieje także obiekt Learner służący do uczenia modelu wykorzystywango przez inne algorytmy. SAMOA posiada także adaptery, które pozwalają jej korzystać z innych bibliotek przetwarzania danych strumieniowych np. MOA. Dzięki adapterowi możemy uruchomić algorytmy MOA np. w topologii Storm. Jest to stosunkowo młody projekt, cały czas rozwijany. Minusem biblioteki - na obecnym etapie - jest mała liczba zaimplementowanych algorytmów oraz brak interfejsów, które pozwalałyby zwrócić wynik przetwarzania każdej przybyłej danej osobno. [21]




Rysunek 2.4: Apache SAMOA - taksonomia

Źródło: https://svn.apache.org/repos/asf/incubator/samoa/site/images/slideshow/

- **SAP HANA Smart Data Streaming** - jest częścią komercyjnej platformy SAP HANA odpowiedzialną za przetwarzanie danych strumieniowych. SAP HANA to platforma analizy danych, która, oprócz samego oprogramowania, dostarcza klientowi dedykowane rozwiązania sprzętowe, a nawet bazę danych. Aplikacja dostarcza pełnej analizy danych strumieniowych, ponadto potrafi wizualizować dane w postaci raportów i interaktywnych pulpitów. Platforma dostarcza kompleksowego narzędzia analizy danych, w którym klient owego systemu nie musi decydować o sprzęcie czy doborze pojedyńczych technologii,

bibliotek bądź aplikacji, z których powinien składać się system. [22]

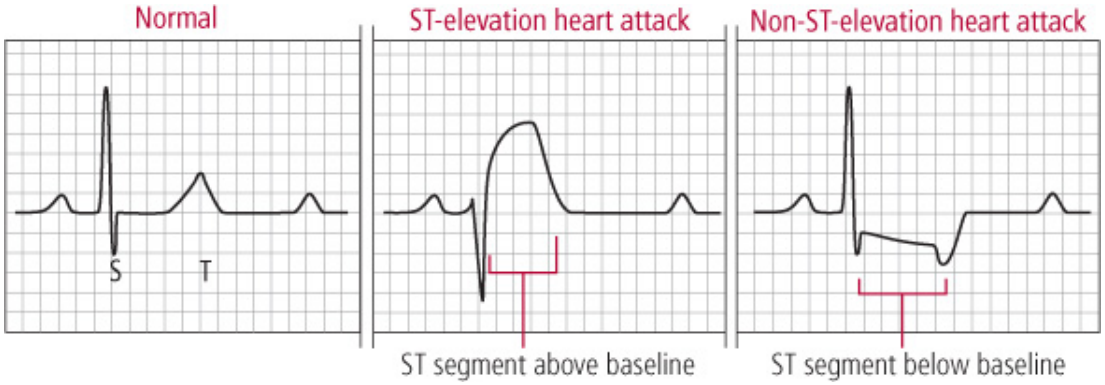
- **TensorFlow** - biblioteka stworzona przez Google wydana na licencji Apache 2.0 zawierająca mechanizmy uczenia maszynowego. Biblioteka bardzo dobrze nadaje się do zastosowania w systemach potrzebujących głębokiego uczenia. Jest wykorzystywana przez Google w swoich komercyjnych projektach, takich jak Gmail, Google Fotografie czy rozpoznawanie mowy. Biblioteka bazuje na przepływie danych poprzez graf skierowany utworzony przez programistę. Węzły grafu reprezentują operacje na danych i nazywane są ops (skrót od angielskiego *operations*). Krawędzie to drogi, po których dane docierają do operacji. Krawędzie w grafie reprezentują tensory, czyli wielowymiarowe tablice, dzięki którym dane są przekazywane z jednej operacji do drugiej. Obliczenia w wierzchołkach są wykonywane asynchronicznie i równolegle, gdy wszystkie tensory, wymagane przez daną operację, są dostępne. Biblioteka jest niezależna od platformy oraz wspiera obliczenia na procesorach graficznych GPU. [23]

# Rozdział 3

# Wyszukiwanie anomalii w danych

## 3.1 Anomalie w danych

Według Słownika Języka Polskiego anomalia to nieprawidłowość, odchylenie od normy. Ta sama definicja odnosi się do anomalii w danych, czyli odchylenia pewnej wartości lub zbioru wartości od przyjętej w zbiorze danych normy dla tych wartości. By zobrazować dokładnie, czym jest anomalia, posłużmy się przykładem wykresu Elektrokardiografii (EKG). EKG jest to zabieg diagnostyczny wykorzystywany w medycynie w celu rozpoznawania chorób serca [24]. Wynikiem badania jest wydruk zwany *elektrokadiogramem*, który obrazuje pracę serca.




Rysunek 3.1: Wykres EKG przedstawiający pracę serca osoby zdrowej oraz osoby z atakiem serca.

Źródło: http://www.patienteducationcenter.org/articles/coping-with-heart-attack/

Na rysunku 3.1 przedstawiono wykresy załamków EKG. Pierwsza ilustracja przedstawia pracę zdrowego serca, dwie następne obrazują wystąpienie zawału mięśnia sercowego: z uniesieniem odcinka ST elektrokardiogramu oraz bez jego uniesienia. Dwa ostatnie wykresy przedstawiają chorobę serca, czyli sytuację, która nie jest normalną pracą zdrowego narządu. Bazując na danych z aparatury, musimy zadać sobie pytanie, jakie wyniki będziemy uważać za odstępstwo - odpowiedź na postawione zapytanie jest pierwszym krokiem w procesie wyszukiwania sytuacji wyjątkowych w danych. Musimy jasno sprecyzować, jakie dane uważamy za normę - pozostałe pomiary, nie mieszczące się w wyznaczonym przez nas progu, będą uważane za anomalię. W powyższym przypadku za normę wyniku EKG przyjęto dane z wykresu pierwszego, czyli pracę serca zdrowego człowieka. Wyniki, których użyto do stworzenia dwóch kolejnych wykresów obrazujących chorobę narządu, są odstępstwem od przyjętej normy, czyli będziemy je uważać za anomalię w pracy serca.

Do wyszukiwania anomalii możemy podejść na dwa sposoby: szukając anomalii negatywnych lub pozytywnych [25]. Anomalią negatywną będziemy nazywać sytuację, gdzie w zbiorze danych za normę przyjmujemy dozwolone wartości danych, a za wyjątki dane, które powodują zaburzenie tej sytuacji w negatywnym znaczeniu. Przykładem takiego odstępstwa jest przytoczony przykład badania EKG, ponieważ

wykryta anomalia oznacza chorobę. Z anomalią pozytywną mamy do czynienia w przypadku, gdy jako normę przyjmiemy wartości, które są wadliwe lub normalne dla badanego zbioru danych, lecz jako anomalię przyjmiemy odstępstwo od tych wartości w pozytywnym znaczeniu - np. wyłonienie najlepiej rokującego zawodnika poprzez badanie cech jego organizmu podczas zawodów sportowych.

Ważnym zagadnieniem jest także określenie progu, od jakiego dane będziemy traktować jako błędy i po przekroczeniu którego system zgłosi np. alert o wystąpieniu błędu. W większości prostych systemów próg określa się manualnie przez podanie miar, które go charakteryzują np. wartości danych, po jakiej ma nastąpić zgłoszenie alertu. W określeniu progu pomaga algorytm *t-digest* pozwalający na wywnioskowanie go ze zgromadzonych danych historycznych [26]. W bardziej złożonych systemach wykrywających anomalie - szczególnie tych, gdzie źródłem są dane strumieniowe - proste określenie progu nie wystarcza. W strumieniach danych często mamy do czynienia z szumem informacyjnym, zjawiskiem nagłego wyskoku czy ze zmianą charakterystyki danych, przez co potrzebujemy bardziej wyrafinowanych metod oczyszczających dane ze zbędnych informacji oraz potrafiących wykryć anomalie w środowiskach, gdzie charakterystyki danych ulegają ciągłym zmianom (zjawisko concept drift). Z pomocą przychodzi nam uczenie maszynowe, z którego algorytmy są bardziej elastyczne niż sztywno przyjęte reguły określające, co jest normalne, a co powinniśmy uznać za anomalię.

Ważnym aspektem w kontekście wyszukiwania anomalii jest także ich natura. Odstępstwa możemy podzielić na następujące kategorie [27]

- anomalie punktowe - mamy z nimi do czynienia, gdy pojedyńcza informacja odstaje od reszty. Taka anomalia jest często zobrazowana poprzez nagły wyskok. Przykładem może być wykrycie przez system bankowy nagłego, dużego upływu pieniędzy z konta

- anomalie wynikające z kontekstu - dane są uważane za anomalie, ale tylko w pewnym kontekście. W innej sytuacji nie zostałyby sklasyfikowane jako dane anormalne. Anomalie tego typu są powszechnie badane w danych uszeregowanych czasowo (*ang. time-series data*) czy danych przestrzennych. Przykładem może być analiza danych pogodowych, szczególnie temperatury. Wystąpienie temperatury bliskiej 0 °Celsjusza w lecie jest uważane za anomalię pogodową, ale wystąpienie owej temperatury w zimę to sytuacja jak najbardziej normalna

- anomalie zbiorowe - pojawiają się, gdy zbiór danych występujących razem, jedna po drugiej, jest nienormalny w odniesieniu do całego zbioru danych. Wystąpienie pojedyńczej danej ze zbioru nie powoduje wystąpienia anomalii - takich przypadków musi wystąpić więcej. Przykładem może być wykrycie ataku typu DoS (*Denial of Service*). Pojedyńczy request nie powinien zostać uznany za odstępstwo, ale zbiór bardzo dużej liczby requestów podobnych do siebie spowodowuje uruchomienie alertu.

## 3.2 Techniki szukania anomalii w danych

Wykrywanie anomalii stosujemy, by ze zbioru danych wyodrębnić informacje, które nie spełniają przyjętych przez nas norm. Normy te powinniśmy umieć zdefiniować i na ich podstawie zbudować model pozwalający na wyodrębnienie anomalii. Mówimy wtedy o nadzorowanym wyszukiwaniu anomalii, ponieważ budujemy model w oparciu o dane, które potrafimy skategoryzować. Niestety, nie zawsze jesteśmy w stanie utworzyć szablon w oparciu o wcześniej zdefiniowane reguły, jak chociażby w przypadku, gdy analizujemy nowe zbiory, z których chcemy się dowiedzieć, czy zawierają informacje niepasujące do ogółu. W takiej sytuacji należy użyć innych narzędzi wyszukiwania nieprawidłowości niepotrzebujących zdefiniowanych z góry reguł - powinny same, na podstawie podobieństwa obiektów, wyekstrahować wiadomości nie pasujące do reszty. W takim przypadku jest mowa o nienadzorowanym

wyszukiwaniu anomalii, ponieważ nie wiemy, które dane są normą, a które wyjątkowe. Istnieje też pół-nadzorowane wyszukiwanie anomalii - mamy z nim do czynienia, gdy model budujemy wyłącznie na podstawie danych sklasyfikowanych jako normalne. Podstawowe techniki wyszukiwania anomalii są oparte o algorytmy związane ze sztuczną inteligencją, a dokładniej z uczeniem maszynowym. Stosuje się następujące techniki wyszukiwana anomalii w danych w oparciu o uczenie maszynowe [27]:

- algorytmy oparte o fukncję gęstości np. K-Najbliższych Sąsiadów
- analiza skupień, grupowanie (*ang. clustering*) - uczenie nienadzorowane, które dzieli obiekty na klastry. W klastrach znajdują się obiekty do siebie podobne
- metoda asocjacyjna, a dokładniej odstępstwa od reguł asocjacyjnych - technika służąca do analizy zbioru danych pod kątem powtarzających się w nim zależności
- maszyna wektorów nośnych (*ang. support vector machine, SVM*) - koncept maszyny, która ma na celu wyznaczenie hiperpłaszczyzny rozdzielającej przykłady należące do dwóch klas; działa jak klasyfikator
- klasyfikatory oparte o drzewa decyzyjne - służą do definicji reguł w postaci drzewa, gdzie klasyfikacja zaczyna się od korzenia, a kończy po osiągnięciu klasy, które reprezentowane są poprzez liście drzewa. Na gałęziach znajdują się warunki, pod jakimi przechodzimy do kolejnych wierzchołków drzewa
- zestawy klasyfikatorów oparte o bagging lub boosting
- techniki oparte o sieci neuronowe
- uczenie głębokie (*ang. deep learning*) - mocno rozwijana dziedzina za sprawą wzrostu mocy obliczeniowej komputerów oraz rozwoju systemów przetwarzania danych typu big data. W tej technice model ma strukturę hierarchiczną, złożoną z wielu warstw. Im głębiej wchodzimy w utworzoną strukturę, tym bardziej szczegółowe informacje jesteśmy w stanie wydobyć z badanego obiektu.
- silniki regułowe - są zbiorem standardów, jakie powinny spełniać dane. Reguły znajdujące się w silniku regułowym mogą być reprezentowane przez różne techniki: kodu napisanego w dedykowanym języku, drzewa decyzyjnego czy tabel decyzyjnych. Informacje, które nie spełniają wyznaczonych zasad, są traktowane jako anomalie. Silniki decyzyjnie nie są tak elastyczne, jak algorytmy uczenia maszynowego, ale w wielu systemach znajdują swoje zastosowanie. Przykładem takiego systemu może być silnik Blaze Advisor firmy Fico, który możemy zastosować w architekturze SOA jako warstwę walidacyjną komunikatów przesyłanych do usług sieciowych.

## 3.3 Systemy zorientowane na wykrywanie anomalii

Systemy informatyczne zorientowane na wyszukiwanie anomalii nie tylko powinny umieć wykryć sytuację nienormalną, lecz także w odpowiedni sposób na nią zareagować. W przypadku anomalii negatywnych reakcja na wykrycie odstępstwa w wielu przypadkach musi być natychmiastowa - tutaj niezwykle istotny jest czas. W związku z powyższym, przetwarzanie wsadowe nie zawsze jest odpowiednim miejscem, w którym powinniśmy wyszukiwać anomalii, ponieważ przeważnie wyniki takich operacji nie są dostępne od razu z uwagi na analizę dużych zbiorów danych. Odpowiedzią na problem otrzymania wyniku analizy danych w możliwe najkrótszym czasie jest umiejscowienie wyszukiwania anomalii w warstwie przetwarzania rzeczywistego, gdzie analizowane są dane strumieniowe w momencie ich pojawienia się. Przykładowymi systemami, gdzie zasadne jest użycie wykrywania anomalii, są systemy medyczne, bankowe, wykrywające błędy w oprogramowaniu czy nadzorujące pracę sieci energetycznych. Przedstawmy działanie systemu wykrywania anomalii

na przykładzie systemu bankowego. Aplikacje bankowe śledzą akcje podejmowane przez klienta i na ich podstawie tworzony jest jego profil. Mechanizmy wykrywające odstępstwa odnajdują akcje użytkownika nie pasujące do jego profilu, a następnie mogą je blokować lub prosić użytkownika o dodatkową autoryzację. Program wykrywający anomalie - oprócz danych wejściowych, dla których ma zgłosić wyjątki - w wielu przypadkach korzysta też z danych zewnętrznych, takich jak pora roku, godzina, temperatura, pora dnia. Jest to niezbędne, gdyż sygnały uważane za anomalie np. w nocy, za dnia mogą być już traktowane jako dane poprawne[26]. Warto nadmienić, że obecnie widać trend rozwijania systemow opartych o duże sieci urządzeń. Projekty takie mają zastosowanie w przestrzeni miejskiej (koncepcja śmart city"), komunikacji czy naszym codziennym życiu. Coraz częściej słychać o nowych urządzeniach, które człowiek może nosić ze sobą, a które monitorują chociażby jego codzienną aktywność bądź wymieniają informacje z sensorami z idei wspomnianych wcześniej inteligentnych miast. Koncepcja urządzeń komunikujących się ze sobą, gromadzących i przetwarzających dane nazywa się internetem rzeczy (*Internet of things, IOT*). Wszystko wskazuje na to, że świat będzie podążał w stronę idei internetu rzeczy, dzięki której powstaną nowe dane do analizy także w kontekście wyszukiwania anomalii. Wskutek tego systemy wykrywające błędy w danych będą odgrywały coraz większą rolę, ponieważ odpowiednio zaprojektowane są źródłem wiedzy, dzięki której użytkownicy - pośredni czy bezpośredni - są w stanie szybciej reagować na wyjątkowe sytuacje np. prędzej rozładowywać ruch uliczny w miastach czy informować człowieka o wystąpieniu symptomów pogorszenia się jego stanu zdrowia.

# Rozdział 4

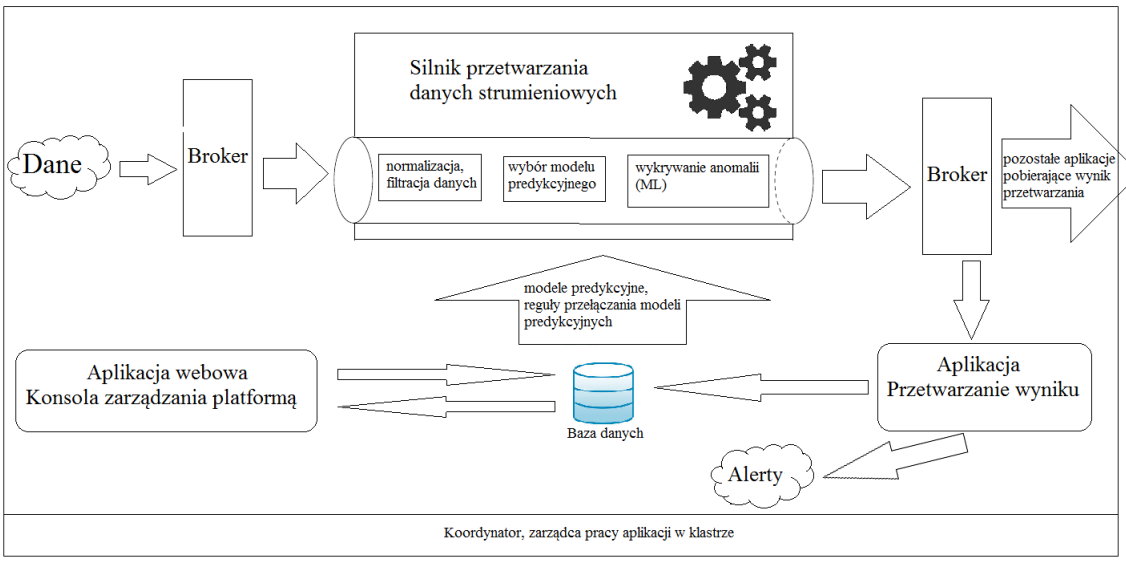
# Architektura systemu

## 4.1 Wprowadzenie

Poprzednie rozdziały dostarczyły informacji o anomaliach, strumieniach danych, narzędziach przetwarzania danych strumieniowych oraz uczenia maszynowego. W niniejszym rozdziale informacje te zostaną zastosowane w praktyce. Rozdział będzie zawierał przedstawienie architektury platformy służącej do wykrywania anomalii. Zostaną zaprezentowane elementy, z których powinien składać się ów system, by był w stanie przetwarzać z dużą szybkością dane strumieniowe. Następnie autor pracy opisze biblioteki i narzędzia mogące posłużyć jako implementacja zaproponowanej architektury.

## 4.2 Opis architektury

Architektura systemu wykrywającego anomalie w danych strumieniowych powinna być skalowalna, odporna na błędy oraz gwarantować przetworzenie wszystkich danych wysłanych do systemu bez zbędnych opóźnień. Może je powodować zapis wyników, nieoptymalne narzędzia przetwarzania danych czy sam sposób implementacji warstwy odpowiedzialnej za odbieranie danych ze źródła. Ważnym elementem jest także zapewnienie monitoringu całego systemu zarówno od strony administracyjnej, by mieć podgląd na stan i działanie platformy z możliwością jej konfiguracji, oraz od strony użytkownika, by ten mógł przeglądać przetworzone informacje oraz powstałe alerty w sposób czytelny np. na dedykowanych pulpitach.




Rysunek 4.1: Architektura systemu wykrywającego anomalie w danych strumieniowych.

Budując system wykrywający anomalie, spełniający powyższe wymagania, powinniśmy użyć takich rozwiązań, które pozwalają na łatwe skalowanie oraz rozszerzanie, by zwiększyć moc obliczeniową, kiedy to wydaje się niezbędne. Dobrym pomysłem jest także taki sposób zaprojektowania systemu, aby możliwa była wymiana jego elementów bez potrzeby ingerencji we wszystkie składowe. Oczywiście w praktyce nie zawsze można takie wymaganie osiągnąć, warto jednak dążyć do tego, by system był złożony z - w miarę autonomicznych - modułów. Wymiana bazy danych nie powinna wpływać na część systemu, która jest odpowiedzialna za dostarczanie do niej danych, ale już zmiana w jednostce obliczającej wynik będzie wymuszała dokonanie aktualizacji warstwy zapisującej dane w przypadku zmiany formatu komunikatu. Aktualizacje architektury lub kodu powinny być propagowane w kierunku od głównej części sytemu do modułów wspomagających, nie odwrotnie. Na rysunku 4.1 zaprezentowano projekt architektury systemu. Została ona podzielona na pięć części: pierwszą (główną), która odpowiada za przetwarzanie danych strumieniowych, drugą - odpowiedzialną za dostarczanie danych do platformy, trzecią zajmującą się odbieraniem wyników oraz ich dalszym propagowaniem, czwartą w której znajduje się baza danych przechowująca wyniki, ustawienia aplikacji oraz modele predykcyjne oraz piątą, gdzie znajduje się konsola służąca do zarządzania systemem. Strzałki na rysunku oznaczają kierunek przepływu danych pomiędzy komponentami.

Centralną częścią platformy jest system odpowiedzialny za przetwarzanie danych strumieniowych. To w nim powinny zostać zaimplementowane wszystkie algorytmy, które będą odpowiedzialne za przetwarzanie danych. Pozwoli nam ona zbudować graf przepływu danych, w którym będzie możliwość implementacji sposobów pozwalających na wyszukiwanie anomalii, ale także filtrowanie danych czy ich normalizacje. Powinna również pozwalać na skalowalność horyzontalną - czyli zwiększającą możliwości obliczeniowe poprzez dokładanie kolejnych węzłów roboczych. Dobrą opcję stanowi także możliwość rozszerzenia jednostek obliczeniowych, by były w stanie wykonywać obliczenia na procesorach graficznych. System przetwarzania danych strumieniowych powinien dostarczać również bibliotekę odpowiedzialną za uczenie maszynowe. Najlepiej, gdyby wspierała ona rozproszone algorytmy uczenia maszynowego. Powinna ona pozwalać nam na wymianę modelu predykcyjnego, gdy zajdzie taka potrzeba. Ma to duże znaczenie, ponieważ - jak opisano wcześniej - istnieją sytuacje, gdy dane uważane za anomalie w jednym kontekście, będą danymi normalnymi w innym. Może pojawić się potrzeba wymiany modelu na odmienny, który będzie bardziej odpowiadał zaistniałym warunkom, w jakich przyjdzie analizować dane. Taki model można przechowywać w pamięci operacyjnej - np. w postaci baz danych NoSQL, co redukuje do minimum narzuty czasowe na komunikację. Kolejną kwestią, istotną dla szybkiego i bezproblemowego działania platformy, jest sposób, w jaki inne systemy będą dostarczały dane. Zastosowano tutaj pośrednika wiadomości, do którego systemy źródłowe będą wysyłały komunikaty o określonym formacie, a z którego następnie będą pobierane przez rdzeń analityczny platformy. Dzięki temu system pobierze wiadomości z odpowiednią dla siebie prędkością, nie zważając na to, że systemy źródłowe będą wysyłały więcej komunikatów niż jest w stanie przetworzyć. Przybyłe informacje będą kolejkowane na brokerze i przetwarzane zgodnie z kolejnością przybycia do systemu. Wiadomości w kolejce powinny zalegać najkrócej jak to możliwe, ponieważ w systemie kluczowa jest szybkość przetworzenia komunikatu po jego otrzymaniu. Użycie warstwy pośredniczącej w komunikacji z systemami źródłowymi daje nam gwarancję, że wiadomość zostanie przetworzona dokładnie jeden raz (exactly-once). Najlepiej, gdyby broker wspierał również skalowalność, co znacznie wpłynie na jego wydajność, ponieważ w przypadku większego zapotrzebowania na moc obliczeniową zwiększymy ją poprzez dodanie kolejnych węzłów roboczych. Powinien także zapewniać mechanizmy serializujące wiadomości na pamięci stałej, by w razie awarii można je było odzyskać.

Kolejna warstwa platformy będzie odpowiedzialna za odebranie wyniku z systemu analitycznego, jego zapis do bazy danych oraz wysłanie alarmu, jeżeli odpowiedź będzie oznaczała anomalię. Tutaj również zastosowano pośrednika danych utworzo-

nego na systemie kolejkującym. Wynik z platformy analitycznej trafia do brokera, skąd może zostać pobrany przez inne systemy np. źródłowy, który chce odczytać wynik przetwarzania, lub serwis zapisujący dane do HDFS, skąd mogą zostać pobrane i przetwarzane przez zadania wsadowe. Elementem składowym platformy jest osobna usługa pobierająca i propagująca dane. Ma ona na celu pobranie danych, a następnie - w zależności od wyniku - zgłoszenie alertu w postaci wiadomości sms, maila czy powiadomienia wysłanego do konsoli obsługiwanej przez operatora systemu. Zastosowanie osobnego serwisu jest zasadne, ponieważ wprowadzamy dzięki temu asynchroniczność w wykonywaniu operacji zapisującej dane. Ponadto wprowadzanie zmian w obsłudze alertów nie pociąga za sobą modyfikacji warstwy analitycznej. Pośrednicy wiadomości, znajdujący się na wejściu i wyjściu systemu, gwarantują nam uniwersalny interfejs do komunikacji z platformą, niezależny od języka programowania. Dodatkowo taki sposób komunikacji jest asynchroniczny, przez co aplikacja źródłowa nie zostaje zablokowana na czas trwania analizy danych. Jej zadanie to wysłanie komunikatu do brokera. Jeżeli potrzebuje wynik przetwarzania, może go pobrać z kolejki wyjściowej - np. poprzez skorelowanie żądania i odpowiedzi przez nadanie im unikalnego numeru identyfikacyjnego.

W warstwie czwartej znajduje się baza danych nosql zorientowana na szybki zapis danych strumieniowych oraz na przetwarzanie analityczne. Służy ona do przechowywania wyników oraz modeli algorytmów odpowiedzialnych za uczenie maszynowe. Z tą bazą danych będą komunikowały się prawie wszystkie warstwy systemu, wskutek czego staje się ona głównym repozytorium danych w zaproponowanej architekturze. Wyniki będą składowane wraz z wartościami podanymi na wejściu, by na ich podstawie dokonywać poprawy modeli uczących lub by ekspert z poziomu panelu administracyjnego był w stanie skorygować wynik, poprawiając w ten sposób działanie systemu. Modele przechowywane w bazie - w zależności od potrzeby - mogą zostać pobrane i zaaplikowane do warstwy analitycznej. Repozytorium danych zawierałoby także informacje pozwalające na decyzje, w jakich przypadkach dokonywać zmiany modelu danych. Baza ta powinna również posiadać mechanizmy pozwalające na jej skalowalność, by móc przyspieszyć zapis danych, a także ich odczyt w przypadku próby pobrania danych, które będą wynikiem skomplikowanej operacji analitycznej.

Kolejną, ostatnią już, warstwą jest konsola zarządzania platformą - dedykowana aplikacja internetowa. Aplikacja ta korzystałaby z relacyjnej bazy danych, w której przechowywane byłyby informacje konfiguracyjne np. lista użytkowników, ustawienia klastra itd. Konsola ma dwa tryby: administracyjny oraz użytkownika. W trybie administracyjnym będzie możliwość podglądu stanu poszczególnych składowych platformy poprzez agregację interfejsów, które udostępniają poszczególne jej składowe. Dla przykładu wiele systemów przetwarzania danych strumieniowych udostępnia aplikacje internetowe czy API, dzięki którym mamy możliwość sprawdzania na żywo stanu i działania klastra oraz uruchomionej na nim aplikacji. Konsola zarządzania udostępniałaby interfejsy wraz z możliwością implementacji dodatkowych wskaźników. Głównym jej zadaniem byłaby konfiguracja platformy. Moglibyśmy dzięki niej uruchamiać zadania wsadowe, które służyłyby do aktualizacji modeli predykcyjnych czy definiować reguły, w jakich przypadkach należy zmienić model. W trybie użytkownika byłby dostęp do edycji wyników algorytmu w celu jego korekcji oraz podgląd wyników na interaktywnym pulpicie odświeżanym w czasie rzeczywistym za pomocą powiadomień oraz dane dotyczące zgłoszonych alertów. Systemy skalowalne, wchodzące w skład architektury, działałyby pod nadzorem aplikacji wspomagającej zarządzanie nimi. Aplikacja ta tworzy warstwę abstrakcji nad systemem operacyjnym i to na niej zostają uruchamiane skalowalne programy. Ideą takiego systemu jest instalacja wielu rozproszonych aplikacji na współdzielonej puli serwerów roboczych, którymi ona zarządza, przydzielając zasoby serwera poszczególnym systemom w zależności od potrzeby. Gdy następuje wzrost zapotrzebowania na moc obliczeniową, do klastra zostają dodane nowe węzły lub już przyłączonym są przydzielane większe zasoby w przypadku, gdy inne aplikacje ich nie potrzebują. Dzięki zastosowaniu takiego rozwiązania platforma lepiej zarządzałaby zasobami serwera, takimi jak pamięć operacyjna, procesor czy dostęp do zasobów dyskowych

w zależności od zapotrzebowania, co jest cechą systemów działających w chmurze. Tak zaprojektowany system z powodzeniem wpisuje się w ramy architektury Kappa służącej do przetwarzania dużych wolumenów danych.

## 4.3 Przykładowa implementacja architektury

Poprzedni podrozdział dostarczył informacji o projekcie architektury systemu wykrywającego anomalie w danych strumieniowych. W obecnym autor naniesie na architekturę przykładowe komponenty, z których może ona zostać zbudowana. Nie będą tutaj przedstawione szczegóły implementacyjne, a wyłącznie przykładowe platformy, biblioteki, które z powodzeniem mogą zostać użyte przy implementacji.

Główna część systemu - odpowiedzialna za przetwarzanie danych strumieniowych - może zostać oparta o platformę Apache Storm. Wybór padł na nią, ponieważ jest to projekt stabilny, używany w wielu projektach oraz posiadający dobrą dokumentację i wsparcie społeczności. Jest także wydajnym, skalowalnym narzędziem, któremu warto przyjrzeć się bliżej, co uczyniono, tworząc niniejszą pracę. Warto spojrzeć na bibliotekę Apache SAMOA, której dokładniejszy opis, przykład uruchomienia oraz zasadność użycia zostanie opisany w następnym rozdziale. Kolejną ważną częścią systemu jest broker. Powinien on być skalowalny, zapewniać szybkie przetwarzanie oraz zawierać mechanizmy pozwalające na odbiór przybyłych komunikatów przez wiele systemów. Przy próbie implementacji architektury jako brokera warto rozważyć Apache Kafka. Jest to stosunkowo nowa biblioteka stworzona z myślą o systemach przetwarzania danych strumieniowych w czasie rzeczywistym. Jej ważną cechą jest skalowalność poprawiająca wydajność. Kolejna interesująca możliwość to obsługa wielu konsumentów, dzięki czemu nie ma problemu z podłączeniem kolejnych aplikacji zapisujących czy odczytujących dane z platformy. Następną rzeczą wartą zastanowienia jest baza danych, która powinna pozwalać na szybki zapis danych oraz ich analizę pod kątem analitycznym. Dobrym wyborem będzie baza Druid. Jej architektura również pozwala na skalowalność oraz jest przystosowana do szybkiego zapisu danych strumieniowych. Posiada także mechanizmy wspomagające przetwarzanie analityczne. Do przechowywania aktualnego modelu predykcyjnego w przypadku zastosowania techniki holdout dobrym rozwiązaniem będzie użycie pamięci podręcznej np. w postaci Memcached - rozproszonego systemu pamięci podręcznej potrafiącego współpracować z biblioteką Trident. W celu zapisu wyników analizy czy zgłaszania alarmów architektura przewiduje utworzenie osobnej aplikacji. Taka aplikacja może zostać zaimplementowana jako serwis systemu operacyjnego, który powinien mieć budowę pozwalającą na dołączanie nowych docelowych miejsc zapisu danych czy zgłaszania informacji o anomaliach. Ostatnim elementem systemu jest aplikacja, która ma za zadanie bycie panelem konfiguracyjnym oraz użytkowym platformy. Najlepiej, by była to aplikacja internetowa, do której użytkownicy będą mieć dostęp bez konieczności instalowania dodatkowego oprogramowania. Aby przyspieszyć implementację takiego programu, można posłużyć się bibliotekami oferującymi gotowe szkielety aplikacji, ktore mają już zaimplementowane zarządzanie użytkownikami, panel administracyjny czy system logowania. Dzięki temu podczas implementacji platformy możemy skupić się wyłącznie na aspekcie, który jej bezpośrednio dotyczy, bez konieczności budowania konsoli platformy od podstaw. Przykładem takiej biblioteki - pozwalającej na szybką budowę aplikacji - jest AppFuse [28], czyli biblioteka do budowania aplikacji internetowej za pomocą narzędzi języka Java. Po ściągnięciu projektu mamy dostępną aplikację, którą możemy rozszerzać o nowe funkcje i ekrany związane z naszym zastosowaniem.

Aplikacje rozproszone do swojego działania potrzebują koordynatora, dzięki któremu węzły główne będą mogły komunikować się z węzłami roboczymi. Powszechnie używanym koordynatorem wspomagającym pracę wewnątrz aplikacji rozproszonych jest Apache ZooKeeper[29]. Powstał on przy tworzeniu platrofmy Hadoop, ale obecnie jest osobnym projektem używanym przez wiele systemów rozproszonych, m.in. Apache Kafka czy Apache Storm. Wspomniano także o zastosowaniu aplikacji,

która pomagałaby w zarządzaniu klastrem aplikacji rozproszonych. Wyborem wartym rozważenia jest Apache Mesos [30]. To otwarta źródłowa aplikacja do zarządzania klastrem. Za jej wyborem przemawia fakt, że posiada wiele bibliotek, które ją rozszerzają, oraz jest używana przez firmy komercyjne - na przykład Microsoft w chmurze Azure.

# Rozdział 5

# Opis przeprowadzonych testów

## 5.1 Wprowadzenie

W ramach pracy dokonano analizy biblioteki Apache Samoa, którą autor wybrał jako docelową bibliotekę wykrywającą anomalie w zaprojektowanym systemie. Zbadano jej możliwości oraz sprawdzono, czy architektura oraz budowa pozwolą na praktyczne zastosowanie. W niniejszym rozdziale zostanie opisana konfiguracja części zaproponowanej architektury. Autor pokaże, jak skonfigurować i uruchomić Apache Kafka jako źródło danych, Apache Storm jako platformę przetwarzania danych oraz Apache SAMOA jako bibliotekę uruchomioną na Apache Storm, której zadaniem będzie wyszukiwanie anomalii. Opisane zostanie także, w jaki sposób tworzyć aplikację na platformę Storm wraz z przykładami kodów. Szczególna uwaga zostanie poświęcona Apache SAMOA - zostaną zaprezentowane braki, na jakie natrafił autor podczas pracy z biblioteką oraz omówi się, jakie modyfikacje należałoby wprowadzić, by bardziej przystosować ją do wymagań systemu i architektury. Ostatecznie autor sprawdzi, czy biblioteka jest gotowa i przystosowana do użycia w systemie wykrywającym anomalie. Środowisko zostało przygotowane w systemie operacyjnym Linux.

Do przeprowadzenia testów użyto zestawu danych *Mammographic Mass Data Set* [31]. Zestaw ten zawiera wyniki badań mammograficznych z komputera diagnostycznego. Wyniki pomagały w podjęciu decyzji, czy należy dokonywać biopsji piersi. Działo się tak, ponieważ zauważono, że 70% zabiegów było przeprowadzanych niepotrzebnie. Rezultaty zostały zebrane w Instytucie Radiologii na Uniwersytecie Erlangen-Nuremberg. Na podstawie skali BI-RADS [32], wieku pacjenta, kształtu, gęstości oraz otoczenia guza możemy przewidzieć, czy wykryta zmiana patologiczna jest łagodna, czy złośliwa i kwalifikuje się do biopsji. Atrybutem decyzyjnym jest atrybut *Severity*. Przyjmuje on dwie wartości: 0 - zmiana łagodna, 1 - zmiana złośliwa. Zbiór danych zawiera 516 klas oznaczających zmiany łagodne oraz 445 klas dla zmian złośliwych.

## 5.2 Apache Kafka

Podczas przeprowadzania testów użyto Apache Kafka w wersji 0.9.0.0. Autor wykorzystał domyślną konfigurację brokera. Skrypty startujące znajdują się w katalogu *bin*, a domyślne pliki konfiguracyjne - w katalogu *config*. W celu uruchomienia Apache Kafka należy najpierw włączyć Apache Zookeeper, który będzie koordynował działanie naszego brokera. Uruchamiamy go za pomocą polecenia:

```
$ /bin/zookeeper-server-start.sh /config/zookeeper.properties
```

Jako parametr należy podać plik konfiguracyjny. W pliku znajdują się informacje o porcie, na jakim uruchomią się koordynator oraz katalog, a także gdzie będą zapisywane pliki tymczasowe.

Kolejnym krokiem jest uruchomienie brokera. Czynimy to za pomocą polecenia:

```
$ /bin/kafka-server-start.sh /config/server.properties
```

Tutaj również jako parametr należy podać plik konfiguracyjny. W pliku znajdują się informacje o porcie, na którym zostanie uruchomiony serwer, liczbie wątków obsługujących wiadomości przychodzące do serwera, rozmiarach buforów, ilości partycji znajdujących się w tematach oraz lokalizacja katalogu, gdzie broker będzie serializował dane.
W przypadku, gdy wyślemy wiadomości na nieistniejący temat (*topic*), Kafka utworzy go z domyślną liczbą partycji ustawioną w pliku konfiguracyjnym. By tego uniknąć, należy w pliku *server.properties* ustawić parametr *auto.create.topics.enable* z wartością *false*. Można także stworzyć temat ręcznie za pomocą polecenia:

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181
    --replication-factor 1 --partitions 1 --topic test
```

Podczas implementacji przydatne jest również polecenie:

```
$ bin/kafka-console-consumer.sh --zookeeper localhost:2181
    --topic topic --from-beginning
```

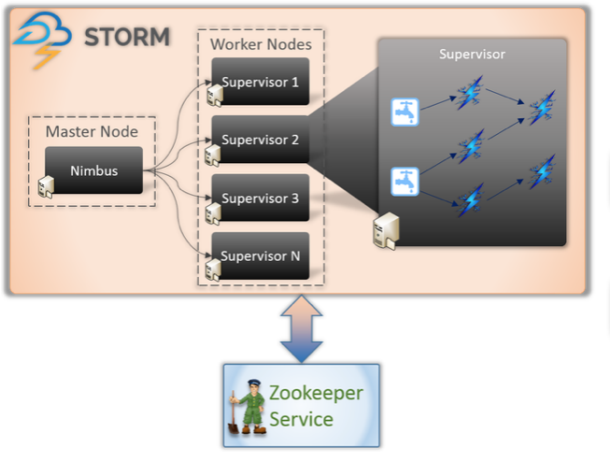
pozwalające na podgląd wiadomości oraz polecenie:

```
$ bin/kafka-console-producer.sh --broker-list
    localhost:9092 --topic topic
```

za pomocą którego możemy - z poziomu konsoli - wysyłać wiadomości do wskazanego tematu. Broker zapisuje informacje znajdujące się w poszczególnych tematach i partycjach w katalogu wskazanym pod parametrem *log.dirs* w pliku *server.properties*. W celu wyczyszczenia wiadomości w interesującym nas temacie należy usunąć pliki z katalogu, którego nazwa jest określona jako nazwa tematu.

## 5.3 Apache Storm

Jako silnik przetwarzania danych strumieniowych wykorzystano Apache Storm. Autor chce tutaj zwrócić uwagę na budowę klastra, uruchomienie platformy oraz podać przykład budowy topologii w języku Java wraz ze sposobem jej uruchomienia na klastrze.




Rysunek 5.1: Schemat klastra Apache Storm

Źródło: https://www.linkedin.com/pulse/apache-storm-architecture-overview-chandan-prakash

Rysunek 5.1 pokazuje architekturę klastra Apache Storm. Możemy wyróżnić węzeł główny (*nimbus*) oraz węzły robocze (*workers*), na które składają się serwery nadzorujące (*supervisor*). Głównym zadaniem serwera Nimbus jest uruchomienie topologii i to on dystrybuuje nowe zadanie do dostępnych serwerów nadzorujących. Serwery nadzorujące mogą mieć uruchomionych jeden lub więcej procesów roboczych. Nadzorca przekazuje zadanie, które dostanie od węzła głównego, do konkretnego procesu roboczego (*worker process*). Proces roboczy jest odpowiedzialny za wykonywanie utworzonej topologii. Wątek wykonujący pojedyńczą topologię w procesie roboczym to wykonawca (*executor*). Jak zostało wspomniane w rozdziale drugim, w topologii wyróżniamy następujące węzły: wylewki (*ang. spouts*) oraz gromy (*ang. bolts*). W architekturze Storm wykonywanie pojedyńczego węzła jest nazywane zadaniem (*task*). Nad komunikacją między węzłem głównym a węzłami roboczymi czuwa koordynator Apache Zookeeper - dzięki niemu Nimbus wie, jaki jest obecnie stan wykonywania zadań na poszczególnych serwerach Supervisor oraz czy jakiś serwer nie uległ awarii.

Platforma Storm zawiera katalog *bin* ze skryptem *storm*, dzięki któremu za pomocą predefiniowanych komend wykonujemy operacje na klastrze. Wszystkie dostępne komendy są opisane w dokumentacji [33]. Tutaj zostaną przytoczone jedynie te niezbędne do uruchomienia topologii, instalacji utworzonych kodów oraz uruchomienia konsoli, dzięki której jesteśmy w stanie nadzorować pracę topologii. Konfiguracja klastra jest zapisana w katalogu conf w pliku *storm.yaml*. W celach testowych wykorzystano domyślną konfigurację klastra.

By uruchomić topologię, w pierwszej kolejności musimy włączyć serwer Zookeeper. Możemy uruchomić go bezpośrednio z katalogu z pobranym serwerem za pomocą polecenia:

```
$ ./bin/zkServer.sh start
```

Następnie należy uruchomić węzeł główny Nimbus. Dokonujemy tego za pomocą polecenia:

```
$ ./bin/storm nimbus
```

Kolejnym krokiem jest włączenie węzłów roboczych. W pliku konfiguracyjnym podajemy adres serwera Zookeeper za pomocą atrybutu *storm.zookeeper.servers*, który koordynuje współpracę węzłów głównego i roboczych. W pliku konfiguracyjnym jest także zapisana informacja, ile węzłów roboczych może zostać uruchomionych na danej maszynie - służy do tego atrybut *supervisor.slots.ports*, w którym podajemy porty, na jakich mają zostać włączone węzły. Węzły robocze uruchamia się poleceniem:

```
$ ./bin/storm supervisor
```

W środowisku produkcyjnym węzły robocze powinny być umiejscowione na osobnych maszynach, lecz podczas testu cały klaster został uruchomiony na pojedyńczym komputerze.

Platforma Storm udostępnia także narzędzia pozwalające na monitorowanie uruchomionych topologii. Przy pomocy polecenia:

```
$ ./bin/storm list
```

na konsoli ukażą się uruchomione topologie wraz z ich statusami. Kolejnym przydatnym narzędziem jest dostępna przez przeglądarkę internetową konsola, dzięki której możemy podejrzeć ustawienia klastra, uruchomione topologie, przeglądać logi z ich pracy czy w nią ingerować np. poprzez ich wyłączenie. Istnieje także narzędzie pozwalające oglądać utworzoną topologię w postaci grafu. Konsola ta jest osobnym demonem uruchamianym poleceniem:

```
$ ./bin/storm logviewer
```

Należy włączyć również demona, dzięki któremu interfejs graficzny jest w stanie zczytywać logi z pracy topologii. Uruchamiamy go za pomocą polecenia:

```
$ ./bin/storm logviewer
```

W domyślnej konfiguracji konsola będzie dostępna lokalnie pod portem 8080. Po uruchomieniu konsoli graficznej mamy także dostęp do udostępnianego przez nią REST API, za pomocą którego możemy pobierać informacje o stanie klastra czy uruchomionych na nim zadaniach. Jest to przydatne, jeżeli chcielibyśmy napisać własną aplikację mającą na celu monitorowanie platformy Storm.

Utworzona topologia w języku Java musi zostać skompilowana do tzw. grubego pliku jar (*fat jar*). Plik ten zawiera w sobie wszystkie inne biblioteki (zależności), z jakich skorzystaliśmy podczas implementacji topologii. Na poniższym listingu widzimy przykład budowy topologii w języku Java.

Listing 5.1: Przykład budowy topologii Apache Storm

```
\\https://github.com/apache/storm/blob/master/examples/storm
    -starter/src/jvm/org/apache/storm/starter/
    ExclamationTopology.java

TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("word", new TestWordSpout(), 10);
builder.setBolt("exclaim1", new ExclamationBolt(), 3).
    shuffleGrouping("word");
builder.setBolt("exclaim2", new ExclamationBolt(), 2).
    shuffleGrouping("exclaim1");

Config conf = new Config();
conf.setDebug(true);
conf.setNumWorkers(3);

StormSubmitter.submitTopologyWithProgressBar("Topology name"
    , conf, builder.createTopology());
```

Pokazany przykład prezentuje budowę topologii z wykorzystaniem podstawowego API platformy. Widzimy tutaj obiekt *TopologyBuilder*, który odpowiedzialny jest za budowę naszej topologii. Posiada on metody, dzięki którym jesteśmy w stanie dodawać do niej wylewki i gromy. Wylewki i gromy implementujemy jako osobne klasy, które muszą dziedziczyć udostępnione przez API platformy klasy bazowe. Widzimy tutaj także obiekt *Config* służący do konfiguracji tworzonej topologii. Możemy sterować poziomem logowania klastra dla zadanej topologii czy podpowiadać, przez ile procesów roboczych powinno zostać uruchomiona topologia. W metodach *setBolt* oraz *setSpout* jako ostatni argument podajemy, ile wątków ma obsługiwać dany komponent. Cała zbudowana topologia wraz z konfiguracją oraz nazwą zostaje poprzez obiekt *StormSubmitter* umieszczona w klastrze.

W rozdziale drugim wspomniano, że Storm posiada także bibliotekę Trident wspomagającą tworzenie topologii oraz wprowadzającą wiele udogodnień dla programisty. W powyższej bibliotece budowanie topologii przebiega analogicznie. Mamy do dyspozycji obiekt *TridentTopology*, do którego dodajemy poszczególne elementy naszej aplikacji. Warto wspomnieć, że Trident wprowadza do topologii pojęcie stanowości(*trident state*), dzięki któremu możemy w o wiele łatwiejszy sposób czytać i zapisywać dane do statycznych źródeł danych np. bazy danych czy plików HDFS.

Dzięki abstrakcyjnemu obiektowi *TridentState* łączymy się z każdym źródłem danych w ten sam sposób oraz jesteśmy w stanie dodać obsługę stanowości do naszej topologii. Przykład topologii zbudowanej za pomocą biblioteki Trident, która wykorzystuje zapis stanu do bufora *Memcached*, prezentuje poniższy listing.

Listing 5.2: Przykład budowy topologii Apache Storm z wykorzystaniem biblioteki Trident

```
\\http://storm.apache.org/releases/1.0.2/Trident-state.html

TridentTopology topology = new TridentTopology();
TridentState wordCounts =
      topology.newStream("spout1", spout)
        .each(new Fields("sentence"), new Split(), new
           Fields("word"))
        .groupBy(new Fields("word"))
        .persistentAggregate(MemcachedState.opaque(
           serverLocations), new Count(), new Fields("count"
           ))
        .parallelismHint(6);
```

Dokładny opis i szczegóły implementacji topologii w podstawowym API oraz za pomocą biblioteki Trident zostały opisane w książce [11] lub na stronie domowej projektu [12]. Skompilowaną do pliku *.jar topologię uruchamiamy na klastrze za pomocą polecenia:

```
$ storm jar topology.jar package.ClassName
```

Polecenie *storm jar* jako parametry w kolejności przyjmuje: plik *fatjar* z kodem topologii oraz nazwę klasy, w której znajduje się kod odpowiedzialny za zbudowanie i uruchomienie topologii. Polecenie to instaluje plik jar na klastrze Storm za pomocą obiektu *StormSubmitter*.

## 5.4 Przygotowanie danych testowych

Zbiór danych *Mammographic Mass Data Set* zawiera braki w danych. Autor nie jest ekspertem w dziedzinie, z której pochodzą dane, dlatego postanowiono usunąć ze zbioru te właśnie przykłady. Oczyszczenia zbioru danych dokonano za pomocą języka R. Puste wartości są reprezentowane za pomocą znaku '?'. Plik z danymi został wczytany do środowiska R, a następnie cały zbiór danych został poddany filtracji, w wyniku której usunięto z niego wiersze zawierające znak '?' w dowolnej kolumnie. Ze zbioru wykluczono 162 klasy. Wynik filtracji został zapisany do pliku wykorzystywanego jako źródło strumienia danych.

Listing 5.3: Usunięcie przykładów posiadających braki w danych

```
library(dplyr)
mydata <- read.table("mammographic_masses_without_missing_
    val.data", header=FALSE, sep = ",")
mydataWithoutMissingVal <- filter(mydataToClean, V1 != '?' &
    V2 != '?' & V3 != '?' & V4 != '?' & V5 != '?')
write.csv(mydataWithoutMissingVal, file = "mammographic_
    masses_without_missing_val.data")
```

Sposób, w jaki dane zostały oczyszczone, pokazuje Listing 5.3. Uzyskany zbiór danych zostanie powielony, by zasymulować strumień danych. W tym celu autor przygotował aplikację, która zaczytuje plik z danymi, a następnie wysyła jego zawartość do Apache Kafka.

Aplikację zaczytującą plik napisano w języku Java. Utworzono ją do symulacji strumienia danych, w którym chcemy dokonać wykrycia anomalii. W ramach programu został napisany kod służący do wysyłania wiadomości do Apache Kafka. Aplikacja korzysta z oficjalnej biblioteki Apache Kafka udostępninej dla języka Java w wersji 0.9.0.1. Adres brokera oraz nazwa tematu, na który mają zostać wysłane wiadomości, są ustawione w pliku konfiguracyjnym programu. Aplikacja przyjmuje dwa obowiązkowe parametry:

- **-fp** - lokalizacja pliku z danymi, które będą wysyłane na kolejkę
- **-r** - parametr określający, ile razy dany plik będzie wysyłany na kolejkę

Poniżej zaprezentowano przykład uruchomienia programu:

```
$ java -jar locust-data-loader.jar
    -fp mammographic_masses_input.data -r 400
```

Aplikacja nie przekształca oraz nie przemapowuje danych z pliku wejściowego - są one wysyłane linia po linii w postaci ciągu znaków, ponieważ Apache SAMOA potrafi interpretować znaki oddzielone ogranicznikiem (np. przecinkiem) jako osobne atrybuty. W zastosowaniu produkcyjnym najbardziej pożądane jest, by na brokera trafiały dane w postaci ustrukturyzowanej np. w formacie JSON, który łatwiej można przekształcić na obiekty. Ponadto system konsumujący takie dane jest prostszy w implementacji i wygodniejszy w obsłudze przez programistów.

## 5.5 Apache SAMOA

### 5.5.1 Opis biblioteki i proponowane modyfikacje

Bibliotekę Apache SAMOA opisano w rozdziale drugim. Tutaj zostaną przybliżone jej aspekty praktyczne, czyli implementacja, dostępne źródła danych oraz przykład uruchomienia. Autor zwróci uwagę na braki w bibliotece w kontekście zastosowania w zaprojektowanym systemie, a także przedstawi, w jaki sposób wspomniane braki uzupełnić. Na samym końcu zaprezentuje, jak uruchomić przetwarzanie danych za pomocą Apache SAMOA, która będzie osadzona na silniku Storm oraz pobierała dane z brokera Kafka.

W pierwszej kolejności warto wspomnieć o udostępnionym API służącym do implementacji nowych zadań oraz algorytmów. Tak jak w przypadku Storm, w SAMOA budujemy topologię, która jest grafem przepływu danych. Wyróżniamy sześć głównych elementów składających się na topologię. Pierwszym z nich jest Procesor (*ang. processor*) - czyli główna jednostka obliczeniowa, w której zawarta jest logika rozwiązująca zadany problem. Dane przetwarzane przez procesor nazwano (*content event*). Połączenia między różnymi procesorami to strumienie (*ant. streams*). Strumień jest osobnym obiektem posiadającym jedno źródło oraz wiele celów. Obiektem, w którym tworzona jest topologia, nazywamy zadanie (*task*). Biblioteka potrafi uruchamiać tylko zadania, dlatego jest to główna klasa służąca do uruchomienia topologii. Przykładem zadania zaimplementowanego na platformie jest *PrequentialEvaluation*. Kolejnym ważnym obiektem jest *TopologyBuilder* odpowiadający za zbudowanie topologii. To on jest odpowiedzialny za powiązanie elementów w graf reprezentujący przepływ danych. Istnieje także obiekt *Learner*, który implementują klasy odpowiedzialne za uczenie maszynowe. Jest on implementowany jako topologia podrzędna głównej topologii utworzonej w zadaniu. Każdy wymieniony wyżej obiekt posiada reprezentujący go interfejs. By utworzyć dany element, należy zapewnić mu implementację odpowiedniego interfejsu.

Na listingu 5.4 widzimy przykład budowy zadania na platformę SAMOA. Pokazuje on implementację metody *init*, która jest odziedziczona z interfejsu *task* i odpowiada za stworzenie topologii. W przykładowej topologii został utworzony jeden procesor źródłowy oraz jeden docelowy, a następnie zostają one powiązane za

pomocą strumienia. Widać także, że wszystkie elementy są dodawane do topologii za pomocą obiektu *builder*, który jest reprezentacją klasy *TopologyBuilder*. Więcej informacji o budowie własnych topologii można znaleźć w dokumentacji projektu [34] w rozdziale przeznaczonym dla deweloperów.

Listing 5.4: Przykład budowy topologii Apache SAMOA

```
        // https://samoa.incubator.apache.org/documentation
           /Developing-New-Tasks-in-SAMOA.htmll
jest dost pny w dokumentacji projektu \cite{website:
   samoaDoc} w rozdziale przeznaczonym dla deweloper w.
    @Override
    public void init() {
        // create source EntranceProcesor
        sourceProcessor = new HelloWorldSourceProcessor(
           instanceLimitOption.getValue());
        builder.addEntranceProcessor(sourceProcessor);

        // create Stream
        Stream stream = builder.createStream(sourceProcessor
           );

        // create destination Processor
        destProcessor = new HelloWorldDestinationProcessor()
           ;
        builder.addProcessor(destProcessor,
           helloWorldParallelismOption.getValue());
        builder.connectInputShuffleStream(stream,
           destProcessor);

        // build the topology
        helloWorldTopology = builder.build();
        logger.debug("Successfully built the topology");
    }
```

SAMOA oferuje bardzo mało predefiniowanych źródeł danych, a ponadto są to źródła danych statycznych, które platforma zamienia w strumień danych. Jako źródło danych możemy podać plik tekstowy lub pliki pochodzące z pamięci HDFS. Zostały także zaimplementowane generatory danych. Mogą one posłużyć jako dane testowe przy implementacji nowych zadań na platformę. Jeden z generatorów stanowi *RandomTreeGenerator* odpowiadający za generowanie losowych drzew decyzyjnych. Jak zauważamy, platforma nie ma zaimplementowanego mechanizmu, który traktowałby Apache Kafka jako źródło danych. Autor samodzielnie musiał zadbać o podłączenie kolejki do platformy. Rozpoczęto implementację klasy odpowiedzialnej za pobieranie danych z brokera, jednak okazało się, że jeden z użytkowników SAMOA również zauważył problem w braku takiego źródła danych i je zaimplementował. W oficjalnym repozytorium Apache SAMOA utworzono *pull request* [35] z powyższym konektorem, ale do dnia dzisiejszego nie został oficjalnie dołączony do biblioteki. Autor pracy ostatecznie zdecydował się na skorzystanie z gotowego rozwiązania. Podczas testów zauważono, że na platformie pojawia się błąd, kiedy na kolejkę przez krótki czas po przetworzniu ostatniego komunikatu nie przybędzie żaden nowy. Autor nie jest pewien, czy jest to błąd po stronie konektora, czy błąd brokera. Wymaga on dalszej analizy i naprawy, gdyby zaszła potrzeba użycia go w środowisku produkcyjnym. Jednak w celu przeprowadzenia testów, wada ta nie była krytyczna.

Przy opise architektury podano, iż wyniki wyszukiwania anomalii zostaną wysłane na brokera, z którego będą pobierać je inne aplikacje. Niestety SAMOA nie

spełnia tego wymagania. Biblioteka nie udostępnia żadnych mechanizmów służących do zapisu wyników przetwarzania do jakiegokolwiek innego systemu, czy to brokera, czy bazy danych. Jedyne wyniki, jakie platforma obecnie oferuje, stanowią statystyki z przetwarzania danych zapisywane do pliku testowego. Wyniki te zostają obliczone i zapisane do pliku co $n$ przetworzonych przykładów, gdzie parametr $n$ jest podawany przy uruchamianiu zadania. Brak otrzymywania wyniku przetwarzania jest dużą wadą platformy. Zadaniem rozwijającym jest z pewnością implementacja procesora, którego zadaniem byłoby udostępnianie wyników przetwarzania. Powinnien zostać stworzony dedykowany pakiet z implementacją warstwy abstrakcyjnej, za pomocą której wyniki byłyby udostępnianie konkretnym systemom. Za implementację modułu wysyłającego wyniki do brokera Kafka, po przeprowadzeniu modyfikacji, mógłby posłużyć kod użyty do wysyłania danych na kolejkę w narzędziu symulującym strumień danych z pliku. Autor dokonał analizy kodu platformy pod kątem zastosowania w systemie wykrywającym anomalie. Podczas niej udało się znaleźć miejsce, gdzie należałoby dokonać modyfikacji w celu pobrania wyniku ostatniego przetwarzania. Modyfikacja dotyczy jedynie przetwarzania *PrequentialEvaluation*, które jako algorytmu klasyfikacji używa *Vertical Hoeffding Tree*. Wyniki z przetwarzania trafiają do procesora o nazwie *EvaluatorProcessor*. Jest to procesor docelowy, do którego trafiają dane po przetworzeniu. To w nim liczone są statystyki zapisywane następnie do pliku. W metodzie *Process* dodano logowanie zmiennej reprezentującej klasę, do której została zakwalifikowana przetwarzana instancja. Fragment kodu, który loguje wynik klasyfikacji:

```
logger.info("predictedClass: "
        + Utils.maxIndex(result.getClassVotes()));
```

Metoda *getClassVotes* zwraca tablicę jednowymiarową, a jej rozmiar odpowiada ilości klas, które chcemy przewidzieć. Pole tablicy o największej w danym czasie wadze oznacza, że algorytm typuje tę klasę jako wynik przetwarzania. Autor doszedł do wniosku, że modyfikacja istniejącego procesora nie jest wskazana. Lepszym rozwiązaniem jest przekształcenie topologii utworzonej w zadaniu *PrequentialEvaluation*, do której zostałby dodany dodatkowy procesor docelowy odpowiedzialny za zapis wyników. Kwestią do sprawdzenia pozostaje jeszcze możliwość implementacji aplikacji na klaster Apache Storm, w którym jednym z elementów topologii byłaby topologia Apache SAMOA. Dzięki temu mielibyśmy dostęp do możliwości przetwarzania i zapisu wyniku, jakie udostępnia platforma Storm.

Autor chciał zastosować algorytm *Vertical Hoeffding Tree* jako narzędzie wykrywające anomalie. Dokonano testów oraz przeglądu implementacji algorytmu. Wyciągnięto następujące wnioski: sam algorytm wraz z połączeniem z techniką *PrequentialEvaluation* nie będzie dobrym narzędziem do wykrywania anomalii, ponieważ nie będziemy mieć tutaj statycznego modelu wskazującego, które klasy traktować jako anomalie, a które nie. Potencjalnie lepszą techniką przetwarzania danych strumienowych - wykorzystujących drzewa decyzyjne - byłaby *holdout*. Podczas analizy kodu drzew decyzyjnych zaimplementowanych na platformie SAMOA autor zauważył, że jest na niej zaimplementowany algorytm ADWIN odpowiedzialny za wykrywanie *concept drift*. Dobrym pomysłem - w kontekście utworzenia mechanizmu wykrywającego anomalie za pomocą SAMOA - byłaby próba wyekstrahowania algorytmu ADWIN do osobnego procesora, którego zadaniem byłoby wykrywanie zmian w charakterystyce danych. Taki procesor z powodzeniem mógłby zostać użyty jako detektor anomalii w przypadku, gdy zmiana charakterystyki danych za każdym razem powinna być traktowana jako anomalia. W platformie przydałaby się również implementacja zadania reprezentującego przetwarzanie *holdout* czy bardziej elastycznych struktur danych przechowujących model. Następnym krokiem byłaby implementacja mechanizmów pozwalających na podmianę modelu w locie.

Podsumowując, platforma Apache SAMOA na obecnym etapie nie nadaje się do zastosowania w projekcie zorientowanym na wykrywanie anomalii. Posiada ona liczne braki, lecz poprzez wprowadzenie zaproponowanych modyfikacji jesteśmy w

stanie tak przygotować platformę, by stała się przydatna do projektowanego systemu.

### 5.5.2 Uruchomienie przetwarzania

Zadania SAMOA uruchamia się z poziomu konsoli. Na samym początku musimy zbudować projekt za pomocą narzędzia *maven*. Aplikacja ma zdefiniowane profile uruchamiające kompilację kodów pod platformę, na którą chcemy dokonać instalacji topologii. Mamy do dyspozycji Apache Storm, Apache Flink, Apache Samza i S4. Brak podania jakiegokolwiek profilu skompiluje aplikację przystosowaną do uruchomienia w środowisku lokalnym bez wykorzystywania silnika przetwarzania danych. By skompilować Apache SAMOA dostosowaną do współpracy z Apache Storm, należy wydać polecenie:

```
$ mvn -Pstorm package
```

Przed uruchomieniem przetwarzania należy dodać do zmiennych środowiskowych systemu lokalizację Apache Storm, by SAMOA mogła pobrać konfigurację za pomocą polecenia:

```
$ export STORM_HOME=/path-to-apache-storm
```

Przykładowe uruchomienie przetwarzania jako topologii Storm wykonujemy poleceniem:

```
$ bin/samoa storm target/SAMOA-Storm-0.0.4-SNAPSHOT.jar
  "PrequentialEvaluation -d /tmp/dump.csv
    -i 1000000 -f 100000
    -l (org.apache.samoa.learners.classifiers.trees.
        VerticalHoeffdingTree -p 4)
     -s (org.apache.samoa.moa.streams.generators.
         RandomTreeGenerator -c 2 -o 10 -u 10)"
```

gdzie jako parametr podajemy, na jakiej platformie uruchamiamy zadanie. Następnie podajemy lokalizację pliku jar dedykowanego dla danej platformy. Kolejnym argumentem jest definicja zadania, które chcemy uruchomić. W przykładzie uruchamiamy zadanie *PrequentialEvaluation*, wyniki (statystyki) przetwarzania będziemy zapisywać do pliku */tmp/dump.csv*. Ponadto przetworzymy *1000000* instancje, a statystyki będziemy obliczali i zapisywali do pliku co 100000 instancji. Jako klasyfikatora użyjemy implementacji zawartej w klasie *VerticalHoeffdingTree*, któremu określimy stopień zrównoleglenia. Jako źródło danych w przykładzie posłuży nam generator *RandomTreeGenerator*. W katalogu *bin* znajduje się plik konfiguracyjny *samoa-storm.properties*, w którym możemy ustawić ilość procesów roboczych Storm obsługujących włączoną topologię, tryb pracy - uruchomienie topologii na klastrze Storm, czyli przesłanie jej do serwera Nimbus, lub włączenie w trybie lokalnym bez potrzeby uruchamiania klastra. Wykonywanie uruchomionego zadania możemy obserwować na platformie Storm za pomocą konsoli Storm UI, a także obserwować plik wynikowy */tmp/dump.csv*, do którego będą trafiały wpisy obrazujące statystyki przetwarzania.

Dokładny opis konfiguracji i przykłady uruchomienia są dostępne w dokumentacji projektu [34] w rozdziale przeznaczonym dla użytkowników platformy.

## 5.6 Uruchomienie testu oraz omówienie wyników

Do tej pory omówiono poszczególne komponenty wchodzące w skład środowiska testowego. W tym rozdziale zwróci sie uwagę na ich współpracę poprzez uruchomienie przykładowego przetwarzania wykorzystującego przygotowany zbiór danych

oraz wszystkie komponenty środowiska testowego.

Na początku należy uruchomić brokera Apache Kafka. Następnie aplikację ładującą dane z przygotowanego pliku z danymi i wysyłającą je do kolejki. Przyjęto, że dane z pliku zostaną powielone 400 razy, przez co otrzymamy 332000 instancji testowych. W ostatniej kolumnie zostanie przesłana klasa, do której przynależą dane, dzięki czemu zbada się wydajność przeprowadzonego przetwarzania. Kolejnym krokiem jest uruchomienie klastra Apache Storm, na którym zostanie uruchomione zadanie Apache SAMOA. Ostatnim krokiem jest zainicjowanie przetwarzania. Autor zdecydował się na przetestowanie działania techniki *prequential evaluation*, w której jako metodę klasyfikacji wybrano *Vertical Hoeffding Tree*, a źródłem danych jest dodany komponent potrafiący pobierać dane z Apache Kafka. Polecenie uruchamiające przetwarzanie ma następującą postać:

```
$ ./bin/samoa storm
    ./target/SAMOA-Storm-0.4.0-incubating-SNAPSHOT.jar
        "PrequentialEvaluation -d /tmp/dump.csv
        -i 322000 -f 830
        -l (classifiers.trees.VerticalHoeffdingTree -p 4)
        -s (kafka.KafkaStream -t topic)"
```

W powyższym poleceniu widać, że uruchamiamy zadanie z wykorzystaniem Apache Storm. Jako technikę przetwarzania wykorzystujemy *prequential evaluation*. Wyniki przetwarzania zapisujemy do domyślnej lokalizacji. Parametr *-i* oznacza, po ilu instancjach przetwarzanie zostanie zakończone. Wskazujemy tutaj liczbę wszystkich instancji, jakie zostały wysłane na kolejkę. Parametr *-f* pokazuje, po ilu instancjach nastąpi podsumowanie i zapis statystyk przetwarzania do pliku. Przyjęto tutaj wartość 830, ponieważ tyle instancji znajduje się w pliku i autor chciał zaobserwować, jak będzie zmieniała się wydajność przetwarzania wraz z kolejnymi iteracjami danych w nim zawartych. Jako algorytmu klasyfikacji użyto wcześniej wspomnianego algorytmu *Vertical Hoeffding Tree*. Ostatni parametr wskazuje na źródło danych. Do projektu dodano kod odpowiedzialny za pobieranie danych z brokera Apache Kafka *KafkaStream*. Jako parametr *-t* podajemy nazwę tematu, z jakiego mają zostać pobrane dane. Adresu IP oraz numeru portu kolejki nie musimy podać, ponieważ broker został uruchomiony na domyślnych ustawieniach klasy *KafkaStream*.

W wyniku przeprowadzonego przetwarzania otrzymaliśmy następujące wyniki:

```
classified instances = 332 000
classifications correct (percent) = 86,57
Kappa Statistic (percent) = 73,063
Kappa Temporal Statistic (percent) = 74,433
```

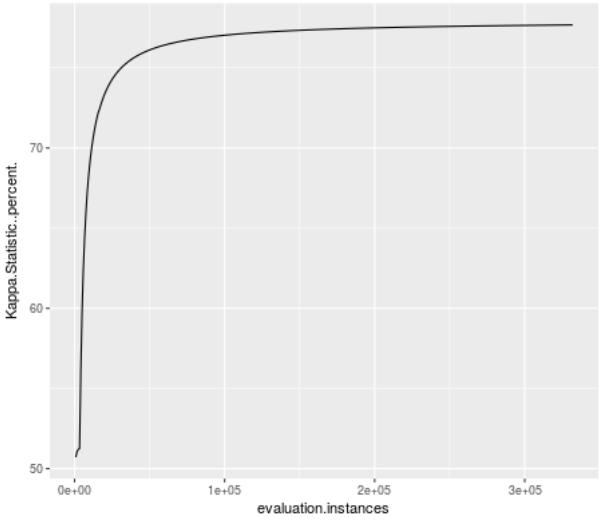
W pliku *dump.csv* podanym w parametrze *-d* otrzymaliśmy 400 wierszy będących reprezentacją zrzutu statystyk badających wydajność przetwarzania. Liczba 400 wierszy wynika z przetworzenia 322000 instancji, podczas którego zbieraliśmy statystyki co 830 instancji (322000 / 830 = 400). Pierwszy wiersz jest wierszem nagłówkowym, opisującym dane w poszczególnych kolumnach:

- **evaluation instances** - liczba przetworzonych instancji
- **classified instances** - liczba sklasyfikowanych instancji
- **classifications correct (percent)** - procent instancji, które zostały sklasyfkowane poprawnie
- **Kappa Statistic (percent)** - statystyka Kappa Cohena
- **Kappa Temporal Statistic (percent)** - statystyka Temporal Kappa

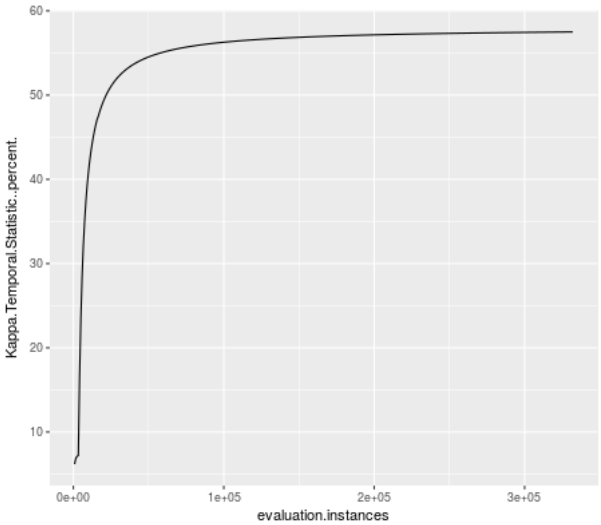
Na szczególną uwagę zasługuje tutaj statystyka Temporal Kappa, która została opisana w artykule *Efficient Online Evaluation of Big Data Stream Classifiers* [36]. Jest to miara uwzględniająca zależności czasowe w danych strumieniowych i wyraża się ją wzorem:
$\mathrm{K}_{per} = \frac{p - p_{per}}{1 - p_{per}}$

gdzie P*per* oznacza klasę trwałą, czyli taką, która przewiduje, że przyszła klasa będzie taka sama, jak ostatnia przewidziana.

Poniżej - za pomocą środowiska R - utworzono wykresy prezentujące przyrost statystyki Kappa i Temporal Kappa wraz ze wzrostem liczby przetworzonych instancji.




Rysunek 5.2: Wykres statystyki Kappa utworzony na podstawie danych uzyskanych z przeprowadzonego testu




Rysunek 5.3: Wykres statystyki Temporal Kappa utworzony na podstawie danych uzyskanych z przeprowadzonego testu

Jak widzimy na powyższych wykresach, przeprowadzone testy potwierdzają tezę, że podczas używania techniki *prequential evaluation*, czym więcej danych zostanie przetworzonych, tym wydajność klasyfikacji rośnie. Na początku widzimy, jak wykresy - wraz ze wzrostem przetwarzanych instancji - szybko rosną, by w końcu

ustablilizować się i wzrastać łagodniej. Razem z przetworzeniem jeszcze większej ilości danych, skuteczność klasyfikacji powinna jeszcze bardziej wzrosnąć.

W podrozdziale przedstawiono wyniki przeprowadzonych eksperymentów. Zostały one wykonane podczas badania działania biblioteki Apache SAMOA jako narzędzia do wyszukiwania anomalii w danych. Przeprowadzono eksperyment polegający na uruchomieniu zaledwie jednego algorytmu, jednak warto kontynuować pracę i przetestować pozostałe zaimplementowane algorytmy. Dobrym pomysłem na kontynuację badań jest również użycie różnych zbiorów danych, które zawierały zmianę charaktetystyki klasy decyzyjnej. Warto także zgłębić wiedzę na temat technik oceny efektywności przetwarzania danych strumieniowych opisanych we wspomnianym wcześniej artykule [36].

# Rozdział 6

# Podsumowanie

Niniejsza praca miała na celu zwrócenie uwagi na zagadnienie wyszukiwania anomalii w danych, a w szczególności w danych strumieniowych. Autor pracy dokonał analizy tematu i - posiłkując się dostępną dokumentacją - opisał problemy, jakie pojawiają się podczas przetwarzania danych strumieniowych. Praca pozwoliła autorowi zgłębić temat wyżej wymienionych zagadnień. Przed przystąpieniem do pisania pracy autor nie zdawał sobie sprawy z wielu zagadanień, które zostały poruszone w pracy. Rozpoznanie tematu dostarczyło wiele nowych doświadczeń i wiedzy, którą warto zgłębić w przyszłości. W pracy poruszono kilka obszarów dotyczących analizy danych oscylujacych wokół wykrywania anomalii, ukazując potencjalne obszary rozwinięcia poszczególnych problemów czy dokonania analizy innych narzędzi niż opisane w pracy.

Autor skupił się na narzędziach wykorzystywanych przez języki programowania implementowane na Wirtualną Maszynę Java, ponieważ to środowisko jest mu najbliższe. Szczególną uwagę poświęcono uczeniu maszynowemu, za pomocą którego możemy wykrywać odstępstwa od normy w danych, a także bibliotece Apache SAMOA, by sprawdzić, czy jest ona dobrym narzędziem do użycia w systemie wykrywającym anomalie. Autor ma świadomość, że problemy opisane w pracy nie należą do trywialnych, a praca prezentuje wstęp do dalszego zgłębiania tematu, pokazując informacje oraz narzędzia, na które warto zwrócić uwagę przy próbie budowy systemu zorientowanego na wykrywanie anomalii. Z pewnością na bycie potencjalnym kierunkiem dalszych badań i testów zasługują technologie wspierające uczenie maszynowe wykorzystujące procesory graficzne jako jednostki obliczeniowe. Warto tutaj zgłębić bibliotekę przygotowaną przez firmę NVidia oraz przyjrzeć się bliżej tematowi ich łączenia ze skalowalnymi systemami przetwarzania danych, czego przykładem jest wspomniany w pracy projekt HeteroSpark. Na początku pisania pracy autor zadecydował, że przyjrzy się bliżej architekturze i możliwościom platformy Apache Storm jako silnika przetwarzającego dane strumieniowe, gdyż silnik ten przetwarza dane w czasie rzeczywistym, a nie wykorzystującym micro-batching (np. Apache Spark), przez co nadawał się do zastosowania w projektowanym systemie. Podczas zbierania informacji o systemach przetwarzających dane strumieniowe okazało się, że powstaje projekt Heron, który ma za zadanie zastąpić Apache Storm, lecz jest on jeszcze w bardzo wczesnej fazie. Warto w przyszłości obserwować jego rozwój - na uwadze szczególnie powinny mieć go osoby, które dotychczas używają Storma, ponieważ Heron ma mieć całkowite wsparcie do topologii zbudowanych na ten silnik. Uwagę autora zwróciła także platforma Apache Flink zdobywająca coraz większą popularność. Warto przyjrzeć się bliżej temu systemowi, jego architekturze, a przy próbie implementacji zaproponowanej architektury warto rozważyć zamianę Apache Storm i biblioteki SAMOA na Flinka wraz z jego dostępnymi metodami uczenia maszynowego. Kolejnym potencjalnym obszarem dalszych badań jest bliższe przyjrzenie się tematowi *concept driftu*, a szczególnie algorytmom pozwalającym na jego wykrywanie, jak chociażby wspomniany algorytm ADWIN. Warto dokonać adaptacji algorytmów wykrywających zmiany w charakterystyce danych do systemu wykrywającego anomalie, ponieważ - zdaniem autora - algorytmy te z powodzeniem

mogą służyć jako detektory anomalii, choć, jak wykazano w pracy, nie zawsze zmiana charakterystyki danych powinna być uważana za anomalie. Podczas tworzenia pracy poświęcono dużo czasu na analizę działania biblioteki Apache SAMOA. W pracy wykazano, że w obecnym stanie nie jest ona idealnym rozwiązaniem do zastosowania w systemie wykrywającym anomalie, lecz jest to projekt z dużym potencjałem i cały czas rozwijany przez autorów oraz społeczność użytkowników. Autorzy podają, że chcą doimplementować na platformę kolejne algorytmy. Warto dokonać próby rozbudowy biblioteki o implementację algorytmu uczenia maszynowego, wykorzystując udostępnione przez twórców API lub dodać do algorytmu VFDT części odpowiedzialnej za wykrywanie *concept driftu*. Bibliotekę można także rozbudować o nowe źródła danych lub moduły odpowiedzialne za zapis wyników przetwarzania nie tylko na zaproponowaną w pracy kolejkę, ale także np. o zapis do bazy danych czy do systemu plików HDFS. Ciekawym tematem badań, polegających na wykorzystaniu dostępnej biblioteki w praktyce, jest zbudowanie systemu opartego o biblitekę TensorFlow. Zdobywając informacje na jej temat autor zauważył, że wiele firm, a w tym Google, wykorzystuje ją jako silnik uczenia maszynowego w swoich projektach.

Jak wyżej wykazano, potencjalnych obszarów dalszych badań czy rozwoju poruszanych tematów jest niezmiernie wiele. Autor starał się przedstawić, oprócz samej tematyki dotyczącej anomalii, także narzędzia oraz problemy wynikające z przetwarzania danych strumieniowych. Praca pozwala poznać tematykę wykrywania anomalii oraz analizy danych strumieniowych, pokazując czytelnikowi temat, który jest obszerny i warty pogłębiania wiedzy. Potęguje to fakt, że w dzisiejszym świecie, gdzie dane przyrastają w niewyobrażalnym tempie, szybka analiza danych, a w szczególności szybkie wykrycie anomalii, jest w wielu przypadkach bardzo ważną, jak nie krytyczną, informacją. Na pewno w przyszłości będzie zapotrzebowanie na inżynierów obeznanych z zagadnieniami przetwarzania dużych wolumenów danych oraz danych strumieniowych, jak również znających temat wykrywania w nich anomalii, przez co niniejsza praca jest dobrym wstępem do tej tematyki.

# Literatura


[1] What is big data? http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html, Grudzień 2016.

[2] Michael Hausenblas, Nathan Bijnens. Lambda Architecture. http://lambda-architecture.net/, Grudzień 2016.

[3] Marz Nathan, Warren James. *Big Data. Najlepsze praktyki budowy skalowalnych systemów obsługi danych w czasie rzeczywistym.* 2016.

[4] Shu Uesugi. Kappa Architecture. http://milinda.pathirage.org/kappa-architecture.com/, Grudzień 2016.

[5] Strumień danych - definicje. https://pl.wikipedia.org/wiki/Strumie%C5%84˙danych, Grudzień 2016.

[6] JOÃO GAMA, NDRĖ ŽLIOBAITĖ, ALBERT BIFET, MYKOLA PECHENIZKIY, ABDELHAMID BOUCHACHIA. A Survey on Concept Drift Adaptation. April 2014.

[7] A. Bifet, J. Gama, M. Pechenizkiy, I. Žliobaitė. Handling Concept Drift:Importance, Challenges Solutions. Maj 2011.

[8] Richard Kirkby Albert Bifet. *DATA STREAM MINING A Practical Approach.* August 2009.

[9] GPU Applications. Transforming computatuinal research and engineering. Machine Learning. http://www.nvidia.com/object/machine-learning.html, Grudzień 2016.

[10] CluStream. http://huawei-noah.github.io/streamDM/docs/CluStream.html, Grudzień 2016.

[11] Ankit Jain, Anand Nalya. *Learning Storm.* Packt Publishing Ltd, 2014.

[12] Apache Storm - strona projektu. http://storm.apache.org/, Grudzień 2016.

[13] Twitter Heron. https://blog.twitter.com/2015/flying-faster-with-twitter-heron, Grudzień 2016.

[14] Apache Kafka. https://kafka.apache.org/, Grudzień 2016.

[15] Apache Samza. http://samza.apache.org/, Grudzień 2016.

[16] Spark Streaming. http://spark.apache.org/streaming/, Grudzień 2016.

[17] Apache Flink. https://flink.apache.org/, Grudzień 2016.

[18] Amazon Kinesis. https://aws.amazon.com/kinesis/streams/, Grudzień 2016.

[19] Druid data store. http://druid.io/, Grudzień 2016.

[20] Massive Online Analysis. http://moa.cms.waikato.ac.nz/, Grudzień 2016.

[21] Scalable Advanced Massive Online Analysis. https://samoa.incubator.apache.org/index.html, Grudzień 2016.

[22] SAP HANA Smart Data Streaming. http://www.sap.com/developer/topics/hana-smart-data-streaming.html, Grudzień 2016.

[23] TensorFlow. https://www.tensorflow.org/, Grudzień 2016.

[24] Elektrokardiografia. https://pl.wikipedia.org/wiki/Elektrokardiografia, Grudzień 2016.

[25] Fernando Esponda, Stephanie Forrest, and Paul Helman. A Formal Framework for Positive and Negative Detection Schemes. Luty 2004.

[26] Ellen Friedman Ted Dunning. *Practical Machine Learning. A New Look at Anomaly Detection.* June 2014.

[27] VARUN CHANDOLA, ARINDAM BANERJEE, VIPIN KUMAR. Anomaly Detection : A Survey. Wrzesień 2009.

[28] AppFuse. http://wiki.appfuse.org/display/APF/Home, Grudzień 2016.

[29] Apache ZooKeeper. https://zookeeper.apache.org/, Grudzień 2016.

[30] Apache Mesos. http://mesos.apache.org/, Grudzień 2016.

[31] Mammographic Mass Data Set. https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass, Grudzień 2016.

[32] skala BI-RADS. https://pl.wikipedia.org/wiki/BI-RADS, Grudzień 2016.

[33] Storm - Command Line Client. http://storm.apache.org/releases/1.0.1/Command-line-client.html, Grudzień 2016.

[34] Apache SAMOA Documentation. https://samoa.incubator.apache.org/documentation/Home.html, Grudzień 2016.

[35] Add Kafka stream reader modules to consume data from Kafka. https://github.com/apache/incubator-samoa/pull/32, Grudzień 2016.

[36] Albert Bifet, Gianmarco De Francisci Morales, Jesse Read, Geoff Holmes. Efficient Online Evaluation of Big Data Stream Classifiers. Grudzień 2016.

# Dodatek A

# Zawartość płyty

Dodatkiem do niniejszej pracy jest płyta DVD. Zawiera ona materiały zgromadzone podczas jej pisania oraz utworzone kody źródłowe. Zawartość płyty może być pomocna, gdy będzie konieczność kontynuacji tematyki poruszonej w pracy.
Na płycie znajdują się następujące rzeczy:

1. Użyty w eksperymentach zbiór danych
2. Kody źródłowe
    a) aplikacji ładującej dane
    b) Apache SAMOA wraz z wprowadzoną modyfikacją
    c) skrpyty wykonane w środowisku R wraz z utworzonym projektem R Studio.
3. wersja uruchomieniowa Apache Kafka
4. wersja uruchomieniowa Apache Storm
5. tekst pracy w formacie pdf oraz tex
6. plik z wynikiem przetwarzania



Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Łukasz Kiszka",
    title = "{Wykrywanie anomalii w strumieniach danych}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2016",
}
```