
# PARALLELIZATION OF USER-DEFINED FUNCTIONS IN AN ETL WORKFLOW

*Author:*
Syed Muhammad Fawad Ali

*Supervisor:*
Prof. Robert Wrembel

April 13, 2021

POZNAN UNIVERSITY OF TECHNOLOGY

# Streszczenie

Przepływy ETL (zwane także procesami) są jednym z najważniejszych komponentów wszystkich architektur integracji danych, m.in. systemów hurtowni danych (ang. data warehouse - DW), jezior danych (ang. data lake) i aplikacji przetwarzania danych przez data science. Przepływy te są odpowiedzialne za: 1) pobieranie danych z różnorodnych źródeł (ang. data sources - DSs), 2) transformowanie heterogenicznych danych do wspólnego modelu i schematu, 3) czyszczenie danych, normalizowanie wartości i eliminowanie duplikatów, 4) wczytywanie danych do centralnego repozytorium, jakim jest DW.

Procesy ETL transportują duże wolumeny danych pomiędzy źródłami a DW i realizują złożone zadania transformacji danych. Z tych powodów, czasy ich wykonania są długie - typowo procesy ETL wykonują się w ciągu kilku godzin. Ponadto, wolumen, różnorodność i szybkość napływania danych do systemu stale wzrastają. Jest to przypadek tzw. gigadanych (ang. big data), co powoduje problemy z wydajnością standardowych architektur przetwarzania danych. W szczególności, różnorodność gigadanych i sposoby ich przygotowania do analizy przez różnego rodzaju aplikacje (m.in. uczenia maszynowego) wymagają rozszerzenia funkcjonalności rozwiązań ETL. Tego typu rozszerzenia w praktyce są realizowane za pomocą tzw. funkcji użytkownika (ang. user-defined functions - UDFs). UDFs są implementowane przez projektanta procesu ETL w językach dostępnych w środowisku projektowym ETL. UDF może realizować dowolną operację na danych, np. eliminowanie duplikatów zestawem algorytmów adekwatnym do przetwarzanych danych, analizę sentymentu. Projektowanie złożonych UDF nie jest łatwe, a powstały kod może być niewydajny lub niewystarczająco dobrze przetestowany.

Z tego powodu, w niniejszej rozprawie adresujemy problem wydajnego przetwarzania kosztownych obliczeniowo UDF poprzez zastosowanie przetwarzania równoległego. Prace badawcze rozprawy zostały poprzedzone szczegółową analizą istniejących rozwiązań w zakresie budowania procesów ETL, tj. modelowania konceptualnego i logicznego procesów oraz ich implementowania, a także w zakresie optymalizowania

wykonania tych procesów.

Na podstawie powyższej analizy wyciągnęliśmy następujące wnioski.

- Większość zaproponowanych metod projektowania procesów ETL wymaga dużego nakładu pracy ze strony projektanta, co czyni je nieodpornymi na błędy, czasochłonnymi i niewydajnymi.
- Większość metod umożliwia projektowanie procesów ETL dla danych o dobrze określonych strukturach (np. relacyjnych), przy niewielkim wsparciu przetwarzania danych częściowo ustrukturyzowanych (ang. semi-structured) lub nieustrukturyzowanych (ang. unstructured). Ponieważ różnorodność formatów integrowanych danych się zwiększa (szczególnie w przypadku gigadanych), więc zachodzi konieczność wspierania projektowania procesów ETL także dla nowych typów danych.
- Zdecydowana większość metod wykorzystuje standardowe zadania ETL, tj. m.in. operacje łączenia tabel, sumy zbiorów, sortowanie, agregowanie, selektywne sięgnięcie do innej tabeli (ang. lookup), konwersję danych. Nie wspierają one jednak efektywnego wykonania procesów ETL z UDF. Tego typu funkcjonalność wydaje się konieczna, ponieważ UDF są w praktyce wykorzystywane bardzo często do implementowania niestandardowych zadań. Optymalizowanie procesów ETL z UDF jest bardzo trudnym wyzwaniem badawczym i nierozwiązanym do tej pory, ponieważ UDF są najczęściej traktowane jako tzw. czarne skrzynki, tj. ich semantyka nie jest znana.
- Większość metod projektowych ETL w ogóle nie uwzględnia w procesie projektowania jakości danych produkowanych przez procesy ETL.
- Wśród analizowanych rozwiązań brakuje takich, które umożliwiałyby automatyczne monitorowanie wydajności poszczególnych komponentów procesu ETL i wskazywałyby w jaki sposób zwiększyć wydajność zadań takiego procesu.

Powyższe wnioski posłużyły do sformułowania następujących wyzwań badawczych zaadresowanych w niniejszej rozprawie.

- **Pierwszym wyzwaniem badawczym** jest zaprojektowanie podejścia wspierającego projektanta procesu ETL w implementowaniu wydajnych UDF poprzez przetwarzanie równoległe. Jest to możliwe dzięki odseparowaniu właściwego kodu UDF od części definiującej sposób jego równoległego przetwarzania. Zmniejsza się w ten sposób prawdopodobieństwo postania błędów w oprogramowaniu i zwiększa produktywność projektanta.
- **Drugim wyzwaniem badawczym** jest konstruowanie modelu kosztów, który umożliwi zdefiniowanie (sub-)optymalnych parametrów architektury przetwarzania równoległego, dla zadanej UDF.

W podpowiedzi na ww. wyzwania badawcze, w ramach rozprawy zaproponowaliśmy architekturę i techniki umożliwiające zbudowanie w pełni autonomicznego systemu do zarządzania procesami ETL. W ramach tej architektury opracowaliśmy cztery rozwiązania, tj. 1) Komponent UDF (the UDF Component), 2) Model Kosztów (the Cost Model), 3) Rekomender (the Recommender), 4) Monitor (the Monitoring Agent) (w dalszej części będziemy stosowali nazwy anglielskie).

**UDF Component** adresuje pierwsze wyzwanie badawcze. Głównym zadaniem tego komponentu jest wspieranie projektanta w implementowaniu wykonywanych równolegle UDF, w taki sposób, aby zapewnić wydajne uruchamianie kodu. W tym celu, **UDF Component** dostarcza predefiniowaną bibliotekę szablonów przetwarzania równoległego (ang. Parallel Algorithmic Skeletons - PASs). Dostępne są dwa rodzaje szablonów, tj. 1) generyczny PASs (ang. generic PASs), np. worker-farm model, divide and conquer, branch and bound, systolic, MapReduce i 2) case-based PASs. Oba rodzaje szablonów zawierają gotowe zrównoleglone kody częstych zadań wykonywanych w procesach ETL dla gigadanych, np. analiza sentymentu (ang. sentiment analysis), deduplikowanie danych (ang. deduplication, entity resolution, entity matching), wykrywanie odchyleń (ang. outlier detection). Dla generycznego PAS, projektant pisze tylko właściwy program, specyfikuje ograniczenie czasowe na jego wykonanie i specyfikacje komputerów w środowisku rozproszonym. Przykładowo, dla szablonu MapReduce, wymagane jest podanie jedynie funkcji Map i funkcji Reduce, a konfiguracja (m.in., parametry partycjonowania, liczba węzłów) zostaną automatycznie uzupełnione przez UDF Component. Implementacja tego mechanizmu bazuje na tzw. procesorze Orchestration Style Sheet, który generuje kod w wersji do wykonania równoległego i alternatywne konfiguracje komputerów w architekturze rozproszonej dla wykonania tego kodu.

Ocena eksperymentalna omówionego rozwiązania umożliwia skrócenie od 50% do około 65% czasu koniecznego do zaimplementowania kodu równoległego. Ponadto, przeprowadziliśmy eksperymenty pozwalające zrozumieć wpływ zrównoleglania wykonania kosztownych obliczeniowo UDF na wydajność całego procesu ETL zawierającego takie UDF. Stwierdziliśmy, że zrównoleglenie tanich obliczeniowo UDF nie wpływa na zwiększenie wydajności procesu ETL, w porównaniu ze scenariuszem, w którym ta sama UDF jest wykonywana bez zrównoleglenia. Natomiast, zrównoleglenie kosztownych obliczeniowo UDF może znacząco zwiększyć wydajność całego procesu ETL.

Wygenerowane warianty kodu i konfiguracji równoległego środowiska uruchomieniowego są wykorzystywane przez komponent **Cost Model**, który wraz z komponentami **Recommender** i **Monitoring Agent** rozwiązuje drugie zadanie badawcze. **Cost Model** wykorzystuje techniki optymalizacji kombinatorycznej i uczenia maszynowego w procesie generowania (sub-)optymalych konfiguracji środowiska uruchomieniowego dla ETL z UDF. Optymalizacja kombinatoryczna jest wykorzystywana dla

case-based PAS. Wykorzystujemy tu Multiple Choice Knapsack Problem (MCKP) do wyboru optymalnego PAS spośród wielu jego wariantów. Przykładowo, rozważmy proces ETL złożony z $m$ kosztownych obliczeniowo UDF. Dla każdej UDF, **UDFs Component** może wygenerować $n$ jej równoległych wariantów, dając w wyniku $n^m$ możliwych kombinacji wariantów kodu.

Z tego powodu, problem znalezienia optymalnego sposobu wykonania UDF został odwzorowany w MCKP. Jako dalszy rozwój opracowanej tu koncepcji, proponujemy zastosowanie technik uczenia maszynowego (ang. machine learning - ML) w celu optymalizacji wykonania UDF opartych o generic PASs. Model uczenia maszynowego zostanie zbudowany w oparciu o historyczne dane z wykonania UDF, a następnie wykorzystany do znalezienia (sub-)optymalnych konfiguracji komputerów w środowisku uruchomieniowym, w sytuacji gdy techniki optymalizacji kombinatorycznej nie znajdą zadowalającego rozwiązania. Moduł **Recommender** bazując na bibliotece modeli kosztów i danych otrzymanych z **Monitoring Agent**, rekomenduje rozwiązania projektowe procesu ETL. **Monitoring Agent** umożliwia gromadzenie danych z monitorowania wykonania procesów ETL.

Zaproponowane w rozprawie rozwiązanie bazujące na modelu kosztów, zostało ocenione eksperymentalnie. Jako przypadek użycia zastosowano znany i kosztowy algorytm Set-Similarity Join, służący do eliminowania duplikatów, zaimplementowany jako proces ETL. Proces zawierał m.in. 3 kosztowne obliczeniowo UDF, podlegające optymalizacji poprzez zrównoleglenie ich wykonania. W eksperymentach wykorzystano klaster Amazon Web Services z 2, 4, 8 i 10-cioma węzłami. Eksperymenty pokazały, że zaproponowany model kosztów umożliwił znalezienie najbardziej wydajnej konfiguracji klastra dla testowanego procesu ETL, przy ograniczeniu budżetu monetarnego na obliczenia. Ten sam model kosztów zaimplementowany na komputerze PC (2,6 GHz 6-Core Intel Core i7) także umożliwił znalezienie tego samego rozwiązania w czasie ułamków sekundy. Na koniec, zaproponowaliśmy rozszerzenie modelu kosztów o zadane miary, tj. dokładność modelu, prcesion, recall i monetarny koszt wykonania procesu.