POLITECHNIKA ŚLĄSKA
WYDZIAŁ INŻYNIERII MATERIAŁOWEJ I METALURGII

Kierunek: Informatyka Przemysłowa

Specjalność:
Bezpieczeństwo Systemów Komputerowych

Rodzaj studiów:
zaoczne

Praca dyplomowa magisterska
Damian WILKOŁEK

TEMAT PRACY DYPLOMOWEJ

ZASTOSOWANIE OBLICZEŃ ROZPROSZONYCH W ZAGADNIENIACH INTELIGENCJI OBLICZENIOWEJ

Kierujący pracą:
Dr inż. Marcin Blachnik

Recenzent:
Prof. zw. dr hab. Tadeusz Wieczorek

Katowice, czerwiec , 2015 r.

Spis treści

1. Wstęp

W dzisiejszych czasach gromadzimy duże ilości danych, ale trudno jest zebrane dane przekształcić w wiedzę, którą można będzie wykorzystać w praktyce. Klasyczne metody statystyczne nie są w stanie sprostać takim wyzwaniom, dlatego też wszystkie wiodące firmy związane z przetwarzaniem danych w swoich flagowych rozwiązaniach implementują narzędzia z obszaru inteligencji obliczeniowej. Pozwalają one na budowę modeli predykcyjnych umożliwiających wnioskowanie o przyszłości na podstawie zebranych danych historycznych. Przykładem mogą tutaj być produkty takie jak MS SQL i SQL Services, IBM SPSS, Statistica DataMiner czy też wyrastający z open source RapidMiner. IBM stosując metody z grupy inteligencji obliczeniowej w swoim oprogramowaniu potrafi uprościć podejmowanie decyzji na podstawie wszelakich dokumentów, którymi mogą być dzienniki, odpowiedzi na ankiety badające rynek, rekordy z bazy danych (np. poczta elektroniczna wysyłana przez klientów do działu obsługi klienta). Wykonując analizę tekstu, oprogramowanie to potrafi zidentyfikować połączenia pomiędzy szerokim zakresem dokumentów i odkryć sentymenty, trendy w nich występujące. Specjaliści, wykorzystując to oprogramowanie, mogą używać algorytmów do opisu zbiorów danych i tworzyć relacje pomiędzy nimi. Tak tworzone encje mogą być przez nich dołączane do modeli używanych w analizie predykcyjnej [20].

We współczesnym biznesie budowa modeli predykcyjnych znalazła szereg zastosowań, wśród których można wymienić:

- analizę sentymentów, czyli predykcję upodobań klientów
- analizę tekstów, czego przykładem może być automatyczna klasyfikacja wiadomości poczty elektronicznej, czy też rozpoznawanie niechcianej poczty (spam)
- rynek ubezpieczeń - przewidywanie ryzyka
- informatyka przemysłowa - wykrywanie i identyfikacja awarii zanim takowa nastąpi, inteligentne systemy doradcze procesów produkcji itp.

Efektywne przetwarzanie dużych ilości danych jest jednak czasochłonne, co znacznie ogranicza możliwości klasycznych systemów inteligencji obliczeniowej. Z pomocą przychodzą tutaj narzędzia i techniki przetwarzania rozproszonego pozwalające na dystrybucję pracy na wiele równorzędnych maszyn obliczeniowych znacząco redukując czas przetwarzania danych. Do popularnych rozwiązań z tej dziedziny należy np. GridGain, Hadoop, Rio czy Java Parallel Processing Framework. Wykorzystując taką technologię wraz z systemami chmurowymi, można stworzyć wydajną i elastyczną platformę pozwalającą na przetwarzanie ogromnych ilości.

Wychodząc naprzeciw tym wyzwaniom, niniejsza praca dyplomowa podejmuje tematykę analizy skalowalności wybranych zagadnień budowy modeli predykcyjnych.

2. Cel i zakres pracy

2.1. Cel

Celem pracy jest weryfikacja możliwości przyspieszenia budowy złożonych modeli predykcyjnych należących do grupy metod inteligencji obliczeniowej poprzez wykorzystanie programowania rozproszonego

2.2. Zakres pracy

Zakres pracy obejmuje:

- stworzenie aplikacji testowej na bazie środowiska RapidMiner z wykorzystaniem biblioteki Apache Ignite
- implementacja różnych scenariuszy zrównoleglenia procesu budowy modeli predykcyjnych:
    - jeden zbiór danych - różne modele predykcyjne,
    - wiele zbiorów danych - jeden model predykcyjny
    - wiele zbiorów danych - wiele modeli predykcyjnych
- badania empiryczne opracowanych rozwiązań na przykładzie zrównoleglonej optymalizacji parametrów konfiguracyjnych sieci neuronowe typu MLP

3. Inteligencja Obliczeniowa

3.1. Definicja

Inteligencja obliczeniowa (ang. Computational Intelligence) to nowa dziedzina nauki zajmująca się tworzeniem rozwiązywań coraz to bardziej złożonych problemów. Złożoność problemu objawia się w trudnościach stworzenia algorytmu do rozwiązania zadania [8]. Pomimo, iż niektóre problemy mogą być rozwiązywane za pomocą metod analitycznych, to są one nieefektywne. Inteligencja obliczeniowa dostarcza metod, które mogą wykonać zadanie efektywniej dzięki prezentacji problemu w formie symbolicznej. Pozwala to na sprowadzenie problemu do prostych zasad [1], których własnoręczne opisanie mogłoby być bardzo trudne lub nawet niewykonalne ze względu na wielkość zbioru danych do przeanalizowania. Problemów takich jest coraz więcej, a wraz ze zwiększającą się ilością przetwarzanych danych konieczne jest tworzenie nowych sposobów na ich rozwiązywanie. Przykładem problemu niealgorytmizowalnego może być rozpoznawanie pisma [14]. Nie da się tego rozwiązać za pomocą jednego algorytmu. Trudno jest też jednoznacznie określić jego zakres ze względu na różnorodność charakteru pisma. Tak jak można opisać cyfrę „8” jako dwa kółka ułożone jedno na drugim, tak każda z osób piszących tę cyfrę może mieć różne tendencje. Jedna osoba będzie pisała ją pochyloną, druga będzie miała skłonność do pisania jej bardzo smukłej tak by była zbliżona formą do cyfry „1”. Wobec tego, nie można określić jak powinien wyglądać szablon tak, aby był on uniwersalny dla każdej napisanej cyfry „8”. Na podstawie tego przykładu możemy wskazać na czym polegają trudności w tworzeniu algorytmu do rozwiązania tego zadania.

Pierwszym z nich będzie brak uniwersalności. Oznacza to że mała zmiana parametrów problemu, tak jak krój pisma, może oznaczać konieczność stworzenia nowego algorytmu uwzględniającego inny szablon. Co więcej, program może natknąć się na nieznane sobie przypadki, których nie będzie potrafił sklasyfikować, ani nauczyć się ich rozpoznawać. Równie ciężko jest określić wszystkie parametry, które mogą ulegać zmianie oraz jej zakres. Powoduje to problemy w generalizacji problemu, ponieważ nie można go sprowadzić do prostych reguł, które pozwolą na jednoznaczne rozwiązanie.

Opisany przykład nie jest odosobnionym problemem, z którymi próbują się mierzyć tworzone algorytmy.

Zastosowanie inteligencji obliczeniowej może być także przydatne w problemach takich jak [14]:

- Klasyfikacja struktur – rozpoznanie struktury i przypisanie jej do pewnej grupy
- Selekcja cech – redukcja złożoności problemu
- Inteligentne wspomaganie decyzji – np. diagnozy medyczne
- Gry – uczenie się na własnych i cudzych błędach
- Kontrola – dostosowywanie parametrów aparatury do istniejących warunków
- Prognozowanie wskaźników ekonomicznych, pogody, intencji człowieka

Tworzone algorytmy w ramach inteligencji obliczeniowej często są zaczerpnięte z innych dziedzin tj. matematyka, biologia, chemia, informatyka.

Duży sukces odniosły inteligentne algorytmy takie jak sztuczne sieci neuronowe, obliczenia ewolucyjne czy algorytmy rojowe.

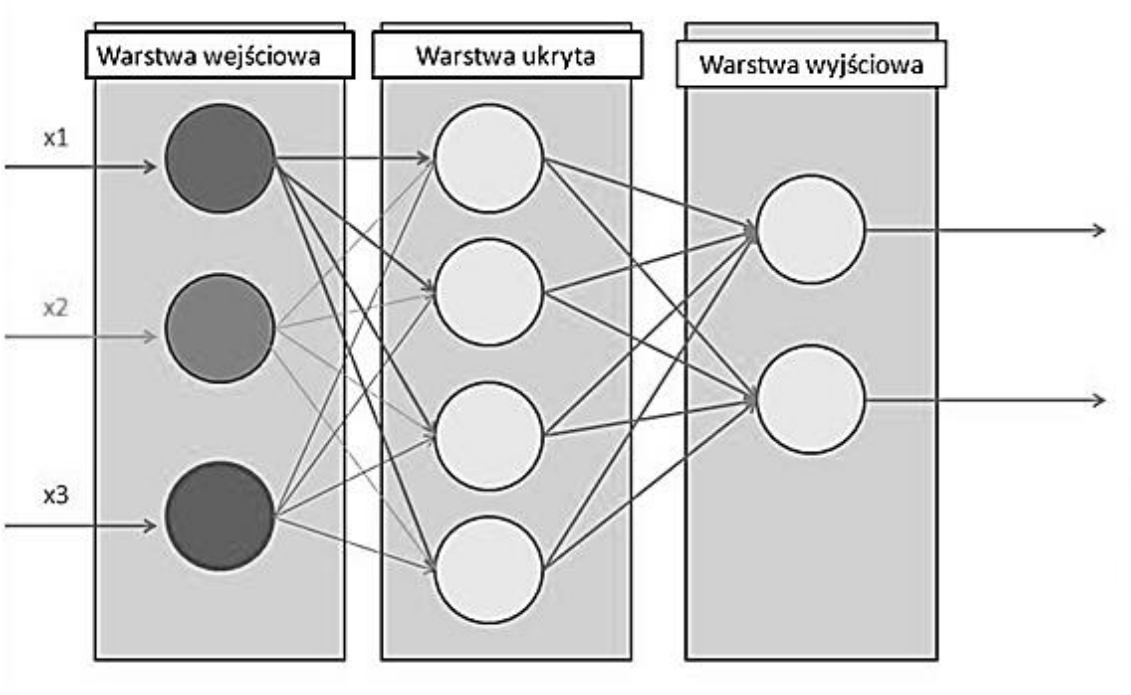
3.2. Sztuczne sieci neuronowe

W roku 1943 został opracowany przez neurologa Warrena S. McCulloch oraz logika Waltera Pitts pierwszy model sztucznej sieci neuronowej bazujący na ludzkim mózgu. Ich praca opisywała koncept neuronu jako pojedynczej komórki w sieci komórek przetwarzających równolegle otrzymane sygnały na wejściu na sygnały wyjściowe [8]. Model ten został stworzony, aby uprościć rozwiązywanie zadań, które stanowiły problem dla komputera, a dla człowieka nie. Przykładowo człowiek bardzo łatwo odróżni dwa przedmioty – co jest bardzo trudnym zadaniem dla komputera. Z drugiej strony, komputer szybko za pomocą wykonania jednej linijki kodu poda ile wynosi pierwiastek kwadratowy z 15345263376, gdzie człowiek zupełnie nie poradzi sobie z rozwiązaniem tego

działania w głowie. Różnice w przetwarzaniu informacji były motorem do prowadzenia badań w tym kierunku.

Jak zostało już wspomniane, sieć neuronowa jest zbiorem połączonych ze sobą neuronów przetwarzających sygnały. W sztucznych sieciach neuronowych neurony zostały poukładane na minimum trzech warstwach [9]:

- Warstwa wejściowa – przekazuje sygnały wejściowe do warstwy ukrytej
- Jedna lub więcej warstw ukrytych – na tych warstwach następuje faktyczne przetwarzanie sygnałów i przesyłanie ich do neuronów na kolejnej warstwie ukrytej lub wyjściowej
- Warstwa wyjściowa – zajmuje się przetwarzaniem sygnałów z warstw ukrytych na sygnały wynikowe



Obraz 1 Schemat sztucznej sieci neuronowej

Neuron jest prostą jednostką, która do każdego wejścia przypisuje wagi i na podstawie ich oraz sygnałów wejściowych generuje określony sygnał na wyjściu.

Działanie takiego neuronu można sprowadzić do prostego algorytmu:

- Wylicz sumę iloczynów
- Wyliczona wartość niech będzie argumentem funkcji aktywacji
- Wynik funkcji aktywacji przekaż na wyjście

W tych 3 krokach została zawarta idea działania neuronu [9]. Wykonuje on proste operacje, na podstawie których możliwe jest wygenerowanie jednego sygnału na podstawie wag oraz sygnałów wejściowych. Różnicując wagi na poszczególnych neuronach możemy mówić o przyjmowaniu różnych stanów przez całą sieć w zależności od podanych sygnałów. Wspomniana na końcu funkcja aktywacji zajmuje się określaniem sygnału wyjściowego na podstawie otrzymanych sygnałów i ich wag [7].

$$y = f_a(\sum_{i=0}^{n} w_i x_i) \quad (1)$$

Gdzie:

- y – wartość sygnału wyjściowego
- x_i – wartość i-tego sygnału wejściowego do neuronu
- w_i – waga przypisana do i-tego wejścia do neuronu
- f_a – funkcja aktywacji

Funkcją aktywacji może być np. funkcja liniowa, która przepisuje sumę ważoną na wyjście w postaci znormalizowanej, lub tak jak w przypadku opisywanego modelu McCullocha i Pittsa – funkcja progowa unipolarna. Wynikiem tej funkcji jest jeden dla liczb dodatnich, natomiast zero dla liczb mniejszych bądź równych zeru [18].

Mówiąc o procesie uczenia sieci neuronowej mamy na myśli dobór wag na każdym z neuronów tak, aby móc w jakimś stopniu poprawnie dokonywać klasyfikacji. Sam proces uczenia jest niestety nieco bardziej skomplikowany i w dużej mierze zależy od architektury sieci neuronowej. W przypadku za małej liczby neuronów będzie dochodziło do zbyt dużego generalizowania problemu

i niemożliwości nauczenia się go. Będzie to skutkowało znaczną ilością błędów w klasyfikacji. W przypadku zastosowania za dużej ilości neuronów można doprowadzić do przeuczenia się sieci neuronowej. Jest to równoznaczne z nauką na pamięć tj. gdy problem ulegnie drobnej zmianie, system nie poradzi sobie z jego klasyfikacją, ponieważ sieć nie rozumie zasady, a zna tylko poszczególne przypadki na pamięć. W skrajnej sytuacji każdy neuron może być odpowiedzialny za jeden przypadek.

Uczenie sieci neuronowych może być wykonywane w dwojaki sposób:

- Nadzorowany – pozwalamy na zbiorze uczącym dokonać klasyfikacji, a następnie przekazujemy neuronowi pożądaną wartość. Dzięki temu neuron może skorygować wagi tak, by spełniał oczekiwania.

- Nienadzorowany – ten sposób jest wykorzystywany, gdy nie ma możliwości weryfikowania wiedzy. Ważne jest, aby sygnały wejściowe dały się sklasyfikować. Wymaga to większej liczby neuronów niż przypadków grup klasyfikacyjnych. Neurony muszą być różnorodne tj. mieć różne upodobania. Potrzebne jest to po to, aby każdy neuron znalazł swój przypadek, który będzie wspierał, promował.

Opisane tutaj metody uczenia i sposób określania sygnału wyjściowego są jednymi z najprostszych jakie zostały stworzone. Jest wiele różnych reguł określania wartości wyjściowych, sposobów doboru wag i ich korekcji. Ze względu na samą nieograniczoność konfiguracji sieci neuronowych (ilość warstw ukrytych i ilość neuronów na każdej warstwie) ciężko jest dobrać najodpowiedniejszą [7]. Najprostszym rozwiązaniem byłoby przetestowanie każdej z tych sieci neuronowych, jednak jest to bardzo czasochłonny proces, z powodu wstępnego nauczania sieci oraz późniejszej walidacji otrzymanych wyników i ich adekwatności. Dodając do tego zwiększającą się ilość przetwarzanych danych konieczne jest tworzenie coraz wydajniejszych systemów, które sprostają stawianym wymaganiom ze względu na wielkość danych, jak i złożoności problemu.

3.3. Algorytm propagacji wstecznej

Zostało już wspomniane w niniejszej pracy, że celem nauki sztucznej sieci neuronowej jest określenie wartości wag neuronów we wszystkich warstwach. Pozwala to na otrzymanie określonego sygnału na wyjściu przy ustalonych wejściach. Jeżeli funkcja celu opisująca to zjawisko jest funkcją ciągłą, możemy się posłużyć metodą gradientową do ustalenia nowych wag.

$$W_{k+1} = W_k + \Delta W \quad (2)$$

$$\Delta W = \eta p(W) \quad (3)$$

gdzie:

W – wektor wag

η – współczynnik uczenia

p(W) – kierunek w przestrzeni wielowymiarowej W

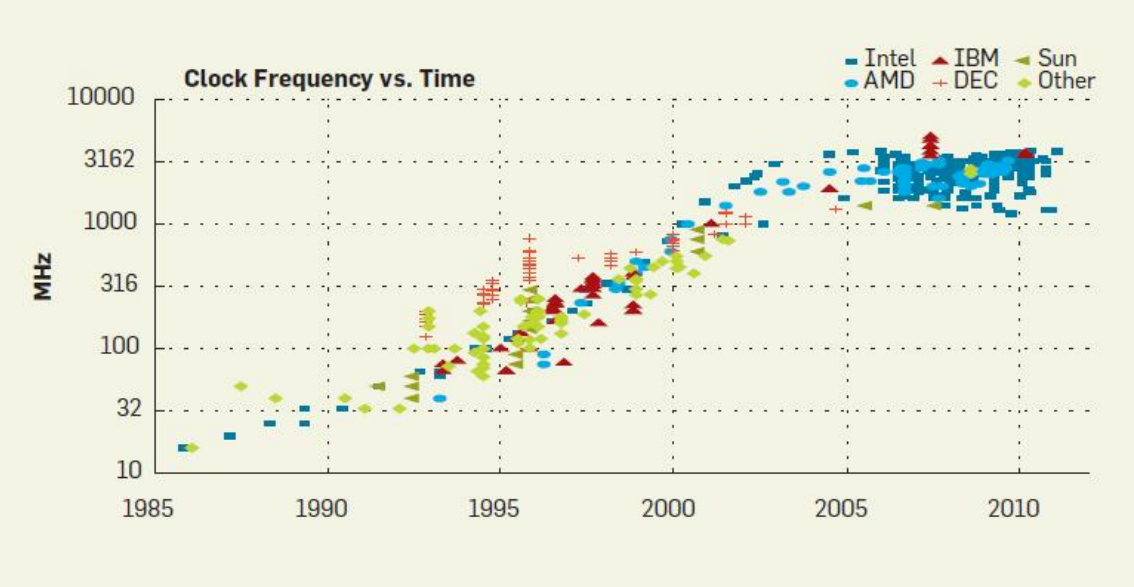
Zastosowanie tej metody do uczenia sieci neuronowej wymaga postępowania wg algorytmu propagacji wstecznej (ang. backpropagation). W tym algorytmie wyróżniane są cztery etapy [7]:

- Analiza sieci neuronowe – wykonanie przebiegu sygnałów przez całą sieć neuronową tak by otrzymać wyjście. Podczas wykonywania przebiegu należy zbierać wartości sygnałów wyjściowych z neuronów ukrytych, warstwy wyjściowej oraz pochodne funkcji aktywacji
- Wykonanie przebiegu w kierunku odwrotnym zastępując funkcje aktywacji ich pochodnymi. Na wejście procesu należy natomiast przekazać sygnał w postaci odpowiedniej różnicy pomiędzy wartością otrzymaną a żądaną
- Dostosowanie wag na podstawie uzyskanych wyników wg odpowiednich wzorów (2)
- Cały algorytm należy powtarzać aż do uzyskania oczekiwanej dokładności odpowiedzi z uczonej sieci neuronowej

W zależności od przyjętej strategii, aktualizacji wag można dokonywać po przejściu jednej próbki danych, jak i dopiero po wykonaniu całej iteracji dla wszystkich próbek w zbiorze uczącym. Ze względu na sam proces uczenia i elementy konfiguracyjne sieci, zbudowanie odpowiedniego modelu predykcyjnego jest trudne oraz wymaga poświęcenia czasu na dokonanie różnych prób. Wykorzystując nowe technologie można przyspieszyć proces tworzenia i weryfikowania modeli predykcyjnych, co starano pokazać się w tej pracy.

4. Programowanie rozproszone

Od czasu pierwszych procesorów następowała poprawa wydajności, poprzez zwiększaną ilość tranzystorów oraz zwiększaną ich częstotliwość [2]. Od roku 2005 wzrost częstotliwość raczej uległ zatrzymaniu, a jedynie jest poprawiana ich optymalizacja.



Rysunek 1 Częstotliwość procesorów od 1985r.
[źródło: http://deliveryimages.acm.org/10.1145/2140000/2133822/figs/f7.jpg]

Aktualnie procesory stoją już prawie na granicy procesu technologicznego. Na rynku zostały wdrożone procesory wytwarzane w procesie technologicznym wynoszącym 14nm. Intel planuje wprowadzić procesory wytwarzane z krzemu w procesie 10nm, 7nm i 5nm. W przypadku tego ostatniego prawdopodobnie krzem będzie musiał być zastąpiony grafenem. Każdy nowy proces technologiczny wiąże się ze znacznie większymi kosztami opracowania nowej

technologii tworzenia. Budując system, który poradzi sobie z trudniejszymi i bardziej czasochłonnymi zadaniami jest droższe oraz mniej efektywne. Stąd coraz częściej firmy sięgają po nowe rozwiązania architektoniczne, aby używać wielu urządzeń jednocześnie. Mogą one wykorzystywać między innymi urządzenia mobilne, komputery stacjonarne, stacje robocze, serwery oraz szeroką gamę systemów wirtualnych. Takie rozwiązania pozwalają na znaczne zwiększenie przepustowości oraz ilości dostępnej pamięci, wydajności systemu jako całego poprzez wykorzystywanie wielu procesorów [16].

Wcześniej tworzenie systemów rozproszonych było bardzo pracochłonnym procesem ze względu na bardzo zróżnicowaną architekturę wszystkich urządzeń. Dodatkowo, do niedawna wzrost wydajności każdej kolejnej platformy był znaczny, co powodowało nieopłacalność takiego przedsięwzięcia. Poświęcenie n-lat na zbudowanie wydajnego i sprawdzonego systemu rozproszonego było niekorzystne ze względu na zyski jakie by przyniosło kupno nowej platformy, która byłaby szybsza. Obecnie środowiska programistyczne wyewoluowały do stanu, w którym tworzenie takich systemów jest wydajne oraz przenośne. Aplikacje stworzone w językach programowania tj. Java czy C# mogą być uruchamiane na wielu platformach niezależnie od fizycznego wyposażenia. Tworząc oprogramowanie, które będzie działało w systemie rozproszonym, cały proces tworzenia go należy podzielić na kilka ważnych etapów [4]:

- Identyfikacja części procesu, które mogą być wykonywane jednocześnie
- Podział tych części na procesy, które mogą być wykonywane równolegle
- Dystrybucja danych potrzebnych dla każdego z procesów
- Zarządzanie dostępem do danych współdzielonych przez wiele jednostek obliczeniowych
- Synchronizacja procesów w różnych stanach wykonywania równoległego programu

Oprogramowanie tworzone na podstawie powyższego algorytmu na pewno niesie wiele korzyści oraz na samym początku wskazuje, czy proces w ogóle

może być zrównoleglony. Co więcej, w zależności od przyjętego podziału procesu oraz zastosowanej architektury można spodziewać się lepszych bądź gorszych wyników. Można tu więc mówić o potrzebie doboru podziału procesu, jak i sposobu wykonywania procesu w zależności od dostępnych zasobów sieci obliczeniowej.

Wracając do systemów rozproszonych należy wspomnieć, czym była sieć obliczeniowa w roku 1998. Była ona definiowana jako infrastruktura łącząca oprogramowanie oraz sprzęt fizyczny, który dostarcza niezawodny, spójny, wszechobecny i tani dostęp do wysokiej klasy możliwości obliczeniowych. W późniejszym czasie definicja ta została rozszerzona o współdzielenie zasobów i systemy wirtualne. Cechami charakterystycznymi takiego systemu są:

- Współdzielenie zasobów – sprzętowych i aplikacyjnych
- Otwartość – wykorzystanie oprogramowania i sprzętu od różnych dostawców
- Równoległość – wykonywanie procesów równolegle, by uzyskać wzrost wydajności
- Skalowalność – zwiększenie przepustowości poprzez zwiększanie zasobów
- Odporność na błędy – możliwość dalszego świadczenia usług, pomimo występujących błędów

Mając na uwadze definicję sieci obliczeniowej oraz cech charakterystycznych, warto określić, co czyni sieć obliczeniową tworem potrzebnym i pożądanym w obecnej sytuacji. Najważniejszym czynnikiem sieci obliczeniowej jest wysoka jakość świadczonych usług, co gwarantuje zadowolenie użytkownika. Aby system dostarczał usług w takim kształcie, konieczne jest sprawdzanie jakości całej sieci jak i pojedynczych jednostek pod kątem dostępnych zasobów, czasu odpowiedzi czy bezpieczeństwa. Pomimo najwyższych parametrów sieci obliczeniowej, musi ona wspierać takie elementy jak bezpieczeństwo przetwarzanych danych (nie powinny one być dostępne dla

każdej jednostki w sieci obliczeniowej), możliwość rozszerzania sieci obliczeniowej, system obsługi błędów np. obsługa pojedynczego procesu, który nie zostanie zakończony sukcesem, zarządzanie siecią i trasowaniem zdarzeń w sieci obliczeniowej.

Systemy rozproszone mogą być tworzone w oparciu o dwie architektury:

- Klient-serwer
- Architektura rozproszonych obiektów

W architekturze klient-serwer wykorzystywane są usługi, których dostarczają serwery, natomiast klienci je tylko konsumują. Klient może równolegle wykorzystywać wiele usług jednego lub wielu serwerów. Problemem w tak tworzonej sieci obliczeniowej jest brak informacji o sposobie ich realizacji. Klient nie ma pojęcia o tym, czy są one wykonywane równolegle, czy sekwencyjnie. Do zalet takiego systemu można zaliczyć łatwość nawiązywania połączenia i wykorzystywania usług, gdyż najczęściej są one opisane w dokumencie zapisanym w języku Web Services Description Language (WSDL) [4], który wykorzystuje język XML. Dokument taki opisuje jakie usługi są dostępne oraz jak ma wyglądać wiadomość skierowana do każdej z metod usługi.

W przypadku architektury wykorzystującej rozproszone obiekty nie ma rozróżnienia pomiędzy klientem a serwerem. Każda rozproszona jednostka jest obiektem dostarczającym usług, jak i otrzymującym usługi od innych obiektów. W odróżnieniu od architektury klient-serwer, komunikacja odbywa się na poziomie middleware za pośrednictwem brokera żądań [5]. Pomimo większego skomplikowania tworzenia systemu opartego na tej architekturze, ma on wiele zalet:

- Pozwala projektantowi systemu na opóźnienie decyzji, gdzie i jak będą dostarczane usługi
- Pozwala na rozbudowanie systemu o nowe zasoby w zależności od potrzeb
- Jest bardziej skalowalny - pozwala na większą dowolność w jego kreowaniu

- Możliwe jest dynamiczne wprowadzanie zmian w sieci obliczeniowej

Niezależnie od tego, z jakiej architektury będzie korzystał system rozproszony dostarczający usług wysokiej jakości, będzie on wartością dodaną zwiększającą wydajność wykonywanych procesów.

5. Wykorzystane technologie i aplikacje

5.1. RapidMiner 5.3

RapidMiner jest aplikacja stworzoną do łatwego projektowania procesu eksploracji danych z użyciem graficznego interfejsu. Korzystanie z Rapidminer nie wymaga znajomości żadnego języka programowania. Całe środowisko zostało oparte o ideę przeciągnij i upuść co czyni je niesamowicie łatwym w użyciu. Poprzez upuszczanie operatorów na okno głównego procesu możemy stworzyć skomplikowane procesy korzystające z wielu różnych algorytmów i modeli przetwarzanych danych. Poza samymi operatorami które wykonują jakiś algorytm, jest też bogata baza operatorów pomagających sterować procesem co daje możliwość łatwiejszego i wygodniejszego projektowania procesu, szczególnie na początkowym stadium gdzie testowane są różne rozwiązania. Wszystkie operatory są pogrupowane wg. następującego schematu:

- Process Control – zawiera operatory do sterowania procesem. Są to np. pętle iterujące po atrybutach, etykietach, wartościach itd., operatory definiujące subproces, operatory pomagające zbierać i przetwarzać kolekcje danych.

- Utility – zawiera operatory odpowiedzialne za deklarację makr, logowanie zdarzeń, wykonywanie skryptów i programów, operacji na plikach, generowania zbiorów danych, wysyłania e-maili czy zarządzania pamięcią JVM.

- Repository Access – główną funkcjonalnością tego zbioru jest dostęp (odczyt i zapis) do repozytoriów danych, które są elementem RapidMinera.

- Import i Export – zawierają szereg operatorów umożliwiających import i export danych do plików, baz danych oraz formatów różnych popularnych narzędzi eksploracji danych takich jak Excel, SPSS, Weka itp.

- Data Transformation – jest to jeden z największych pakietów operatorów. Zawiera on operatory odpowiedzialne za konwersję typów, transformację i redukcję atrybutów, modyfikację wartości, filtrowanie, sortowanie.

- Modeling – zawiera operatory odpowiedzialne za budowanie modeli predykcyjnych tj. klasyfikacja, regresja, klasteryzacja, segmentacja i wiele innych. W tym pakiecie znajdziemy operatory odpowiedzialne np. za budowę i ewaluację modelu sieci neuronowej typu MLP

- Evaluation – ten pakiet jest to zbiór operatorów do pomiarów wydajności, wizualizacji procesów oraz walidacji zbudowanych modeli.

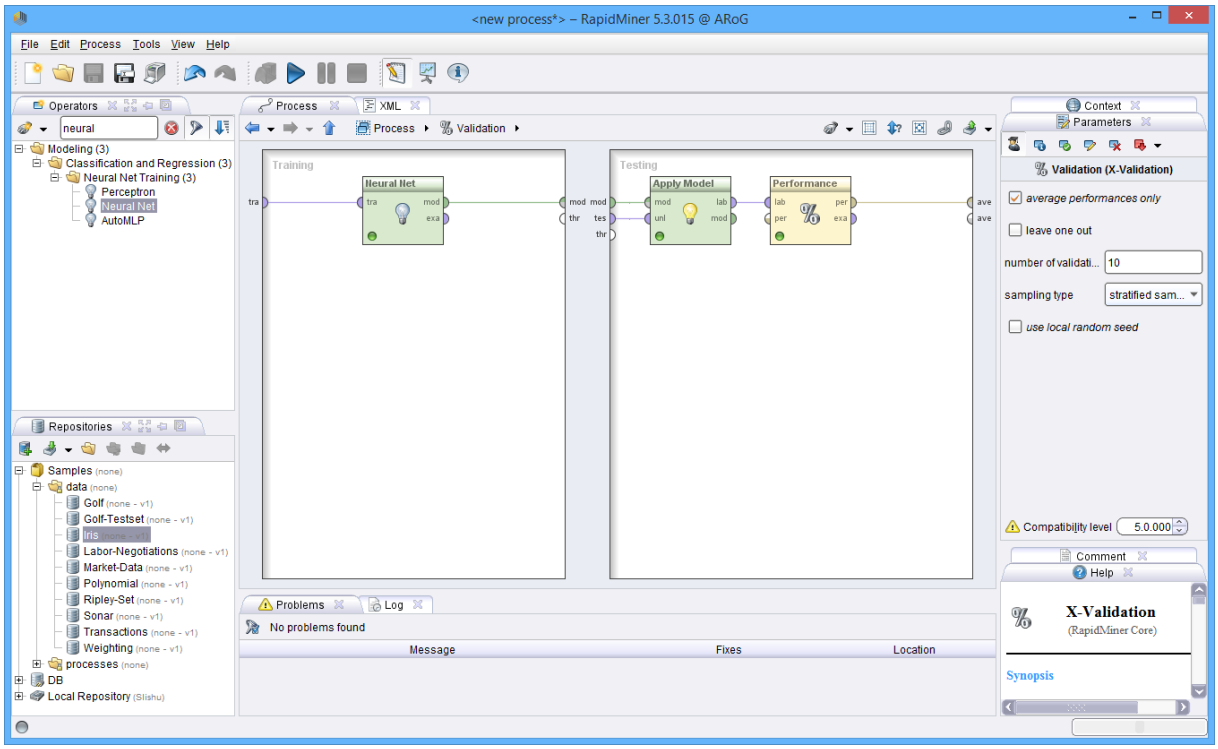
Jak widać, jest to bardzo rozbudowane oprogramowanie dające bardzo szeroką gamę narzędzi do wykonywania zadań związanych z maszynowym uczeniem, eksploracją danych, czy procesami z zakresu analityki biznesowej.

RapidMiner jest zbudowany modułowo, co pozwala na łatwą rozbudowę o nowe operatory [12]. Dzięki temu zostało już stworzone wiele niezależnych dodatków zajmujących się np. analizą tekstu czy analizą szeregów czasowych.

Niestety pomimo dużej funkcjonalności, nie zostały dostarczone w nim żadne metody pozwalające na zrównoleglenie procesów obliczeniowych. Mimo, iż istnieje dodatek dostarczający tej funkcjonalności, to posiada on liczne ograniczenia, jak chociażby konieczność prowadzenia obliczeń na tej jednej maszynie. Wychodząc naprzeciw istniejącemu zapotrzebowaniu na takie rozwiązania, został stworzony dodatek, który dostarcza możliwości przetwarzania procesów równolegle na wielu jednostkach obliczeniowych.

Rapidminer do wersji 5 był dystrybuowany na zasadach licencji AGPL v3.

W wersji 6 dostępnych jest wiele różnych wersji aplikacji w zależności od rodzaju i funkcjonalności. Wersja bezpłatna (Starter) oferuje nieco większą funkcjonalność tj. więcej i bardziej dopracowane operatory niż wersja piąta.



Obraz 2 Przykładowy proces stworzony w RapidMiner 5.3

5.2. Apache Ignite 1.0.0

Apache Ignite to zintegrowana platforma pozwalająca na wykonywanie równoległych obliczeń rozproszonych na dużych zbiorach danych w czasie rzeczywistym. Jest to bardzo wydajne rozwiązanie, ponieważ wszystkie dane wykorzystywane do obliczeń są trzymane w pamięci RAM. Apache Ignite składa się z następujących funkcjonalności [11]:

- Data Grid – do przechowywania danych wykorzystuje pamięć RAM w odróżnieniu do tradycyjnych baz danych które wykorzystują dysk twardy jako główne miejsce przechowywania danych. Wspiera system rozproszonej pamięci podręcznej (JCache zgodne z JSR-107), rozproszone transakcję oraz rozproszone zapytania SQL, wspiera JDBC dla potrzeb korzystania z bazy danych.

- Compute Grid – dostarcza możliwość równoległego i rozproszonego przetwarzania zadań. Wspiera dynamiczne zmiany w sieci obliczeniowej, harmonogramowanie zadań, adaptacyjne zarządzanie obciążeniem

- Service Grid – system tworzenia usług. Pozwala określić jak dużo instancji serwisu ma być uruchomionych. Zapewnia że serwisy będą

zawsze dostępne, nawet w przypadku wystąpienia błędu w jednostce obliczeniowej.

- Streaming – system obsługi zdarzeń pozwalający na przetwarzanie danych w trybie ciągłym. Dostarcza on możliwość sprawdzenia jakie wartości były przetwarzane w danym czasie dzięki rozproszonym zapytaniom

- Hadoop Acceleration – pozwala na przyspieszenie wykorzystanej już platformy Hadoop poprzez szybsze dostarczanie danych. Składa się z rozproszonego systemu plików w pełni zgodnego z Hadoop HDFS oraz zaimplementowanego zoptymalizowanego MapReduce'a do korzystania z dyskowego HDFS. Pozwala na wzrost wydajności do 100x. Dostarcza także opcjonalnej warstwy cache'ującej dla HDFS.

- Advanced Clustering – pozwala na zaawansowane zarządzanie jednostkami obliczeniowymi oraz grupowaniem ich w klastry. Wspiera automatycznie wyszukiwanie innych jednostek obliczeniowych, brak konieczności dodawania bibliotek do classpath dzięki rozproszonemu classloader'owi, monitorowanie stanu klastra jak i każdej jednostki obliczeniowej

- Distributed File System – jest to system zbliżony do Hadoop HDFS pozwalający na przechowywanie i dostęp do plików zapisanych w rozproszonym cache'u. Apache Ignite domyślnie dzieli duże pliki na bloki i zapisuje na wielu wybranych jednostkach obliczeniowych tak, by były one łatwo dostępne.

- Distributed Messaging – dostarcza funkcjonalności wiadomości i wymiany danych zgodnego z modelami komunikacyjnymi publish-subscribe oraz point-to-point.

- Distributed Events – dostarcza możliwości odbierania powiadomień o wszystkich zdarzeniach, jakie mają miejsce w cache'u czy jednostkach obliczeniowych jak np. zakończone przetwarzanie zadania, dodanie zadania itp. Aby zaoszczędzić na ruchu sieciowym są one pakowane i wysyłane grupami zgodnie z wybranym interwałem czasowym.

- Distributed Data Structures – Apache Ignite pozwala na zapisywanie struktur z pakietu java.util.concurrent w systemie rozproszonym i korzystanie z nich na różnych jednostkach obliczeniowych jak ze struktury jednocześnie nie przesyłając jej całej.

Apache Ignite był początkowo rozwijany jako projekt GridGain, który został podzielony na dwie części. W wersji podstawowej i bezpłatnej jako open source został udostępniony pod nazwą Apache Ignite. Druga wersja jest aplikacją przygotowaną dla rozwiązań z grupy enterprise. Wspiera dodatkowo takie rozwiązania jak Rolling Updates czy ochrona przed błędami sieci oraz zwiększa bezpieczeństwo danych jak i całego rozwiązania. Dostarcza również aplikację z graficznym interfejsem umożliwiającą wygodne zarządzanie wszystkimi elementami sieci obliczeniowej.

Podsumowując, jest to bardzo rozbudowane rozwiązanie, które wspiera wiele nowoczesnych modeli komunikacji jak i upraszcza tworzenie rozproszonych systemów.

Decydując się na wybór tego frameworka spośród wielu innych dostępnych, kierowano się łatwością nauki. Jest to jedno z najbardziej rozbudowanych rozwiązań do budowania systemów rozproszonych. Pomimo swej złożoności, udostępnia ono prosty i dość przystępny interfejs aplikacji, co pozwala na wykonanie pierwszych testów i zapoznanie się z możliwościami tego rozwiązania bez konieczności wykorzystywania skomplikowanej architektury. Wraz z implementacją faktycznej aplikacji realizującej zagadnienia poruszane w tej pracy, płynnie odkrywano go w coraz większym stopniu poprzez wykorzystywanie bardziej zaawansowanych funkcjonalności oraz przede wszystkim wygodę w wykorzystywaniu ich.

6. Środowisko rozproszone

6.1. Konfiguracja jednostek obliczeniowych

Każda z jednostek obliczeniowych była maszyną o następujących parametrach:

- Procesor: vCPU@2.5-3.3GHz
- Pamięć RAM: 1GB
- Dysk: 8GB SSD
- Karta sieciowa: 100Mbit
- System operacyjny: Ubuntu 14.04 64bit

Na każdej maszynie zostało zainstalowane OpenJDK oraz git. OpenJDK to otwarta platforma dostarczającej wirtualnej maszyny javy potrzebnej do uruchomienia Apache Ignite. Natomiast gita wykorzystywano do pobierania gotowej paczki zawierającej skonfigurowanego Apache Ignite oraz RapidMinera. Przed uruchomieniem Apache Ignite każda maszyna musiała najpierw połączyć się z VPN. Było to konieczne ze względu na sposób przydzielania adresów przez dostawcę maszyn. Za każdym razem otrzymywały one inny adres, co wymusiłoby rekonfigurację całego rozproszonego systemu lub poszukiwanie instancji Apache Ignite na tysiącach wirtualnych maszyn. Dla utrzymania stabilności połączenia, MTU zostało obniżone do 1400.

6.2. Konfiguracja sieci obliczeniowej

Konfiguracja sieci obliczeniowej Apache Ignite opiera się, w najwygodniejszej i najprostszej wersji, na uruchomieniu instancji Apache Ignite przekazując w parametrze lokalizację pliku z konfiguracją zapisaną w pliku XML zgodną ze schemą frameworka Spring.

Na poniższym listingu pokazano jak stworzyć pustą konfigurację dla Ignite. Plik rozpoczyna się od określenia wersji xml i sposobu kodowania. Następnie są podane adresy, pod którymi znajdziemy plik xsd na podstawie którego będą tworzone obiekty javowe.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="grid.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">
  </bean>
</beans>
```

Listing 1 Pusta konfiguracja Apache Ignite w xml

Następnie do beana grid.cfg który będzie instancją klasy IgniteConfiguration zostały dodane kolejne parametry, które będą wykorzystane przy jej tworzeniu. Na początku została dodana konfiguracja cache wykorzystywanego w aplikacji. W konfiguracji zostały określone 2 cache o takiej samej konfiguracji i innych nazwach. Każdy z nich jest partycjonowany pomiędzy wszystkie jednostki obliczeniowe, które mają go w konfiguracji. Dodatkowo wszystkie operacje mają być atomowe.

```
    <!-- Cache configurations. -->
    <property name="cacheConfiguration">
      <list>
        <!--Replicated cache configuration.-->
        <bean class="org.apache.ignite.configuration.CacheConfiguration">
          <property name="name" value="dane"/>
          <property name="cacheMode" value="PARTITIONED"/>
          <property name="atomicityMode" value="ATOMIC"/>
        </bean>
        <bean class="org.apache.ignite.configuration.CacheConfiguration">
          <property name="name" value="result"/>
          <property name="cacheMode" value="PARTITIONED"/>
          <property name="atomicityMode" value="ATOMIC"/>
        </bean>
      </list>
    </property>
```

Listing 2 Konfiguracja Cache

Kolejnym ustawieniem było określenie maksymalnej liczby zadań oraz rodzaju kolejki. Została wybrana kolejka FIFO oraz ustalono, że każda jednostka obliczeniowa może wykonywać maksymalnie 10 zadań. Maksymalną liczbę zadań dobrze by było określać dla każdej jednostki obliczeniowej z osobna w zależności od jej zasobów.

```
    <property name="collisionSpi">
      <bean class="org.apache.ignite.spi.collision.fifoqueue.FifoQueueCollisionSpi">
        <property name="parallelJobsNumber" value="10"/>
      </bean>
    </property>
```

Listing 3 Definicja kolejki i maksymalnej liczby zadań

Właściwość discoverySpi odpowiada za sposób wyszukiwania innych jednostek obliczeniowych. W tym wypadku jednostka obliczeniowa będzie szukała uruchomionych instancji Apache Ignite'a na jednostkach obliczeniowych pod adresami podanymi na liście. W tej konfiguracji są to adresy przypisywane w skonfigurowanej sieci VPN.

```
<property name="discoverySpi">
  <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
    <property name="ipFinder">
      <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
        <property name="addresses">
          <list>
            <value>192.168.100.1</value>
            <value>192.168.100.100</value>
            <value>192.168.100.101</value>
            <value>192.168.100.102</value>
            <value>192.168.100.103</value>
            <value>192.168.100.104</value>
            <value>192.168.100.105</value>
            <value>192.168.100.106</value>
            <value>192.168.100.107</value>
            <value>192.168.100.108</value>
            <value>192.168.100.109</value>
            <value>127.0.0.1</value>
          </list>
        </property>
      </bean>
    </property>
  </bean>
</property>
```

Listing 4 Definicja adresów jednostek obliczeniowych

Ostatnią właściwością jest userAttributes, która przechowuje wartości użytkownika. Są one pomocne do kierowania zadań dla określonej grupy jednostek obliczeniowych. W tej aplikacji wszystkie zadania są kierowane do jednostek obliczeniowych z atrybutem ROLE=worker. Dzięki temu odróżniane są jednostki obliczeniowe od innych uczestników sieci Apache Ignite takich jak np. aplikacje monitorujące stan jednostek obliczeniowych czy aplikacje zlecające zadania.

```
<property name="userAttributes">
  <map>
    <entry key="ROLE" value="worker"/>
  </map>
</property>
```

Listing 5 Definicja własnego atrybutu

7. Aplikacja

7.1. Opis aplikacji

W ramach pracy magisterskiej został stworzony dodatek do aplikacji RapidMiner. Dostarcza on nowych operatorów, które pozwalają na przetworzenie zadania w rozproszonej sieci Apache Ignite. Głównym elementem jest operator IgniteJobManager. Jego zadaniem jest podłączenie się do sieci obliczeniowej oraz stworzenie zbioru narzędzi, z którego będą korzystać inne operatory tego dodatku. Dodatkowo zostały stworzone dwa operatory:

- Parallel Loop – zamiast wykonywać w pętli subproces i zbierać wyniki z każdego przebiegu, tworzy zadania dla jednostek obliczeniowych i rozsyła wszystkie zadania na raz do sieci obliczeniowej, a potem zbiera wszystkie wyniki.
- Parallel Subprocesses – dla każdego zdefiniowanego subprocesu tworzy zadania i tak, jak w przypadku operatora „Parallel Loop" wysyła wszystkie do sieci obliczeniowej a potem zbiera wyniki.

Dodatek został tak zaprojektowany, by można było stworzyć zadanie, które wewnątrz siebie będzie tworzyć kolejne podczas przetwarzania zadania przez RapidMiner na jednostce obliczeniowej. Tutaj jednak warto wziąć po uwagę fakt, by tworzony proces w RapidMinerze nie przekroczył możliwości sieci obliczeniowej i nie doprowadził do deadlocka poprzez wewnętrze tworzenie kolejnych zadań.

Stworzone operatory z grupy „Universal Blocks" są tworzone tak, by mogły być wykonywane niezależnie od środowiska, na którym zostaną oparte. Dlatego aby wyspecyfikować w jakim środowisku zostaną uruchomione należy je umieszczać wewnątrz operatorów definiujących środowisko oraz dostarczających koniecznych metod do wykonania procesu. Przykładem takiego operatora jest *IgniteJobManager*.

7.2. Model logiczny

W modelu logicznym możemy wydzielić dwie najważniejsze klasy abstrakcyjne:

- *AbstractJobManager* – implementacja tej klasy pozwala na stworzenie operatora specyficznego dla każdego środowiska rozproszonego.
- *AbstractJobManagerHelper* – implementacja tej klasy jest ważna, ponieważ to ona zawiera metody, które są wykorzystywane w operatorach do tworzenia zadań dla rozproszonej sieci obliczeniowej, pobierania i udostępniania danych innym jednostkom obliczeniowym, czy przetwarzania odpowiedzi z jednostek obliczeniowych na faktyczne obiekty zawierające dane.

Dla frameworka Apache Ignite zostały stworzone implementacje tych klas abstrakcyjnych. Dzięki takiej strukturze zastąpienie tego frameworka innym opiera się tylko na implementacji dwóch klas, które zawierają komplet metod wykorzystywanych w całym procesie. Można powiedzieć, że same operatory wykonujące zadanie są niezależne względem frameworka je obsługującego.

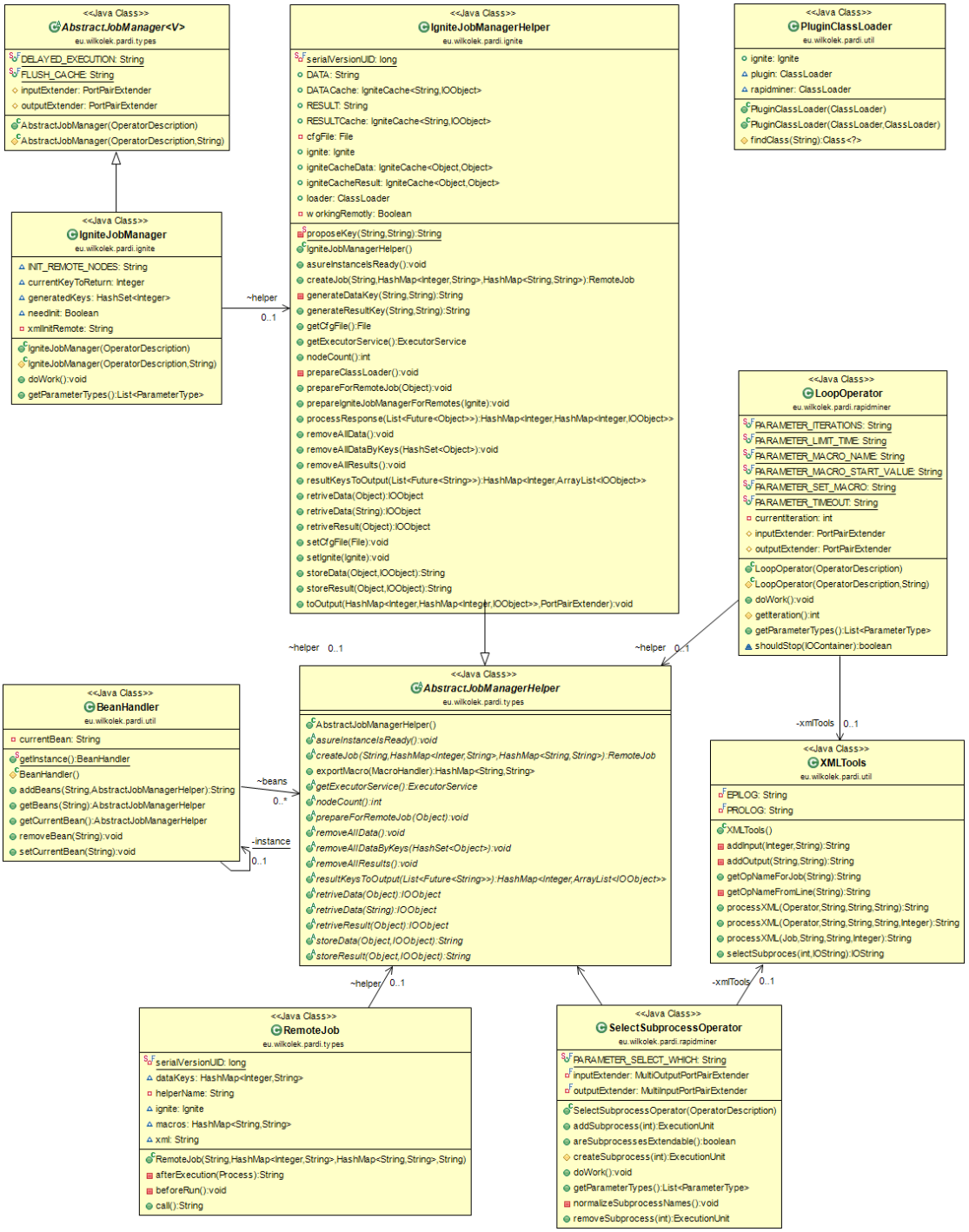
Kolejnymi elementem jest singleton *BeanHandler*, który trzyma mapę obiektów AbstractJobManagerHelper, co pozwala na wykorzystanie różnych implementacji wewnątrz jednego procesu RapidMinera.

Każdy operator korzystając z tego singletona za pomocą metody *getCurrentBean()* może pobrać aktualną instancję *AbstractJobManagerHelpera* i korzystać z jego metod na potrzeby tworzenia zadań dla środowiska rozproszonego. Każda z klas operatorów korzysta także z klasy *XMLTools*, która zawiera metody do przetworzenia procesu zapisanego w postaci dokumentu xml na zadania które zostaną wykonane na poszczególnych jednostkach obliczeniowych. Operatory wykorzystują klasę *RemoteJob* jako nośnik informacji o procesie do wykonania jak i danych które muszą zostać użyte.

Jednym z problemów, które pojawiły się podczas tworzenia dodatku, był brak możliwości uruchomienia instancji Apache Ignite korzystając z konfiguracji zapisanej w pliku xml. Problem wynikał z implementacji RapidMinera i dedykowania osobnych class loaderów dla każdego dodatku. Z racji tego, że

domyślnym class loaderem dla wirtualnej maszyny javy był class loader RapidMinera, to nie miał on pojęcia o klasach dołączanych przez stworzony dodatek. Problem można było rozwiązać na dwa sposoby. Pierwszym było porzucenie konfiguracji w formie xml i zastąpienie jej niewygodną zapisaną w języku java lub zaimplementowanie własnego class loadera. Finalnie został stworzony class loader, dzięki któremu cała konfiguracja Apache Ignite nie została zaszyta w skompilowanym kodzie, a pozostała czytelna i dostępna dla użytkownika.

Klasa *PluginClassLoader* implementuje omawiane rozwiązanie, zastępując domyślny class loader wirtualnej maszyny javy, jednocześnie rozbudowując go o możliwość korzystania z klas zawartych w dodatku.



Obraz 3 Diagram klas

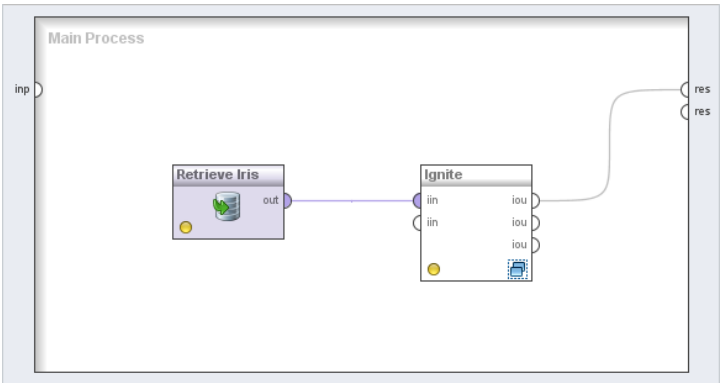
7.3. Instalacja oraz tworzenie procesu dla środowiska rozproszonego

Tworzenie procesu, który będzie przetwarzany w środowisku rozproszonym jest bardzo łatwe. Zanim jednak będzie to możliwe należy zainstalować dodatek poprzez umieszczenie go w katalogu rapidminer/lib/plugins. Następnie należy do katalogu rapidminer/libs wkopiować wszystkie potrzebne pliki jar konkretnej wersji Apache Ignite tj.:

- ignite-core-1.0.0.jar
- cache-api-1.0.0.jar
- commons-logging-1.1.1.jar
- ignite-spring-1.0.0.jar
- spring-aop-4.1.0.RELEASE.jar
- spring-beans-4.1.0.RELEASE.jar
- spring-context-4.1.0.RELEASE.jar
- spring-core-4.1.0.RELEASE.jar
- spring-expression-4.1.0.RELEASE.jar
- spring-tx-4.1.0.RELEASE.jar

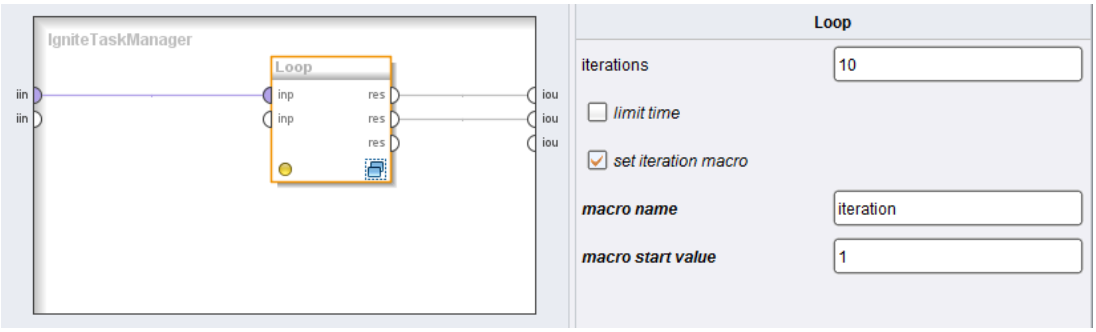
Aby wskazać, z jakiej konfiguracji ma korzystać Apache Ignite należy uruchomić RapidMiner z parametrem -Dignite.configuration.file="plik.xml".

Tak uruchomiony RapidMiner powinien zawierać nową gałąź operatorów *Parallel Distributed Jobs*. Na samym początku należy wybrać operator, który będzie definiował z jakiej implementacji *AbstractJobManagera* chcemy korzystać. W tym wypadku dostępny jest tylko jeden: *IgniteJobManager*.

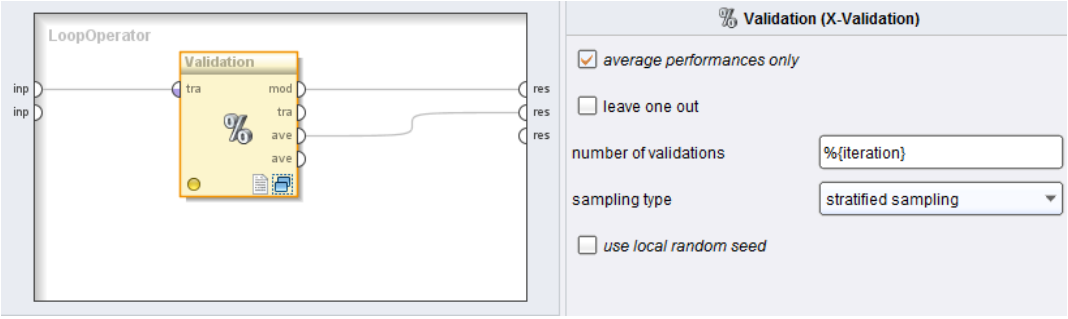


Obraz 4 Inicjalizacja IgniteJobManagera

Wewnątrz niego możemy już korzystać z dowolnych operatorów zawartych w gałęzi „Universal Blocks". Operator *Parallel Loop* z tego dodatku zawiera dokładnie te same parametry, jakie zawiera operator Loop dostarczony przez RapidMinera, różnica jest tylko w sposobie wykonania obliczeń. W przypadku operatora *Parallel Subprocesses* został usunięty parametr odpowiedzialny za wybór subprocesu do wykonania, ponieważ wszystkie subprocesy będą wykonane w jednym przebiegu. Tak jak już było wspomniane operatory dla obliczeń rozproszonych mogą się zagnieżdżać, co będzie skutkowało tworzeniem kolejnych zadań przez jednostki obliczeniowe.



Obraz 5 Użycie operatora Parallel Loop do obliczeń rozproszonych



Obraz 6 Przykładowy subproces operatora Parallel Loop

7.4. Opis działania dodatku

Opisując sposób działania dodatku będę się posługiwał procesem stworzonym i zaprezentowanym na obrazach 3-5. Na samym początku wczytywany jest zbiór danych i przekazywany do operatora *IgniteJobManager*. Jeżeli instancja *IgniteJobManagerHelpera* istnieje już w *BeanHandlerze*, to jest pobierana jego referencja, a w przeciwnym wypadku jest tworzona nowa instancja. Potem uruchamiana jest metoda *asureInstanceIsReady(),* która sprawdza czy Apache Ignite jest uruchomiony i jeśli jest to konieczne to jest on uruchamiany. Następnie w *BeanHandlerze* stworzony *IgniteJobManagerHelper* ustawiany jest jako aktywny. Kolejnym etapem jest przekazanie danych do subprocesu, wykonanie go i przekazanie obiektów z wyjścia subprocesu na wyjście operatora *IgniteJobManager*. Po przekazaniu danych na wyjście czyszczony jest cache, w którym były przechowywane obiekty klasy *IOObject*. Dodatkowo w tym operatorze jest też obsługa wszystkich błędów, które mogą wystąpić w subprocesach. W przypadku błędów zamykany jest Apache Ignite.

Podczas wywoływania podprocesu uruchamiany jest operator *Parallel Loop*, który z pomocą klasy implementującej *AbstractJobManagerHelper* pobranej z *BeanHandlera* pobiera dane z wejścia i zapisuje w globalnej pamięci podręcznej systemu Ignite o nazwie dane. W kolejnym kroku wykonuje kopię wszystkich aktualnie ustawionych parametrów procesu oraz z pomocą klasy *XMLTools* tworzy dokument xml, z którego będzie tworzony proces dla RapidMinera na jednostce obliczeniowej. Następnie tworzona jest lista zadań składająca się z obiektów *RemoteJob* zawierających:

- dokument xml z procesem,
- listę parametrów procesu dla zadania,
- listę kluczy pod którymi są zapisane obiekty z danymi dla procesu
- nazwę klasy implementującej AbstractJobManagerHelpera, która była wykorzystana w procesie tworzenia zadania

Dodawana nazwa AbstractJobManagerHelpera jest potrzebna jednostce obliczeniowej, żeby ta wiedziała z jakiej implementacji ma skorzystać do

przetworzenia tego zadania tj. jak pobierać dane na podstawie kluczy oraz jak odpowiadać i je zapisywać.

Tak stworzona lista zadań jest przekazywana do obliczenia przez jednostki obliczeniowe zawierające atrybut ROLE=worker. Gdy zadanie jest uruchomione na jednostce obliczeniowej, następuje pobranie referencji do uruchomionej instancji Ignite za pomocą adnotacji *@IgniteInstanceResource*. Potem następuje stworzenie obiektu implementującego klasę *AbstractJobManagerHelper* na podstawie przekazanej nazwy. Następnie za pomocą przekazanego dokumentu xml budowany jest obiekt procesu dla RapidMinera oraz pobierane są wszystkie obiekty z cache do których zostały przekazane klucze. W kolejnym kroku następuje sprawdzenie czy RapidMiner jest uruchomiony, w przypadku gdy nie jest następuje uruchomienie go. Mając wszystkie dane gotowe wykonujemy proces podając za parametry listę parametrów procesu, oraz dane wstępne do procesu. Po wykonaniu procesu wszystkie obiekty wyjściowe są wysyłane do cache oraz zapisywane są ich klucze. Wynikiem całego zadania jest lista kluczy. Operator który stworzył te zadania dostaje w odpowiedzi kolekcję list kluczy. Następnie w kolejności w jakiej otrzymał zestawy kluczy, pobiera wszystkie obiekty z cache. Na koniec całego procesu pakuje obiekty w kolekcje i przekazuje na odpowiednie wyjścia w RapidMinerze. Cały proces został przedstawiony na poniższym diagramie.

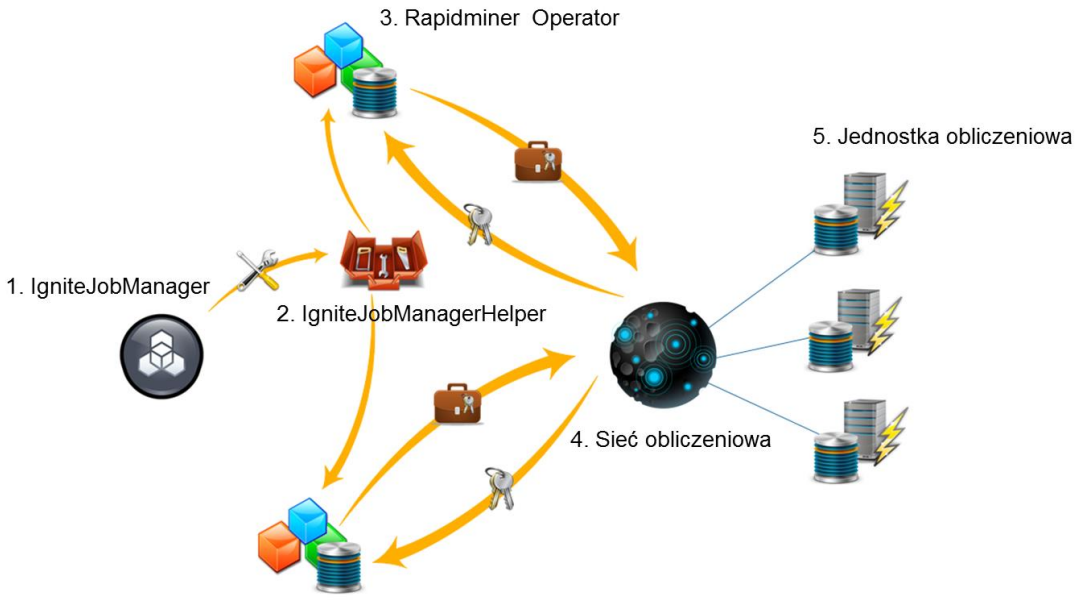


Diagram 1 Przepływ danych

8. Testowanie i analiza wyników

8.1. Opis wykonanych testów

Ważnym elementem każdego projektu programistycznego jest wykonanie niezbędnych testów, by móc ocenić wydajność i poprawność rozwiązania. W tym celu zostały zaprojektowane 3 testy. Pierwszym z nich jest wykonanie dużej ilości obliczeń, które łatwo można podzielić na zadania i przetestować wydajność rozwiązania. W drugim sprawdzane jest czy wydajność procesu obliczeniowego jest zależna od wielkości danych wykorzystywanych w procesie. W trzecim teście sprawdzane jest czy rozwiązanie to równie dobrze może być wykorzystywane w celu wykonywania małych procesów, które niekoniecznie musiałyby być wykonywane w środowisku rozproszonym. Pozwoli to ocenić jaki narzut czasowy jest związany z korzystaniem z tego rozwiązania w porównaniu do wykonania tych obliczeń na maszynie lokalnej.

8.2. Test wydajności dla zadania łatwo podzielnego z wykorzystaniem niewielkiej ilości danych

8.2.1. Opis testu

W tym teście wykorzystano zbiór danych Iris. Jest to zbiór dostępny w repozytorium UCI Machine Learning Repository. Opisuje on 3 gatunki kwiatów irysa z każdego po 50 sztuk. Dwa z nich są liniowo separowalne i opisują je 4 parametry:

- Długość kielicha w cm
- Szerokość kielicha w cm
- Długość płatka w cm
- Szerokość płatka w cm

W tym procesie została wykonana klasyfikacja za pomocą sztucznej sieci neuronowej. Badane były sieci neuronowe składające się z 1 warstwy, którą przetestowano w różnych konfiguracjach tj. począwszy od 1 neuronu do 20 neuronów w warstwie. Każde niezależne zadanie składało się z jednej sieci neuronowej do zbadania.

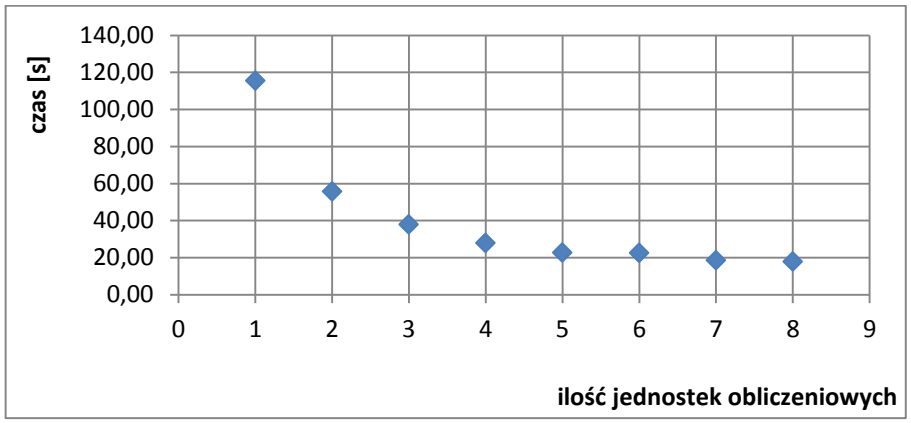
8.2.2. Wyniki

Czas potrzebny maszynie lokalnej: 87s (czas średni z 5 prób)

ilość JO	1	2	3	4	5	6	7	8
nr próby	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]
1	143,72	55,93	38,29	31,18	21,44	20,79	18,97	17,09
2	103,74	56,11	35,73	29,59	23,99	19,95	17,73	18,84
3	104,40	54,06	40,23	26,30	23,69	28,66	19,71	17,85
4	118,22	62,29	39,98	25,87	22,20	21,32	17,90	17,60
5	107,32	50,03	35,61	26,20	21,92	21,59	18,48	17,86
średnia	115,48	55,68	37,97	27,83	22,65	22,46	18,56	17,85

Tabela 1 Wyniki z pierwszego testu

Wykres czasu w zależności od użytych ilości jednostek obliczeniowych



Wykres 1 Prezentacja danych z pierwszego testu

8.2.3. Analiza wyników

Analizując wyniki, można zauważyć że, czas potrzebny na otrzymanie wyników spadł o 79,48% w porównaniu z obliczeniami prowadzonymi jedynie na lokalnej maszynie. Przy 6 jednostkach obliczeniowych średni czas obliczeń wzrósł. Jest to spowodowane trzecim pomiarem, w którym najprawdopodobniej jeden z systemów był bardziej obciążony samymi zadaniami systemowymi lub miał problemy z przesłaniem danych. Poza technicznymi problemami mogło tam dojść do nierównomiernego rozłożenia zadań. Bazując na wykonanym teście wystarczy, że jedna z jednostek obliczeniowych nie zdążyła skończyć swojego zadania nr 2, a inna jednostka pobrała 4 zadanie dla siebie. W wyniku tego, wszystkie jednostki obliczeniowe w sieci muszą czekać aż ta jedna jednostka policzy to ostatnie zadanie.

8.3. Test wydajności dla zadania łatwo podzielnego z wykorzystaniem dużej ilości danych

8.3.1. Opis testu

Analogicznie jak w teście nr.1 wykonana była klasyfikacja za pomocą sieci neuronowej o takiej samej konfiguracji. Jednak ze względu na zastosowany zbiór danych została zmniejszona liczba walidacji i cykli treningowych do wykonania, aby zadanie było obliczalne w krótszym czasie. W tym przypadku zastosowano zbiór Spambase opisujący pocztę elektroniczną oznaczoną jako niechciana (spam) oraz chcianą.. Zbiór danych składa się z 4601 rekordów. Każdy z rekordów opisany jest przez 57 atrybutów reprezentujących częstość występowania wybranych wyrazów.

Pojedynczy rekord pliku SpamBase składa się z następujących atrybutów:

- 48 atrybutów opisujących częstość występowania wybranego słowa w mailu w stosunku do ilości wszystkich słów
- 6 atrybutów opisujących częstość występowania wybranych znaków
- Atrybut zawierający maksymalną długość najdłuższego nieprzerwanego ciągu wielkich liter
- Atrybut zawierający średnią długość najdłuższego nieprzerwanego ciągu wielkich liter
- Atrybut zawierający sumę znaków wszystkich nieprzerwanych ciągów wielkich liter

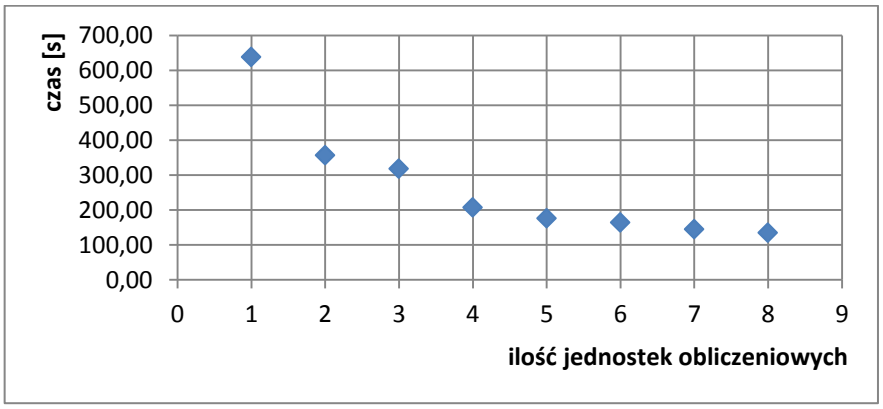
8.3.2. Wyniki

Czas potrzebny maszynie lokalnej: 648s (czas średni z 5 prób)

ilość JO	1	2	3	4	5	6	7	8
nr próby	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]	czas [s]
1	662,58	382,04	306,65	207,53	182,01	156,56	137,34	134,68
2	634,18	333,83	322,86	217,53	169,44	160,14	145,20	133,21
3	613,28	365,70	275,93	205,62	179,18	188,32	138,93	137,88
4	662,51	330,23	338,68	194,51	176,39	157,19	144,51	132,97
5	616,14	367,36	344,85	209,09	169,89	157,81	157,87	132,91
średnia	637,74	355,83	317,79	206,86	175,38	164,00	144,77	134,33

Tabela 2 Dane z drugiego testu

Wykres czasu w zależności od użytych ilości jednostek obliczeniowych



Wykres 2 Prezentacja danych z drugiego testu

Wyniki otrzymane podczas spowalniania jednej jednostki obliczeniowej

średnie zużycie procesora przez Ignite zawieszającej się jednostki [%]	100	95	80	50	25
czas [s]	186,90	185,76	243,39	582,90	1030,90

Tabela 3 Wyniki z przebiegu ze spowalniającą jednostką obliczeniową

Wykres czasu potrzebnego na otrzymanie wyników w zależności
od wykorzystania procesora przez najsłabszą jednostkę obliczeniową

Wykres 3 Wzrost czasu obliczeń w zależności od spowolnienia pracy jednej jednostki obliczeniowej

8.3.3. Analiza wyników

Przyglądając się wynikom tego testu, czas potrzebny na otrzymanie wyników spadł o 79,27% względem czasu potrzebnego przez maszynę lokalną. Porównując to z wynikami otrzymanymi w poprzednim teście, można zauważyć, że wzrost wydajności jest na podobnym poziomie, pomimo wzrostu poziomu trudności problemu. Zestaw wygenerowanych zadań był różny ze względu na coraz to bardziej rozbudowaną konfigurację sieci neuronowej. W przypadku zadań o równym stopniu trudności, np. budując jeden model predykcyjny i testując go na różnych zbiorach danych o takiej samej skali złożoności, prawdopodobnie wykres byłby bardziej liniowy. W tym procesie różnica pomiędzy najprostszym a najtrudniejszym zadaniem jest znacznie większa, ponieważ złożoność architektury sieci neuronowej jest 20-krotnie większa.

W tym teście, jak zostało wspomniane, konieczne było zmniejszenie ilości walidacji z 10 do 3 oraz zmniejszenie cykli treningowych z 1000 do 250. Powodem tego były bardzo czasochłonne testy, które często się nie powodziły ze względu na parametry jednostek obliczeniowych. Na wykresie 3 widać jak zmienia się czas obliczeń, gdy doszło do zawieszenia się pojedynczej jednostki obliczeniowej. Pomimo, że te wyniki nie obrazują korzyści z korzystania z tego rozwiązania, to w przypadku występujących problemów pokazują ich skalę. Obniżenie wydajności jednej jednostki do 25% powoduje ponad 5-krotny wzrost

czasu obliczeń. Aby zapobiegać takim problemom, należałoby ściśle monitorować stan jednostki obliczeniowej oraz automatycznie wykluczać je z sieci obliczeniowej na czas naprawy. Zbyt duże procesy są za trudne do przetworzenia ze względu na ilość dostępnej pamięci RAM oraz sposób dystrybucji tych jednostek obliczeniowych. Nie są to maszyny typu barebone, a wydzielone instancje z rozproszonego systemu. Możliwe jest otrzymanie odpowiednio mocnej maszyny z zabezpieczeniem, że jej wszystkie parametry będą stabilne przez cały czas korzystania. Jest to drogie rozwiązanie, ale dla organizacji mogą to być wciąż niższe koszty niż utrzymywanie własnej infrastruktury.

8.4. Test wydajności na zadaniu niepodzielnym z użyciem różnej wielkości danych

8.4.1. Opis testu

W tym teście został wykonany proces na jednostce obliczeniowej, który nie będzie zależny od wielkości przesyłanych danych, tak by móc jednoznacznie określić czas potrzebny na przesył danych.

Test zostanie wykonany w trzech wariantach:

- bez przesyłania danych
- z przesyłaniem zbioru danych Iris
- z przesyłaniem zbioru danych Spambase

Dodatkowo każdy z tych wariantów będzie wykonany wraz z czasem potrzebnym na podłączenie do sieci jak i z wykorzystaniem istniejącego połączenia.

8.4.2. Wyniki

	czas całego procesu [s]			czas procesu bez podłączania do sieci obl. [s]		
przesyłany zbiór danych	Iris	Spambase	Brak	Iris	Spambase	Brak
1	90	128	151	57	72	57
2	129	128	103	57	70	56
3	136	154	87	57	71	56
4	121	151	116	57	71	56
5	91	149	82	57	72	55
średni czas	113,4	142	107,8	57	71,2	56

Tabela 4 Wyniki z trzeciego testu

Wyznaczana wartość	czas[s]
średni czas procesu z połączeniem	121,07
średni czas procesu bez połączenia	61,4
średni czas potrzebny na połączenie	59,67
średni czas potrzebny na przesłanie bazy Iris	1
średni czas potrzebny na przesłanie bazy Spambase	15,2

Tabela 5 Wyznaczone wielkości na postawie wyników z testu trzeciego

8.4.3. Analiza wyników

W teście trzecim bazując na wszystkich wynikach można określić, że w tych warunkach średni czas na połączenie wynosi niecałą minutę. Dla niektórych procesów może to stanowić większą część czasu, aniżeli sam proces obliczeniowy. Jednocześnie można zauważyć, że przesłanie pliku o rozmiarach 1,4MB (przesłanie zbioru Spambase w obie strony) zajmuje ok 15s. Może to być spowodowane ustawieniem najwyższego poziomu atomowości operacji, a co się z tym wiąże najwyższego narzutu czasowego związanego z weryfikacją spójności danych. W przypadku bazy Iris czas przesyłu jest pomijalny. Jednak ze względu na rozmiar danych sprawdzenie spójności pomiędzy dwoma jednostkami obliczeniowymi przebiegło znacznie sprawniej. Przy konfiguracji sieci VPN zostało określone MTU na wartość 1400 (maksymalny datagram może wynosić 1400B). Dodatkowo wszystko jest tunelowane i wysyłane przez serwer pośredniczący (serwer VPN), co może dodatkowo zwiększać czas przesyłu.

8.5. Podsumowanie przeprowadzonych testów

Z przeprowadzonych testów jasno wynika, że wykorzystanie rozproszonych obliczeń w procesach inteligencji obliczeniowej jest przydatne i widocznie może zmniejszyć czas potrzebny na otrzymanie wyników. Wykorzystanie takiego rozwiązania wiąże się z dodatkowym czasem potrzebnym na uruchamianie odpowiedniej liczby instancji i utrzymywaniem ich. Podczas przeprowadzanych testów niejednokrotnie konieczne było uruchomienie oraz konfiguracja jednostki obliczeniowej od początku. Z racji tego, że cały system był połączony przez serwer VPN, który był w innej lokalizacji niż wszystkie maszyny, wymagane było łączenie się z nim poprzez Internet. Ma to swoje wady. Niejednokrotnie połączenia były zrywane, co powodowało znaczny wzrost czasu potrzebnego na wykonanie obliczeń. Wynikało to z faktu, że zadania, które już zostały przetworzone albo były w trakcie przetwarzania, trzeba było przetworzyć jeszcze raz. Apache Ignite sam sobie z tym radzi więc nie wymagało to dodatkowego czasu potrzebnego na implementację takiego rozwiązania. W przypadku większych problemów sieciowych całe obliczenia rozproszone stają się nieefektywne. Biorąc pod uwagę długo trwające obliczenia np. 5 dni, źle działająca sieć będzie potrafiła skutecznie uniemożliwić wykonanie procesu w całości oraz otrzymanie wyników. Problemy z siecią to nie jedyne jakie mogą wystąpić podczas procesu przetwarzania zadań. Dodatkowym problemem może być źle działająca jedna z maszyn. W trakcie testowania rozwiązania, często dochodziło do zapchania maszyny ilością procesów, co skutkowało posiadaniem bardzo wolnej maszyny, która nie radziła sobie z obciążeniem. Wynikało to z dwóch faktów. Po pierwsze, w systemie operacyjnym było uruchomione wiele niepotrzebnych usług. Maszyny testowe nie były najsilniejsze, co tylko uwidoczniło problem jakim jest poprawna optymalizacja samego systemu operacyjnego. Po drugie, źle skonfigurowana sieć obliczeniowa może doprowadzić do zatykania się całej jednostki obliczeniowej. Pozwalając danej jednostce obliczać 100 zadań na raz, wiedząc równocześnie, że ma ona jeden procesor jedno-rdzeniowy doprowadza do sytuacji, w której 100 zadań jest zabranych z kolejki, a żadne z nich nie jest efektywnie przetwarzane.

Rozpraszając proces obliczeniowy na wszystkie jednostki obliczeniowe w sieci, musimy liczyć się z tym, że czas potrzebny na wykonanie wszystkich zadań jest w głównej mierze zależny od wydajności najsłabszego ogniwa całej infrastruktury.

Abstrahując od infrastruktury i wszelakich rozwiązań systemowych, równie ważna jest dobra implementacja rozwiązania w postaci, w tym wypadku, operatorów. Jeżeli operatory nie będą usuwały niewykorzystywanych już zasobów, może to powodować zatrzymywanie się kolejnych jednostek obliczeniowych ze względu na za małą przydzieloną pamięć. Bardzo ważnym elementem jest odpowiednie zarządzanie pamięcią i przechowywanymi obiektami, aby nie dopuszczać do sytuacji, w której przechowywane są dane w pliku stronicowania, czy pliku wymiany ze względu na za dużą przydzieloną pamięć – co bardzo spowalnia procesy obliczeniowe, czy za małą pamięć przydzieloną niezdolną do przechowywania chociażby wyników obliczeń.

Generując bardzo dużą ilość wyników powodowany jest bardzo duży ruch w sieci. W takich przypadkach warto by było nie czekać aż wyniki zostaną przesłane na maszynę, z której został uruchomiony proces. Lepszym rozwiązaniem byłoby zapisywanie ich w bazach danych, czy przeznaczonych do tego miejscach, które mają bardzo dobre połączenie – najlepiej bezpośrednie – z maszyną obliczeniową. W przedsiębiorstwach często wykorzystuje się raz zbudowany model predykcyjny do wykonywania analiz z danego zakresu na przesyłanych danych pojedynczego przypadku. W takich sytuacjach nie będzie dochodziło do przesyłania ogromnych ilości informacji w obie strony, ponieważ odpowiedź od takiego systemu będzie konkretna i niewymagająca przesyłania ogromnych zbiorów danych w celu dobrego nauczenia sieci neuronowej. Po przesłaniu wstępnej dawki wiedzy, która wymagałaby przesłanie dużej ilości danych, sieć neuronowa może się sama szkolić wraz z otrzymywanymi nowymi przypadkami.

9. Wnioski

Stosowanie rozproszonych systemów obliczeniowych ma sens, płynie z tego wiele korzyści. Po pierwsze, znacznie lepiej są wykorzystywane zasoby w ramach organizacji, ponieważ wiele osób może je wykorzystywać w zależności od swoich potrzeb. Dedykując pracę systemowi rozproszonemu, odciążamy lokalną maszynę, co dalej pozwala na wydajną pracę pomimo biegnącego w tle procesu obliczeniowego. Ponadto, otrzymywanie szybciej wyników może się wprost przełożyć na wydajność i zyski firmy. Skracając czas wykonania obliczeń 5 razy, pozwalamy pracownikom na wykonanie większej ilości pracy. Ba, w pewnych przypadkach pozwalamy wypełniać pracownikowi pracę należycie, ponieważ nie spędza on większości czasu na oczekiwaniu na wyniki, by móc poprawić niedoskonałości poprzedniej iteracji. Wykorzystywanie rozproszonych sieci obliczeniowych wiąże się też z pewnymi wadami. Bardzo ważnym elementem jest sieć obliczeniowa, jak i każda jednostka obliczeniowa w niej zawarta. To wymusza na organizacji przeznaczenie dodatkowych środków na zakup oraz utrzymanie odpowiednio wielkiej i rozbudowanej sieci obliczeniowej. W dzisiejszych czasach jest wiele firm oferujących systemy chmurowe. Odpowiednio konfigurując, można je włączać i wyłączać w zależności od potrzeb. Co więcej, płacimy tylko za wykorzystane zasoby (odpowiednio od procenta użycia procesora/pamięci/sieci w czasie), a nie za cały dostarczony system, który mógłby pracować na 100% możliwości.

Metody z grupy inteligencji obliczeniowej są potrzebne i dostarczają metod przetwarzania danych. O tyle, o ile można powiedzieć że są one efektywne, tak odpowiednia konfiguracja i dobór parametrów dla niej jest to proces czasochłonny. Przetwarzanie kilku takich procesów równolegle na pewno jest czymś pożądanym w tej dziedzinie. Zrównoleglenie całych procesów nie jest zadaniem najprostszym ze względu na tworzenie oprogramowania wykonującego go, jednak dostarczającym wymiernych korzyści w postaci zaoszczędzonego czasu. Sprzężenie rozwiązań informatycznych z zagadnieniami z dziedziny inteligencji obliczeniowej jest ciekawym polem do prowadzenia badań pozwalającym rozpatrywanie zagadnienia od strony teoretycznej po faktyczne rozwiązania od strony technicznej. Mogą one

dostarczyć wielu nowych rozwiązań, które w przyszłości mogą zaowocować szybkością i wydajnością tworzonych systemów do przetwarzania danych.

Cel pracy został osiągnięty. Udało się zrównoleglić pewne procesy wykorzystując rozproszoną infrastrukturę. Na badanych przykładach udało się przyspieszyć proces obliczania i otrzymywania wyników pięciokrotnie. Stosując jeszcze bardziej rozbudowane systemy, korzyści mogą być jeszcze większe. W procesach trwających po kilkadziesiąt godzin, skrócenie o 5 razy jest wyraźne widoczne. Dodatkowo maszyna lokalna, z której były wykonywane testy nie była praktycznie w ogóle obciążona, co pozwala na swobodną pracę pomimo prowadzonych obliczeń.

Bibliografia

Spis obrazów

Spis tabel

Spis wykresów

Spis listingów

Spis literatury

[1] Engelbrecht A. P., *Computational Intelligence: An Introduction*; Wydawnictwo WILEY, Wydanie II; 2007r

[2] Grama A, Gupta A, Karypis G., Kumar V., *Introduction to Parallel Computing*, Addison Wesley; 2003r

[3] Hofmann M., Klinkenberg R., *RapidMiner: Data Mining Use Cases and Business Analytics Applications*; Wydawnictwo CRC Press; 2013r

[4] Joseph J., Ernest M., Fellenstein C., *Evolution of grid computing architecture and grid adoption models*, IBM SYSTEMS JOURNAL; 2004r

[5] Juhász Z., Kacsuk P., Kranzlmüller D., *Distributed and Parallel Systems. Cluster and grid computing.*, Springer Science + Business Media, Inc.; 2005r

[6] Marks II R.J., *Intelligence: Computational Versus Artificial* [W:] IEEE Transactions on Neural Networks, tom 4, numer 5, wrzesień 1993

[7] Osowski S., *Sieci neuronowe w ujęciu algorytmicznym*, Wydawnictwa Naukowo-Techniczne, 2013r

[8] Rutkowski L., *Computational Intelligence: Methods and Techniques;* Wydawnictwo naukowe PWN SA; 2005r

[9] Shiffman D., *The nature of code*; 2012r

[10] Sommerville I., *Software Engineering*; 2004r

Spis stron internetowych

[11] http://www.gridgain.com/wp-content/uploads/2015/02/Apache_Ignite_White_Paper.pdf

[12] https://rapidminer.com/wp-content/uploads/2013/10/How-to-Extend-RapidMiner-5.pdf

[13] http://apacheignite.readme.io/v1.0/docs

[14] http://dydaktyka.polsl.pl/kwmimkm/MIO%20wyklad1%20do%20druku.pdf

[15] https://technet.microsoft.com/pl-pl/library/cc739678%28v=ws.10%29.aspx

[16] http://riad.usk.pk.edu.pl/~kbanas/PRR/PRiR_skrypt.pdf

[17] http://www.impan.pl/~zakopane/35/Kosinski.pdf

[18] http://www.ai.c-labtech.net/sn/pod_prakt.html

[19] http://www.ai.c-labtech.net/sn/sneuro.html

[20] https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/ef2b3bf0-86fe-4369-9c18-92bcddcac54b/document/cab990c3-f9ea-4b05-ba90-2811a6ee54cd/media/Mastering New Challenges in Text Analytics.pdf