(19) URZĄD PATENTOWY RZECZYPOSPOLITEJ POLSKIEJ

(10) **PL 246006 B1**

(12) # Opis patentowy

(21) Numer zgłoszenia: **439828**
(22) Data zgłoszenia: **2021.12.14**
(43) Data publikacji o zgłoszeniu: **2023.06.19 BUP 25/2023**
(45) Data publikacji o udzieleniu patentu: **2024.11.18 WUP 47/2024**

(51) MKP:
***G06F 17/40*** *(2006.01)*
***G06N 3/02*** *(2006.01)*

(73) Uprawniony z patentu:
**INFOKLINIKA SPÓŁKA AKCYJNA, Warszawa, PL**

(72) Twórca(-y) wynalazku:
**DZMITRY PADKAPAYEU, Warszawa, PL**
**JANUSZ MIROFORIDIS, Długołęka, PL**
**IGNACY KALISZEWSKI VEL KIELISZEWSKI, Pruszków, PL**
**PRZEMYSŁAW JUSZCZUK, Goląsza Górna, PL**
**GRZEGORZ ŚLADOWSKI, Jerzmanowice, PL**
**MARIUSZ TKACZYK, Raszyn, PL**

(74) Pełnomocnik:
**rzecz. pat. Mariusz Kondrat, Warszawa, PL**

(54) Tytuł:
**System oraz wspomagany komputerowo sposób zarządzania pamięcią masową**

PL 246006 B1

## Opis wynalazku

Przedmiotem wynalazku jest system oraz wspomagany komputerowo sposób zarządzania pamięcią masową.

Ze stanu techniki znanych jest wiele rozwiązań dotyczących przetwarzania danych, zarządzania systemami oraz optymalizacji działania systemów. Na przykład zgłoszenie US2021011860 A1 ujawnia system przetwarzania danych obejmujący urządzenie hosta skonfigurowane do wybierania trybu szybkości przepustowości pamięci w oparciu o model sieci lub wielkość partii, albo oba te czynniki, moc obliczeniową i moc pamięci akceleratora dopasowywaną zgodnie z wyborem trybu szybkości; oraz urządzenie do przechowywania danych zawierające akcelerator skonfigurowany do zmiany struktury macierzy elementów przetwarzających (PE) poprzez sterowanie ścieżkami transmisji pierwszych danych wejściowych i drugich danych wejściowych, które są wprowadzane do macierzy PE w oparciu o tryb prędkości szerokości pasma pamięci, w którym moc pamięci w całkowitym budżecie mocy jest zwiększana od wartości początkowej mocy pamięci, jeśli wybrany jest tryb pierwszej prędkości, a moc obliczeniowa w całkowitym budżecie mocy jest zwiększana od wartości początkowej mocy obliczeniowej, jeśli tryb drugiej prędkości jest zaznaczony.

Zgłoszenie EP3729279 A1 dotyczy sposobu organizowania danych w urządzeniu zunifikowanej pamięci mającym zunifikowany nośnik pamięci i jedną lub więcej jednostek przetwarzających. Ujawniony sposób obejmuje kolejno: konfigurowanie pierwszego modułu pamięci zunifikowanego nośnika pamięci w celu komunikowania się z jedną lub większą liczbą jednostek przetwarzających i zawierających pierwszy zbiór komórek identyfikowanych przez unikalny identyfikator komórki pamięci, które są skonfigurowane do przechowywania danych; konfigurowanie drugiego modułu pamięci zunifikowanego nośnika pamięci w celu komunikowania się z jedną lub większą liczbą jednostek przetwarzających i zawierania drugiego zbioru komórek pamięci, które są skonfigurowane do przechowywania danych, przy czym drugi zbiór komórek pamięci jest identyfikowany przez unikalny identyfikator komórki; oraz konfigurowanie jednostki przetwarzania jednej lub większej liczby jednostek przetwarzania w celu: odbierania pierwszych danych wejściowych z jednej z pierwszych wielu komórek pamięci, odbierania drugich danych wejściowych z jednej z drugich wielu komórek pamięci i generowania danych wyjściowych na podstawie na pierwszych i drugich danych wejściowych.

Zgłoszenie US2017277628 A1 dotyczy urządzenia obliczeniowego do zarządzania pamięcią sieci neuronowej, które zawiera pamięć zawierającą dane o łączności związane z pierwszym neuronem sieci neuronowej, przy czym pierwszy neuron jest związany z wieloma neuronami wejściowymi i wieloma neuronami wyjściowymi, przy czym dane o łączności zawierają sekwencję bitową wskazującą, które z wielu połączeń sieciowych do wielu neuronów wyjściowych miało wagę niezerową i które z wielu połączeń sieciowych do wielu neuronów wyjściowych ma wagę zerową; moduł przetwarzania pakietów do odebrania pakietów danych z drugiego neuronu oraz do ustalenia, że pakiet danych wskazuje skok z drugiego neuronu jako wejście do pierwszego neuronu; moduł określania łączności do odczytu pamięci w celu uzyskania dostępu do danych o łączności związanych z pierwszym neuronem sieci neuronowej w odpowiedzi na ustalenie, że pakiet danych wskazuje skok z drugiego neuronu jako wejście do pierwszego neuronu; oraz moduł określania wagi do (i) określania, dla każdego z wielu połączeń sieciowych do wielu neuronów wyjściowych, które mają niezerową wagę, adresu pamięci, pod którym przechowywana jest odpowiednia niezerowa waga oraz (ii) dostępu, dla każdego z wielu połączeń sieciowych do wielu neuronów wyjściowych, które mają niezerową wagę, odpowiednią niezerową wagę z lokalizacji pamięci odpowiadającej odpowiadającemu adresowi pamięci. Przy czym moduł przetwarzania pakietów przetwarza pakiet danych w oparciu o wagę wielu połączeń sieciowych do wielu neuronów wyjściowych, które mają niezerową wagę, oraz w którym wagi wielu połączeń sieciowych do wielu neuronów wyjściowych mają zerową wagę nie są bezpośrednio przechowywane w pamięci.

Zgłoszenie EP3557485 A1 dotyczy sposobu uzyskiwania dostępu do danych i ich przetwarzania przez urządzenie akceleratora, który obejmuje kolejno: pobieranie co najmniej części danych wejściowych do przetworzenia przez urządzenie akceleratora; segmentowanie danych wejściowych na wiele sąsiednich płytek wejściowych, przy czym płytki wejściowe mają z góry określony rozmiar; przechowywanie co najmniej jednej z wielu sąsiednich płytek wejściowych w buforze danych urządzenia akceleratora; dostęp do elementów danych co najmniej jednego z wielu sąsiednich płytek wejściowych przechowywanych w buforze danych poprzez umieszczenie okna wejściowego nad elementami danych przechowywanymi w buforze danych, w celu wygenerowania wielu obszarów wejściowych, przy czym okno wejściowe jest regulowane zgodnie z zestawem parametrów, i w którym dostęp do elementów danych

za pośrednictwem okna wejściowego jest na stałe w obwodzie realizującym bufor danych; oraz sekwencyjne przetwarzanie co najmniej jednego z wielu obszarów wejściowych w celu co najmniej częściowego przetwarzania co najmniej jednego z wielu sąsiednich płytek wejściowych przez urządzenie przyspieszające. Ponadto zgłoszenie ujawnia system obejmujący host przetwarzania; co najmniej jedną pamięć; aparat akceleratorowy do przetwarzania danych wejściowych oraz połączenie łączące host przetwarzania, co najmniej jedną pamięć i urządzenie, przy czym urządzenie jest skonfigurowane do pobierania danych wejściowych z co najmniej jednej pamięci za pośrednictwem połączenia. Przy czym, wspomniany aparat akceleratorowy zawiera interfejs skonfigurowany do łączenia urządzenia z połączeniem oraz co najmniej jeden moduł przetwarzania skonfigurowany do przetwarzania danych wejściowych, przy czym urządzenie akceleratora jest skonfigurowane do pobierania co najmniej niektórych danych wejściowych za pośrednictwem interfejsu i przetwarzania danych wejściowych, przy użyciu co najmniej jednego modułu przetwarzania.

Natomiast zgłoszenie US2021049459 A1 ujawnia systemy i metody wydajnego wykonywania w czasie rzeczywistym wielu procesów uczenia maszynowego. W jednym przykładzie wykonania platforma obliczeniowa z wieloma elementami obliczeniowymi odbiera wiele strumieni danych, przy czym każdy taki strumień jest powiązany z własnym odpowiednim procesem uczenia maszynowego. Każdy proces uczenia maszynowego wykorzystuje swój strumień danych jako dane wejściowe do trenowania w czasie rzeczywistym odpowiedniego modelu matematycznego. Każdy z procesów ma szczyty i spadki wymagań przetwarzania. System ponownie przydziela w czasie rzeczywistym elementy obliczeniowe z procesów o niższym zapotrzebowaniu na przetwarzanie do procesów o wyższym zapotrzebowaniu na przetwarzanie, tym samym obsługując wszystkie liczne procesy w locie, zapobiegając tym samym przestojom systemu, oraz zmniejszenie ogólnych zasobów obliczeniowych wymaganych przez system. Przy czym, ujawniony sposób efektywnego wykonywania wielu procesów uczenia maszynowego, obejmuje następujące etapy: odbieranie na platformie obliczeniowej zawierającej wiele elementów obliczeniowych wielu strumieni zestawów danych; ciągłe uczenie, za pomocą wielu elementów obliczeniowych, wielu modeli matematycznych z wykorzystaniem odpowiednio wielu strumieni działających jako dane wejściowe, w którym wspomniane ciągłe uczenie jest wykonywane odpowiednio jako wiele procesów uczenia maszynowego w połączeniu z wieloma elementami obliczeniowymi, przy czym wiele modeli matematycznych obejmuje pierwszy model matematyczny i drugi model matematyczny, przy czym wiele elementów obliczeniowych obejmuje jeden lub więcej pierwszych elementów obliczeniowych przydzielonych do ciągłego uczenia się pierwszego modelu matematycznego i jeden lub więcej drugich elementów obliczeniowych przydzielonych do ciągłego trenowania drugiego modelu matematycznego; wykrycie na platformie obliczeniowej podczas wykonywania wielu procesów uczenia maszynowego stanu tymczasowego, w którym ciągłe uczenie drugiego modelu matematycznego pozostaje w tyle za odpowiednim strumieniem wielu strumieni zbiorów danych w wyniku tymczasowego stanu obliczeniowego związanego z drugim modelem matematycznym i odpowiednim strumieniem; oraz co najmniej tymczasowe ponowne przydzielenie co najmniej jednego z co najmniej jednego pierwszego elementu obliczeniowego z uczenia ciągłego pierwszego modelu matematycznego do uczenia ciągłego drugiego modelu matematycznego na podstawie, przynajmniej częściowo, wykrytego stanu tymczasowego i wykorzystanie ponownie przydzielonego co najmniej jednego pierwszego elementu obliczeniowego jest niższe niż wykorzystanie co najmniej jednego drugiego elementu obliczeniowego, zwiększając w ten sposób wydajność wspomnianego ciągłego szkolenia, które pozostaje w tyle za odpowiednim strumieniem, umożliwiając w ten sposób platformie obliczeniowej radzenie sobie ze stanem tymczasowym.

Celem wynalazku jest zapewnienie systemu oraz sposobu optymalizacji i zwiększenia efektywności systemów pamięci masowej.

Istotą wynalazku jest system zarządzania pamięcią masową oparty o sieci neuronowe, charakteryzujący się tym, że zawiera:

a) moduł akceleracyjny AIL z wbudowanymi algorytmami sztucznej inteligencji w postaci sieci neuronowej skonfigurowany do wielokryterialnej zmiany reguł alokacji i relokacji na podstawie danych wejściowych stanowiących dane o operacjach na plikach otrzymane z modułu DMFS i obejmujące co najmniej kombinację następujących danych o wagach niezerowych KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB, przy czym reguły alokacji i relokacji są wypracowywane co epokę na podstawie trenowania w czasie rzeczywistym sieci neuronowej na podstawie wzorców pochodzących z matematycznego modelu, generowanych na podstawie wspomnianych danych wejściowych; oraz

b) moduł zarządzający DMFS skonfigurowany do zbierania informacji dotyczących operacji na plikach i przekazywania ich do komponentu AIL oraz do wybierania trybu alokacji i relokacji plików w oparciu o reguły wypracowane przez algorytmy modułu AIL.

Korzystnie sieć neuronową stanowi sieć neuronowa typu perceptron wielowarstwowy.

Korzystnie perceptron wielowarstwowy składa się z warstwy wejściowej, 1–3 warstw ukrytych oraz warstwy wyjściowej.

Korzystnie moduł akceleracyjny AIL jest wyposażony w kartę akceleracyjną bazującą na układzie GPGPU.

Kolejną istotą wynalazku jest wspomagany komputerowo sposób zarządzania pamięcią masową znamienny tym, że obejmuje:

a) zbieranie co epokę przez moduł zarządzający DMFS informacji dotyczących operacji na plikach obejmujących co najmniej KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB;
b) przekazywanie co epokę przez DMFS danych zebranych w etapie a) wraz z ich konfiguracją do modułu akceleracyjnego AIL;
c) trenowanie w czasie rzeczywistym algorytmów sztucznej inteligencji w postaci sieci neuronowej wbudowanej w moduł AIL, z wykorzystaniem danych wejściowych stanowiących kombinację co najmniej danych o wagach niezerowych obejmujących KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB, oraz modelu matematycznego, którym jest model programowania liniowego całkowitoliczbowego;
d) wypracowanie co epokę przez moduł AIL reguł alokacji i relokacji plików;
e) przekazanie co epokę przez moduł AIL reguł wypracowanych w etapie d) do modułu zarządzającego DMFS;
f) alokację i relokację plików przez moduł zarządzający DMFS w oparciu o reguły wypracowane w etapie d) przez algorytmy modułu AIL.

Korzystnie sieć neuronową zastosowaną w sposobie według wynalazku stanowi sieć neuronowa typu perceptron wielowarstwowy.

Korzystnie perceptron wielowarstwowy składa się z warstwy wejściowej, 1–3 warstw ukrytych oraz warstwy wyjściowej.

Przy czym, w rozumieniu niniejszego wynalazku stosuje się następujące definicje pojęć:

**SYSTEM** – system zarządzania pamięcią masową.

**MACIERZ** – urządzenie do składowania plików w SYSTEMIE (pamięć masowa), w którego skład wchodzi zestaw nośników.

**AIL** – (Artificial Intelligence Layer) – komponent SYSTEMU, wykorzystujący metody optymalizacji, sztucznej inteligencji i uczenia maszynowego do zarządzania rozmieszczaniem plików na nośnikach.

**DMFS** – (Data Management File System) – komponent SYSTEMU odpowiadający za operacje na nośnikach.

**AIDTOOL** – narzędzie w ramach komponentu DMFS do zarządzania konfiguracją SYSTEMU.

**REGUŁY ALOKACJI** – wielowariantowa reguła o alokacji pliku tworzonego w MACIERZY.

**REGUŁY RELOKACJI** – wielowariantowa reguła o relokacji pliku istniejącego w MACIERZY.

**REGUŁY DOMYŚLNE** – reguły określone w konfiguracji SYSTEMU przez jego administratora za pomocą narzędzia AIDTOOL, które są używane do alokacji plików bezpośrednio po uruchomieniu SYSTEMU oraz wtedy, gdy komponent DMFS nie mógł wykorzystać REGUŁY ALOKACJI lub REGUŁY RELOKACJI.

**EPOKA** – okres (niekoniecznie stałej długości), po którym komponent AIL udostępnia wypracowane „co EPOKĘ" REGUŁY ALOKACJI i REGUŁY RELOKACJI.

**PARAMETRY PLIKU** – dane o pliku w postaci ustalonego zestawu trójek (nazwa_parametru, typ_parametru, wartość_parametru), które zapisywane są na nośniku wraz z danymi, tworzącymi zawartość pliku.

**PARAMETRY NOŚNIKA** – dane o nośniku w postaci ustalonego zestawu trójek (nazwa_parametru, typ_parametru, wartość_parametru), które przechowywane są w SYSTEMIE.

**KRYTERIA** – lista kryteriów, którymi kieruje się AIL, podczas opracowywania reguł alokacji i relokacji dla wielokryterialnej optymalizacji rozkładu plików na macierzy.

**TABELA PLIKÓW** – lista danych o plikach o rozmiarze p, zawierająca parametry (statyczne i dynamiczne) tych plików, do których w określonym okresie czasu SYSTEM miał dostęp. Na podstawie TABELI PLIKÓW komponent AIL wypracowuje REGUŁY ALOKACJI i REGUŁY RELOKACJI. AIL może

w dowolnym momencie skorzystać z TABELI PLIKÓW za pośrednictwem odpowiednich funkcji komponentu DMFS.

**TABELA NOŚNIKÓW** – lista nośników w MACIERZY zawierająca parametry (statyczne i wyliczane) tych nośników. Na podstawie TABELI NOŚNIKÓW komponent AIL wypracowuje REGUŁY ALOKACJI i REGUŁY RELOKACJI. AIL może w dowolnym momencie skorzystać z TABELI NOŚNIKÓW za pośrednictwem odpowiednich funkcji komponentu DMFS.

**TABELA WAG** – lista wag określająca stopień uwzględnienia kryterium przy maksymalizacji łącznej korzyści względem wszystkich kryteriów łącznie. Suma wag musi wynosić 1 (jeden), a kryterium, które ma być pominięte, ma przypisaną wagę 0 (zero).

Wynalazek dostarcza następujących korzyści:

- Algorytmy sztucznej inteligencji modułu AIL pozwalają na automatyczną, inteligentną alokację i relokację danych zgodnie z ustalonymi kryteriami np. szybkość dostępu, bezpieczeństwo, koszt składowania, poprzez wyznaczanie wzorca za pomocą optymalizacji wielokryterialnej;
- System i sposób według wynalazku zapewniają optymalizację rozkładu danych i przez to zwiększenie efektywności systemów pamięci masowej;
- Zaadoptowanie i zintegrowanie algorytmów decyzyjnych z akceleracją sprzętową pozwala na wyeliminowanie opóźnień charakterystycznych dla bibliotek uruchamianych tylko w środowisku opartym o standardowe procesory;
- Rozwiązanie według wynalazku zapewnia zbieranie informacji o stanie i użytkowaniu pamięci masowej do nauki wielokryterialnie optymalnego rozkładu danych (np. koszt, wydajność, bezpieczeństwo) wykorzystując automatyczne próbkowanie i analizę kluczowych parametrów (takich jak: stan danych cyfrowych w trakcie ich pełnego cyklu życia, stan nośników w czasie, operacje relokacji danych cyfrowych) oraz automatyczne uczenie się sposobu użytkowania pamięci masowej;
- Rozwiązanie według wynalazku zapewnia automatyczną optymalną alokację i relokację przechowywanych danych cyfrowych;
- Rozwiązanie według wynalazku umożliwia przesunięcie kluczowych operacji związanych z zarządzaniem danymi cyfrowymi oraz powiązanych z nimi informacjami ze strony użytkownika na obszar algorytmów sztucznej inteligencji.

Wynalazek przedstawiono w przykładach wykonania na rysunku, na którym fig. 1 przedstawia schemat systemu według wynalazku; fig. 2 przedstawia schemat działania algorytmów decyzyjnych w systemie i sposobie według wynalazku; fig. 3 przedstawia schemat sieci neuronowej wykorzystanej w systemie oraz sposobie według wynalazku.

**Przykład 1**

System zarządzania pamięcią masową oparty o sieci neuronowe według wynalazku przedstawiono na fig. 1. Operacje na systemie plików w jądrze systemu operacyjnego są przekazywane poprzez bibliotekę libFuse do oprogramowania DMFS znajdującego się w przestrzeni użytkownika, który to określa ostateczną decyzję wykonania operacji na systemie plików zapamiętując przy tym istotne informacje dotyczące tych operacji i decyzji, przekazuje je następnie do modułu sztucznej inteligencji AIL, podejmującej decyzję o ewentualnej zmianie położenia plików w systemie plików zwracając je w celu wykonania do oprogramowania DMFS.

Głównym założeniem dla działania Systemu według wynalazku jest wielokryterialna optymalizacja rozkładu plików na macierzy, przy kryteriach takich jak:

- minimalizacja kosztu nośników,
- minimalizacja zużycia energii przez nośniki,
- maksymalizacja średniej szybkości odczytu plików z nośników,
- maksymalizacja średniej szybkości zapisu plików na nośniki,
- minimalizacja średniego opóźnienia dostępu do plików na nośnikach,
- maksymalizacja bezpieczeństwa plików przechowywanych na nośnikach,
- maksymalizacja gęstości upakowania danych na nośnikach,
- minimalizacja wibracji spowodowanych pracą nośników.

Szczegółowy zakres kryteriów i sposób ich wyliczania przez System według wynalazku przedstawiono w Tabeli 1:

**Tabela 1**. Kryterium optymalizacji rozkładu plików na macierzy

| **Nazwa kryterium/akronim** | **Sposób wyliczenia** | **Typ** |
|---|---|---|
| KOSZT<br><br>**KOSZT** | Koszt przestrzeni zajętej przez AIL.<br><br>$KOSZT := \sum_{j=1}^{n} KOSZT_j \times \left(\sum_{i \in m(j)} SWP_i / PAIL_j\right)$,<br><br>gdzie:<br>$m(j)$ – zbiór identyfikatorów tych plików, o których informacje zawarte są w TABELI PLIKÓW, które to pliki umieszczone są nośniku $j$.<br><br>*minimalizować*<br>$m(j)$ – zbiór identyfikatorów tych plików, o których informacje zawarte są w TABELI PLIKÓW, które to pliki umieszczone są nośniku $j$.<br><br>*minimalizować* | real |
| ZUŻYCIE ENERGII<br><br>**ENERGIA** | Łączne zużycie energii przez nośniki ponad tryb spoczynku.<br><br>$$ENERGIA := \sum_{j=1}^{n}\left(\frac{\mathrm{ZEO}_j}{\mathrm{PODCZ}_j}\sum_{i \in m(j)} WO_i + \frac{\mathrm{ZEZ}_j}{\mathrm{POZ}_j}\sum_{i \in m(j)} WZ_i\right)$$<br><br>*minimalizować* | real |
| ŚREDNIA SZYBKOŚĆ ODCZYTU<br><br>DANYCH<br><br>**SSO** | Średnia szybkość odczytu danych z plików.<br><br>$$SSO := \frac{\sum_{j=1}^{n}\sum_{i \in m(i)} WO_i}{\sum_{j=1}^{n}\frac{1}{\mathrm{PODCZ}_j}\sum_{i \in m(j)} WO_i}$$<br><br>*maksymalizować* | real |
| ŚREDNIA SZYBKOŚĆ ZAPISU DANYCH<br><br>**SSZ** | Średnia szybkość zapisu danych do plików.<br><br>$$SSOZ := \frac{\sum_{j=1}^{n}\sum_{i \in m(j)} WZ_i}{\sum_{j=1}^{n}\frac{1}{\mathrm{POZ}_j}\sum_{i \in m(j)} WZ_i}$$<br><br>*maksymalizować* | real |

| | | |
|---|---|---|
| ŚREDNIE OPÓŹNIENIE DOSTĘPU<br><br>**SOD** | $$SOD := \frac{A}{\sum_{j=1}^{n}\left(\sum_{i\in m(j)} LOO_i + \sum_{i\in m(j)} LZAP_i\right)},$$<br>gdzie<br>$$A := \sum_{j=1}^{n}\left(\mathrm{ODS}_j\left(\sum_{i\in m(j)} LOO_i + \sum_{i\in m(j)} LZAP_i\right)\right)$$<br>*minimalizować* | real |
| BEZPIECZEŃSTWO PLIKÓW<br><br>**BP** | Współczynnik bezpieczeństwa plików.<br>$$BP := \frac{\sum_{j=1}^{n}(100 - SZ_j)\sum_{i\in m(j)} SWP_i}{n}$$<br>*maksymalizować* | real |
| GĘSTOŚĆ UPAKOWANIA<br><br>**GUP** | Średnia gęstość upakowania danych na nośnikach.<br>$$GUP := \frac{\sum_{j=1}^{n}\left(\sum_{i\in m(j)} SWP_i / GAB_j\right)}{n}$$<br>*maksymalizować* | real |
| WIBRACJE NOŚNIKÓW<br><br>**WIB** | Średni poziom wibracji nośników.<br>$$WIB := \frac{\sum_{j=1}^{n} WIB_j \sum_{i\in m(j)}(WO_i + WZ_i)}{n}$$<br>*minimalizować* | real |

Moduł akceleracyjny AIL z wbudowanymi algorytmami sztucznej inteligencji w postaci sieci neuronowej (np. perceptron wielowarstwowy) jest skonfigurowany do wielokryterialnej zmiany reguł alokacji i relokacji na podstawie danych wejściowych stanowiących dane o operacjach na plikach otrzymane z modułu DMFS i obejmujące co najmniej kombinację następujących danych o wagach niezerowych KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB. Przy czym reguły alokacji i relokacji są wypracowywane co epokę na podstawie trenowania w czasie rzeczywistym matematycznego

modelu programowania liniowego całkowitoliczbowego na podstawie wspomnianych danych wejściowych. Ponadto w tym przykładzie wykonania moduł AIL jest wyposażony w kartę akceleracyjną bazującą na układzie GPGPU.

Natomiast moduł zarządzający DMFS skonfigurowany do zbierania informacji dotyczących operacji na plikach i przekazywania ich do komponentu AIL oraz do wybierania trybu alokacji i relokacji plików w oparciu o reguły wypracowane przez algorytmy modułu AIL.

W tym przykładzie wykonania moduł zarządzający DMFS (Data Management File System) jest warstwą systemu plików, pozwalającą na odseparowanie rzeczywistego systemu plików od klientów korzystających z pamięci masowej – użytkowników, aplikacji, bazy danych, itp. System plików DMFS rozmieszcza dane na urządzeniach pamięci różnych typów wraz z możliwością relokacji danych pomiędzy nimi, przy uwzględnieniu iż relokacja danych będzie transparentem dla klientów. Dodatkowo, za pomocą narzędzia AIDTOOL (stanowiącego sub-komponent DMFS) są w nim zapisywane i przekazywane do modułu AIL dane dotyczące plików, nośników, wag kryteriów i reguł domyślnych. Przez DMFS są zbierane również dane o operacjach na plikach w Systemie w postaci tabeli plików, odczytywanej następnie przez moduł AIL.

System plików DMFS jest parametryzowany za pośrednictwem narzędzia AIDTOOL, wyszczególniając konfigurację wydajności oraz bezpieczeństwa plików poprzez nadawanie im dodatkowych cech, profilowanie systemu poprzez regulację wag oraz określanie wstępnego rozkładu plików na wskazane nośniki poprzez definiowanie reguł (Reguł Administratora).

Charakterystykę modułu DMFS oraz modułu AIL przedstawiono poniżej.

**MODUŁ DMFS**

Moduł DMFS składa się z następujących elementów:

- Narzędzia AIDTOOL
- Komponent Ctx
- Komponent Fs, Alloc
- Filesystem DMFS
- Metadane
- Komponent Recfg
- Komponent Reloc
- Komponent Aip

**Narzędzie AIDTOOL**

DMFS w trakcie swojej pracy zbiera informacje dotyczące operacji na plikach, takie jak częstotliwość i wielkość odczytu/zapisu oraz aktualny rozmiaru pliku wraz z ilością zmian. Informacje te wraz z konfiguracją wprowadzoną za pośrednictwem narzędzia AIDTOOL, przekazywane są do modułu sztucznej inteligencji AIL, który na tej podstawie wypracowuje własne decyzje dotyczące rozkładu plików na nośnikach wchodzących w skład macierzy DMFS. Decyzje AIL są następnie przekazywane do DMFS i są podstawą do relokacji plików oraz do alokacji nowych plików, co jest równoznaczne z zastąpieniem reguł tworzonych za pośrednictwem AIDTOOL.

**Komponent Ctx**

Komponent Ctx stanowi główny konfiguracji DMFS tworzący kontekst systemu. Jest uruchamiany wraz ze startem DMFS. Inicjalizacja Ctx rozpoczyna się od wywołania metody ‘ctx(Ctx * pSelf)’,która w pierwszej kolejności inicjalizuje blokadę rwlock (mutex r/w) a następnie wywołuje operację ‘DmfsCfgInit()', która rozpoczyna tworzenie instancji klas przechowujących konfigurację Dmfs (Diagramie 3, 4).

Zadaniem blokady rwlock jest blokowanie innych wątków korzystających z kontekstu systemu w przypadku rekonfiguracji online.

Poprawne zakończenie ‘ctx()’ zwraca wartość zero (0).

W każdym przypadku powstania błędu, inicjalizacja zostaje przerwana a informacja o błędzie (<0) zwracana jest do komponentu wywołującego Ctx.

Usuwanie instancji Ctx odbywa się poprzez wywołanie ‘_ctx()’ a proces przebiega w odwrotny sposób jak inicjalizacja a ewentualne błędy są pomijane.

Wywołanie 'dmfsCfgInit()' uruchamia inicjalizacje poszczególnych komponentów przechowujących informacje dotyczące:

- Cfg – konfiguracji systemowej [2]
- Drives – konfiguracji nośników [1][2]

- Rules – reguł tworzenia plików [1][2]
- Filemask – dodatkowej parametryzacj i plików [1][2]
- Weights – wag [2][5]

Poprawne zakończenie 'dmfsCfgInit()' zwraca wartość zero (0).

W każdym przypadku powstania błędu, inicjalizacja zostaje przerwana a informacja o błędzie (<0) zostaje zwrócona.

Inicjalizacja komponentów dziedziczących po abstrakcyjnej klasie BaseCfg rozpoczyna się od otwarcia i rozpoczęcia odczytywania pliku z wartościami parametrów. Do odczytu sekwencyjnego linia po linii, BaseCfg używa komponentu klasowego GetFile z gałęzi narzędzi Tools:

- BaseCfg przesyła do Tools: :GetFile komendę 'fOpen(pName, ppF)'
- Tools::GetFile zwraca do BaseCfg informację o statusie
- Tools::GetFile przesyła do BaseCfg komendę 'CfgFileCB(pOb, p line)', która jest wskaźnikiem na Callback podanym podczas wywołania fOpen

Każda linia zawierająca wartości parametrów zostaje zapisana do listy parametrów

- BaseCfg przesyła do List komendę 'addToEndList(pData, size, ppList)' w celu dodania bufora danych konfiguracyjnych

Po zakończeniu czytania pliku, deskryptor zostaje zamknięty oraz inicjalizacja zostaje zakończona.

W każdym przypadku powstania błędu (wyłączając zamknięcie pliku), inicjalizacja zostaje przerwana a odpowiednia informacja o błędzie zostaje zwrócona (patrz: AIDTOOL błędy).

Rekonfiguracja DMFS rozpoczyna się od przesłania do Ctx przez Aidtool komendy 'recfg()', która w pierwszej kolejności wywołuje operację 'Przypisz nową konfigurację do zmiennej tymczasowej', umożliwiając w ten sposób tworzenie nowego kontekstu systemu w tle. Ctx wywołuje operację 'DmfsCfgInit', która uruchamia proces tworzenia obiektów kolejnych klas dziedziczących po komponencie 'BaseCfg' zapisywanych w zmiennej tymczasowej. W przypadku powstania błędu, zostaje on zwrócony do komponentu Aidtool ('DmfsCfgInit' zwraca do Ctx informację o błędzie, Ctx zwraca do Aidtool informację o błędzie, zmodyfikowaną w zależności od typu komponentu, w którym powstał błąd). Więcej informacji na temat typów błędów, znajduje się w niniejszym opracowaniu w sekcji „Typy błędów". Więcej informacji na temat 'DmfsCfgInit' znajduje się w niniejszym opracowaniu w sekcji „Inicjalizacja komponentów konfiguracyjnych". W przypadku gdy operacja 'DmfsCfgInit' zakończyła się sukcesem, Ctx wykonuje 'Oczekuj na zablokowanie rwlock (mutex r/w) do zapisu', w przypadku którego niepowodzenie określone czasem 30 sek (w wersji prototypowej jest to wartość wprowadzona do kodu), zwraca błąd 'AIDTOOL_ERR_LOCK', oznaczający brak możliwości założenia blokady, co skutkuje niewykonaniem rekonfiguracji i zakończeniem procesu.

W przypadku założeniu blokady do zapisu (mutex r/w), Ctx wywołuje operację 'Zapisz nową konfigurację', która w miejsce jej aktualnej wersji wprowadza nową, następnie Ctx aktualizuje wskaźnik nowej wersji konfiguracji dla zarejestrowanych komponentów poprzez 'Wskaż nową wersję kontekstu dla zarejestrowanych komponentów' (w wersji prototypowej inne komponenty korzystają z tej wartości w celu sprawdzenia poprawności kontekstu na jakim pracują). Dalej Ctx zwalnia blokadę do zapisu 'Zwolnij blokadę rwlock (mutex r/w) do zapisu' oraz usuwa poprzedni kontekst, zwalniając pamięć ('Usuń poprzednią konfigurację').

Na końcu procesu rekonfiguracji Ctx zwraca do Aidtool informację '0', oznaczającą poprawne jej wykonanie. Wyrejestrowanie nie jest wymagane w tej wersji DMFS ponieważ żaden zarejestrowany komponent nie kończy życia w trakcie działania DMFS.

**Komponent Fs, Alloc**

Komponent Fs odpowiedzialny jest w DMFS za komunikację z FUSE. Operacje użytkownika na plikach i katalogach przekazywane są przez FUSE do Fs, który to współpracując głównie z komponentem Alloc obsługują system plików DMFS.

**Filesystem DMFS**

System plików DMFS w ogólnym opisie wykorzystuje wiele dostępnych nośników w sposób transparentny dla użytkownika, to jest w taki sposób by użytkownik korzystał z ich przestrzeni jako z jednej całości. Istotą rozwiązania jest możliwość doboru nośników w celu wielokryterialnej optymalizacji działania macierzy DMFS.

**Metadane**

System plików DMFS wykorzystuje atrybuty plików (xattr) w celu przechowywania wymaganych dla zrealizowania swoich zadań danych.

Nazwy tych atrybutów nie mogą być dostępne dla użytkownika.

Fs sprawdza zarezerwowane atrybuty X w FileTab. Fs obsługuje zarezerwowane atrybuty X. Zarezerwowane atrybutów X (nazwy) uniemożliwiają modyfikację atrybutów X używanych przez DMFS. Dodatkowo umożliwiają uruchamianie dodatkowych funkcjonalności DMFS z poziomu systemu plików użytkownika. Fs resetuje się w Alloc. Fs wyszukuje plik lub katalog w Alloc. W obecnym rozwiązaniu funkcjonalność ta jest identyczna jak wyszukiwanie katalogów z tym że wyszukuje również pliki. Fs wykonuje operacje na pliku lub katalogu. Fs informuje o zmianie pojemności nośnika w Drives.

FUSE tworzy nowy plik Fs. Fs resetuje się w Alloc. Ze względu na wymaganie wskazujące na minimalizację opóźnienia dotyczącego ruchu na plikach obiekt Alloc tworzony jest dla wątku a nie dla pojedynczej operacji na pliku (wiele operacji na pliku dla jednego wątku Alloc) dlatego też musi zostać „zresetowany" czyli wyczyszczony jego stan z poprzedniej operacji. FileTab zapisuje dane nośnika. Alloc sprawdza aktualność kontekstu systemu. Alloc rozpoczyna tworzenie nowego pliku.

Alloc włącza obsługę długich nazw plików. Alloc poszukuje odpowiedniej maski dla pliku. Alloc wysyła zapytanie do AIL. Alloc sprawdza reguły Administratora. Alloc sprawdza reguły domyślne. Reguły domyślne polegają na znalezieniu nośnika o jak największej dostępnej pojemności. Fs poszukuje lokalizacji dla nowego pliku w Alloc. Alloc zwraca nośnik dla nowego pliku do Fs. Fs tworzy nowy plik na wskazanym nośniku. W przypadku błędu Fs ponawia poszukiwanie lokalizacji dla nowego pliku przez komponent Alloc. Fs tworzy nowy plik w FileTab. Sformułowanie "tworzy nowy plik" jest umowne ponieważ komponent FileTab implementuje swoje rozwiązanie dla takiej funkcjonalności. W wersji prototypowej DMFS FileTab tworzy nowy wpis w Tabeli Plików lub usuwa najstarszy plik jeżeli Tabela Plików jest przepełniona. FileTab generuje ID nowego pliku. FileTab tworzy strukturę dla nowego pliku. Utworzoną strukturę wypełnia wartościami domyślnymi. FileTab tworzy deskryptor pliku. FileTab poszukuje miejsca w Tabeli Plików. Pliki w trakcie relokacji nie mogą być usunięte z Tabeli Plików.

FileTab sprawdza kontekst systemu. FileTab dodaje strukturę do Tabeli Plików. FileTab przesuwa strukturę na szczyt Tabeli Plików. Ta operacja umożliwia sortowanie plików względem najczęstszego ich używania. FileTab zapisuje wymagane parametry w atrybutach X. FileTab zwraca uchwyt do pliku do Fs. Uchwyt do pliku to numer pozycji w Tabeli Plików oraz utworzone ID pliku. Fs zapisuje uchwyt do struktury w Tabeli Plików. Komponent FUSE umożliwia przechowywanie jednej wartości danych jako uchwyt.

Fs resetuje się w Alloc. Fs wyszukuje plik w Alloc. Fs wykonuje operację na pliku. Fs aktualizuje informacje o pliku w FileTab.

Operacje na pliku (odczyt, zapis) wykonywane przez Użytkownika przedstawione są na przykładzie zapisu do pliku, gdzie Fs pobiera deskryptor pliku z FileTab. FileTab zwraca dwa deskryptory, jeden dla pliku, na którym obecnie pracuje system oraz drugi w przypadku gdy trwa relokacja wskazujący na plik docelowy. FileTab odczytuje deskryptor pliku na podstawie uchwytu. FileTab sprawdza czy pozycja zawarta w uchwycie należy do odpowiedniego ID pliku. Zakłada się że nie będzie możliwości żeby plik znajdował się pod inną pozycją niż ta zapisana w uchwycie ponieważ tylko jeden system (czyli DMFS) zarządza plikami. Może zaistnieć sytuacja że pliku nie będzie w Tabeli Plików wtedy zostaje on załadowany z nośnika. Istnieje jeszcze możliwość taka, że odczytany plik nie będzie utworzony przez DMFS wtedy powinny być dla niego stworzone nowe atrybuty. Fs zapisuje dane do pliku. W przypadku gdy plik jest relokowany zapis dotyczy obydwu zwróconych przez FileTab deskryptorów z użyciem systemowej blokady danych, na których wykonywana jest operacja. Fs aktualizuje informacje o pliku w FileTab. Fs informuje o zmianie pojemności nośnika.

FUSE tworzy nowy katalog Fs. Fs pobiera identyfikator dla nowego katalogu w FileTab. Fs resetuje się w Alloc. Fs poszukuje lokalizacji dla nowego katalogu w Alloc. Obecnie na wszystkich nośnikach tworzony jest ten katalog. Komponent Alloc może przerwać operację w razie błędu. Fs tworzy nowy katalog na wskazanym nośniku. Nieutworzenie katalogu na nośniku na którym ma się znaleźć plik nie jest błędem krytycznym. Fs aktualizuje informacje o katalogu w FileTab. FileTab zapisuje wymagane parametry w atrybutach X. Fs informuje o zmianie pojemności nośnika w Drives.

FUSE usuwa katalog poprzez Fs. Fs resetuje się w Alloc. Fs wyszukuje katalog w Alloc. Alloc przeszukuje aktualny nośnik. Alloc przesuwa wskaźnik nośnika. Fs usuwa katalog na wskazanym nośniku. Operacja jest powtarzana aż do końca listy nośników.

**Komponent Recfg**

ReCfg jest odpowiedzialny za proces rekonfiguracji DMFS online czyli podczas pracy Macierzy. Rekonfigurację rozpoczyna przesłanie komunikatu recfg z AIDTOOL.

DMFS inicjalizuje ReCfg. ReCfg oczekuje na sygnał od AIDTOOL. Na obecną chwilę jedynym sygnałem od AIDTOOL jest sygnał rekonfiguracji. W wersji prototypowej komunikacja z AIDTOOL odbywa się za pośrednictwem "Named Pipe". ReCfg odczytuje sygnał od AIDTOOL. Sygnałem do relokacji jest wartość jeden w pierwszym bajcie przesyłanych danych. W przypadku rozwinięcia komunikacji z AIDTOOL do DMFS należy określić inne wartości pierwszego bajta danych. Mechanizm odczytu sygnału od AIDTOOL musi blokować inne odczyty do czasu zakończenia obsługi sygnału. ReCfg uruchamia rekonfigurację w Ctx. Ctx alokuje pamięć dla nowej konfiguracji. Ctx odczytuje konfigurację Weights. Ctx odczytuje konfigurację Rules. Ctx odczytuje konfigurację Filemask. Ctx odczytuje konfigurację Drives. Ctx odczytuje konfigurację Cfg. Ctx sprawdza limit wielkości Tabeli Plików.

Ctx wysyła sygnał do zarejestrowanych komponentów. Wiele komponentów korzysta z aktualnego kontekstu systemu(konfiguracji), który to zmieni się po rekonfiguracji. Każdy z komponentów musi odrębnie implementować fakt zmiany kontekstu podczas pracy systemu. Zdecydowano się na formę rejestracji komponentów dla poinformowania o zmianie kontekstu podczas pracy systemu (inne to np. suma kontrolna CRC, wersjonowanie konfiguracji) ponieważ wyklucza możliwość przypadkowego powtórzenia się podobnej wartości. Przypadek powtórzenia wartości przy CRC lub wersjonowaniu ma małe prawdopodobieństwo powstania jednak w systemach embedded należy wykluczać możliwie każdą awarię, która może spowodować błędną pracą systemu. Na końcu Ctx usuwa poprzednią konfigurację.

**Komponent Reloc**

Reloc odpowiada za relokację plików pomiędzy nośnikami wchodzącymi w skład Macierzy. Relokowanie pliku jest inicjalizowane poprzez Kolejkę relokacji pochodzącą z modułu sztucznej inteligencji lub poprzez potrzebę przesunięcia pliku z nadmiernie zajętego nośnika.

Poniższy opis przedstawia przykładowy proces inicjalizacji komponentu Reloc oraz proces relokacji na podstawie kolejki Relokacji.

DMFS inicjalizuje Reloc. Reloc rejestruje się w Ctx. Reloc inicjalizuje Logger. Reloc inicjalizuje AilThread. AilThread oczekuje na sygnał od AIL. W przypadku prototypu, komunikacja z AIL odbywa się za pośrednictwem pliku wymiany danych. Sygnał wykonywany jest poprzez usunięcie pliku wymiany danych przez AIL. AilThread konwertuje Kolejkę Relokacji. W wersji prototypowej Kolejka Relokacji jest plikiem wymiany danych w formacje który nie posiada wymaganych parametrów w związku z tym wymaga konwersji. W pierwszej kolejności nazwa pliku wymiany danych zostaje zmieniona na plik o nazwie "/tmp/reloctmp".Umiejscowienie pliku w katalogu "/tmp" powoduje, że po uruchomieniu maszyny plik zostanie systemowo usunięty podczas inicjalizacji systemu operacyjnego. AilThread odczytuje Kolejkę Relokacji. AilThread sprawdza czy istnieją pliki do relokacji. AilThread pobiera informacje o pliku do relokacji. AilThread aktualizuje Listę Relokacji. Lista relokacji jest obszarem w pamięci w którym kolejkowane są zadania relokacji. Istnieje po to by zadania relokacji mogły być tworzone przez wiele różnych komponentów-relokacja zagnieżdżona. Lista relokacji umożliwia również wielowątkowe relokowanie plików. AilThread odczytuje Listę Relokacji. AilThread pobiera pierwszy plik do relokacji. Pliki do Relokacji pobierane są z list Relokacji. AilThread inicjalizuje RelocProc. AilThread uruchamia relokację w RelocProc. RelocProc sprawdza pojemność docelowego nośnika.

RelocProc inicjalizuje plik do relokacji w I_Reloc. Plik zdatny do relokacji musi znajdować się w Tabeli Plików oraz spełniać wymagane warunki kwalifikujące go do relokacji RelocProc sprawdza czy nie zmienił się kontekst systemu. W przypadku zmiany kontekstu systemu najprawdopodobniej zmieni się również charakterystyka macierzy. Dlatego też należy zaprzestać relokacji na podstawie obecnej Kolejki Relokacji. RelocProc sprawdza czy docelowy nośnik istnieje. RelocProc tworzy docelowy plik. I_Reloc sprawdza czy plik nie jest usunięty. I_Reloc sprawdza czy plik nie ma zgłoszonego błędu. I_Reloc sprawdza czy plik nie jest już relokowany. I_Reloc sprawdza czy nie posiada atrybutu X. wykluczające I_Reloc kopiuje deskryptor pliku. I_Reloc ustawia flagę relokacji. RelocProc wyłącza aktualizację daty. RelocProc odczytuje statystykę pliku źródłowego. RelocProc sprawdza czy nie posiada dowiązań. W przypadku dowiązań należałoby stworzyć mechanizm umożliwiający mechanizm przeniesienia dowiązań na plik docelowy. Na tym etapie wymaganie to nie jest konieczne. RelocProc odczytuje statystykę pliku docelowego. RelocProc porównuje czy nośnik docelowy nie jest nośnikiem źródłowym. RelocProc alokuje pamięć w pliku docelowym. RelocProc rozpoczyna relokację. RelocProc inicjalizuje RelocFileThread. RelocFileThread sprawdza

statystykę pliku źródłowego. RelocFileThread informuje o rozpoczęciu relokacji w I_Reloc. I_Reloc sprawdza flagę relokacji. I_Reloc sprawdza czy plik nie jest już relokowany. I_Reloc ustawia deskryptor pliku docelowego. RelocFileThread rozpoczyna kopiowanie danych. Kopiowanie rozpoczyna się od końca pliku. Operacje nadpisywanie i dopisywanie danych przez użytkownika w relokowanym pliku wykonywane są na pliku źródłowym i docelowym. Operacje które anulują relokację. RelocFileThread zakłada blokadę odczytu na pliku źródłowym. RelocFileThread zakłada blokadę zapisu na pliku docelowym. RelocFileThread kopiuje dane z pliku źródłowego. RelocFileThread odblokowuje plik źródłowy. RelocFileThread kopiuje dane do pliku docelowego. RelocFileThread odblokowuje plik docelowy. RelocFileThread ustawia parametry pliku docelowego. RelocFileThread kopiuje czasy pliku. RelocFileThread kopiuje atrybuty X. RelocFileThread przenosi plik docelowy z katalogu tymczasowego. RelocFileThread informuje o zakończeniu relokacji w I_Reloc. Od tego momentu rozpoczyna się sekcja krytyczna trwająca aż do usunięcia pliku źródłowego. Jednym z problemów jest fakt, iż przez pewien moment na macierzy będą istniały dwa takie same pliki na różnych nośnikach, ale zamiana deskryptorów w Tablicy Plików rozwiązuje ten problem.

I_Reloc ustawia offset deskryptora. I_Reloc zamyka deskryptor pliku źródłowego. I_Reloc ustawia deskryptor docelowy na źródłowy. I_Reloc ustawia docelowy nośnik dla pliku. I_Reloc zeruje flagę relokacji. RelocFileThread usuwa plik źródłowy. AilThread powtarza operację. Po zakończeniu wyłącza Reloc oraz usuwa w AilThread.

**Komponent Aip**

Komponent Aip jest ściśle związany z zadaniami dotyczącymi modułu sztucznej inteligencji AIL jak:

- Allocate_file – odpytanie modułu sztucznej inteligencji alokacje nowego pliku
- Realocate_file – odpytanie modułu sztucznej inteligencji relokacje istniejącego pliku
- Inicjalizacja komponentu Ail odpowiadającego za przygotowanie pliku z informacjami o ruchu na plikach i innych aktualnych parametrach

Zadania Allocate_file, Relocatefile odpowiadają za odpytywanie modułu sztucznej inteligencji o miejsce alokacji lub relokacji plików. W wersji prototypowej DMFS wymagają dokończenia implementacji. Relocate file jest bliźniaczo podobnym technicznie rozwiązaniem do Allocate_file.

DMFS utwórz plik z parametrami pliku DMFS files. allocate_file.py Pobierz plik z parametrami pliku DMFS files. allocate_file.py Wczytaj pliki binarne files. allocate_file.py Zapisz listę nośników dla pliku DMFS files, allocate_ file.py Przetwarzaj. DMFS Pobierz listę nośników dla pliku DMFS files. User Utwórz nowy plik DMFS. DMFS Utwórz nowy plik . DMFS Uruchom skrypt sh allocate file_py. Działanie realizuje rozwiązanie zaproponowane przez dostawców modułu sztucznej inteligencji w wykorzystaniem odrębnych wątków.

**MODUŁ AIL**

**Komponent Ail** odpowiada za przygotowanie pliku z informacjami o ruchu na plikach i innych aktualnych parametrach dla modułu sztucznej inteligencji AIL.

Aip inicjalizuje Ail. Ail rejestruje się w Ctx. Ail oczekuje na sygnał od modułu sztucznej inteligencji. W przypadku prototypu, komunikacja z AIL odbywa się za pośrednictwem pliku wymiany danych. Komunikat wykonywany jest poprzez usunięcie pliku wymiany danych przez moduł sztucznej inteligencji. Ail tworzy Tabelę Plików w FileTab. Ail tworzy Tabelę Wag w Weights. Ail tworzy Tabelę Nośników w Drives. DMFS inicjalizuje Aip. Ail odpowiada na sygnał od modułu AIL. Następnie wraca do oczekiwania na sygnał od AIL.

**Komponent FileTab** zbiera informacje o ruchu na plikach. Wywoływanie funkcjonalności FileTab dotyczy komunikacji z innymi komponentem Fs i opisane jest właśnie w nim. Poniższy diagram przedstawia natomiast FileTab w przebiegu pełnego procesu otwarcia pliku oraz operacji na pliku.

**Algorytmy decyzyjne modułu AIL**

Zaimplementowane na karcie akceleracyjnej algorytmy wypracowującego reguły alokacji i realokacji plików w macierzy należą do klasy metod nadążnych, tj. metod które na podstawie bieżącego stanu systemu (macierzy pamięci) wypracowując decyzje co do pracy systemu na pewien okres w przód. Efektywność algorytmów decyzyjnych, rozumiana jako wzrost efektywności macierzy pamięci działającego wraz z algorytmami decyzyjnymi, w stosunku do efektywności macierzy pamięci działającej bez tego komponentu, jest wypadkową wielu elementów. Kluczowymi elementami są: zakres informacji dostępnych dla wypracowania decyzji (reguł alokacji i realokacji plików), czas wypracowania decyzji oraz długość okresu, na który wypracowana decyzja ma zastosowanie.

**Zakres informacji wykorzystywanych przez algorytmy modułu AIL do wypracowania decyzji (reguł alokacji i relokacji plików)**

Możliwa liczba plików w strumieniu wyklucza możliwość operowania na wszystkich plikach, co z kolei wyklucza możliwość ustalanie bieżącego stanu macierzy pamięci i wypracowania decyzji z wykorzystaniem pełnej informacji o strumieniu plików. Dla efektywnej pracy algorytmów decyzyjnych i macierzy pamięci, stan bieżący oraz prognoza stanu ustalana jest i wypracowywana jedynie na wyselekcjonowanym, reprezentatywnym podzbiorze plików. Ponieważ każdy z plików w systemie jest opisany wektorem parametrów niosących istotne, różnicujące pliki informacje, efektywne operowanie nawet na takich podzbiorach również nie jest możliwe i dlatego w tym przykładzie wykonania zastosowano agregację informacji niesionej przez wyselekcjonowany podzbiór plików w modelu wielowymiarowej kostki. Każdy bok kostki reprezentuje pewien parametr. Długość boku kostki to zakres zmienności parametru. Zakresy zmienności parametrów dzielone są na odcinki, co prowadzi do podziału wielowymiarowej kostki na mniejsze kostki (dla wygody nazywamy je kategoriami). Pliki o parametrach mieszczących się w zakresach danej kategorii są do tej kategorii przypisywane i są dalej reprezentowane przez wartości parametrów przypisane kategoriom (w najprostszym przypadku wartość średnia dolnego i górnego zakres parametru danej kategorii). W ten sposób powstaje agregat strumienia informacji. Kostka może być parametryzowana na wiele sposobów: poprzez ilość wymiarów kostki, zakresy zmienności parametrów, skale zmienności parametrów w zakresie (równomierna, potęgowa, ekspotencjalna).

**Wypracowanie decyzji**

Wypracowania wzorca optymalnego przydziału plików do nośników pamięci na agregacie strumienia plików można dokonać jedynie przy odpowiednim stopniu agregacji informacji o strumieniu plików. Problem przypisywania plików bez pełnej informacji do nośników rozwiązano za pomocą uczenia maszynowego. Na agregacie strumienia plików wyuczono klasyfikator jak przydzielać do nośników pliki nie niosące pełnej informacji. W tym celu zastosowano standardowe podejście uczenia maszynowego: na wejście klasyfikatora podawano parametry plików należących do podzbioru plików, który został użyty do wytworzenia agregatu. Etykietami obiektów (tu: plików) wymaganych przy nauczaniu z nadzorem, były wyniki optymalizacji, a więc każdy plik z podzbioru plików miał przypisany nośnik wskazany przez optymalizator. Klasyfikacja była poprawna, jeżeli plik został zaklasyfikowany do tej samej kategorii. Proces uczenia klasyfikatora oparty jest na obserwacji tzw. funkcji kosztu. Uczenie kończy się, gdy nie następuje istotna poprawa wartości tej funkcji.

Problem przypisywania plików z pełną informacją do nośników rozwiązano, posiłkując się agregatem. Dla danego pliku, na podstawie jego parametrów określa się, do jakiej kategorii należy ten plik. Znając wyniki optymalizacji (przypisanie kategorii plików do nośników), przypisanie nośnika do pliku jest natychmiastowe.

Mając optymalny przydział kategorii plików do nośników, wypracowany na etapie optymalizacji, przydział plików bez pełnej informacji do kategorii, wypracowany na etapie uczenia maszynowego w formie klasyfikatora, oraz przydział plików do nośników oparty na agregacie, można było utworzyć reguły przydziału dowolnego pliku do najwłaściwszego, w sensie przyjętych kryteriów pracy macierzy pamięci, nośnika.

Algorytmy decyzyjne wykorzystują bezpośrednio techniki agregacji danych i sztuczne sieci neuronowe. Rozwiązanie hybrydowe, polegające na połączenie tych dwóch elementów z metodami optymalizacji, dla uzyskania konkretnych funkcjonalności w rozważanym zastosowaniu, jest rozwiązaniem nowatorskim. Schemat ideowy działania algorytmów decyzyjnych w systemie i sposobie według wynalazku przedstawiono na fig. 2.

**Dane wejściowe modułu AIL**

Informacje wejściowe dla AIL pobierane od DMFS zawarte są w:

- tabeli plików (Tabela 2),
- tabeli nośników (Tabela 3),
- zestawie wag kryteriów (Tabela 4).

**Tabela 2**. Parametry plików wymagane do pracy modułu AIL

| **Nazwa parametru/akronim** | **Znaczenie** | **Typ** | **Uwagi** |
|---|---|---|---|
| IDENTYFIKATOR<br>**ID** | Identyfikator pliku. | integer > 0 | Niezmienny. |
| CIĄG IDENTYFIKATORÓW KATALOGÓW<br>**CIDK** | Ciąg identyfikatorów katalogów. | Lista integer > 0 | Identyfikatory katalogów w ścieżce dostępu do pliku, np. [34 56 67 89], gdzie 34 oznacza katalog stojący najwyżej w hierarchii, a 89 – najniżej. |
| IDENTYFIKATOR NOŚNIKA<br>**IDN** | Identyfikator nośnika. | integer > 0 lub 0:= ∅ | ∅ - nośnik nie jest określony dla nowych plików. |
| ROZSZERZENIE<br>**EX** | Rozszerzenie pliku. | tekst | Ciąg znaków od ostatniej kropki występującej w nazwie pliku do ostatniego znaku (włącznie) nazwy pliku, np. „.doc", „.". |
| TYP DANYCH<br>**TD** | Faktyczny typ danych zawartych w pliku. | tekst lub „":= ∅ | Np.: PDF, JPEG. ∅ - typ danych nie jest określony dla nowych plików. |
| WŁAŚCICIEL<br>**IW** | Identyfikator właściciela pliku. | integer > 0 | UID (User ID). |
| BEZPIECZEŃSTWO<br>**BP** | Parametr określający pożądany stopień ochrony pliku (im większa wartość, tym pożądany stopień ochrony większy). | integer [0;100] lub -1:= ∅ | ∅ - Administrator nie podał żadnej wartości. |
| UPRAWNIENIA<br>**UP** | Tablica uprawnień zapisana na 4 bajtach w postaci int. | Integer > 0 | |

| | | | |
|---|---|---|---|
| SZYFROWANIE<br><br>**SzP** | Flaga oznaczająca, czy plik ma być szyfrowany. | Liczba 0, lub 1 | |
| INTENSYWNOŚĆ ODCZYTU<br><br>**IO** | Parametr określający przewidywaną intensywność odczytu danych z pliku. | integer [0;100] lub -1:= ∅ | ∅ - Administrator nie podał żadnej wartości. |
| INTENSYWNOŚĆ ZAPISU<br><br>**IZ** | Parametr określający przewidywaną intensywność zapisu danych z pliku. | integer [0;100] lub -1:= ∅ | ∅ - Administrator nie podał żadnej wartości. |
| PLIK TYLKO DO ODCZYTU<br><br>**PTO** | Flaga określająca czy plik jest tylko do odczytu. | Liczba 0, lub 1 | |
| PLIK NIE PODLEGAJĄCY RELOKACJI<br><br>**HOLD** | Flaga określająca czy plik podlega relokacji. | Liczba 0, lub 1 | |
| CZAS OSTATNIEJ RELOKACJI<br><br>**COR** | Czas ostatniej relokacji pliku. | data_czas lub 0:= ∅ | Wartość przekazywana w strukturze time_t. ∅ - plik nie był relokowany. |
| CZAS OSTATNIEGO DOSTĘPU<br><br>**CUP** | Czas ostatniego dostępu do pliku | data_czas lub 0:= ∅ | Wartość przekazywana w strukturze time_t. ∅ - dla nowego pliku. |
| HISTORIA WIELKOŚCI PLIKU<br><br>**HWP** | Historia zmian wielkości pliku w formie listy par: (czas operacji zmieniającej wielkość pliku, wielkość pliku). | tablica h par (data_czas, real) | h jest maksymalną przechowywaną liczbą par (pojemność) na nośniku.<br><br>Dla nowych plików na liście umieszczana jest para (data_czas, 0). |

| | | | |
|---|---|---|---|
| HISTORIA OPERACJI ODCZYTU<br>**HOO** | Historia operacji odczytów pliku. | tablica h par (data_czas, real)<br>lub lista pusta [] | Dla nowych plików na liście umieszczana jest para (data_czas, 0). |
| HISTORIA OPERACJI ZAPISU<br>**HOZ** | Historia operacji zapisów do pliku . | tablica h par (data_czas, real)<br>lub lista pusta [] | Dla nowych plików na liście umieszczana jest para (data_czas, 0). |
| **Parametry wyliczane przez AIL** | | | |
| ŚREDNIA WIELKOŚĆ PLIKU<br>**SWP** | Średnia wielkość pliku w epoce. | real | |
| LICZBA OPERACJI ODCZYTU PLIKU<br>**LOO** | Liczba operacji odczytu z pliku w epoce. | integer | |
| WOLUMEN ODCZYTÓW PLIKU<br>**WO** | Wielkość danych odczytanych z pliku w epoce. | real | |
| LICZBA OPERACJI ZAPISU PLIKU<br>**LZAP** | Liczba operacji zapisu pliku w epoce. | integer | |
| WOLUMEN ZAPISÓW PLIKU<br>**WZ** | Wielkość danych zapisanych do pliku w epoce. | real | |

**Tabela 3**. Parametry nośników, wymagane do pracy AIL

| **Nazwa parametru/akronim** | **Znaczenie** | **Typ** |
|---|---|---|
| IDENTYFIKATOR<br>**ID** | Identyfikator nośnika. | integer lub tekst |
| POJEMNOŚĆ AIL<br>**PAIL** | Pojemność przestrzeni do dyspozycji AIL. | liczba |
| PRZESTRZEŃ ZAJĘTA AIL<br>**ZPAIL** | Stopień zajętości pojemności AIL przez wszystkie pliki na nośniku. | liczba |
| KOSZT<br>**KOSZT** | Koszt nabycia przestrzeni do dyspozycji AIL. | integer |
| OPÓŹNIENIE DOSTĘPU<br>**ODS** | Średni czas oczekiwania przed rozpoczęciem każdej operacji odczytu/zapisu. | integer |
| PRĘDKOŚĆ ODCZYTU **PODCZ** | Średnia szybkość odczytu. | liczba |
| PRĘDKOŚĆ ZAPISU **POZ** | Średnia szybkość zapisu. | liczba |
| ZUŻYCIE ENERGII - ODCZYT<br>**ZEO** | Pobór mocy przy odczycie danych ponad pobór w trybie spoczynku. | real |
| ZUŻYCIE ENERGII - ZAPIS<br>**ZEZ** | Pobór mocy przy zapisie danych ponad pobór w trybie spoczynku. | real |

| | | |
|---|---|---|
| SZYFROWANIE<br>**SzN** | Określa, czy nośnik wykonuje szyfrowanie na poziomie sprzętowym. | Boolean |
| BEZPIECZEŃSTWO<br>**BN** | Określa poziom bezpieczeństwa nośnika. | integer |
| GABARYTY<br>**GAB** | Gabaryty nośnika.<br>Interpretację tego parametru określa wzór obliczania kryterium GĘSTOŚĆ UPAKOWANIA. | real |
| WIBRACJE<br>**WIB** | Określa poziom wibracji nośnika.<br>Interpretację tego parametru określa wzór obliczania kryterium WIBRACJE NOŚNIKÓW. | real |
| ZUŻYCIE<br>**SZ** | Stopień zużycia nośnika. | integer<br>[0;100] |

**Tabela 4**. Opis wag kryteriów wymaganych do pracy AIL

| Nazwa | Znaczenie | Typ | Zakres |
|---|---|---|---|
| KOSZT | Koszt przestrzeni zajętej przez AIL | **liczba** | Min = 0.00;<br>Max = 1.00 |
| ENERGIA | Łączne zużycie energii przez nośniki ponad tryb spoczynku | **liczba** | Min = 0.00;<br>Max = 1.00 |
| SSO | Średnia szybkość odczytu danych z plików. | **liczba** | Min = 0.00;<br>Max = 1.00 |
| SSZ | Średnia szybkość zapisu danych do plików. | **liczba** | Min = 0.00;<br>Max = 1.00 |

| | | | |
|---|---|---|---|
| SOD | Średnie opóźnienie dostępu do plików | **liczba** | Min = 0.00; Max = 1.00 |
| GUP | Średnia gęstość upakowania danych na nośnikach. | **liczba** | Min = 0.00; Max = 1.00 |
| WIB | Średni poziom wibracji nośników. | **liczba** | Min = 0.00; Max = 1.00 |

**Działanie AIL**

Na podstawie historii wartości parametrów każdego pliku ujętego w TABELI PLIKÓW (Tabela 2), AIL będzie obliczał wartości tzw. parametrów wyliczanych pliku (parametry wyliczane są na podstawie parametrów: HWP, HOO, HOZ). Parametry wyliczane plików zidentyfikowano dla potrzeb wypracowywania przez AIL reguł alokacji i relokacji. Dla potrzeb wypracowywania tychże reguł, komponent ten uwzględnia także inne parametry plików.

W tym przykładzie wykonania na podstawie informacji wejściowych: parametry plików, parametry nośników oraz zestaw wag kryteriów (Tabele 2–4), moduł AIL tworzy reguły, aktualizowane co epokę, które rekomendują alokację i relokację plików na nośnikach. Długość epoki może być zmienna i jest powiązana z aktywnością samego Systemu. Będzie skracana w sytuacji, gdy zestaw plików wymagany do uruchomienia AIL (sprawdzany w funkcji Continue_Work) został wygenerowany odpowiednio szybko – w przeciwnym wypadku, będzie wydłużana.

Reguły pozwalają na wskazanie nośników (wskazanie ID nośników) na podstawie przedstawionych parametrów pliku. Moduł AIL podejmuje więc (wielowariantowe) decyzje o rozmieszczeniu plików na nośnikach.

W tym przykładzie wykonania w systemie według wynalazku zastosowano sztuczną sieć neuronową typu perceptron wielowarstwowy (ang. MultiLayer Perceptron – MLP; Micheli-Tzanakou, E., (red.), 2000, "Supervised and Unsupervised Pattern Recognition: feature Extraction and *Computational Intelligence – Feature Extraction and Computational Intelligence", CRC Press, Boca Raton*), w której przepływ sygnałów jest jednokierunkowy – od warstwy wejściowej, poprzez warstwę ukrytą, do warstwy wyjściowej. Sygnałami wejściowymi są parametry plików, a sygnałami wyjściowymi są identyfikatory nośników. Zbiór uczący perceptronu jest wynikiem działania Bloku Optymalizacji. Sieć MLP tworzona jest wraz z każdym wywołaniem komponentu AIL, tak więc jej struktura ulega zmianie z wywołania na wywołanie tegoż komponentu. Domyślnie stosowana jest jedna warstwa ukryta, ale rozwiązanie według wynalazku przewiduje zastosowanie większej ich liczby. Minimalna liczba warstw ukrytych to 1, maksymalna warstwa liczb ukrytych to 3. Schemat użytego w rozwiązaniu perceptronu wielowarstwowego przedstawiono na fig. 3.

**Dane wyjściowe modułu AIL**

Dane wyjściowe modułu AIL są jednocześnie danymi wejściowymi modułu DMFS. Moduł AIL przekazuje do systemu plików DMFS reguły:

a. alokacji w postaci:
   - tabeli zawierającej listę plików (ID pliku) i reguł relokacji dla każdego z nich w postaci identyfikatora nośnika, na który ma być plik przeniesiony; oraz

b. relokacji w postaci:
   - pliku binarnego, zawierającego wytrenowaną sieć neuronową (perceptronową) typu MLP (Multi-Layered Perceptron),

- pliku binarnego, zawierającego koder etykiet, wykorzystany do kodowania etykiet przy uczeniu sieci MLP,
- pliku binarnego, zawierającego koder zmiennych nominalnych, wykorzystany do kodowania etykiet przy uczeniu sieci MLP.

**Algorytm A_R, tworzący reguły alokacji/relokacji przez AIL**

W Systemie zaimplementowana jest KOLEJKA RELOKACJI, którą obsługuje DMFS poprzez swój komponent składowy o roboczej nazwie RELOKATOR. Algorytm A_R stanowi wątek, który może być uaktywniany okresowo. Na początku każdej epoki AIL wykonuje Algorytm A_R. Przy pierwszym uruchomieniu algorytmu algorytm nie ma dostępu do żadnych informacji historycznych.

Algorytm A_R korzysta z następujących funkcji:

- Continue_Work,
- Sleep_AIL,
- Build_Category_Set,
- Select_Files,
- Solve_Optimization_Problem,
- Make_Allocation_Rules,
- Reallocate_File,
- Enqueue,
- Save_Category_Set,
- Inform_DMFS,

Algorytm A_R korzysta z następujących zmiennych globalnych:

- TABELA PLIKÓW,
- TABELA NOŚNIKÓW,
- KOLEJKA RELOKACJI,
- WAGI,
- HISTORYCZNE KATEGORIE,
- TABELA ZDATNYCH PLIKÓW.

Pierwsze cztery zmienne to struktury danych implementowane po stronie DMFS (do których dostęp ma AIL), a kolejne trzy – zmienne, utworzone na potrzeby Algorytmu A_R.

**Przykładowy pseudokod Algorytmu A_R**

```
Begin
TABELA PLIKÓW := DMFS.Get_File_Table(ø); // uzyskaj dostęp do wszystkich zapisów
                                          w TABELI PLIKÓW
// Wybierz tylko te pliki, dla których zgromadzono dostatecznie dużą ilość danych
TABELA ZDATNYCH PLIKÓW := Select_Files(PARAMETRY WYBORU);
if Continue_Work(PARAMETRY KONTYNUACJI) == false then
Sleep_AIL(EL, true); // AIL zasypia na kwant czasu EL (domyślna długość epoki) lub pochodną
                     tego kwantu ustaloną wewnątrz procedury Sleep_AIL;
else // AIL kontynuuje pracę, gdyż w TABELI PLIKÓW zebrano dostatecznie wiele informacji
     wymaganych do opracowania reguł alokacji/relokacji
Inform_DMFS(SYGNAŁ:ALOKACJA_W_TOKU); // Poinformuj DMFS, że ustalanie reguł alokacji
                                        jest w toku
WAGI := DMFS.Get_Weights();          // pobierz wagi kryteriów
// Zbuduj zestaw kategorii plików o zadanych parametrach (parametry dyskretyzacji etc.) na
  podstawie aktualnych danych o plikach oraz na podstawie historycznych kategorii plików
C_SET := Build_Category_Set(PARAMETRY BUDOWY KATEGORII);
// Rozwiąż problem optymalizacyjny przydziału kategorii plików (uwzględniając wagi kryteriów) do
nośników
ROZWIĄZANIE := Solve_Optimization_Problem(C_SET);
// Na podstawie otrzymanego ROZWIĄZANIA oraz innych danych, dostępnych podczas pracy
tego algorytmu, opracuj struktury danych określające reguły alokacji/relokacji
Make_Allocation_Rules(); // ustalone działanie funkcji: AIL.Allocate_File, AIL.Reallocate_File
// Relokuj pliki
KANDYDACI DO RELOKACJI := Zbiór tych plików z tabeli ROZWIĄZANIE,
                          dla których IDN <> IDNOP
```

```
for each f in KANDYDACI DO RELOKACJI do
RELOKUJ := Reallocate_File(f.ID); // podejmij decyzję dla pliku f
if RELOKUJ nie jest listą pustą then
        Enqueue(RELOKUJ); // wstaw decyzję do KOLEJKI RELOKACJI
done
        Save_Category_Set(C_SET);  // Uaktualnij historię AIL'a
Inform_DMFS(SYGNAŁ:ALOKACJA_GOTOWA); // Poinformuj DMFS, że reguły
                                                alokacji/relokacji są dostępne
        Sleep_AIL(EL, false);
fi
End
```

Po każdym wykonaniu Algorytmu A_R, DMFS korzysta z uaktualnionych reguł alokacji/relokacji, dostępnych poprzez wywołania funkcji AIL.Allocate_File/AIL.Reallocate_File, implementowanych w interfejsie AIL-DMFS.

Reguły relokacji dla plików z tabeli KANDYDACI DO RELOKACJI, w postaci wyników działania funkcji Reallocate_File, umieszczone są w KOLEJCE RELOKACJI, którą stale obsługuje RELOKATOR (w sposób zależny od obciążenia Systemu).

**Bloki modułu AIL**

Od strony funkcjonalnej i algorytmicznej moduł AIL składa się z trzech bloków:

- bloku agregacji,
- bloku wyznaczania wzorca,
- bloku reguł.

**Blok agregacji (kategoryzacja), BA**

W tym bloku, na podstawie charakterystyk plików zawartych w TABELI ZDATNYCH PLIKÓW, pliki są agregowane w kategorie. Ma to na celu zmniejszenie rozmiaru danych przekazywanych do kolejnego bloku – bloku wyznaczania wzorca.

Możliwe do zastosowania w tym bloku mechanizmy agregacji to wszystkie metody sztucznej inteligencji, określane terminem klasteryzacja. Zastosowano metodę kategoryzacji polegającej na alokacji plików wejściowych do elementów wielowymiarowej tablicy (hiperkostka) o wymiarze liczby rozpatrywanych parametrów wyliczanych plików. Zarówno wymiar tablicy jak i podział zakresu wartości parametrów dla każdego pojedynczego wymiaru są parametrami, które mogą podlegać strojeniu.

Zaletą przyjętej metody jest to, że przyjęty w niej model struktury danych jest intuicyjny i interpretowalny, pozwalający zatem na jego adaptację i dostrajanie według wiedzy eksperckiej. W szczególności obsługiwana jest sytuacja, w której ze względu na charakter danych (brak operacji zapisu na plikach dla dużej liczby elementów w TABELA ZDATNYCH PLIKÓW) początkowo zakładana liczba kategorii w hiperkostce nie jest możliwa do utworzenia. W takiej sytuacji liczba kategorii adaptowana jest w sposób automatyczny do rodzaju danych przekazanych do AIL.

Drugą zaletą przyjętej metody jest jej skalowalność, tzn. możliwość jej efektywnej pracy na zbiorach plików o wielkiej liczności. Czas działania funkcji w bloku agregacji jest liniowo zależny od rozmiaru TABELI ZDATNYCH PLIKÓW.

Wyniki testów mogą wskazać na celowość zmiany metody kategoryzacji dla uzyskania lepszej charakterystyki pracy systemu w aspekcie efektywności działania AIL względem nakładu obliczeń.

Proces generowania zestawu kategorii plików (hiperkostki) składa się z etapów wykonywanych sekwencyjnie realizowanych w kolejnych metodach opisanych poniżej:

**Funkcja Binary_API(String wej, String data_dir**): Funkcja, w której realizowane są kolejno dwie operacje. Po pierwsze, pobierane są informacje o plikach w formacie binarnym zapisywane następnie do pliku Tabela_plikow.txt. W dalszej części funkcji z pliku binarnego pobierane są informacje dotyczące dostępnych podczas optymalizacji dysków. Informacje te zapisywane są do pliku disks.csv znajdującego się w katalogu "data_dir" podanym jako parametr funkcji.

INPUT:

- String wej – nazwa pliku binarnego, z którego pobierane są dane;
- String data_dir – katalog z danymi, do którego zapisywane są informacje pobrane z pliku binarnego: dane dotyczące plików (zapisywane w pliku Tabela_plikow.txt" oraz dane dotyczące dysków).

OUTPUT – na wyjściu funkcji nie jest zwracana żadna informacja. Przy czym po zakończeniu działania funkcji na podstawie pliku binarnego z danymi źródłowymi budowany jest plik "Tabela_plikow.txt".

Funkcja Read_Wszystkie_Pliki():

INPUT – void;

OUTPUT – tablica String[] TABLICA_PLIKOW zawierająca wszystkie opisy plików (włącznie z tymi, dla których historia jest zbyt krótka, żeby wyliczyć parametry wyliczane).

Funkcja String[] Select_Zdatne_Pliki(String[] TABLICA_PLIKOW)

INPUT – parametr TABLICA PLIKOW, który zawierający opisy plików przekazane do AIL.

OUTPUT - tablica zdatnych plików.

Funkcja Set_TINT(int[] par_, String[] TABLICA_ZDATNYCH_PLIKOW):

INPUT: int[] par_ – informacja o tym, na ile interwałów dzielony będzie każdy z zakresów parametrów wyliczanych. Dostępnych jest 5 parametrów, więc wielkość tej tablicy, to zawsze 5. Domyślnie ustalony jest podział każdego parametru na 5 części (wartość może zostać zmieniona i ustalana jest jako parametr). Drugi parametr, to tablica zdatnych plików, z której pobierane są parametry wyliczane wszystkich plików. Każdy z parametrów znajduje się w osobnej tablicy, która następnie jest sortowana. Tablica double[] SWP przechowuje wartości parametru wyliczanego SWP.

Opcjonalnie możliwe jest wykorzystanie funkcji set_TINT_from_file(...), w której na podstawie zmiennych "par_" oraz "ile_interwalow" budowana jest tablica TINT, przy czym wartości przedziałów odczytywane są z pliku Intervals.txt.

OUTPUT – tablica TINT obiektów klasy Par_interval opisująca wszystkie wyliczone przedziały. Klasa ma dwa pola publiczne String: dol i gora, które wskazują odpowiednio dolną i górną granicę interwału. Do parametrów przygotowane są gettery i settery oraz funkcja Get_interval(), która zwraca interwał w postaci "[" + dol + + gora + "]"

Funkcja void Create_Vocabularies(Par_interval[][] TINT)

INPUT: funkcja przyjmuje tablicę 2D obiektów klasy Par_interval, na podstawie której budowane są wszystkie słowniki.

OUTPUT – Funkcja nic nie zwraca, ale po jej wywołaniu zostaje utworzonych 6 słowników globalnych.

Funkcja do budowy wszystkich słowników. Słowniki będą stosowane przy odwoływaniu się do poszczególnych kategorii. Zakładamy, że mamy 3 możliwości adresowania kategorii:

- indeks globalny;
- indeksy parametrów;
- interwały parametrów.

Przykładowo zakładając, że dostępnych jest 3125 kategorii (5 parametrów, każdy dzielony na 5 interwałów, co daje 5*5*5*5*5 możliwych kategorii), to:

- indeks globalny wskazuje konkretny numer kategorii. Kategoria opisywana jest przez klasę TCell, a wszystkie kategorie trzymane są w tablicy jednowymiarowej o wielkości 3125 (lub innej, w zależności, od liczby interwałów dla parametrów – to można zmieniać).
- indeksy parametrów: danych jest 5 parametrów wyliczanych, każdy podzielony (przykładowo) na 5 interwałów. Można zatem odwoływać się do kategorii opisanej pięcioma indeksami (pięciu kolejnych parametrów). Przykładowo: 4 3 0 0 0 oznacza kategorię, gdzie dla pierwszego parametru (SWP) pod uwagę brany jest interwał ostatni (indeksowanie od zera), dla drugiego parametru interwał czwarty, a dla pozostałych trzech – interwały pierwsze. Ten sposób adresowania przedstawiony jest jako String.
- interwały parametrów – podobnie, jak wyżej, ale zamiast indeksów wskazywane są konkretne zakresy interwałów dla każdego z parametrów wyliczanych. Przykładowe adresowanie może być następujące:

[20.072:39.688] [1201.0:2394.0] [0.0:2.968] [238.0:357.0] [100.226:INF]

Zbudowanych jest łącznie 6 słowników (struktura mapy):

- przedzial_indeksy_przedzialow – podając interwały parametrów otrzymywane są indeksy parametrów;

Przykład:

[20.072:39.688] [3615.0:4799.0] [8.972:11.965] [478.0:INF] [0.0:25.165] → 1 3 3 40

przedzial_indeks – podając interwały przedziałów otrzymywany jest indeks globalny kategorii;

Przykład:

[59.509:79.927] [3615.0:4799.0] [2.968:5.964] [478.0:INF] [25.165:50.17] → 1168

- indeksy_przedzialow_przedzial – podając indeksy parametrów otrzymywane są interwały parametrów;

Przykład:

4 1123 → [79.927:INF] [1201.0:2394.0] [2.968:5.964] [238.0:357.0] [75.718:100.226]

- indeksy_przedzialow_indeks - podając indeksy parametrów otrzymywany jest indeks globalny kategorii;

Przykład:

3 4 4 3 2 → 1748

indeks_przedzial – podając indeks globalny kategorii otrzymywane są interwały parametrów;

Przykład:

707 → [39.688:59.509] [1201.0:2394.0] [8.972:11.965] [0.0:117.0] [25.165:50.17]

indeks_indeksy_przedzialow – podając indeks globalny kategorii otrzymywane są indeksy parametrów:

Przykład:

303 → 3 0 2 2 0

Zarówno przy indeksach parametrów, jak i przy interwałach parametrów dostępny jest zestaw tylu elementów, ile danych jest kategorii. W samej funkcji wykorzystywane są dwie metody pomocnicze do generowania zestawów:

- String[] generate_przedzialy(Par_interval[][] sets) – funkcja z tablicy Par_interval buduje wszystkie dopuszczalne przedziały i zwraca w formie tablicy łańcuchów;
- String[] generate_indeksy(Par_interval[][] sets) – jak wyżej, ale zwraca indeksy przedziałów w formie tablicy łańcuchów.

Finalnie, na podstawie OUTPUT z tych dwóch powyższych funkcji budowane są sowniki.

Funkcja void Initialize_Category_set(int ile_kategorii)

INPUT Informacja o tym, ile kategorii ma zostać zbudowanych, czyli długość tablicy obiektów TCell.

OUTPUT Funkcja zwraca void, ale budowana jest tablica obiektów klasy TCell.

Funkcja wywoływana przed przydzieleniem plików do poszczególnych kategorii. Tablica kategorii składa się z obiektów, więc w tej funkcji tworzona jest tablica pustych obiektów.

Klasa TCell:

Obiekty klasy TCell przechowują pełną informację o jednej kategorii. Posiadają następujące pola:

- private String klucz – zestaw interwałów przedziałów opisujących daną kategorię, np.: [39.688:59.509] [1201.0:2394.0] [8.972:11.965] [0.0:117.0] [25.165:50.17];
- private String indeksy – zestaw indeksów przedziałów parametrów, np. 1 2 3 1 0;
- private int index – globalny indeks kategorii.
- public ArrayList<double[]> lista_parametrow_wyliczanych – do tej tablicy dynamicznej dodawane są parametry wyliczane wszystkich plików, które wpadły do danej kategorii;
- public ArrayList<String> full_file_description – do tej tablicy dynamicznej dodawane są pełne opisy plików z TABLICA_ ZDATNYCH_PLIKOW, dla których parametry wyliczane wpadły do przedziału tej kategorii;
- public int NofFiles – łączna liczba plików w tej kategorii.

Konstruktor klasy public TCell(int indeks_globalny)

W konstruktorze inicjalizowane są pola:

this.lista_parametrow_wyliczanych = new ArrayList<>() – pusta tablica dynamiczna

this.full_file_description = new ArrayList<>() – pusta tablica dynamiczna

this.klucz = Category_set.indeks_przedzial.get(indeks_globalny) – przedziały parametrów wyznaczone na podstawie słownika. Mamy już informację o przedziałach wszystkich plików, zatem możemy korzystać ze słownika wyznaczającego na podstawie indeksu globalnego przedziały parametrów;

this.indeksy = Category_set.indeks_indeksy_przedzialow.get(indeks_globalny) – jw. ale tym razem dla indeksów przedziałów.

this.index = indeks_globalny;

Gettery: public String Get_indeksy(),public String Get_klucz(), public int Get_index().

public void Add_parameters(double[] tab) – zestaw parametrów wyliczanych pliku dodajemy do naszej tablicy dynamicznej przez funkcję Add_parameters. Jej parametrem jest tablica pięciu elementów (kolejne parametry wyliczane pliku przydzielonego do tej kategorii). Dla uproszczenia double [], ale w środku są też inty.

Dalej dostępnych jest zestaw metod:

public double Get_sumSWP()

public double Get_sumLOO()

public double Get_sumWO()

public double Get_sumWZ()

public double Get_sumZAP()

wyznaczających sumy poszczególnych parametrów wyliczanych dla plików z danej kategorii. Budowa zestawu kategorii plików (funkcja Build_Category_Set) została opisana w dalszej części opracowania.

Dla celów badawczych programy w tym bloku zostały zrealizowane w języku Java.

**Blok wyznaczania wzorca (optymalizacja), BWW**

W systemie według wynalazku, ze względu na a priori nieznane, zmienne w czasie natężenie strumienia obsługiwanych plików, zmienne w czasie charakterystyki plików, a także zmienne w czasie charakterystyki nośników pamięci optymalny wzorzec (w czasie) rozkładu plików pomiędzy nośniki, obowiązujący na pewien okres w przód, należy wypracować na podstawie bieżącego stanu sytemu.

W tym celu charakterystyki pewnego podzbioru plików (TABELA ZDATNYCH PLIKÓW), kierowane są do modułu AIL celem wypracowania reguł alokacji (przydział do nośnika pamięci pliku pojawiającego się w systemie pamięci po raz pierwszy) i relokacji (przemieszczanie plików pomiędzy nośnikami). Na tym podzbiorze plików wypracowany jest wzorzec optymalnego rozkładu plików pomiędzy nośniki. Zadaniem bloku optymalizacji jest wypracowanie takiego wzorca.

W tym bloku zastosowano mechanizmy wypracowania optymalnego wzorca metodami optymalizacji dokładnej, w których, w zależności od rozmiaru i złożoności zadania optymalizacyjnego możliwe jest uzyskanie wzorca optymalnego lub, o ile obliczenia optymalizacyjne nie mogą być prowadzone odpowiednio długo, tylko suboptymalnego.

***Blok reguł (uczenie maszynowe), BWW***

Wypracowanie w bloku BWW wzorca (optymalnego lub suboptymalnego) rozkładu plików pomiędzy nośniki pozwala na zastosowanie metod uczenia maszynowego do wypracowania reguł alokacji i relokacji plików w systemie. Podstawą do uczenia maszynowego jest założenie, że charakterystyki plików rozpatrywanych przez AIL są reprezentatywne dla wszystkich plików w systemie macierzy nośników. Innymi słowy, gdyby populacja plików w systemie miała te same statystyczne własności co zbiór plików rozpatrywanych przez moduł AIL, to wypracowane przez AIL metodami uczenia maszynowego reguły byłyby statystycznie optymalne.

W tym bloku wypracowywany jest mechanizm (automat) przydziału plików do nośników najbardziej zgodny z wypracowanym w bloku BWW wzorcem. Jest to istota uczenia maszynowego. Tak wypracowany mechanizm jest następnie stosowany do całej populacji plików w systemie do bieżącej alokacji i relokacji plików.

Dla uczenia maszynowego wykorzystano sieć neuronową. Wykorzystano architekturę sieci typu perceptron wielowarstwowy.

**Funkcje modułu AIL**

**Funkcja AIL.Allocate_File** (Parametry pliku BP, IO, IZ, CIDK, EX, TD, SzP, IW, UP, PTO) → Reguła alokacji (hierarchiczna lista nośników).

Funkcja używa utworzonych w funkcji Make_Allocation_Rules i zapisanych przez nią do plików obiektów binarnych: kodera zmiennych nominalnych (KODER_KATEGOR), kodera etykiet (KODER_ETYKIETY) oraz perceptronu wielowarstwowego (MLP). Obiekty te muszą być wczytane do pamięci, aby można było wyznaczyć regułę alokacji. Wyznaczenie reguły alokacji odbywa się według następujących kroków.

1. **Kodowanie CIDK**. Ponieważ CIDK jest listą zmiennej długości, dla potrzeb klasyfikacji parametr ten zamieniany jest na MAX_KAT_GLEB (> 0) parametrów $SCIDK_1$, ... , $SCIDK_{MAX\_KAT\_GLEB}$. Jeżeli rozmiar(CIDK) < MAX_KAT_GLEB, to parametry $SCIDK_1$, ... , $SCIDK_{roz-}$

$_{miar(CIDK)}$ przyjmują wartości odpowiednich elementów z listy CIDK, a parametry $SCIDK_{roz-miar(CIDK)+1}$, ... , $SCIDK_{MAX\_KAT\_GLEB}$ przyjmują wartość '0'. Jeżeli długość CIDK ≥ MAX_KAT_GLEB, to $SCIDK_1=CIDK_1$, ... , $SCIDK_{MAX\_KAT\_GLEB}=CIDK_{MAX\_KAT\_GLEB}$.

2. **Podział parametrów**. Parametry plików dzielone są na dwa typy: parametry ciągłe (BP, IO, IZ); parametry nominalne ($SCIDK_1$, ... , $SCIDK_{MAX\_KAT\_GLEB}$, EX, TD, SzP, IW, UP, PTO).
3. **Kodowanie parametrów**. Parametry nominalne kodowane są koderem KODER_KATEGOR. Obiekt do rozpoznania (wektor X) tworzą: parametry ciągłe, zakodowane parametry nominalne.
4. **Klasyfikacja**. PREDYKCJA := MLP.predict(X). PREDYKCJA jest wektorem prawdopodobieństw, którego współrzędna i określa prawdopodobieństwo przynależności opisanego parametrami wejściowymi pliku do klasy i. Koder KODER_ETYKIETY pozwala jednoznacznie stwierdzić, jaki identyfikator nośnika odpowiada współrzędnej i.
5. **Szeregowanie identyfikatorów nośników**. Na podstawie wektora PREDYKCJA tworzona jest uszeregowana nierosnąco lista nośników (LN), a szeregowanie odbywa się na podstawie prawdopodobieństw przynależności klasyfikowanego obiektu do klas.
6. **Korekta ze względu na konieczność szyfrowania**. Z listy LN usuwane są te nośniki, które nie zapewniają szyfrowania sprzętowego. Po tej operacji lista LN jest regułą alokacji.

**Funkcja AIL.Reallocate_File** (ID istniejącego pliku) → Reguła relokacji

Zachowanie tej funkcji zależy od dostępności parametrów wyliczanych pliku o zadanym ID.

Jeżeli nie są znane parametry wyliczane pliku, to funkcja ta wykorzystuje funkcję AIL.Allocate_File. Wyznaczenie reguły relokacji odbywa się wtedy następująco.

1. Odczytanie wartości parametrów BP, IO, IZ, CIDK, EX, TD, SzP, IW, UP, PTO pliku o zadanym ID.
2. Reguła relokacji := AIL.Allocate_File(BP, IO, IZ, CIDK, EX, TD, SzP, IW, UP, PTO).

Jeżeli są znane parametry wyliczane pliku (W), to działanie funkcji jest następujące.

1. Na podstawie zestawu kategorii C_SET (wynik funkcji Build_Category_Set()) oraz W, ustal komórkę (kategorię plików) KOM, do której należy plik o zadanym ID.
2. Na podstawie ROZWIĄZANIA ustal identyfikator nośnika IDNKOM dla komórki KOM, będący jednocześnie identyfikatorem docelowego nośnika dla pliku o zadanym ID. Umieść IDNKOM na liście REGUŁA.
3. Ustalenie mniej preferowanych nośników. Ustal listę SĄSIEDZI_KOM wszystkich sąsiednich komórek komórki KOM. Dla każdej komórki K z listy SĄSIEDZI KOM oblicz: jej centrum C(K) w przestrzeni parametrów określających wymiary zestawu kategorii; dystans DYST(K) wektora parametrów W do C(K). Niech IDN(K) oznacza identyfikator nośnika przypisany do komórki K. Dodaj IDN(K) do listy REGUŁA dla wszystkich spotkanych nośników. W zależności od tego, do ilu z sąsiednich komórek jest przypisany nośnik IDN(K) i jakie te komórki mają dystanse DYST(K), ustal kolejność identyfikatorów nośników na liście REGUŁA tak, żeby na początku listy był nośnik najbardziej preferowany, a na końcu – najmniej. Jeżeli plik o zadanym ID wymaga szyfrowania, usuń identyfikatory nośników, które nie są szyfrowane. Reguła alokacji := REGUŁA.

Funkcja AIL.Reallocate_File() jest wywoływana przez DFMS dla plików, które znajdują się na nośnikach macierzy, a nie są dostępne w tabeli TABELA ZDATNYCH PLIKÓW (za relokację podzbioru plików tej tabeli odpowiada wewnętrzna funkcja Algorytmu A_R o nazwie Reallocate_File()). Umożliwia to zastosowanie przez DMFS reguł relokacji podczas przeglądania plików na macierzy, które spełniają zadane przez Administratora warunki (np. nie było do nich żądań dostępu w ciągu ostatnich trzech miesięcy).

**Wewnętrzne funkcje**

**Funkcja Select_Files**

CEL: Wybór plików z TABELI PLIKÓW, spełniających narzucone warunki.

INPUT: PRMW – parametr określające warunki wyboru.

OUTPUT: Podzbiór TABELI PLIKÓW.

OPIS DZIAŁANIA: Na podstawie TABELI PLIKÓW oraz PRMW, funkcja ustala listę plików, dla których zgromadzono wystarczającą ilość danych, przechowywanych w tablicach HWP, HOO i HOZ. Argument wejściowy PRMW jest argumentem złożonym, a jego składowe to, co najmniej: PRMW.min-

HWP, PRMW.minHOO, PRMW minHOZ. Składowe te oznaczają, odpowiednio, minimalną długość tablic HWP oraz jedną z dwóch: HOO lub HOZ, jakie muszą charakteryzować dany plik, aby mógł być on wybrany z TABELI PLIKÓW. Funkcja ta pozwoli wyszczególnić tylko te pliki, które ze względu na rozbudowaną historię poszczególnych parametrów, zawierają dostateczny dla potrzeb kategoryzacji plików ładunek informacyjny.

**Funkcja Build_Category_Set**

CEL: Wyznaczenie kategorii plików na podstawie TABELI ZDATNYCH PLIKÓW, HISTORYCZNYCH KATEGORII oraz zadanych parametrów.

INPUT: PRMH – parametr określający charakterystykę wynikowych kategorii plików.

OUTPUT: Kategorie plików.

OPIS DZIAŁANIA: Do budowy hiperkostki stosowana jest klasa pomocnicza Par_interval, przechowująca informację o podziale każdego z parametrów wyliczanych na zakresy. Liczba zakresów jest parametrem i może być dowolnie modyfikowana. Po ustaleniu liczby zakresów wyznaczane jest minimum i maksimum każdego z parametrów wyliczanych. Następnie przedział <minimum, maksimum> dzielony jest na pewną ustaloną liczbę przedziałów. Podział następuje w taki sposób, że do każdego z utworzonych przedziałów należy tyle samo plików. Procedura powtarzana jest dla każdego z parametrów wyliczanych. W sytuacji, kiedy dwa lub więcej przedziałów są równe, następuje zmniejszenie liczby przedziałów, na jakie podzielony zostanie dany parametr. Pierwszy element hiperkostki odpowiada sytuacji, kiedy dla każdego z parametrów wyliczanych rozważany jest jego pierwszy zakres. Ostatni element hiperkostki to sytuacja, kiedy analizowany jest ostatni zakres. Zatem, przykładowo, przy pięciu parametrach i podziale każdego parametru na 5 zakresów liczba komórek (kategorii) hiperkostki wynosi 5^5=3125.

Kolejnym etapem jest budowa słowników, umożliwiających odwołanie się do poszczególnych elementów hiperkostki. Łącznie budowanych jest 6 słowników umożliwiających dowolne rzutowanie zakresów przedziałów na numery przedziałów hiperkostki (i odwrotnie), a także na rzutowanie zakresów przedziałów na indeks globalny komórek hiperkostki (np., dla hiperkostki zbudowanej z 3125 komórek, każda z komórek ma swój unikalny indeks globalny, będący liczbą ze zbioru liczb {0,1,...,3124}).

Ostatni etap działania funkcji to analiza wszystkich plików znajdujących się w strukturze TABELA_ZDATNYCH_PLIKOW i ich przypisanie do odpowiedniej komórki hiperkostki. Przypisanie odbywa się poprzez sprawdzenie wartości wszystkich pięciu parametrów wyliczanych pliku. To sprawdzenie pozwala określić, w jakich interwałach znajdują się wartości tych parametrów, a tym samym umożliwia umiejscowienie pliku w odpowiedniej komórce hiperkostki. Wyjściowa struktura bloku agregacji zawiera informacje o wszystkich kategoriach, ich licznościach oraz parametrach plików przypisanych do danej kategorii. Dodatkowo istnieje możliwość sprawdzenia bezpośredniego sąsiedztwa każdej z kategorii w hiperkostce, co jest wykorzystywane przy ustalaniu reguł relokacji.

Rezultat wykonania funkcji Build_Category_Set jest podstawowym argumentem wejściowym funkcji Solve_Optimization_Problem.

**Funkcja Solve_Optimization_Problem**

CEL: Wyznaczenie optymalnego przydziału kategorii plików do nośników.

INPUT: Kategorie plików; dla każdej kategorii, dla każdego parametru wyliczanego plików, suma wartości parametru po wszystkich plikach w kategorii. Poniżej oznaczamy tę sumę tak samo jak odpowiedni parametr plików, czyli np. $WO_i$ to wielkość odczytu kategorii i.

OUTPUT: Optymalny przydział kategorii plików do nośników $(\tilde{X})$, czyli dla każdej kategorii, ID przydzielonego nośnika.

OPIS DZIAŁANIA: Funkcja ta ma za zadanie wyznaczyć taki przydział kategorii plików do nośników, który jest optymalny względem ważonej wagami agregacji wszystkich kryteriów. Każda kategoria jest interpretowana jako jeden zbiorczy plik, reprezentujący wszystkie pliki w kategorii.

**Formułowanie modelu matematycznego**

Używany model matematyczny to model programowania liniowego całkowitoliczbowego (binarnego), gdzie wszystkie kryteria przekształcone są w formę minimalizacji:

$$\text{minimalizuj } f_k(X) := \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^{k} x_{ij}, \quad k \in \overline{1,K} \tag{1}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i \in \overline{1,m} \tag{2}$$

$$SP \cdot MP \cdot \sum_{i=1}^{m} SWP_i x_{ij} \leq PAIL_j, \quad j \in \overline{1,n} \tag{3}$$

- *Przestrzeń rozwiązań* to zbiór macierzy binarnych $X = (x_{ij})_{m \times n}$, $x_{ij} \in \{0,1\}$, z ograniczeniami (2), gdzie m – liczba kategorii plików, n – liczba nośników. Interpretacja: $x_{ij} = 1$, wtedy i tylko wtedy, gdy kategoria i jest przydzielona do nośnika j. Ograniczenia (2) gwarantują że każda kategoria jest przydzielona do jednego nośnika.
- *Zbiór rozwiązań* dopuszczalnych to zbiór elementów przestrzeni spełniających dodatkowe ograniczenia (3) na nieprzekraczanie pojemności poszczególnych nośników.

  $MP = \sum_{j=1}^{n} PAIL_j / \sum_{i=1}^{m} SWP_i$ to 'multiplikator przestrzeni", 0<SP<1 to „skala przestrzeni". Interpretacja (np., w przypadku SP=0.5): *jeżeli reguły alokacji/realokacji na podstawie danego przydziału kategorii będą zastosowane do zbioru plików, których całkowity rozmiar stanowi 50% całkowitej przestrzeni, to żaden z nośników nie będzie przepełniony.* Celem tego ograniczenia jest zapobieganie sytuacji, kiedy wszystkie pliki będą przydzielone do jednego/kilku najlepszych nośników szybko wyczerpując przestrzeń. Jeżeli SP jest blisko 1, to zwęża się zbiór dopuszczalnych rozwiązań co może pogorszyć jakość optymalnego rozwiązania. Jeżeli SP jest blisko 0, to zbyt wiele plików będzie przydzielone do najlepszych nośników. Parametr SP jest ustalony empirycznie na poziomie 0.5.
- Każda funkcja celu $f_k$, k od 1 do K, reprezentuje jedno kryterium jakości działania macierzy. W naszym przypadku K=8. Współczynniki $c_{ij}$ są wyprowadzone z formuł odpowiednich kryteriów (rozdział **Błąd! Nie można odnaleźć źródła odwołania**.) po ich przekształceniu w formę liniowych funkcji minimalizacji jak poniżej

k=1 (KOSZT): $c_{ij}^{k} = SWP_i \cdot KOSZT_j / PAIL_j$

k=2 (ENERGIA): $c_{ij}^{k} = WO_i \frac{\text{ZEO}_j}{\text{PODCZ}_j} + WZ_i \frac{\text{ZEZ}_j}{\text{POZ}_j}$

k=3 (SSO): $c_{ij}^{k} = \frac{WO_i}{\text{PODCZ}_j}$

k=4 (SSZ): $c_{ij}^{k} = \frac{WZ_i}{\text{POZ}_j}$

k=5 (SOD): $c_{ij}^{k} = \frac{\text{ODS}_j (LOO_i + LZAP_i)}{\sum_{q=1}^{m} (LOO_q + LZAP_q)}$,

k=6 (BP): $c_{ij}^{k} = SWP_i (100 - SZ_j)/m$

k=7 (GUP): $c_{ij}^{k} = -\frac{SWP_i}{n \cdot GAB_j}$

k=8 (WIB): $c_{ij}^{k} = \frac{WIB_j (WO_i + WZ_i)}{n}$

**Formułowanie problemu optymalizacji**

Przy danych wagach kryteriów $w_1$ ..., $w_K$, do odnalezienia optymalnego przydziału kategorii na nośniki rozwiązujemy następujący problem optymalizacji, który jest skalaryzacją (Kaliszewski I. (2015) On Variant Selection Mechanisms in Interactive MCDA-Engineering versus Reverse Engineering. Journal of Multi-Criteria Decision Analysis, vol. 23, pp. 40–8) modelu przedstawionego powyżej:

$$\text{minimalizuj } \max_k w_k\sigma_k(f_k(X) - z_k^*) + \rho\sum_k w_k\sigma_k f_k(X) \qquad (4)$$

przy ograniczeniach (2), (3), gdzie:

- $\sigma_K$ to multiplikatory, służące do sprowadzenia wszystkich funkcji celów do takiej samej skali: $\sigma_k = 1/(Z_k^{nad} - Z_k^*)$ jeżeli $Z_k^{nad} > Z_k^*$, oraz $\sigma_K = 0$ jeżeli $Z_k^{nad} = Z_k^*$ (funkcja celu jest zdegenerowana, czyli ma taką samą wartość dla wszystkich rozwiązań Pareto-optymalnych),
  - zdegenerowane funkcje calu są wykluczane ze skalaryzowanej funkcji celu, jednak dla każdej zdegenerowanej funkcji celu k dodajemy ograniczenie $f_k(X) \leq Z^{nad}$ do modelu;
- $Z_k^*$ to wektor idealny, składający się z optymalnych wartości każdej funkcji celu minimalizowanej niezależnie;
- $Z_k^{nad}$ to wektor nadir, składający się z najgorszych (maksymalnych) wartości każdej funkcji celu na zbiorze Pareto;
- $\rho$ to bardzo mały współczynnik, np. $10^{-4}$.

Do linearyzacji funkcji celu (4) używamy następującego przekształcenia.

- Każdą funkcję celu k reprezentujemy poprzez odpowiednią zmienną fikcyjną $g_K$, dodając liniowe ograniczenie $g_k \geq f_k(X)$.
- Wprowadzamy jeszcze jedną zmienną fikcyjną t i łączymy ją z każdą ze zmiennych $g_k$ dla niezdegenerowanych funkcji celu k poprzez ograniczenie $t \geq w_k\sigma_k(g_k - z_k^*)$;
- Dla każdej zdegenerowanej funkcji celu k dodajemy do modelu ograniczenie $g_k \leq z^{nad}$.
- Zamieniamy nieliniową funkcję celu (4) przez liniową funkcję celu:

$$\text{minimalizuj } t + \rho\sum_k w_k\sigma_k f_k(X) \qquad (5)$$

***Rozwiązywanie problemu optymalizacji***

Najpierw oceniane są wektory z* i $z^{nad}$. W tym celu zaimplementowane są dwie metody:

1) Regular – powszechna (Miettinen K. (1999) Nonlinear Multiobjective Optimization. Boston: *Kluwer Academic Publishers; rozdział 2.4.)* metoda na podstawie rozwiązywania K problemów minimalizacji poszczególnych funkcji celów i budowania macierzy payoff. Jest obliczeniowo kosztowna bo wymaga rozwiązywania K problemów o złożoności porównywalnej z głównym problemem.
2) Naive – bardzo szybka ale niedokładna metoda. Dla każdej funkcji celu k, oceniamy wartości $z_k^*$ i $z_k^{*nad}$ w następujący sposób.
   - Dla każdego i od 1 do m, obliczamy $c_i^{k,min} = \min_j c_{ij}^k$, $c_i^{k,max} = \max_j c_{ij}^k$
   - Oceniamy $z_k^* = \sum_i c_i^{k,min}$, $z_k^{nad} = \sum_i c_i^{k,max}$

Po odnalezieniu z* i $z^{nad}$, obliczamy multiplikatory $\sigma_K$, sprawdzamy które z funkcji celu są zdegenerowane, i rozwiązujemy problem skalaryzowany (5)(5)(5) przy ograniczeniach (2), (3) sformułowany powyżej.

Rezultat wykonania funkcji Solve_Optimization_Problem jest argumentem wejściowym funkcji Make_Allocation_Rules.

**Funkcja Make_Allocation_Rules**

CEL: Wypracowanie reguł alokacji.

INPUT: ∅.

OUTPUT: ∅.

OPIS DZIAŁANIA: Na podstawie wyników działania funkcji Build_Category_Set i Solve_Optimization_Problem, funkcja ta zbuduje mechanizm, który dla każdego nowotworzonego pliku określi regułę alokacji. Dla każdego nośnika wskazana zostanie wartość określająca stopień przekonania o tym, że nowotworzony plik powinien zostać powiązany z konkretnym nośnikiem. W miarę spadku wartości stopnia przynależności, kolejne nośniki będą znajdowały się na kolejnych pozycjach listy.

Tabela ROZWIĄZANIE zawiera następujące parametry plików: ID, BP, IO, IZ, CIDK, EX, TD, SzP, IW, UP, PTO, IDNOP (IDNOP jest identyfikatorem nośnika przypisanym do pliki w wyniku optymalizacji). Parametry te dzielą się na następujące rodzaje:

a) parametry ciągłe: BP, IO, IZ;
b) parametry nominalne: CIDK, EX, TD, SzP, IW, UP, PTO;
c) etykieta (identyfikator nośnika przypisany do pliku po optymalizacji): IDNOP.

Działanie funkcji realizowane jest w następujących krokach:

1. **Kodowanie CIDK**. Ponieważ CIDK jest listą zmiennej długości, dla potrzeb uczenia maszynowego parametr ten zamieniany jest na MAX_KAT_GLEB (> 0) parametrów $SCIDK_1$, ... , $SCIDK_{MAX\_KAT\_GLEB}$. Jeżeli rozmiar(CIDK) < MAX_KAT_GLEB, to parametry $SCIDK_1$, ... ,$SCIDK_{rozmiar(CIDK)}$ przyjmują wartości odpowiednich elementów z listy CIDK, a parametry $SCIDK_{rozmiar(CIDK)+1}$, ...,$SCIDK_{MAX\_KAT\_GLEB}$ przyjmują wartość '0'. Jeżeli długość CIDK ≥ MAX_KAT_GLEB, to $SCIDK_1$,=$CIDK_1$,..., $SCIDK_{MAX\_KAT\_GLEB}$= $CIDK_{MAX\_KAT\_GLEB}$.
2. **Podział parametrów.** Parametry plików dzielone są na cztery typy:
   a) identyfikator pliku: ID;
   b) parametry ciągłe: BP, IO, IZ;
   c) parametry nominalne: $SCIDK_1$, ... ,$SCIDK_{MAX\_KAT\_GLEB}$, EX, TD, SzP, IW, UP, PTO;
   d) etykieta kategorii: IDNOP.
3. **Kodowanie parametrów**. W procesie nauczania sieci neuronowej, jej wejście (X) tworzą parametry ciągłe oraz zakodowane parametry nominalne. Parametry nominalne kodowane są koderem KODER_KATEGOR typu 'One-HOT'. Wyjście (Y) tworzy parametr IDNOP, zakodowany koderem KODER_ETYKIETY typu 'LabelBinarizer'. Oba kodery, jako obiekty binarne, zapisywane są do plików, odpowiednio, PLIK_KODER_KATEGOR, PLIK_KODER_ETYKIETY. Będą one wykorzystywane w funkcjach AIL.Allocate_File i AIL.Reallocate_File.
4. **Budowa perceptrona wielowarstwowego**. Budowany jest perceptron wielowarstwowy MLP (sztuczna sieć neuronowa), zawierający LICZBA_WARSTW_WEW ≥ 1 warstw ukrytych. Liczba neuronów w każdej warstwie ukrytej dobierana jest metodami heurystycznymi. Opracowano trzy heurystyki. W jednej z nich uzależnia się tę liczbę nie tylko od liczby neuronów warstw wejściowej i wyjściowej, ale także od liczby elementów w zbiorze treningowym.
5. **Uczenie sieci MLP z nadzorem**. Zakodowane dane o plikach (X) oraz zakodowane etykiety kategorii (Y) dzielone są na dane treningowe (X_TRENING, Y_TRENING) oraz na dane testowe (X_TEST, Y_TEST). Frakcja danych testowych wynosi FRAKCJA_TEST (domyślnie 25%). Liczba epok w procesie uczenia wynosi MAX_LICZBA_EPOK (domyślnie 1000), ale proces uczenia monitorowany jest za pomocą tzw. mechanizmu „callback". Za pomocą tego mechanizmu przechwytywana jest także najlepsza, w sensie dokładności przewidywania dla danych testowych, wersja MLP. Jest ona zapisywana jako obiekt binarny do pliku PLIK_NAJLEPSZY_MLP i będzie wykorzystywana w funkcjach AIL.Allocate_File i AIL.Reallocate_File.

**Funkcja Reallocate_File**

CEF: Przygotowanie reguły relokacji dla pliku z tabeli KANDYDACI DO RELOKACJI.
INPUT: ID pliku.
OUTPUT: Reguła relokacji lub lista pusta.
OPIS DZIAŁANIA: Na podstawie tabeli KANDYDACI DO RELOKACJI ustalana jest dla zadanego pliku reguła relokacji. Jeżeli SzP == **true**, a wskazany w tabeli KANDYDACI DO RELOKACJI nośnik dla zadanego pliku (parametr IDNOP) nie jest nośnikiem szyfrowanym, to wybierany jest w sposób losowy jeden z dostępnych nośników szyfrowanych. Jeżeli identyfikator wybranego tak nośnika jest różny od identyfikatora nośnika, na którym aktualnie znajduje się plik, to ten pierwszy umieszczany jest na liście, tworzącej regułę relokacji. W przeciwnym wypadku reguła relokacji jest listą pustą, co oznacza, że plik nie ma być relokowany. Reguła relokacji wypracowana przez tę funkcję jest listą co najwyżej 1-elementową, w odróżnieniu od reguł relokacji wypracowywanych przez funkcję AIL.Reallocate_File.

**Funkcja Enqueue**

CEL: wstawienie do KOLEJKI RELOKACJI reguły relokacji dla pliku.
INPUT: R – reguła relokacji dla pliku.
OUTPUT: ∅.
OPIS DZIAŁANIA: Wstawienie do KOLEJKI RELOKACJI reguły relokacji R.

**Funkcja Save_Category_Set**

CEL: Zachowanie aktualnej kategorii plików w archiwum HISTORYCZNE KATEGORIE.

INPUT: Aktualne kategorie plików.

OUTPUT: ∅.

OPIS DZIAŁANIA: Zarchiwizowanie aktualnej kategorii plików w archiwum HISTORYCZNE KATEGORIE. Zakłada się, że kolejne kategorie plików będą przechowywane w archiwum. Zakłada się również, że archiwum mieści ograniczoną liczbę kolejnych kategorii. Zarchiwizowane kategorie plików, charakteryzują dynamikę Systemu.

**Karty akceleracyjne**

Karty akceleracyjne określane są jako płyty elektroniczne, wyposażone w układy obliczeniowe, komunikujące się z docelowym komputerem poprzez złącze Peripheral Component Interconnect Express (PCIe). Obecnie na rynku istnieją dwie technologie, które dominują w domenie kart akceleracyjnych: General Purpose Graphical Processing Unit (GPGPU) oraz Field Programmable Gate Array (FPGA). Pierwsze z nich, GPGPU, są kartami z układami o stałej architekturze wewnętrznej charakteryzującej się dużą ilością jednolitych rdzeni realizujących obliczenia w trybie Single Instruction Multiple Data (SIMD). Będąc najpopularniejszą rodziną układów akceleracyjnych posiadają najszersze wsparcie od strony pakietów oprogramowania, co sprawia ze ich użycie jest relatywnie łatwe. Drugie z nich, układy rekonfigurowalne FPGA, charakteryzują się możliwością definiowania architektury obliczeniowej przez użytkownika, dzięki czemu możliwe jest osiągniecie większej wydajności obliczeń w stosunku do stałych architektur. Brak stałej architektury wprowadza jednak odejście od klasycznych metod rozwijania oprogramowania, przez co otrzymanie działającego rozwiązania kosztuje znacznie więcej czasu i wysiłku.

W tym przykładzie wykonania w systemie według wynalazku do przyspieszenia obliczeń związanych z siecią neuronową w module AIL stosowane są karty akceleracyjne wyłonione zostały bazujące na układzie GPGPU, najpopularniejszym rozwiązaniu w dziedzinie sieci neuronowych. Pakiety oprogramowania posiadają najczęściej wbudowaną obsługę tego typu układów co ułatwia ich użycie.

Przeprowadzone badania oraz analiza rozwiązań i wymagań dotyczących implementacji modułu AIL pozwoliły wyznaczyć platformę Nvidia AGX oraz bibliotekę Tensorflow jako te, które w najlepszy sposób spełniają stawiane zadania. Będąc niezależną platforma, NVidia AGX odciąża system główny od wszystkich obliczeń związanych z modułem AIL. Dotyczy to nie tylko samej sieci neuronowej przetwarzanej na układzie GPU ale również operacji przygotowujących dane, które mogą być realizowane na zintegrowanym procesorze ARM. Takie rozwiązanie zapewnia bardzo dużą elastyczność dzięki temu, że na procesorze ARM uruchomiony jest system operacyjny Linux dostarczający wszystkich mechanizmów do komunikacji z systemem głównym. Do implementacji modułu na karcie akceleracyjnej wykorzystano platformę Keras dostępną z poziomu języka Python, która wewnętrznie wykorzystuje zoptymalizowane elementy napisane w języku C. Wszystkie rozważane platformy sprzętowe wspierają język Python i C, więc możliwa jest kompilacja programu napisanego w tych językach bezpośrednio do postaci wykonywalnej na kartach akceleratorowych.

**Przykład 2**

**Praca systemu według wynalazku z wykorzystaniem reguł domyślnych**

Po uruchomieniu systemu Administrator Systemu wprowadza parametry statyczne nośników i plików z wykorzystaniem narzędzia Aidtool. Do wartości tych parametrów ma dostęp komponent AIL za pomocą odpowiedniego interfejsu. Następnie Administrator wprowadza reguły domyślne. W dalszej kolejności Administrator wprowadza wagi kryteriów. W przypadku pominięcia tego kroku przez Administratora obowiązują wagi domyślne. Reguły domyślne są używane do alokacji plików bezpośrednio po uruchomieniu systemu.

Następnie, co epokę, moduł zarządzający DMFS zbiera informacje dotyczące operacji na plikach obejmujących co najmniej KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB oraz przekazuje je wraz z ich konfiguracją do modułu akceleracyjnego AIL, w którym następuje trenowanie w czasie rzeczywistym algorytmów sztucznej inteligencji w postaci sieci neuronowej neuronowa typu perceptron wielowarstwowy wbudowanej w moduł AIL, z wykorzystaniem danych wejściowych stanowiących kombinację co najmniej danych o wagach niezerowych obejmujących KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB, oraz modelu matematycznego, którym jest model programowania liniowego całkowitoliczbowego. Co epokę moduł AIL wypracowuje reguły alokacji i relokacji plików, które następnie przekazuje do modułu

zarządzającego DMFS, który w oparciu o wspomniane reguły co epokę przeprowadza alokację i relokację plików. Przy czym, jeżeli informacje o operacjach na plikach nie pozwalają na wytworzenie innych reguł niż te wypracowane w poprzedniej epoce, obowiązują reguły wypracowane w poprzedniej epoce.

Ponadto Administrator ma możliwość zmiany wag kryteriów. W takim wypadku AIL użyje nowych wag w pierwszej epoce następującej po epoce, w której zmieniono wagi.

## Zastrzeżenia patentowe

1. System zarządzania pamięcią masową oparty o sieci neuronowe, **znamienny tym**, że zawiera:
   a) moduł akceleracyjny AIL z wbudowanymi algorytmami sztucznej inteligencji w postaci sieci neuronowej skonfigurowany do wielokryterialnej zmiany reguł alokacji i relokacji na podstawie danych wejściowych stanowiących dane o operacjach na plikach otrzymane z modułu DMFS i obejmujące co najmniej kombinację następujących danych o wagach niezerowych KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB, przy czym reguły alokacji i relokacji są wypracowywane co epokę na podstawie trenowania w czasie rzeczywistym sieci neuronowej na podstawie wzorców pochodzących z matematycznego modelu, generowanych na podstawie wspomnianych danych wejściowych; oraz
   b) moduł zarządzający DMFS skonfigurowany do zbierania informacji dotyczących operacji na plikach i przekazywania ich do komponentu AIL oraz do wybierania trybu alokacji i relokacji plików w oparciu o reguły wypracowane przez algorytmy modułu AIL.
2. System według zastrz. 1, **znamienny tym**, że sieć neuronową stanowi sieć neuronowa typu perceptron wielowarstwowy.
3. System według zastrz. 2, **znamienny tym**, że perceptron wielowarstwowy składa się z warstwy wejściowej, 1–3 warstw ukrytych oraz warstwy wyjściowej.
4. System według dowolnego z poprzednich zastrz. od 1 do 3, **znamienny tym**, że moduł akceleracyjny AIL jest wyposażony w kartę akceleracyjną bazującą na układzie GPGPU.
5. Wspomagany komputerowo sposób zarządzania pamięcią masową znamienny tym, że obejmuje:
   a) zbieranie co epokę przez moduł zarządzający DMFS informacji dotyczących operacji na plikach obejmujących co najmniej KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB;
   b) przekazywanie co epokę przez DMFS danych zebranych w etapie a) wraz z ich konfiguracją do modułu akceleracyjnego AIL;
   c) trenowanie w czasie rzeczywistym algorytmów sztucznej inteligencji w postaci sieci neuronowej wbudowanej w moduł AIL, z wykorzystaniem danych wejściowych stanowiących kombinację co najmniej danych o wagach niezerowych obejmujących KOSZT, ENERGIA, SSO, SOD, BP, GUP, WIB, oraz modelu matematycznego, którym jest model programowania liniowego całkowitoliczbowego;
   d) wypracowanie co epokę przez moduł AIL reguł alokacji i relokacji plików;
   e) przekazanie co epokę przez moduł AIL reguł wypracowanych w etapie d) do modułu zarządzającego DMFS;
   f) alokację i relokację plików przez moduł zarządzający DMFS w oparciu o reguły wypracowane w etapie d) przez algorytmy modułu AIL.
6. Sposób według zastrz. 5, **znamienny tym**, że sieć neuronową stanowi sieć neuronowa typu perceptron wielowarstwowy.
7. Sposób według zastrz. 6, **znamienny tym**, że perceptron wielowarstwowy składa się z warstwy wejściowej, 1–3 warstw ukrytych oraz warstwy wyjściowej.

## Rysunki

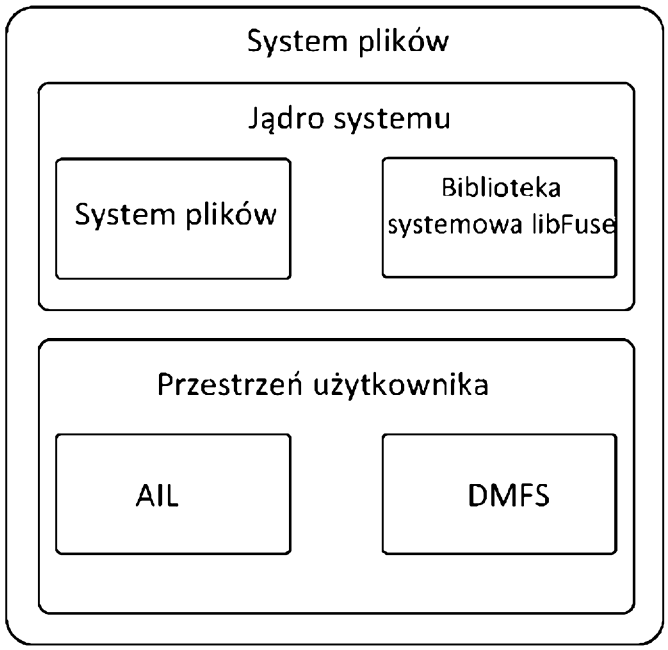


Fig. 1

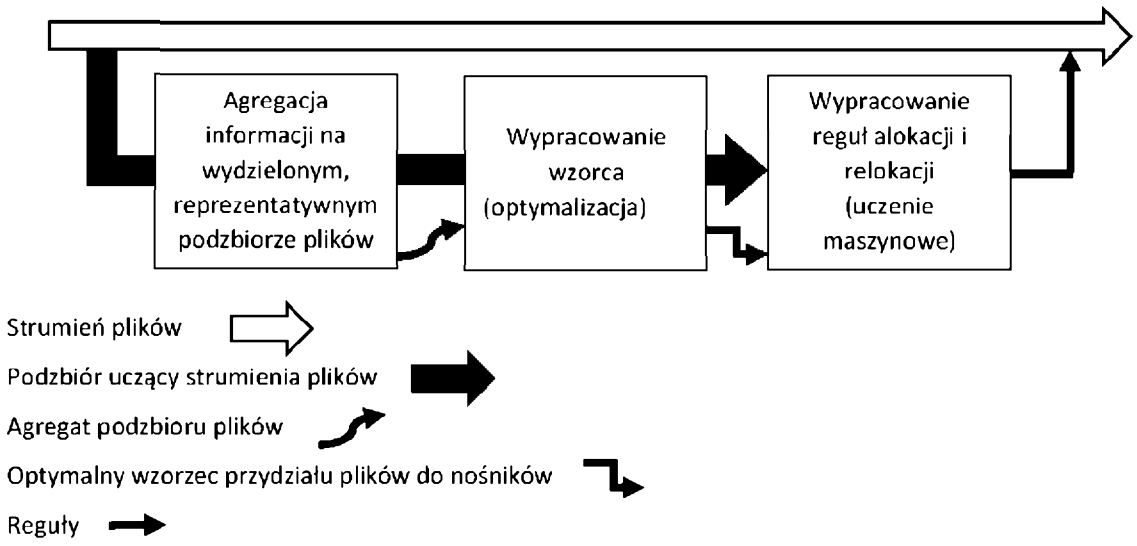


Fig. 2

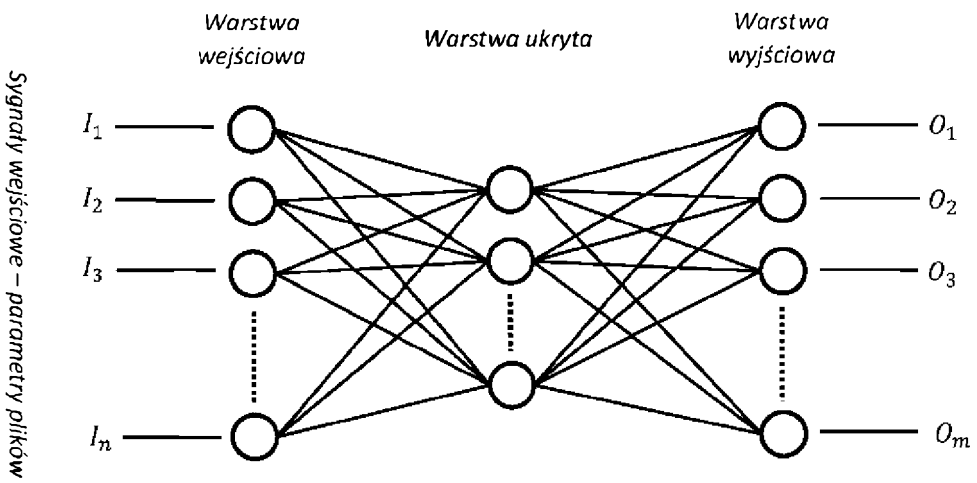
Warstwa wejściowa
Warstwa ukryta
Warstwa wyjściowa
Sygnały wejściowe – parametry plików
$I_1$
$I_2$
$I_3$
$I_n$
$O_1$
$O_2$
$O_3$
$O_m$

Fig. 3