# Złożoność czasowa algorytmów wypełniania stron w metodzie materializowanej listy agregatów

**Marcin Gorawski**[1]

**Streszczenie:** Materializowana Lista Agregatów (MAL) pozwala efektywnie przechowywać i przetwarzać długie listy agregatów. Struktura MAL zawiera tablicę iteratora podzieloną na logiczne strony, które przechowują pewną określoną liczbę agregatów. Wyznaczono złożoność czasową trzech różnych algorytmów i porównano z wynikami eksperymentalnymi estymującymi najlepsze kombinacje parametrów konfiguracyjnych MAL tj.: liczba stron, rozmiar pojedynczej strony oraz liczba dostępnych połączeń z bazą danych. MAL może być zastosowana także do każdego poziomu agregacji w różnych strukturach indeksacji, takiej jak np. aR-drzewo.

**Słowa kluczowe:** Przestrzenna hurtownia danych, materializacja, indeksacja, materializowana lista agregatów, złożoność czasowa algorytmów MAL

## 1. Wprowadzenie

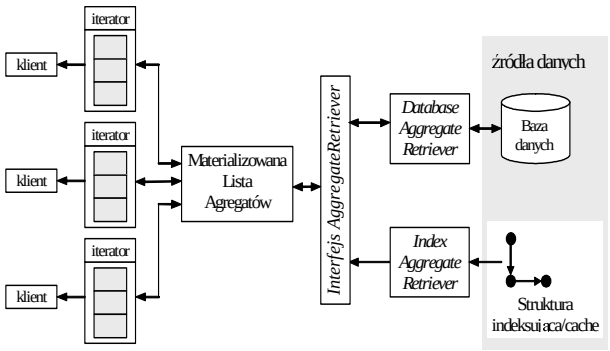
W relacyjnych hurtowniach danych, czas obliczania zapytań może być poprawiony poprzez zastosowanie odpowiednich metod indeksacji i materializacji. Materializacja perspektyw to proces wstępnego przetwarzania i zmagazynowania zbioru częściowych agregatów, które umożliwiają minimalizację kosztu wykonania zapytania dla danego obciążenia oraz dla danego ograniczenia przestrzeni dyskowej (Theodorato, Bouzehoub, 2000). W (Harinarayan, Rajaraman, Ullman, 1996) autorzy po raz pierwszy użyli przestrzennej sieci do przechowywania relacji pomiędzy zagregowanymi perspektywami. W pracach (Baralis, Paraboschi, 1997, Gupta, 1997, Gupta, Mumick, 1999) materializacja perspektyw jest determinowana poprzez obciążenie oraz ograniczenie przestrzeni dyskowej. Ponieważ indeksy mogą być tworzone na wszystkich zmaterializowanych perspektywach, w celu zredukowania złożo-ności problemu, materializacja oraz indeksacja są często stosowane oddzielnie. Oznacza to, że wybierany jest optymalny schemat indeksacji, dla danego ograniczenia przestrzeni, zaraz po określeniu zbioru perspektyw przeznaczonych do materializacji (patrz Golfarelli, Rizzi, Saltarelli, 2002). W pracy (Labio, Quass, Adelberg, 1997) zaproponowano zbiór heurystycznych kryteriów dla wyboru perspektyw oraz indeksów dla hurtowni danych. W artykule (Rizzi, Saltarelli, 2003) zaprezentowano porównawczą ocenę korzyści, wynikających z zastosowania materializacji perspektyw oraz indeksacji w hurtowni danych, w funkcji właściwości zapytania. Następnie zaproponowano heurystyczną metodę oszacowania, dla danego obciążenia oraz globalnego ograniczenia przestrzeni dyskowej, optymalnego

[1] Instytut Informatyki, Politechnika Śląska, ul. Akademicka 16, 44-100 Gliwice
e-mail: {M.Gorawski}@polsl.pl

ustępstwa pomiędzy przestrzenią przeznaczoną dla materializacji perspektyw oraz przestrzenią poświęconą indeksom. Artykuł jest zorganizowany w następujący sposób: rozdział 2 krótko wyjaśnia motywację naszych prac. W rozdziale 3 zamieszczono architekturę MAL oraz nowe definicje pojęć dla integratora MAL. Rozdział 4 zawiera opis szczegółów proponowanych algorytmów wypełniania stron w metodzie MAL oraz wyprowadzono po raz pierwszy wzory na ich złożoność czasową. W rozdziale tym dokonano porównania teoretycznych wykresów algorytmów MAL z uzyskanymi wcześniej wynikami eksperymentalnymi (Gorawski, Malczok, 2005). Rozdział 5 to krótkie podsumowanie.

## 2. Motywacja

Nasz system Rozproszonej Przestrzennej Hurtowni Danych (ang. *Distributed Spatial Data Warehouse - DSDW*) zaprezentowany w (Gorawski, Malczok, 2004) jest hurtownią danych (DW) gromadzącą i przetwarzającą ogromną liczbę danych telemetrycznych, wygenerowanych przez system telemetryczny zintegrowanych odczytów liczników. Odczyty liczników wody, gazu i energii są wysyłane drogą radiową poprzez węzły odbiorcze do serwera telemetrycznego. Pojedynczy odczyt (pomiar) wysłany przez licznik do serwera zawiera stempel czasowy, identyfikator licznika, oraz odczytane wartości. Periodycznie, system ekstrakcji ładuje dane do bazy danych systemu DSDW. W naszych bieżących badaniach staramy się znaleźć najsłabszy punkt rozwiązania systemu DSDW. Po różnych seriach testów (ze zmiennymi okresami agregacji, liczbą obiektów telemetrycznych itd.) odkryto, że kluczowym problemem jest stworzenie i zarządzanie długą listą agregatów. Lista agregatów jest listą odczytów liczników, zagregowanych zgodnie z odpowiednim oknem czasowym. Okno czasowe jest to przedział czasu, w którym chcemy badać zużycie mediów. Agregat składa się ze stempla czasowego i wartości zagregowanych. Gdy chcemy przeanalizować poziom poboru mediów, musimy zbadać historię tego zużycia. Jest to moment, w którym lista agregatów jest przydatna. Każdy z agregatów tworzących listę, przechowuje kilka wartości, więc konsumpcja pamięci jest wysoka. Aby zapobiegać przepełnieniom pamięci opracowano algorytm zarządzania pamięcią, zastosowany w systemie zaprezentowanym w (Gorawski, Malczok, 2004). Jednak nawet, jeśli obliczenie wartości agregatu sprowadza się w najprostszym przypadku do zsumowania kilku wartości, czas obliczenia wartości pojedynczego agregatu pomnożony przez długość listy agregatów powoduje, że proces obliczania wartości agregatów staje się długotrwały. Dlatego też w wielu przypadkach podejmowana jest decyzja o całkowitym lub częściowym zachowaniu już obliczonych agregatów - tzw. materializacji. W związku z tym zaproponowaliśmy nową metodę składowania i materializowania agregatów. Dla tej metody materializacji list agregatów zostały przyjęte następujące założenia, że metoda: a) eliminuje ograniczenia pamięciowe, b) umożliwia przechowywanie dowolnych typów agregatów i dopuszcza różne rodzaje źródeł danych, c) bazuje na uznanej strukturze programowej. Nazwano nasze rozwiązanie Materializowaną Listą Agregatów (ang. Materalized Aggregate List - MAL) (patrz Gorawski, Malczok, 2005).




Rysunek 1. Struktura logiczna systemu

## 3. Architektura Materializowanej Listy Agregatów

Materializowana Lista Agregatów została zaprojektowana w oparciu o wzorzec projektowy listy dostępnej w języku Java (java.util.List (patrz http://java.sun.com.)). Rysunek 1 przedstawia schemat architektury Materializowanej Listy Agregatów. W porównaniu do *java.util.List* architektura MAL jest znacznie bardziej rozbudowana. Pierwszą znaczącą różnicą jest źródło danych listy. Elementy MAL nie są wstawiane do listy przez jej klientów, ale są pobierane ze źródła danych, zdefiniowanego dla danej instancji listy. Dla klienta Materializowanej Listy Agregatów (klient MAL) wykorzystującego listę typ źródła danych jest zupełnie przezroczysty. Drugim aspektem silnie odróżniającym listę materializowaną od listy standardowej, jest przeniesienie znacznej części funkcjonalności do iteratorów. Klienci mogą komunikować się z MAL tylko przy wykorzystaniu iteratorów. Iteratory mają złożoną strukturę, zapewniającą odpowiednie mechanizmy cache oraz materializację agregatów. Ostatnią ważną różnicą, o której należy wspomnieć, jest fakt, iż obiekt listy nie przechowuje żadnych danych (agregatów). Jego rola ogranicza się tylko do pośredniczenia pomiędzy iteratorami wykorzystywanymi przez klientów, a źródłem danych. Klient MAL przegląda listę, pobiera z niej agregaty i przetwarza je w specyficzny dla siebie sposób.

### 3.1. Iterator MAL

Iteratory Materializowanej Listy Agregatów (Iterator MAL) pozwalają jedynie na przeglądanie listy i pobieranie agregatów przechowywanych w liście. Operacja usuwania agregatów nie jest dostępna zgodnie z ideą DSDW. Przeglądanie i pobieranie elementów listy odbywa się przy wykorzystaniu dwóch funkcji iteratora:

- *hasNext()* - metoda zwracająca wartość logiczną typu boolean. Funkcja ta zwraca wartość prawda, jeśli lista zawiera kolejny element, lub fałsz w przypadku, kiedy nie zawiera kolejnych elementów,
- *next()* - metoda pozwala na pobranie bieżącego elementu listy i automatycznie przesuwa wskaźnik (kursor) iteratora na kolejny element listy.

Architektura iteratora MAL jest znacznie bardziej złożona w porównaniu do iteratora standardowej listy. Każdy iterator zawiera strukturę pamięciową - tablicę o ściśle określonym rozmiarze.

Definicja 1. *Tablica B iteratora MAL (B) jest strukturą pamięciową przechowującą agregaty. Tablica B jest zbiorem logicznych fragmentów nazywanych stronami.* $B := S_i : S_i \ - \ i-ta$ *strona iteratora* $MAL, i \in 1, \ldots, n$, *gdzie n - liczba stron*

Strony przechowują pewną określoną liczbę agregatów. Dla wszystkich stron liczba przechowywanych agregatów jest taka sama. Istnienie logicznych stron znajduje wykorzystanie podczas operacji tworzenia agregatów przechowywanych w tablicy. Wielkością charakteryzującą stronę tablicy iteratora MAL jest znacznik czasowy nazywany datą graniczną. Data graniczna strony jest generowana w oparciu o znacznik czasowy pierwszego agregatu przechowywanego na stronie.

Definicja 2. *Strona S tablicy iteratora MAL (S) jest logicznym fragmentem tablicy B. Strona S jest zbiorem agregatów:* $S := A_i : A_i - \ i-ty\ agregat, i \in 1, \ldots, M$, *gdzie M - liczba agregatów Strona S jest charakteryzowana przez datę graniczną BD (ang. border date), równą znacznikowi czasowemu pierwszego agregatu przechowywanego na stronie, pomniejszonemu o wartość okna czasowego tego agregatu.*

Najmniejszą strukturą przechowywaną w tablicy iteratora jest agregat. Agregat składa się ze stempla czasowego oraz zbioru wartości. Wartości agregatu mają określone typy (np. liczba zmiennoprzecinkowa, liczba całkowita). Agregaty obliczane są dla pewnego przedziału czasu, nazywanego oknem czasowym agregatu.

Definicja 3. *Agregat A jest najmniejszą strukturą przechowywaną w tablicy iteratora MAL. Agregat A definiujemy jako* $A := (TS, V_A)$,*gdzie:TS jest znacznikiem czasowym agregatu,* $V_A$ *jest zbiorem wartości agregatu takim, że* $V_A := \{V_i : V_i$ *- wartość i-tego agregatu,* $i \in 1, \ldots, n\}$ *oraz* $\forall i \ \forall V_i \ \exists! t_v$, *gdzie* $t_v$ *oznacza typ agregatu.*

Liczność zbioru $V_A, |V_A|$, nie jest ograniczona i jest określana przez potrzebę konkretnego zastosowania agregatu. Typ agregatu jest określony przez liczność zbioru $V_A$ oraz typy elementów tego zbioru. Lista wartości zawiera wstępnie przetworzone dane dla obliczenia agregatu (np. zużycie mediów za dany okres czasu).

Definicja 4. *Dwa agregaty* $A_m$, $A_n$ *są tego samego typu, jeśli:*

- *Liczność zbioru wartości agregatu* $A_m$ *jest równa liczności zbioru wartości agregatu* $A_n$ $|V_{A_m}| = |V_{A_n}|$
- *Odpowiadające sobie elementy zbiorów wartości agregatów* $A_m$ *i* $A_n$ *są tego samego typu* $i = j \Rightarrow t_{V_i} = t_{V_j}; i, j = 1|V_{A_m}|$.

Możliwe jest użycie MAL jako składnika węzłów hierarchicznej struktury indeksującej. Przy takim wykorzystaniu, źródłem danych list w węzłach wyższych poziomów struktury są listy umieszczone w węzłach niższych poziomów. Agregaty w listach na wyższych poziomach są tworzone poprzez sumowanie agregatów z list węzłów niższych poziomów. Sumowane mogą być tylko agregaty o identycznych znacznikach czasowych i równych typach.

Definicja 5. *Operacja sumowania agregatów $A_s = A_m + A_n$ może zostać wykonana dla dwóch agregatów, które mają równe znaczniki czasowe: $TS_{A_m} = TS_{A_n}$ oraz są tego samego typu. Wynikiem operacji sumowania jest agregat $A_s$, w którym:*

- *znacznik czasowy TS jest równy znacznikom czasowym agregatów $A_m$ oraz $A_n$: $TS_{A_s} = TS_{A_m} = TS_{A_n}$.*
- *$V_A$ jest zbiorem wartości agregatu; kolejne elementy zbioru są sumą elementów agregatów składników sumy: $V_i \in V_{A_s}, V_i = W_i + Q_i, i = 1|V_{A_s}|$, gdzie $W_i \in V_{A_m}, Q_i \in V_{A_n}$*

## 4. Algorytmy MAL wypełniania stron i ich złożoność czasowa

Tablica iteratora MAL zawiera agregaty, które mogą być przeglądane i pobierane przez klientów MAL. Proces wypełniania tablicy iteratora odbywa się z wykorzystaniem specjalnie zaprojektowanych algorytmów MAL, które działają w oparciu o koncepcję strony - logicznej części tablicy. Analizując aktualny stan wypełnienia tablicy i aktualne położenie kursora, którym posługuje się klient (wywołując funkcję iteratora *next()*), algorytm MAL uruchamia proces wypełniania odpowiedniej strony tablicy iteratora MAL. Działanie klienta MAL może być scharakteryzowane czasem konsumpcji (przetwarzania) pojedynczego agregatu $T_a$. Czas ten jest zależny od specyfiki analizy działania klienta. Analiza ta przeprowadzona pod kątem doboru i konfiguracji algorytmu wypełniania stron staje się łatwiejsza, kiedy zdefiniuje się dwie wielkości czasowe, a mianowicie:

1. $T_k$ - czas konsumpcji pojedynczej strony charakteryzujący proces klienta; czas $T_k$ jest wprost proporcjonalny do czasu konsumpcji agregatu $T_a$, a współczynnikiem proporcjonalności jest rozmiar strony: $T_k = |S| * T_a$,|S| - jeśli strona będzie przechowywać agregaty za okres jednego dnia to jej rozmiar wynosi 48 agregatów (agregaty gromadzone co 30 min).

2. $T_w$ - czas wypełniania strony; czas wypełniania strony jest bezpośrednio zależny od zastosowanego algorytmu wypełniania stron. Czynniki wywierające największy wpływ na wartość $T_w$ zostaną przedstawione poniżej.

Założenie wykorzystania MAL jako substytutu listy pamięciowej powoduje, że działanie klienta nie może zostać wstrzymywane podczas przeglądania listy. Działanie klienta przeglądającego zawartość listy jest wstrzymywane wtedy, gdy żądana strona nie zawiera agregatów, ponieważ agregaty nie zostały jeszcze obliczone. Bazując na pojęciu iteratora MAL oraz na definicjach określających jego strukturę można określić, kiedy działanie klienta nie będzie wstrzymywane. W tym celu należy zdefiniować sposób, w jaki są wypełniane strony tablicy iteratora. W MAL zostało przyjęte założenie, że wypełnienie każdej strony jest wykonywane w odrębnym wątku działającym niezależnie od wątku klienta listy. Różne podejścia do problemu wypełniania stron tablicy iteratora MAL zostały nazwane **algorytmami MAL wypełniania stron** lub zamiennie **algorytmami MAL**. Poniżej zostały

scharakteryzowane trzy różne algorytmy wypełniania stron. Opisy algorytmów MAL mają pewne części wspólne, ale każdy jest charakteryzowany przez:

1. Nadmiarowość algorytmu, oznaczona symbolem N. Przed rozpoczęciem właściwego działania algorytmy wypełniają pewną liczbę stron, zabezpieczając w ten sposób działanie klienta przed zablokowaniem w przypadku, gdy czas $T_k$ jest krótki. Jednostką nadmiarowości algorytmu jest strona tablicy iteratora MAL.

2. Graniczny stosunek czasu wypełniania strony $T_w$ i czasu konsumpcji strony $T_k$, $R = \frac{T_w}{T_k}$, powyżej którego dany algorytm wypełniania stron nie jest w stanie zapewnić ciągłości działania klienta MAL.

Czasowa złożoność T(n) algorytmów MAL, gdzie n oznacza liczbę agregatów pobranych z tablicy iteratora, jest bardzo zbliżona, różnice wynikają jedynie ze współczynnika nadmiarowości N różnego dla różnych algorytmów. Za dominujące czasowo można przyjąć operacje dostępu do źródła danych, a także operacje obliczania agregatów. Złożoność czasowa algorytmów MAL jest wprost proporcjonalna do liczby stron, które zostały wypełnione agregatami. Pamięciowa złożoność S(n) tych algorytmów jest identyczna dla wszystkich wersji. Pojemność pamięci (przestrzeni) wykorzystywanej do dostarczenia agregatów klientowi MAL ściśle zależy od rozmiarów listy wartości agregatów przechowywanych w tablicy iteratora, parametrów tablicy iteratora oraz parametrów puli tablic iteratora.

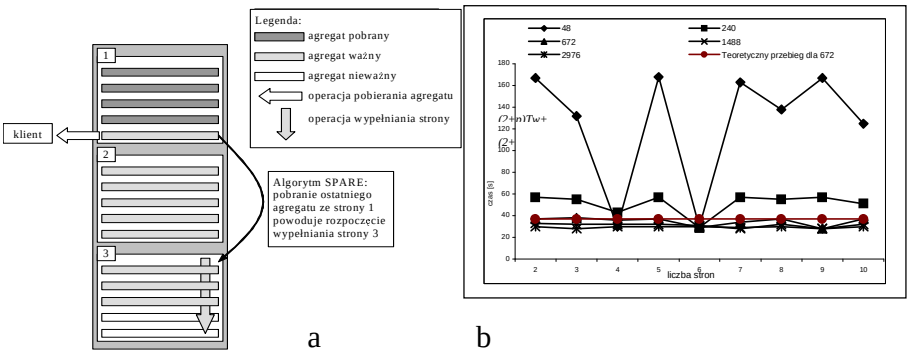
### 4.1. Algorytm SPARE

Pierwszym algorytmem MAL jest algorytm wypełnienia stron o nazwie SPARE. Przed rozpoczęciem właściwego działania algorytm SPARE wypełnia dwie pierwsze strony tablicy iteratora. Następnie, podczas pobierania agregatów algorytm sprawdza, czy ostatni agregat został pobrany ze strony $n \; mod \; |B|$. Jeśli tak, wtedy rozpoczyna się wypełnianie strony o numerze $(n + 2)mod|B|$ podczas, gdy klient pobiera agregaty ze strony$(n+1)mod|B|$. Jedna strona tablicy iteratora jest zawsze wypełniana nadmiarowo, stanowiąc stronę rezerwową (ang. *spare page*). Działanie algorytmu zostało schematycznie zobrazowane na rysunku 2a. Charakterystyka algorytmu SPARE jest następująca:

1. Nadmiarowość algorytmu stanowi jedna strona: $N_{SPARE} = 1$,

2. Złożoność czasowa obliczana jest dla 2 wariantów przy założeniu, że:

**Założenie 1.** Algorytm SPARE zapewni nieprzerwane działanie klienta MAL, jeśli czas wypełniania strony będzie mniejszy bądź równy od czasu konsumpcji strony: $T_w \leq T_k$ . Graniczną wartością jest zatem:$R = \frac{T_w}{T_k} = 1$.
**2a.** Dostępne tylko dwie strony tablicy iteratora MAL W takim przypadku algorytm SPARE potrzebuje do poprawnego działania trzech stron oraz dwóch połączeń do bazy danych. Strony są wykorzystywane następująco: jedna strona jest w danym momencie czytana (przetwarzanie agregatów), druga jest przygotowana w




Rysunek 2. a.Działanie algorytmu wypełniania stron SPARE, b.Czas wykonania zadania z wykorzystaniem algorytmu SPARE w zależności od rozmiaru i liczby stron.

zapasie, by rozpocząć z niej odczyt w momencie zakończenia przetwarzania pierwszej strony. Trzecia strona jest potrzebna by móc rozpocząć jej wypełnianie w momencie pobrania ostatniego agregatu z pierwszej strony. Przy takich założeniach możemy przedstawić zależność na złożoność czasową algorytmu SPARE, jeśli mamy dostępne tylko dwie strony tablicy iteratora, to:

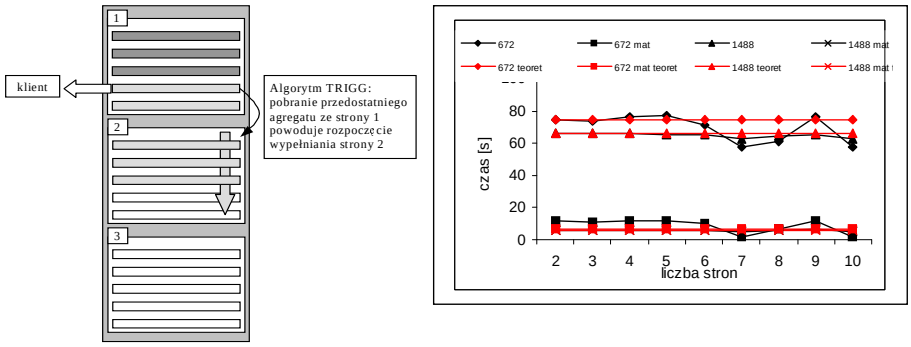
$$T(n) = (2 + p) * T_w + p * T_a \tag{1}$$

gdzie: $T_w$ - czas wypełnienia strony, $T_a$ - czas konsumpcji jednego agregatu, p - liczba odczytanych stron. Czas $T_a$ wynika z faktu, iż w momencie rozpoczęcia pobierania ostatniego agregatu z pierwszej strony, algorytm SPARE nie może rozpocząć wypełniania kolejnej strony, ponieważ nie ma dostępnej żadnej strony. Algorytm ten musi więc oczekiwać na zwolnienie strony przez wątek konsumujący agregaty przez czas Ta potrzebny na konsumpcję ostatniego agregatu.

**2b**. Dostępne 3 i więcej stron tablicy iteratora MAL Gdy dostępne są 3 i więcej stron wtedy złożoność czasowa algorytmu SPARE ma postać:

$$T(n) = 2 * T_w + p * T_a = (2 + p) * T_w \tag{2}$$

Większa liczba dostępnych stron, z punktu widzenia teoretycznego nie ma wpływu na wydajność tego algorytmu - nie zakłada on wykorzystania większej liczby stron. Wyniki eksperymentów (patrz Gorawski, Malczok, 2005) wykazują, iż najlepsze czasy zostały uzyskane dla większej liczby stron, niż wynikałoby to z rozważań teoretycznych (rys. 2b). Wynika to z działania mechanizmu synchronizacji wątków, który nie nadąża zwalniać strony, które zostały już w całości odczytane. Dzięki dodatkowej stronie, wątek wypełniający stronę, nie musi oczekiwać na jej udostępnienie. Jednak dodawanie kolejnych stron, powoduje dodatkowe koszty zarządzania pamięcią, co jak widać na wykresie 3b ma wpływ na całkowitą wydajność algorytmu SPARE.




a b

Rysunek 3. a.Działanie algorytmu wypełniania stron TRIGG, b.Pozytywny wpływ materializacji na działanie systemu

### 4.2. Algorytm TRIGG

Algorytm TRIGG wypełnia tylko jedną stronę tablicy iteratora MAL przed rozpoczęciem właściwego działania. Podczas przeglądania agregatów ze strony $nmod|B|$, algorytm sprawdza czy został pobrany przedostatni agregat. Pobranie przedostatniego agregatu ze strony $nmod|B|$ powoduje uruchomienie (ang. *trigger*) wątku wypełniającego stronę $(n+1)mod|B|$. Algorytm TRIGG nie wypełnia nadmiarowo stron tablicy, dopóki klient MAL nie zgłosi takiego żądania (rys. 3a). Poniżej przedstawiona została charakterystyka algorytmu TRIGG:

- Algorytm TRIGG nie posiada nadmiarowości, $N_{TRIGG} = 0$.
- Złożoność czasowa algorytmu TRIGG wynosi $T(n) = p * T_w$.
- Algorytm TRIGG zapewnia płynne działanie klienta MAL jedynie wtedy, kiedy czas konsumpcji strony jest znacznie większy od czasu jej wypełnienia. $T_w \leq \frac{T_k}{|S|}$ , $T_w \leq T_k$ . Zatem graniczna wartość stosunku czasów $R = \frac{T_w}{T_k} = \frac{1}{|S|}$.

Przy wyznaczaniu złożoności czasowej algorytmu TRIGG, dominującą operacją jest pobieranie danych z bazy (rys. 3a). Na pobieranie danych wpływa przede wszystkim wyszukiwanie odpowiednich danych w tabeli pomiarów. Ponieważ zmaterializowane dane są łatwo dostępne, a ich liczba jest o wiele mniejsza, niż danych surowych, stąd dostęp do nich jest o wiele mniej kosztowny. Przykładowo zmaterializowane dane dla jednego licznika i jednego roku będą zapisane tylko w 365 wierszach zamiast w 17520. Stąd współczynnik przyśpieszenia można zdefiniować jako stosunek pobrania danych zmaterializowanych do czasu pobrania danych surowych. Dla algorytmu TRIGG można zapisać tę zależność następująco:

$$T(n) = \frac{T_m}{T_n}(p * T_w) \tag{3}$$

gdzie: $T_w$ - czas wypełnienia strony, $T_m$ - czas pobrania danych zmaterializowanych, $T_n$ - czas pobrania danych niezmaterializowanych.

### 4.3. Algorytm RENEW

Przed rozpoczęciem obsługi klienta algorytm RENEW wypełnia wszystkie strony tablicy iteratora MAL. Następnie, kiedy agregaty są przeglądane i pobierane przez klienta, algorytm sprawdza, czy ze strony $n \ mod \ |B|$ został pobrany ostatni agregat. Jeśli klient pobrał ostatni agregat ze strony $n \ mod \ |B$ i przeszedł na stronę $(n+1)mod|B|$, wtedy algorytm uruchamia wątek, który wypełnia agregatami stronę $n \ mod \ |B|$ - strona jest na nowo przygotowywana do przeglądania przez proces klienta (jest odnawiana - ang. *renew*). Charakterystyka algorytmu RENEW przedstawia się następująco:

1. Nadmiarowość najwyższa z prezentowanych algorytmów MAL - zależy od liczby stron tablicy iteratora i wynosi $N_{RENEW} = |B| - 1$.

2. Złożoność czasowa także jest największa:

$$T(n) = (|B| - 1) * T_w + p * T_w = (|B| + p - 1) * T_w \tag{4}$$

3. W celu zdefiniowana wartości współczynnika R koniecznie jest określenie dodatkowych warunków.

Poszczególne strony tablicy iteratora MAL są wypełniane przez niezależne wątki uruchamiane podczas działania algorytmu wypełniania stron. W przypadku dwóch poprzednich algorytmów, fakt ten nie miał znaczenia, jednak dla algorytmu RENEW, w celu precyzyjnego określenia warunków definiujących, kiedy użycie algorytmu RENEW zapewnia płynne działanie klienta MAL konieczne jest przyjęcie dodatkowego założenia 2.

**Założenie 2.** *Czas wypełniania strony $T_w$ jest niezależny od liczby w równocześnie działających wątków.* W pierwszym przybliżeniu problemu przyjęto, że system, na którym działa MAL ma wystarczającą liczbę zasobów, aby uruchomić dowolną liczbę równoległych wątków wypełniających strony tablicy i jednocześnie wartość czasu $T_w$ dla wszystkich wątków nie ulegnie wydłużeniu. W tym przypadku czas wypełniania strony tablicy iteratora $T_w$ musi być mniejszy bądź równy iloczynowi czasu konsumpcji strony $T_k$ oraz liczbie stron pomniejszonej o jeden:$T_w \leq (|B| - 1)T_k$. Graniczna wartość stosunku czasów R, przy której działania klienta listy nie będzie wstrzymywane przez brak agregatów na którejkolwiek ze stron tablicy iteratora ma wartość: $R = \frac{T_w}{T_k} = |B| - 1$ . Dalsza analiza problemu pokazuje, że założenie 2 jest niemożliwe do spełnienia. System spełniający to założenie pozwalałby na równoczesne wykonywanie nieskończonej liczby zadań ze stałą prędkością. Stąd konieczna jest weryfikacja tego założenia.

**Założenie 3.** *Czas wypełniania strony tablicy iteratora jest zależny od liczby równolegle działających wątków wypełniania stron.* Zamiana założenia powoduje konieczność wprowadzenia nowej wielkości czasowej - efektywnego czasu wypełniania

strony $T_{we}$. Czas ten jest wprost proporcjonalny do liczby równolegle działających wątków wypełniania stron, z pominięciem wątku, dla którego jest obliczany. Wartość $T_{we}$ jest obliczana według wzoru:

$$T_{we} = (1 + k * w)T_w \tag{5}$$

gdzie: k - współczynnik wzajemnego wpływu wątków;w - liczba równolegle działających wątków; $T_w$ - czas wypełnienia strony.
Współczynnik k określa wpływ działania innych wątków na działanie tego wątku, dla którego jest obliczana wartość $T_{we}$; współczynnik przyjmuje wartości k $\in [0, 1]$. Wartości parametru k pozwalają na odpowiednie dostosowanie wzoru 3 do możliwości konkretnego systemu DSDW. W celu zapewnienia nieprzerwanego działania klienta MAL konieczne jest spełnienie warunku: $T_{we} \leq (|B| - 1)T_k$ ,$T_{we} <= \frac{(|B|-1)T_k}{1+k*w}$. Wartość R określająca graniczną wartość stosunku czasu wypełniania strony do czasu konsumpcji strony dla algorytmu RENEW wynosi $R = \frac{|B|-1}{1+k*w}$ .

### 4.4. Wpływ liczby połączeń do bazy danych na poszczególne algorytmy MAL

Algorytm TRIGG wykorzystuje tylko jedno połączenie do bazy danych i stąd złożoność czasową algorytmu TRIGG jest następująca:

$$T(n) = p * T_w \tag{6}$$

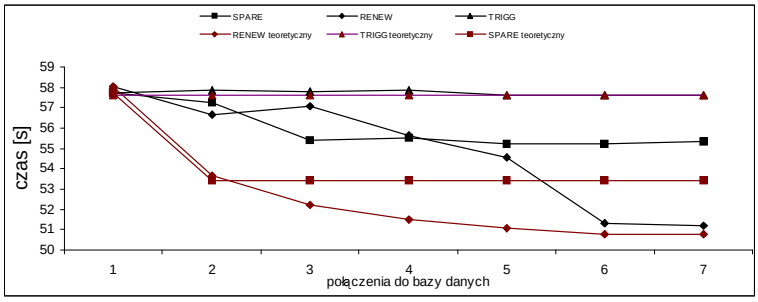
Drugi z algorytmów, RENEW, potrafi wykorzystać wszystkie dostępne strony iteratora MAL stąd, zależność $T(n)$ ma postać:

$$T(n) = \begin{cases} \frac{(|B|+p-1)(1+k*(w-1))T_w}{|C|} & \text{dla } |C| \leq |B| \\ \frac{(|B|+p-1)(1+k*(w-1))T_w}{|B|} & \text{dla } |C| > |B| \end{cases} \tag{7}$$

Podczas tworzenia wykresu 6 dla tego algorytmu przyjęto współczynnik k = 0.85, ponieważ dobrze oddaje wzajemny wpływ wątków w systemie jednoprocesorowym. Złożoność czasowa algorytmu SPARE $T_w$ wynosi:

$$T(n) = \begin{cases} \frac{(2+p)(1+k*(w-1))T_w}{|C|} & \text{dla } |C| < 3 \\ \frac{(2+p)(1+k*)T_w}{|B|} & \text{dla } |C| \geq 3 \end{cases} \tag{8}$$

Wyniki eksperymentów uzyskane w pracy (patrz Gorawski, Malczok, 2005) w tym przypadku, są dość zbieżne z rozważaniami teoretycznymi (rys. 4). Różne wartości pomiarów w zależności od liczby dostępnych połączeń dla algorytmu TRIGG, są w granicach błędu pomiarowego. Wyniki dla algorytmu SPARE sugerują, iż instancja MAL wykorzystuje jedno z połączeń dostępnych w puli połączeń. Ponieważ zależność na złożoność czasową wg wzoru 8 nie uwzględnia tego wpływu, stąd wynikają różnice na wykresie dla tego algorytmu. Z wykresu dla algorytmu RENEW wynika, iż również jedno z połączeń jest wykorzystywane przez instancję MAL. Pomimo,




Rysunek 4. Wpływ liczby połączeń do bazy danych dla poszczególnych algorytmów MAL

iż kształty charakterystyk rzeczywistych (patrz Gorawski, Malczok, 2005) i teoretycznych dla algorytmu RENEW różnią się, to jednak w końcowym ich punkcie zaczynają się one do siebie zbliżać. Różnice te wynikają z losowego zachowania środowiska Java (usuwanie nieużywanych obiektów tylko w sobie wiado-mym momencie), oczekiwanie przez wątek konsumujący stronę na wypełnienie potrzebnej strony, niedoskonałej synchronizacji ("zużyta" strona nie ma ustawionego znacznika).

## 5. Wnioski

Przedstawiono trzy wersje algorytmów wypełniania stron tablicy iteratora MAL. Dla klienta MAL wersje algorytmów MAL różnią się od siebie przede wszystkim nadmiarowością oraz możliwością płynnego dostarczania agregatów dla różnych czasów konsumpcji strony. Pierwszym i najważniejszym czynników doboru algorytmu jest czas przetwarzania agregatu przez proces klienta MAL. W przypadku, kiedy agregaty są bardzo długo przetwarzane lub płynna praca klienta MAL nie jest wymagana, wtedy warto zastosować algorytm TRIGG. W przypadku, kiedy czas konsumpcji strony tablicy iteratora MAL jest porównywalny do czasu wypełniania strony najlepszym wyborem jest algorytm SPARE, który przy niewielkim narzucie potrafi zapewnić płynną pracę klienta. Jeśli jednak czasy konsumpcji stron są krótkie, bądź obliczenie agregatów jest bardzo czasochłonne, wtedy należy zastosować algorytm RENEW. Kolejnym czynnikiem decydującym o wyborze odpowiedniego algorytmu jest charakterystyka systemu, na którym zos-tała uruchomiana MAL. Jeśli zasoby systemowe (a w szczególności dostępna przepustowość kanałów I/O) są niewielkie, wtedy nawet użycie algorytmu RENEW nie zagwarantuje płynnej pracy klienta MAL. Na obecnym etapie projektowania algorytmy wypełniania strony, jak również pozostałe parametry MAL, są określane w pliku konfiguracyjnym XML, a więc nie są one dynamicznie balansowane zgodnie z obciążeniem systemu DSDW. Innym sposobem przyśpieszenia działanie MAL jest zastosowanie bufora LRU jako wyższego poziomu hierarchicznego mechanizmu cache.

## Literatura

Baralis, E, Paraboschi, S. and Teniente, E. (1997) Materialized view selection in multidimensional database. *VLDB,*,pages 156-165, Athens, 1997.

Golfarelli, M, Rizzi, S. and Saltarelli, E.(2002) Index selection for data warehousing. *DMDW*, Toronto, 2002.

Gorawski, M.and Malczok, R. (2004) Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation *5th STDBM VLDB'04*. Toronto.

Gorawski, M.and Malczok, R. (2005) On Efficient Storing and Processing of Long Aggregate Lists. *7th International Conference Data Warehousing and Knowledge Discovery, DaWaK* . Copenhagen, LNCS 3589, pp.190-199.

Gupta, H. (1997) Selection of views to materialize in a data warehouse. *ICDT*, pages 98-112, 1997.

Gupta, H and Mumick, I.(1999) Selection of views to materialize under a maintenance cost constraint. *ICDT*, Jerusalem, Israel, 1999.

Harinarayan, V,Rajaraman, V and Ullman, J. (1996) Implementing data cubes efficiently. *In. Proc. ACM SIGMOD Conf.*, Montreal, 1996.

Labio, W J, Quass, D. andAdelberg, B.(1997) Physical database design for data warehouses. *In. Proc. ICDE*, pages 277-288.

Rizzi, S and Saltarelli, E. (2003) View Materialization vs. Indexing: Balancing Space Constraints in Data Warehouse Design *CAISE*, Austria.

Theodorato, D and Bouzehoub, M. (2000) A general framework for the view selection problem for data warehouse design and evolution. *DOLAP*, McLean.

Sun Microsystems JavaTM 2 Platform Standard Edition 5.0 API Specification `http://java.sun.com`

## Time Complexity of Page Filling Algorithms in Materialized Aggregate List Method

The Materialized Aggregate List (MAL) allows efficiently storing and processing of long aggregate lists. The MAL contains an iteration table divided into three logical parts that store certain aggregate quantity. We appointed the time complexity of three different algorithms. Obtained test results were used for estimation of the best combination of the MAL's configuration parameters: number of pages, size of a single page and number of available database connections. The Materialized Aggregate List can be applied on every aggregation level in various indexing structures, such as, an aR-tree.