Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marek Dopiera
Nr albumu: 219418

Techniki reorganizacji w locie obiektowych baz danych

Praca licencjacka
na kierunku MATEMATYKA

Praca wykonana pod kierunkiem
dra Marcina Szczuki

Wrzesień 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Praca obejmuje przegląd technik reorganizacji obiektowych baz danych w locie z uwzględnieniem ich wydajności, poziomu skomplikowania oraz przydatności. Jej celem jest stworzenie takiego połączenia i modyfikacji opisywanych technik, które nadawałoby się do implementacji w eksperymentalnej, obiektowej bazie danych, rozwijanej na Uniwersytecie Warszawskim pod opieką dra hab. Krzysztofa Stencla.

Słowa kluczowe

obiektowe bazy danych, reorganizacja, łańcuchy Markowa, symulowane wyżarzanie

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.0 Matematyka, Informatyka

Klasyfikacja tematyczna

Computer science
Theory of data
68P20 Information storage and retrieval

Statistics
Inference from stochastic processes
62M05 Markov processes: estimation

Tytuł pracy w języku angielskim

On-line reorganization techniques in object databases

Spis treści

Wprowadzenie

Jednym z zasadniczych problemów związanych z obiektowymi bazami danych jest fakt, że optymalizowanie zadania rozmieszczenia obiektów na dysku jest dużo trudniejsze niż w bazach relacyjnych. Większa trudność polega na tym, że standardowy sposób znany z relacyjnych baz danych, czyli układanie obiektów tego samego typu koło siebie, źle sprawdza się przy nawigowaniu po drzewie obiektów.

Powstało już wiele koncepcji w jaki sposób radzić sobie z tą trudnością. Ta praca opisuje techniki reorganizacji ułożenia obiektów w trakcie działania bazy. Zarówno wykrywanie jakiej reorganizacji dokonać oraz sam proces reorganizacji przeplata się, bądź zachodzi równolegle z operacjami bazodanowymi zleconymi przez klientów.

Do samej reorganizacji w locie powstało również kilka zupełnie różnych pomysłów. Część z nich jest oparta na statystyce, część na przemyślanych heurystykach, które, wg ich autorów, sprawdzają się w praktyce.

Zakłada się, że czytelnik posiada podstawową wiedzę z zakresu obiektowych baz danych. Większość opisywanych zagadnień i pojęć dotyczy baz danych w ogóle. Informacje na ten temat można znaleźć np. w [3] i [11]. Do modelowania opisywanych problemów wykorzystywany jest rachunek prawdopodobieństwa, między innymi łańcuchy Markowa. Potrzebna wiedza zawarta jest np. w [6]. W pracy pojawia się też pojęcie NP-zupełności wyjaśnione np. w [2].

Dla uproszczenia zapisu pojęcie system zarządzania bazami danych będzie się pojawiać w tekście jako SZBD.

Rozdział 1

Nieformalne sformułowanie problemu

Sylvain Guinepain i Le Gruenwald w swojej pracy [4], podsumowującej charakterystykę tego problemu, wyróżniają następujące podproblemy:

- Co reorganizować?
  Różne koncepcje przewidują między innymi całą bazę, jej ustalone wcześniej fragmenty, fragmenty ustalane dynamicznie lub np. tylko pojedyncze strony.
- W jakim porządku układać obiekty?
  Sposób w jaki dane zostaną ułożone ma wpływ na szybkość odczytu, czas potrzebny na dostęp do danych, ilość zajętego przez bazę miejsca oraz inne parametry, więc w zależności od tego jak ważne są one dla konkretnego rozwiązania, stosowane są różne algorytmy.
- Kiedy uruchamiać proces reorganizacji?
  Reorganizacja zużywa, na ogół w zauważalny sposób, pamięć oraz czas procesora, więc jeżeli baza jest zorganizowana podobnie do pożądanego stanu, może być warto nie przeprowadzać reorganizacji bez przerwy.
- Jakie statystyki zbierać?
  Często algorytmy służące do reorganizacji zbierają statystyki dotyczące bazy oraz sposobu jej użycia. Ograniczenia pamięciowe narzucają na ogół mocne ograniczenie na ich ilość, więc nie jest łatwym zadaniem zdecydować, które mają być zbierane.
- Jak wykryć, że aktualna organizacja nie jest optymalna?
  Stosowane heurystyki na ogół zliczają jak często zdarza się, że dobra strategia rozkładu obiektów pozwala na uniknięcie odczytu z dysku. Sposób liczenia tych statystyk bywa różny. Oddzielnym problemem jest ustalenie limitu, którego przekroczenie oznacza, że aktualna reorganizacja nie jest optymalna.
- Jak przeprowadzać sam proces reorganizacji?
  W przypadku reorganizacji w locie, można założyć, że będzie się ona przeplatać bądź wykonywać równolegle z transakcjami użytkowników, więc pożądane jest używanie mechanizmu, który pozwoli zachować interakcyjność bazy.

Nie istnieje jedyny słuszny sposób na sformalizowanie powyższych problemów, między innymi dlatego, że różne strategie skupiają się na różnych celach. W następnym rozdziale znajduje się jeden ze sposobów.

Rozdział 2

Model stochastyczny

Jedną z pierwszych prac wprowadzającą ścisła, matematyczną definicję problemu jest [9]. Definiuje ona model dla systemu zarządzania bazami danych (SZBD) opartego na architekturze klient-serwer.

2.1. Model działania SZBD

Niech S będzie zbiorem obiektów w bazie, $F = \{0, 1, 2, \ldots, n\}, n \in \mathbb{N}$ zbiorem bloków dyskowych, a funkcja $d\colon S \to F$ niech reprezentuje przynależność obiektu do konkretnego bloku. d będzie czasem przedstawiana jako złożenie dwóch innych funkcji:

$$d_\pi\colon S \to F$$

$$d_S\colon F \to F$$

$$d(s) = d_S(d_\pi(s))$$

Dla uproszczenia zakładamy, że S i F nie zmieniają się. Ukryte jest też tutaj założenie, że obiekty muszą być niewiększe niż jeden blok dyskowy. Ustalmy konwencję, że s_k oznacza element zbioru S, a f_k — F.

Niech nieskończony ciąg zapytań konkretnego klienta nazywa się $X_{\{n\}}$. Elementy tego ciągu są zmiennymi losowymi o wartościach w S. W dalszej części będziemy rozważać dwa warianty: kiedy X_n będą miały jednakowe rozkłady i będą niezależne oraz kiedy będzie to łańcuch Markowa. W drugim przypadku przyjmiemy, że znamy macierz $P = \{p_{ij}\}$ definiującą prawdopodobieństwa warunkowe zapytań o obiekty.

$$p_{ij} = P(X_{n+1} = s_j | X_n = s_i)$$

Do uzyskania takiej macierzy służą odrębne algorytmy. Aby mieć pewność, że istnieje rozkład stacjonarny, założymy dalej, że $\forall_{i,j} p_{ij} > 0$. Tak zdefiniowany łańcuch jest ergodyczny. Niech π_S będzie rozkładem stacjonarnym, czyli:

$$\pi_S P = \pi_S$$

Przyjmujemy, że czas odczytu obiektu z dysku zależy wyłącznie od tego, jaki obiekt był odczytywany przed nim. Założenie jest bliskie rzeczywistemu światu, o ile poprzednie odczyty nie są nigdzie buforowane. Formalnie, jeżeli przez T_n oznaczymy czas potrzebny na wykonanie

pierwszych n zapytań i przez $q(x, y)$ czas potrzebny na odczytanie obiektu y o ile przed nim odczytywany był x, to

$$T_n = \begin{cases} T_{n-1} + q(X_{n-1}, X_n) & n > 0 \\ 0 & n = 0 \end{cases}$$

oznacza czas potrzebny na odczytanie pierwszych n obiektów. Rozważmy $\bar{T_n}$ zdefiniowane jako

$$\bar{T_n} = \frac{T_n}{n}$$

Założyliśmy ergodyczność łańcucha $X_{\{n\}}$, więc możemy napisać

$$T = \lim_{n \to \infty} \bar{T_n} = E(q(X_{n-1}, X_n))$$

T ma interpretację średniego kosztu dostępu do danych.

2.2. Sformułowanie problemu w terminologii modelu

Mając dane:

- charakterystykę $X_{\{n\}}$ — sposobu dostępu klinta do SZBD
- formułę q na czas potrzebny na odczyt strony

znaleźć funkcję d przypisującą obiekty do bloków dyskowych, która:

- minimalizuje koszt T
- przypisuje do każdego bloku nie więcej niż L obiektów
- być może spełnia inne ograniczenia

Niestety, gdy nałożone są inne ograniczenia, problem staje się dużo bardziej skomplikowany. W praktyce każdy blok ma limit rozmiaru w bajtach, więc ograniczenie na liczbę obiektów w bloku, to tak naprawdę założenie, że obiekty są równego rozmiaru. Odrzucenie tego postulatu komplikuje problem do problemu plecakowego, o którym wiadomo, że jest NP-zupełny.

2.3. Rozwiązania

W tym rozdziale przeanalizowane zostanie kilka przypadków założeń, które można przyjąć oraz ich rozwiązania.

2.3.1. Identyczne rozkłady, zmienne losowe niezależne

Przyjmujemy, że $X_1, X_2, X_3 \ldots$ są niezależne i mają jednakowy rozkład. Rozważymy przypadki kiedy $q(X_n, X_{n-1}) = \alpha(1 - \delta_{X_{n-1}, X_n})$, gdzie $\alpha > 0$, a $\delta_{X_{n-1}, X_n} = 1$ gdy $X_n = X_{n-1}$ i 0 w przeciwnym wypadku, oraz gdy $q(X_n, X_{n-1}) = c(|d(X_{n-1}) - d(X_n)|)$, gdzie c jest monotoniczną funkcją z F w $\mathbb{R}$.

Rozwiązanie pierwsze

Funkcja q próbuje symulować bufor na jeden blok dyskowy. Problem ten, analizowany w pracy [13] ma łatwe rozwiązanie. Funkcja d_π sortuje malejąco obiekty względem ich prawdopodobieństwa użycia, czyli przydziela do bloków w ten sposób, że do pierwszego trafia L obiektów o największym prawdopodobieństwie, do następnego kolejne L itd. d_S jest identycznością.

Rozwiązanie drugie

W tym przypadku funkcja q ma za zadanie odzwierciedlić działanie dysku twardego: formuła odzwierciedla czas potrzebny na przesunięcie ruchu głowicy znad jednego cylindra nad inny. Rozwiązanie tego problemu również znajduje się w pracy [13]. d_π pozostaje takie jak w powyższym przypadku, natomiast zmienia się d_s: przyjmując, że numerem ostatniego bloku dyskowego jest $2n$

$$d_S(k) = \begin{cases} 2(n-k) & 0 \leqslant k \leqslant n \\ 2(k-n)-1 & n+1 \leqslant k \leqslant 2n \end{cases}$$

Autorzy [9] twierdzą, że zastosowanie tej strategii w systemie plików może zmniejszyć czas wyszukiwania nawet o 40–50%.

2.3.2. Łańcuch Markowa, pełniejszy model

Motywacja

Model, w którym nie jest uwzględnione żadne buforowanie, a na raz działa tylko jeden klient jest niestety zbyt mało realistyczny, dlatego rozszerzono go.

Zmiany w modelu

Przyjęto, że na raz działa wielu klientów. Każdy generuje zapytania o takich samych własnościach probabilistycznych. Wszystkie te zapytania muszą przejść przez jedną kolejkę zapytań. Kolejka ta może być priorytetowa. Im więcej klientów, tym rozkład zapytań wychodzących z tej kolejki jest bliższy modelowi niezależnych zmiennych losowych o tym samym rozkładzie. Dodatkowo, przyjęto, że każdy klient ma własny bufor bloków dyskowych. Przyjęto, że jeżeli żądany blok znajduje się w buforze, to koszt jego odczytania jest równy 0, w przeciwnym wypadku — 1. Oczywiście, to jak działa bufor, wpływa na zapytania docierające do serwera, zatem bufor będzie miał wpływ na sposób reorganizacji bazy.

Aby ominąć problem, który pojawił się po dodaniu do modelu buforów wprowadzono miarę lokalności zapytań. Jest ona rozmiarem zbioru roboczego WSS, tj. dla M odczytów, $WSS(M)$ jest średnią liczbą różnych odczytów spośród wszystkich M. Oznaczmy przez $K_t^{(M)}$ średni rozmiar zbioru roboczego dla M odczytów począwszy od chwili t. Jeżeli $X_{\{n\}}$ jest łańcuchem Markowa lub serią niezależnych zmiennych losowych, to jest ergodyczne, stąd $K_t^{(M)}$ jest niezmiennicze ze względu na t, więc odtąd t będziemy opuszczać.

Jako, że zmieniła się charakterystyka zapytań docierających do serwera, zmienia się też problem do rozwiązania. To co ma być optymalizowane to:

- Funkcja przypisująca obiekty do bloków dyskowych. Chcemy minimalizować wynikający z niej średni rozmiar zbioru roboczego, tj. generowany przez ciąg zapytań o bloki dyskowe $d(X_i(n))$ dla określonego M. To zaowocuje lepszym wykorzystaniem buforów.
- Funkcję d_s permutującą bloki dyskowe tak, aby minimalizować koszt potrzebny serwerowi na odczytywanie bloków.

Rozwiązanie problemu

Jak już zostało stwierdzone, należy uznać, że zapytania docierające do serwera od wielu klientów przez bufory mają jednakowy rozkład i są niezależne. W takim wypadku, drugi problem został już rozwiązany w 2.3.1.

Gdy $X_{\{n\}}$ są niezależne, o jednakowym rozkładzie, problem ma łatwe rozwiązanie.

$$K^{(M)} = M - \sum_{q=0}^{|F|-1} (1 - \sum_{y \in d^{-1}(q)} \pi_s(q))^M$$

Da się pokazać, że K^M minimalizowane jest przy takiej funkcji d jak w 2.3.1.

Gdy $X_{\{n\}}$ jest łańcuchem Markowa, rozwiązanie jest trudniejsze. Możemy przyjąć, że gdy zamiast rozważać elementy przestrzeni stanów łańcucha, zaczniemy rozważać podział przestrzeni na podzbiory, to będą one asymptotycznie tworzyć łańcuch Markowa. W ten sposób przyjmiemy, że zapytania o konkretne bloki dyskowe tworzą łańcuch Markowa.

Zdefiniujmy funkcję

$$W(M,q) = \begin{cases} 1 & \text{gdy } q \in \{X_1, X_2, \ldots, X_M\} \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

Wtedy

$$K^{(M)} = \sum_{q=0}^{|F|-1} W(M,q)$$

Dla $M = 1$, widać że $K = 1$. Dla $M = 2$

$$K^{(2)} = 2 - \sum_{q=0}^{|F|-1} \pi_f(q) P_f(q,q)$$

gdzie π_f jest rozkładem stacjonarnym łańcucha bloków dyskowych, a P_f — jego macierzą przejść.

Dla $M > 2$, problem staje się dużo trudniejszy, natomiast dla $M = 2$, jest równoważny maksymalizowaniu L_2

$$L_2 = \sum_{q=0}^{|F|-1} \pi_f(q) P_f(q,q) = \sum_{q=0}^{|F|-1} \sum_{x \in d^{-1}(q)} \sum_{y \in d^{-1}(q)} \pi_S(x) P_f(x,y)$$

lub też minimalizowaniu G_2

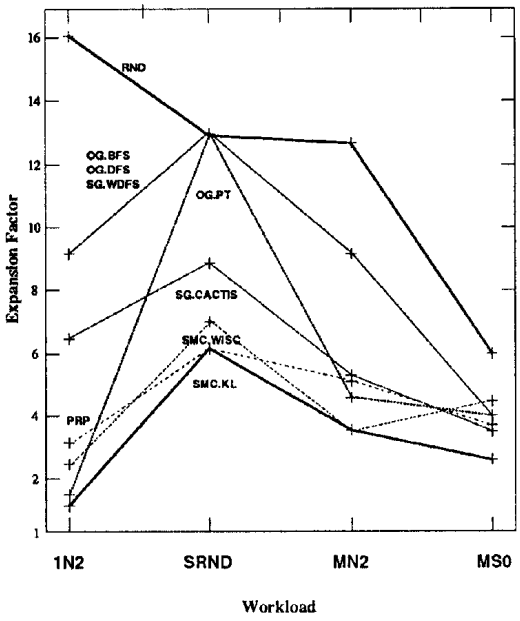
$$G_2 = \sum_{q=0}^{|F|-1} \pi_f(q)(1 - P_f(q,q)) = \sum_{q=0}^{|F|-1} \sum_{x \in d^{-1}(q)} \sum_{y : d(y) \neq q} \pi_S(x) P_f(x,y)$$

Minimalizowanie G_2 jest problemem podziału grafu tak, żeby suma krawędzi przechodzących pomiędzy różnymi jego fragmentami była możliwie najmniejsza. NP-zupełny jest nawet algorytm rozstrzygający czy, przy danym limicie obiektów przypadających na jeden podzbiór oraz limicie górnym na G_2, istnieje taki podział grafu. Są znane jednak algorytmy heurystyczne, pozwalające na wyznaczenie dobrego przybliżenia rozwiązania w czasie $O(n^2)$ lub nawet $O(n \log n)$, gdzie n to liczba wierzchołków w grafie.

Testy

Autorzy poświęcili całą pracę [10] testom. Są one tam dokładnie opisane. Głównym ich wnioskiem jest, że podejście autorów jest najbardziej efektywnym z porównywanych.

Na wszystkich wykresach, dane oznaczone jako SMC.* odnoszą się do algorytmów opartych o ten model ale z różnymi algorytmami do podziału grafu.

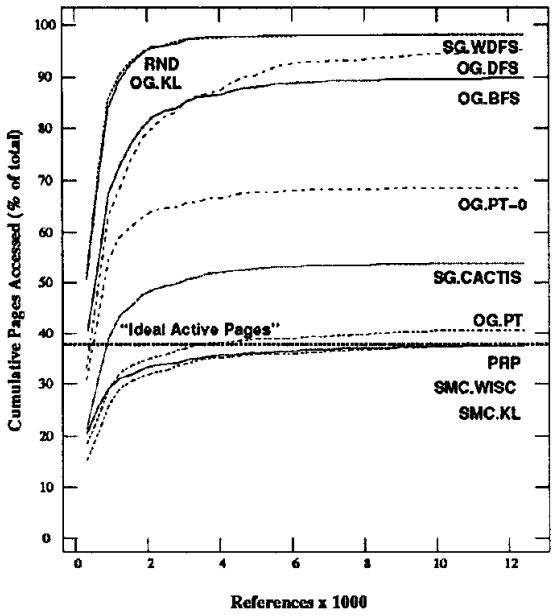


Rysunek 2.1: Expansion factor

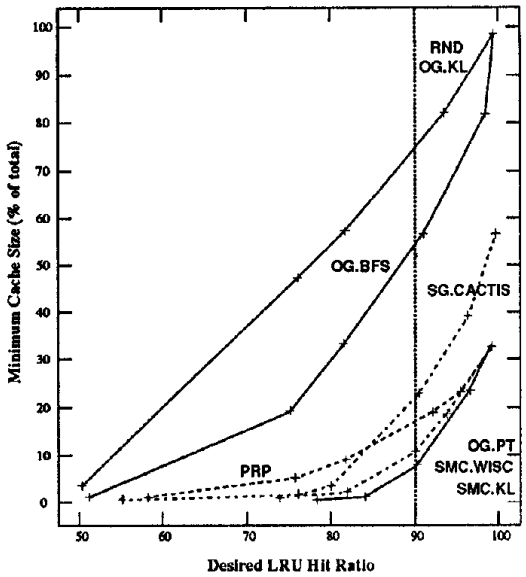
Nazwijmy iloraz liczby zajętych bloków do ilorazu liczby wszystkich obiektów do liczby obiektów mieszczących się w jednym bloku, czyli idealnego ułożenia jako „expansion factor". Im jest większy, tym gorzej ułożone są dane. Na wykresie 2.1 widać jego różnice w zależności od testu i algorytmu.

Na wykresie 2.2 widać ile razy poszczególne algorytmy musiały odczytać blok dyskowy. Widać, że część algorytmów, w tym ten opisany tutaj, po wypełnieniu buforów, praktycznie przestała odczytywać bloki z dysku.

Wykres 2.3.2 pokazuje potrzebny rozmiar bufora (zakładając, że będzie działać na zasadzie LRU) funkcji żądanego poziomu trafień w bufor przy dostępie do danych.



Rysunek 2.2: Odczyty z dysku



Rysunek 2.3: Rozmiar bufora

Rozdział 3

Symulowane wyżarzanie

Autorzy artykułu [5] stosują inną metodę rozwiązania tego problemu. Wykorzystują oni algorytm symulowanego wyżarzania do konstrukcji optymalnego podziału grafu. Zastosowany tam model jest prostszy od tego w [9]. Informacje na temat symulowanego wyżarzania można znaleźć w [1]

3.1. Definicje

Niech, tak jak poprzednio, S oznacza zbiór obiektów. Przyjmijmy, że mamy dany skierowany graf obiektów $G = (S, A)$, gdzie $A \subseteq S \times S$ oznacza zbiór krawędzi. Obiekty połączone są krawędzią wtedy i tylko wtedy, gdy odczyt drugiego z nich nastąpił bezpośrednio po odczycie pierwszego.

Przyjmijmy ponadto, że mamy funkcję wagową $w\colon A \to \mathbb{R}$, która odzwierciedla częstość używania danej krawędzi, tj. jeżeli mamy dwie krawędzie: $a_1 = (o_1, o_2)$ i $a_2 = (o_3, o_4)$, to jeżeli częściej zdarzało się, że o_2 był odczytywany bezpośrednio po o_1 niż o_4 po o_3, to $w(a_1) > w(a_2)$.

Przyjmijmy, tak jak poprzednio, że obiekty są tego samego rozmiaru, mamy zbiór F bloków dyskowych o stałym rozmiarze i chcemy znaleźć funkcję d przypisującą obiekty do bloków.

Zdefiniujmy też funkcję kosztu c dla danej funkcji d.

$$c(d) = \sum_{(o_1,o_2)\in A} w((o_1, o_2))(1 - \delta_{d(o_1),d(o_2)})$$

Chcemy znaleźć funkcję d minimalizującą c. Funkcja d to po prostu podział grafu.

3.2. Trudności

Ten problem jest NP-zupełny. Algorytm symulowanego wyżarzania dobrze sprawdza się dla problemów kombinatorycznych takich jak ten. Niestety, w klasycznej wersji, potrzeba mu bardzo dużo pamięci operacyjnej, ponieważ musiałby przechowywać w niej cały graf obiektów. Dlatego autorzy tego pomysłu wprowadzili swoje usprawnienia do tego algorytmu. Ich testy wydajnościowe oraz analiza pokazują, że zmodyfikowany algorytm zachowuje się bardzo dobrze.

3.3. Klasyczny algorytm wyżarzania

$d \leftarrow$ jakikolwiek naiwny podział grafu G
$T \leftarrow T_0$ **while**(**not** koniec)**do**
  while(**not** równowaga)**do**
    $d' \leftarrow d$ po losowej zamianie 2 obiektów z różnych bloków
    $\Delta c \leftarrow c(d') - c(d)$
    if($\Delta c < 0$)
      $d \leftarrow d'$
    else
      $d \leftarrow d'$ z prawdopodobieństwem $e^{\frac{-\Delta c}{T}}$
  end-while
  $T = \alpha T$
end-while

Ideą tego algorytmu jest błądzenie losowe, ale w określony sposób: na ogół błądzimy tylko tak, że wartość funkcji c się zmniejsza, natomiast czasem, im wcześniej, tym z większym prawdopodobieństwem — zwiększa się. Prawdopodobieństwo zależy od zmiennej T, zwanej dalej temperaturą. Efektywność tego algorytmu zależy od wyboru parametrów T_0 i α oraz zdefiniowania warunków „końca" i „równowagi". Autorzy przyjęli wartość $\alpha = 0.9$, założyli, że równowagę osiąga się po $\beta|S|$ obrotach wewnętrznej pętli dla $\beta \in \{5, 6, 7, 8\}$, zaś warunek końca to temperatura mniejsza od 0.001 oraz koszt, który nie zmienia się przez przynajmniej 2 obroty zewnętrznej pętli. T_0 ustalane jest za pomocą wzoru

$$T_0 = \frac{\overline{\Delta c}^{(+)}}{ln(\frac{m_2}{m_2\chi_0 - m_1(1-\chi_0)})}$$

gdzie χ_0 jest początkowym prawdopodobieństwem zaakceptowania ruchu pogarszającego rozwiązanie, a m_1, m_2 oraz $\overline{\Delta c}^{(+)}$ są wynikiem krótkiej symulacji algorytmu z przyjętym $T_0 = 0$ i oznaczają: m_1 liczbę kroków poprawiających wynik, m_2 liczbę kroków pogarszających wynik, a $\overline{\Delta c}^{(+)}$ — średnią wartość różnicy przy krokach pogarszających wynik. Przyjęto, że $\chi_0 = 0.95$. Analiza numeryczna pokazała szybką zbieżność tej formuły do stabilnego stanu.

3.4. Właściwy algorytm

Przyjmijmy, że $S_i = d^{-1}(i)$, czyli obiekty w i-tym bloku. Niech $v = |F|$. Weźmy $k < v$, takie, żeby k stron dyskowych zmieściło się w pamięci. Algorytm rozpoczyna się od jakiegokolwiek naiwnego podziału. Zakładamy, że k bloków jest w pamięci. Tym razem w wewnętrznej pętli znajdują się dwa możliwe typy zmian:

- zmiana załadowanych bloków
  Zamiast aktualnie załadowanych, do pamięci wczytywane są wylosowane od nowa bloki.

- zamiana obiektów
  Zmiana dwóch obiektów pomiędzy blokami będącymi w pamięci tak jak w oryginalnej wersji

$d \leftarrow$ jakikolwiek naiwny podział grafu G
$T \leftarrow T_0$ **while**(**not** koniec)**do**
  while(**not** równowaga)**do**
    if(zamiana obiektów)
      $d' \leftarrow d$ po losowej zamianie 2 obiektów z różnych bloków
      $\Delta c \leftarrow c(d') - c(d)$
      if($\Delta c < 0$)
        $d \leftarrow d'$
      else
        $d \leftarrow d'$ z prawdopodobieństwem $e^{\frac{-\Delta c}{T}}$
    else
      wczytaj nowy zestaw bloków
  end-while
  $T = \alpha T$
end-while

Wybór, czy zmieniać obiekty, czy załadowane bloki jest wynikiem losowania i z prawdopodobieństwem b będzie to zamiana załadowanych bloków. Okazuje się, że zbieżność do rozwiązania przy takim algorytmie można udowodnić czysto teoretycznie, jak i też istnieją testy praktyczne.

3.5. Analiza kosztu algorytmu

Podczas każdej zamiany wczytanych obiektów, nastąpi co najwyżej $2k$ operacji wejścia/wyjścia. Naturalne jest pytanie, ile razy, przy stałej temperaturze należy wymienić załadowane do pamięci bloki, aby każda para miała dużą szansę na znalezienie się w całości w pamięci. Niech n oznacza liczbę wymian, a zmienna losowa X — ile razy w pamięci była załadowana pewna konkretna para bloków. Rozkład zmiennej X jest rozkładem dwumianowym

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, x \in \{0 \dots n\}$$

gdzie

$$p = \frac{\binom{v-2}{k-2}}{\binom{v}{k}} = \frac{k(k-1)}{v(v-1)}$$

Stąd $\mu_x = np$. Jeżeli przyjmiemy rozsądne założenie, że chcemy aby $\mu_x = 1$, to otrzymamy $n = \frac{1}{p}$. Oczywiście wartości niecałkowite nas nie interesują, więc przez TD oznaczymy zaokrąglenie w górę. Ostatecznie

$$TD = \left\lceil \frac{v(v-1)}{k(k-1)} \right\rceil$$

W takim razie prawdopodobieństwo b powinno wynosić $\frac{TD}{M}$, gdzie M jest liczbą obrotów wewnętrznej pętli przy stałym T.

Niech h_i oznacza liczbę operacji wejścia/wyjścia potrzebnych do wymiany załadowanych bloków na inne, zakładając, że i z nich sie nie zmienia. Oczywiście, $h_i = 2(k-i)$. Niech p_i oznacza prawdopodobieństwo, że przy losowej wymianie bloków, i z nich zostanie. p_i ma rozkład hipergeometryczny.

$$p_i = \frac{\binom{k}{i}\binom{v-k}{k-i}}{\binom{v}{k}}, 0 \leqslant i \leqslant k$$

Zatem wartość oczekiwana liczby odczytanych bloków wynosi

$$\sum_{i=1}^{k} p_i h_i = \frac{2k(v-k)}{v}$$

Niech f oznacza liczbę iteracji zewnętrznej pętli. Wtedy całkowita oczekiwana liczba operacji we/wy wynosi

$$k + (fTD - 1)\frac{2k(v-k)}{v}$$

Dla porównania, dla klasycznego algorytmu wyżarzania jest to

$$\frac{2fM(v-k)}{v}$$

Jeżeli by zbadać iloraz, to otrzymamy w przybliżeniu

$$\frac{M(k-1)}{v(v-1)}$$

co przyjmując przykładowe dane $k = 10$, $v = 100$, $f = 110$, $|S| = 2048$, daje wynik co najmniej 15. Czyli znaczący wzrost wydajności.

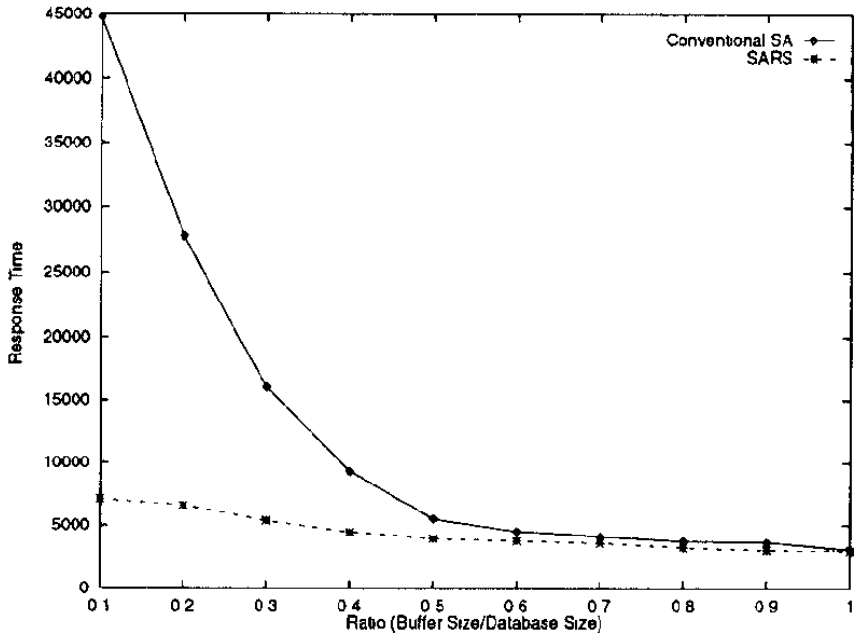
3.6. Analiza jakości rozwiązania

Znanym jest fakt, że algorytm symulowanego wyżarzania znajduje asymptotycznie poprawne rozwiązanie z prawdopodobieństwem 1. Okazuje się, że pomimo modyfikacji, ta własność nadal jest w mocy. Dowód tego faktu znajduje się w [5].

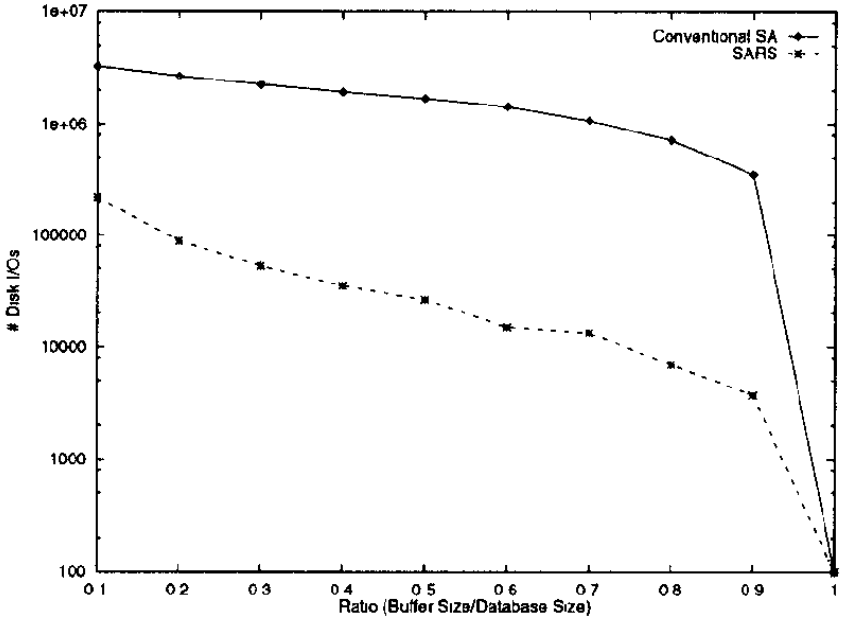
3.7. Testy

Poniżej znajdują się wyniki testów przeprowadzonych przez autorów. Dokładna metoda i środowisko przeprowadzania testów opisane są w artykule. Linią przerywaną zaznaczony jest zmodyfikowany algorytm wyżarzania (SARS), natomiast ciągłą — standardowy (SA). Wykres 3.7, przy przykładowych danych pokazuje zależność czasu reakcji od wielkości bufora. Widać, że przewaga SARS nad SA jest tym większa im mniejszy jest bufor, co było celem autorów.

Na wykresie 3.7 widać, że tak jak się spodziewano, w SARS liczba operacji wejścia/wyjścia spadła w stosunku do zwykłego SA.



Rysunek 3.1: Czas reakcji



Rysunek 3.2: Operacje we/wy

Rozdział 4

Od modelu do implementacji

Opisane powyżej modele abstrahują od sposobu ich implementacji. Niestety, luka, którą zostawiają powyższe prace jest duża. W pierwszym rozdziale stwierdziliśmy, że istotna jest np. interakcyjność SZBD. Powyższe podejścia zakładają, że nie zmienia się liczba obiektów w bazie i wszystkie one mają jednakowy rozmiar. Ukryte jest też założenie, że bazę należy reorganizować raz na jakiś czas. Nie podają one też strategii którą należy zastosować przy dodawaniu nowych obiektów i usuwaniu starych.

Poniżej znajdują się problemy, które powyższe prace pomijają:

- zmienny rozmiar obiektów
- jak przeplatać reorganizację z działaniem SZBD
- kiedy uruchamiać reorganizację
- jaką przyjmować strategię przy dodawaniu, usuwaniu i zmienianiu obiektów

Istnieją prace adresujące te problemy. Np [12] podaje ciekawą heurystykę tworzenia porządku w jakim obiekty powinny być ułożone, ale proponuje też architekturę systemu z wyróżnionymi trzema elementami:

- zbieraczem statystyk
  Jest to moduł działający asynchronicznie w stosunku do reszty SZBD informowany przez egzekutor lub dziennik o aktywności bazy danych. W tym konkretnym przypadku moduł ten zbiera informacje będące pewną formą bezwzględnego prawdopodobieństwa oraz prawdopodobieństw warunkowych w dostępie do obiektów.

- analizatorem statystyk
  Jest to moduł odpowiedzialny za zaprojektowanie optymalnego rozłożenia obiektów na dysku na podstawie zebranych statystyk. Jego funkcjonalność odpowiada mniej więcej temu, co jest wyliczane w rozdziałach 2 i 3. Jest on aktywowany przez zbieracza statystyk i po zakończeniu swojej pracy, aktywuje reorganizatora.

- reorganizatorem
  Jest to moduł odpowiedzialny fizyczne przenoszenie obiektów. Implementacja podana w tej pracy, jak sami autorzy zaznaczają, jest dość naiwna, natomiast można zrobić to lepiej, czego przykład znajduje się w [7].

4.1. Przykładowe rozwiązania części z nieomówionych problemów

Zastosowanie podziału całej funkcjonalności na moduły tak jak zaproponowano w [12] stanowi dobry punkt wyjścia do rozważań na temat problemów implementacyjnych. Ta praca podała jedno z możliwych rozwiązań pierwszych trzech wymienionych wyżej problemów. Streścić można je następująco:

- zmienny rozmiar obiektów
  Zastosowany sposób przypisywania obiektów do bloków dyskowych abstrahuje od ich pojemności. Sprowadza się on do wprowadzenia na nich porządku liniowego i następnie przypisywania ich do kolejnych bloków póki jest w nich miejsce. Może się okazać, że istniałoby bardziej optymalne rozwiązanie, ponieważ algorytm układający te obiekty zakłada, że skoro obiekty są blisko siebie względem tego porządku, to prawdopodobnie na raz znajdą się w pamięci, a nietrudno wyobrazić sobie sytuację, gdy znajdą się w niedalekich, ale różnych blokach dyskowych. Należy jednak pamiętać, że współczesne systemy operacyjne używają prefetchingu, więc to założenie nieźle oddaje rzeczywistość.

- jak przeplatać reorganizację z działaniem SZBD
  Niestety, zastosowano tu naiwne rozwiązanie polegające na chwilowym zablokowaniu dostępu do bazy innym transakcjom. Ciekawszy sposób znajduje się w pracy [7], który reorganizację traktuje jako jedną z transakcji bazodanowych i wykorzystuje partycjonowanie bazy w celu modyfikacji odnośników do przenoszonych obiektów. Sposób ten pozwala na dużą interakcyjność ze względu na to, że minimalizuje liczbę zamków, które reorganizator musi zamknąć, żeby móc dokonywać reorganizacji.

- kiedy uruchamiać reorganizację
  Podano tam warunek który opiera się na stosunku liczby zapytań o konkretne obiekty, przy których nie udało się uniknąć odczytu z dysku w stosunku do wszystkich zadanych.

4.2. Dalszy rozwój

Wydaje się, że omówiona tu architektura pasuje do omawianych w początkowych rozdziałach strategii partycjonowania grafu obiektów pomiędzy strony dyskowe. Trzeba jednak mieć na uwadze, że wszystkie omówione jak dotąd rozwiązania podążają za założeniem, że dodając nowe obiekty lub modyfikując istniejące, nie musimy dbać o ich efektywne układanie. To naprawić ma uruchamiana raz na jakiś czas reorganizacja. Inne podejście zastosowano np. w [8], gdzie skupiono się na lokalnej reorganizacji podając strategie dla wszelkich operacji. Nie znalazłem jak dotąd pracy która łączyłaby w sobie zarówno formalne podejście do sposobu reorganizacji w skali całej bazy danych, wspomniane strategie dla utrzymywania bazy względnie dobrze ułożonej oraz podano by sposób na efektywne fizyczne przemieszczanie obiektów.

4.3. Podsumowanie

Widać, że stosując metody statystyczne, można poprawić szeroko rozumianą wydajność składu obiektowej bazy danych. Jest wiele koncepcji w jaki sposób to robić. Dokładnie przedstawione zostały dwie z nich, obie wykorzystujące rachunek prawdopodobieństwa jako główne

narzędzie służące do ich projektowania. Istnieją jednak pomysły, które poparte są w większości tylko praktyką, a mimo to są efektywne. Na ogół ich projekt bliższy jest ich implementacji, natomiast przedstawione tutaj algorytmy pozostawiają jeszcze dużo miejsca do badań nad faktycznym ich wykorzystaniem.

Bibliografia

[1] Emile Aarts, Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, Cambridge, MA, 2001.

[3] Hector Garcia-Molina, Jennifer Widom, Jeffrey D. Ullman. *Database System Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[4] Sylvain Guinepain, Le Gruenwald. Research issues in automatic database clustering. *SIGMOD Record*, 34(1):33–38, 2005.

[5] Kien A. Hua, S. D. Lang, Wen K. Lee. A decomposition-based simulated annealing technique for data clustering. *PODS '94: Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, strony 117–128, New York, NY, USA, 1994. ACM.

[6] Jacek Jakubowski, Rafał Sztencel. *Wstęp do teorii prawdopodobieństwa*. Wydawnictwo Script, Warszawa, wydanie 3, 2004.

[7] Mohana K. Lakhamraju, Rajeev Rastogi, S. Seshadri, S. Sudarshan. On-line reorganization in object databases. *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, strony 58–69, New York, NY, USA, 2000. ACM.

[8] Mark L. Mcauliffe, Michael J. Carey, Marvin H. Solomon. Vclusters: a flexible, fine-grained object clustering mechanism. *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, strony 230–243, New York, NY, USA, 1998. ACM.

[9] Manolis M. Tsangaris, Jeffrey F. Naughton. A stochastic approach for clustering in object bases. *SIGMOD Record*, 20(2):12–21, 1991.

[10] Manolis M. Tsangaris, Jeffrey F. Naughton. On the performance of object clustering techniques. *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, strony 144–153, New York, NY, USA, 1992. ACM.

[11] Jeffrey D. Ullman, Jennifer D. Widom. *A First Course in Database Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, wydanie 3, 2007.

[12] Jr. William J. McIver, Roger King. Self-adaptive, on-line reclustering of complex object data. *SIGMOD Record*, 23(2):407–418, 1994.

[13] P. C. Yue, C. K. Wong. On the optimality of the probability ranking scheme in storage applications. *J. ACM*, 20(4):624–633, 1973.