STUDIA INFORMATICA 2011
Volume 32 Number 2A (96)

Michał GORAWSKI, Jakub ISAKOW
Politechnika Śląska, Instytut Informatyki

ALGORYTM ADAPTACYJNEGO BALANSOWANIA OBCIĄŻENIA ZAPYTAŃ W ROZPROSZONYCH PRZESTRZENNO-TEMPORALNYCH HURTOWNIACH DANYCH

Streszczenie. Artykuł opisuje algorytm adaptacyjnego balansowania obciążenia zapytań w przestrzenno-temporalnych hurtowniach danych. Przedstawione zostały istniejące algorytmy oraz porównanie ich działania z nowym algorytmem. Ponadto, omówione zostały podstawy teoretyczne algorytmu ALBQ (ang. Adaptive Load Balancing for Queries) oraz wyniki testów działania algorytmu, w zależności od wartości parametrów.

Słowa kluczowe: hurtownia danych, balansowanie obciążenia, systemy rozproszone

ADAPTIVE LOAD BALANCING FOR QUERIES ALGORITHM (AQLB) IN DISTRIBUTED SPATIAL DATA WAREHOUSES

Summary. The article presents an adaptive load balancing for queries algorithm (ALBQ algorithm) in distributed special data warehouses. It contains a description of currently used algorithms and a comparison of their behaviour with new algorithm. Moreover, the article describes theoretical basis of ALBQ algorithm and test results of using it accordingly to parameters' values.

Keywords: data warehouse, load balancing, distributed systems

1. Wstęp

W ostatnich latach liczba wykorzystywanych systemów przestrzenno-temporalnych hurtowni danych stale rośnie [12]. Są one wyspecjalizowane w przechowywaniu i przetwarzaniu danych, które charakteryzowane są przez pozycję w przestrzeni i przez czas. Przykładowo, może być to występowanie osobników danych gatunków ptaków na danym obszarze i w da-

nym czasie. Dane takie, zawarte w hurtowni, mogą stworzyć wiedzę o migracji ptaków. Innym przykładem mogą być pomiary temperatury i ciśnienia powietrza w zależności od pozycji miernika i czasu wykonywania pomiaru.

Przetwarzanie danych w przestrzenno-temporalnej hurtowni danych najczęściej sprowadza się do agregacji faktów zawartych w systemie. Agregaty mogą być przechowywane w systemie lub wyliczane *ad hoc*. To, czy wymagane będzie wyliczanie agregatu w dużej mierze zależy od typu zapytań, jakie są kierowane do systemu. Jeśli zapytanie wymaga przeprowadzenia agregacji *ad hoc*, niezbędne jest przeanalizowanie wszystkich faktów zawartych w systemie, celem wyznaczenia tych, które spełniają warunki zapytania. Jest to proces czasochłonny i w zależności od rozmiarów hurtowni, może trwać bardzo długo.

Jednym ze sposobów przyśpieszenia wyznaczania odpowiedzi na zapytania *ad hoc* jest wprowadzenie rozproszenia hurtowni danych tak, aby obliczenia były wykonywane jednocześnie na kilku jednostkach operacyjnych (węzłach) i były maksymalnie zrównoleglone. Podejście takie znacznie przyśpiesza realizację zapytania, należy jednak uwzględnić wszystkie ograniczenia tego rozwiązania.

Po pierwsze, poszczególne komputery składające się na hurtownie danych mogą nie być jednakowe (system heterogeniczny). Problemem ten rozwiązują algorytmy balansowania obciążenia, które przed rozesłaniem krotek do poszczególnych węzłów systemu analizują możliwości obliczeniowe danych jednostek. Zastosowanie takiego algorytmu wydaje się wystarczające, jednak należy zaznaczyć, że poszczególne jednostki obliczeniowe mogą wykonywać inne obliczenia (często niezwiązane z działaniem systemu hurtowni danych). Z punktu widzenia hurtowni danych, węzeł systemu nie wykonuje żadnych operacji, a mimo to obciążenie komputera się zmienia – z tego też powodu przetwarzanie zapytania w danej jednostce może zajmować różny czas dla poszczególnych prób.

Autorzy artykułu prezentują algorytm balansowania obciążeń, który uwzględnia zmieniający się stan poszczególnych węzłów rozproszonej hurtowni przestrzenno-temporalnej. W rozdziale 2. artykułu zostały przedstawione obecnie stosowane metody balansowania obciążeń. Rozdział 3. zawiera opis ideowy algorytmu ALBQ (ang. Adaptive Load Balancing for Queries). W rozdziale 4. zawarto wyniki testów przykładowej implementacji, natomiast 5. przedstawia wnioski i spostrzeżenia, jakie poczyniono podczas prac nad algorytmem.

2. Wcześniejsze rozwiązania

Istotą balansowania systemu rozproszonego jest osiągnięcie stanu maksymalnej wydajności. Zdefiniowanie pojęcia wydajności zależy od celów określonego systemu, jednak w przypadku hurtowni danych można stwierdzić, że najważniejszym zadaniem jest wyznaczanie wyników na

zapytania kierowane do systemu. Dlatego też w kontekście systemu rozproszonej hurtowni danych uznaje się, że maksymalizacja wydajności polega na zapewnieniu jak najkrótszego średniego czasu wyznaczenia odpowiedzi na zapytanie przesłane do systemu. Dodać należy, że w przypadku hurtowni przestrzenno-temporalnych zapytania kierowane do systemu są przeważnie zakresowe. Z takiego założenia można wysunąć wniosek, że wydajność takiego systemu (średni czas przetwarzania zapytania) jest ściśle związana z podziałem tablicy faktów pomiędzy poszczególne węzły. Dla zapewnienia maksymalnego zbalansowania, niezbędne jest takie rozmieszczenie danych, aby typowe zapytania zakresowe były przetwarzane równolegle.

W zależności od charakterystyki systemu oraz parametrów poszczególnych węzłów stosuje się jedną z metod balansowania. Można je podzielić na dwie kategorie: metody balansowania statycznego oraz metody balansowania dynamicznego. Algorytmy statyczne dzielą tablicę faktów wg ustalonej odgórnie polityki, często wybranej w trakcie projektowania systemu – podział nie uwzględnia stanu systemu. W odróżnieniu od algorytmów statycznych, algorytmy balansowania dynamicznego biorą pod uwagę aktualny stan systemu i na podstawie jego wartości decydują, w jaki sposób podzielić dane pomiędzy węzły. Stan systemu, na podstawie którego ustalany jest podział danych przez algorytmy dynamiczne, może być definiowanych przez np. topologie sieci, jej szybkość, wartości parametrów (obliczeniowych i pamięciowych) poszczególnych węzłów, częstość napływania zapytań itd. Poprzednie rozwiązania balansowania przedstawiono w [1] oraz [2].

3. Algorytm ALBQ w rozproszonych przestrzenno-temporalnych hurtowniach danych

Algorytm ALBQ proponowany w niniejszym artykule powstał, by sprostać wymaganiom balansowania w systemach DSTDW (ang. *Distibuted Spatio-Temporal Data Warehouse*) oraz uniknąć wad istniejących rozwiązań, jednocześnie łącząc ich zalety. Prezentowane rozwiązanie bazuje na idei dostosowywania (adaptacji) stanu systemu do każdego wykonywanego zapytania. Wiąże się to z koniecznością sprawdzania i ewentualnego modyfikowania stanu systemu przed wykonaniem zapytania. Jednak algorytm potrafi zbalansować system niezależnie od zmian w nim zachodzących.

Podstawowym warunkiem działania algorytmu jest uruchomienie modułu nadzorcy procesu balansowania (dalej nazywanego serwerem). Steruje on wszystkimi procesami zachodzącymi w systemie oraz stanowi widoczną dla użytkownika „fasadę" systemu – jego podstawowym zadaniem jest zbieranie danych statystycznych, dotyczących rozmieszczenia danych w systemie, jak również działania samych węzłów.

Węzeł natomiast zachowuje swoje najważniejsze zadanie przechowywania i przetwarzania danych, lecz dodatkowo uzyskuje możliwość przesyłania danych do innych węzłów oraz

określenia swojego obciążenia. Obciążenie to powinno uwzględniać moc obliczeniową danego węzła i może być np. średnim obciążeniem procesorów w określonym przedziale czasu. Ponadto, algorytm wymaga, aby węzeł był w stanie oszacować (lub wyznaczyć) koszt przetwarzania danego zapytania, zanim podjęta zostanie decyzja o jego wykonaniu.

Inną cechą stworzonego algorytmu jest próba takiego zarządzania obciążeniem, aby w przypadku awarii któregoś z elementów systemu utrata danych była jak najmniej dotkliwa. W takiej sytuacji przestrzeń zawierającą krotki podzielono na regiony oraz zdefiniowano założenie, że krotki leżące w jednym regionie nie mogą być przechowywane tylko i wyłącznie w jednym węźle. Założenie to jest bliskie idei zastosowania atrybutu partycjonującego, stosowanego w algorytmie opisanym w [2].

3.1. Parametry algorytmu

Parametry serwera

Na działanie serwera wpływa wiele parametrów dostosowujących działanie algorytmu według potrzeb użytkowników. Do najważniejszych parametrów należą współczynnik bezwładności algorytmu w oraz próg przeciążenia *p*. Pierwszy decyduje o tym, jak mają być modyfikowane otrzymane wyniki metod przeprowadzających balansowanie. Wartości współczynnika znajdują się w zakresie $w \in \langle 0,1)$. Drugi, próg przeciążenia *p*, decyduje o częstości uruchamiania procesu balansowania – opisuje wymagane odchylenie obciążenia węzła od wartości odniesienia. Niska wartość tego parametru sprawi, że nawet najmniejsze odchylenie wartości obciążenia przez algorytm uznane zostanie jako stan niezbalansowania. Dodatkowo, podczas tworzenia regionów wykorzystywany jest parametr określający ile regionów (partycji) ma zostać wydzielonych.

Parametry węzła

Najważniejszym parametrem węzła jest jego moc obliczeniowa *P*, która może zmieniać się w trakcie działania systemu, oddając zmiany w obciążeniu maszyny. Wartość parametru nie musi być bezpośrednio powiązana z jakąkolwiek cechą węzła, jednak stosunek parametrów *P* poszczególnych węzłów powinien odzwierciedlać stosunek ich możliwości obliczeniowych.

3.2. Fazy przetwarzania zapytania

Algorytm przetwarzania zapytania
Niech *Q* to zapytanie kierowane do DSTDW
Po otrzymaniu zapytania *Q* serwer wysyła je do wszystkich węzłów systemu
Każdy węzeł $N_i, 0 \le i < n$
    Wysyła do serwera l_i = szacowany koszt przetworzenia Q przez N_i
Serwer wyznacza próg niezbalansowania (B_P = średnie obciążenie + *p*)

```
Serwer wyznacza niezbalansowanie każdego węzła b_i = l_i − B_P, 0 ≤ i < n
Każdy węzeł N_i, 0 ≤ i < n
      Otrzymuje wartość swojego niezbalansowania b_i
      Jeśli b_i > 0 wtedy N_i przeprowadza balansowanie
Serwer wysyła informację do wszystkich węzłów o zakończeniu balansowania
   Każdy węzeł N_i, 0 ≤ i < n
      Wykonuje Q, a wynik wysyła do serwera
Serwer zbiera i łączy wyniki poszczególnych węzłów
Wynik końcowy wysyłany jest jako odpowiedź na zapytanie Q
```

Zapytanie kierowane do hurtowni danych jest przesyłane do serwera systemu. Użytkownik wykorzystujący hurtownię danych nie jest świadomy jej integralnej budowy – komunikacja z systemem odbywa się poprzez serwer.

Po otrzymaniu zapytania serwer rozsyła je do wszystkich zarejestrowanych węzłów. Każdy z węzłów, który bierze udział w wykonywaniu zapytania analizuje je pod względem wymaganej mocy obliczeniowej, potrzebnej do wykonania zapytania. Ponieważ moc niezbędna do wyznaczenia wyniku jest wprost proporcjonalna do liczby krotek, biorących udział w zapytaniu (cechy krotki muszą spełniać warunki zapytania), wyznaczenie mocy obliczeniowej sprowadza się do oszacowania liczby krotek, których wartość złoży się na wynik zapytania. Oszacowana wartość wraz z aktualnym obciążeniem węzła zostaje przesłana do serwera. Należy zwrócić uwagę na fakt, że obciążenie węzła nie zostaje zmodyfikowane, gdyż zgodnie z założeniami szacowanie jest względnie prostą operacją i nie powoduje wzrostu obciążenia.

Po otrzymaniu danych od wszystkich węzłów biorących udział w wykonywaniu zapytania serwer normalizuje obciążenia węzłów z uwzględnieniem szacowanego wzrostu ich wartości oraz wyznacza współczynnik niezbalansowania b_i dla każdego z nich. Jeśli obliczona wartość jest większa od dopuszczalnego progu, to jest ona przesyłana do węzła, gdzie na jej podstawie wyznaczana jest liczba krotek, które powinny zostać przesunięte do innych węzłów. Liczba ta ustalana jest w taki sposób, by po balansowaniu obciążenie węzła było w granicach progu od średniego obciążenia węzłów w systemie.

$$\Delta t = \frac{b_i \times P}{100}, \tag{1}$$

gdzie: Δt – zmiana liczby krotek, b_i – przeciążenie (zmiana obciążenia wymagana do powrotu do stanu zbalansowania), a P – moc obliczeniowa węzła.

Po ustaleniu liczby krotek węzeł pobiera odpowiednią ich liczbę oraz tworzy statystykę regionów, do których należą. Statystyka ta jest przesyłana do serwera, ustalającego, do których węzłów przesłać jaką liczbę krotek. Po zakończeniu balansowania węzły rozpoczynają wyznaczanie odpowiedzi na zapytanie, które przesłane do serwera i połączone tworzą odpowiedź DSTDW na zapytanie użytkownika.

3.3. Koszty wykonywanych operacji

Dla działania systemu najważniejszą cechą każdego węzła jest jego obciążenie. Na tę wartość składa się zarówno liczba przechowywanych krotek, jak i zapytania, w których dany węzeł brał udział. Obciążenie węzła rośnie, gdy otrzymuje on krotki do przechowania lub wykonuje zapytanie, natomiast maleje, gdy wysyła dane do innego węzła (np. w wyniku balansowania systemu). Zmiany obciążenia węzła są odpowiednio modyfikowane współczynnikiem mocy węzła P.

Węzeł po wykonaniu zapytania aktualizuje swoje obciążenie, zgodnie ze wzorem:

$$\Delta l = 100\frac{t}{P}, \tag{2}$$

gdzie: Δl – zmiana obciążenia węzła, t – liczba krotek, do których wymagany był dostęp, P – moc obliczeniowa węzła.

Ponadto, zarówno otrzymywanie nowych, jak i wysyłanie już przechowywanych krotek przez węzeł wpływa na jego obciążenie. Zmiana obciążenia podczas otrzymywania danych wyrażona jest wzorem (3), natomiast podczas ich wysyłania wzorem (4):

$$\Delta l = \left\lceil \frac{\Delta t \times 100}{P} \right\rceil \tag{3}$$

$$\Delta l = \left\lfloor \frac{\Delta t \times 100}{P} \right\rfloor, \tag{4}$$

gdzie: Δl – zmiana obciążenia węzła; Δt – zmiana liczby przechowywanych krotek; P – moc obliczeniowa węzła.

3.4. Balansowanie systemu

Proces balansowania systemu rozbito na dwa etapy. Podział ten wymuszony jest koniecznością zapewnienia stabilności działania węzłów, które nie wymagają balansowania przy jednoczesnym balansowaniu węzłów tego potrzebujących.

```
Algorytm balansowania
Dla każdego węzła N_i
Oszacowanie zmiany obciążenia
Wysłanie szacowanej zmiany obciążenia do serwera
Wyznaczenie niezbędnych przesunięć obciążenia {metoda Balance}
Dla każdego węzła N_i
       Przesłanie wartości niezbalansowania b_i do węzła N_i
       Jeśli b_i > 0 wtedy
          Wybranie paczki danych do przesłania
          Przesłanie statystyk paczki do serwera
          Wyznaczenie węzłów docelowych przez serwer{metoda Dispatch}
Wysłanie danych do odpowiednich węzłów
Wysłanie informacji o zakończeniu balansowania
```

Pierwszym etapem balansowania jest zbieranie danych przesyłanych przez węzły systemu oraz ustalanie, w jaki sposób obciążenie poszczególnych węzłów powinno zostać zmodyfi-

kowane, aby cały system został zbalansowany. Zadaniem tym zajmuje się metoda *Balance*, która zwraca dane sumarycznie dla danego węzła. Wartość ta oznacza jedynie, o ile powinno zmniejszyć się obciążenie węzła. To węzeł decyduje, w jaki sposób powinno to nastąpić. Po otrzymaniu wartości przeciążenia węzeł za pomocą wzoru 1 określa liczbę krotek do wysłania. Następnie pobiera wyznaczoną liczbę krotek i wyznacza statystyki regionów. Statystyki te, przesłane do serwera, biorą udział w drugim etapie balansowania systemu. Ustaleniem, w jaki sposób krotki danego węzła powinny zostać rozesłane zajmuje się metoda *Dispatch*, wywoływana przez serwer.

Metoda Balance

```
Metoda Balance
Niech L_P = obciążenie odniesienia
Niech L_max = L(N_max) = max(L(N_i)) = maksymalne obciążenie wśród węzłów
Jeśli L_max ≤ L_P + p wtedy zakończ działanie metody
Niech L_min = L(N_min) = min(L(N_i)) = minimalne obciążenie wśród węzłów
Przenieś odpowiednią ilość obciążenia l z N_max do N_min
       Niech l = min(L_max-L_P, L_P-L_min)
       Uwzględnij współczynnik bezwładności
       Zapisz przesunięcie w macierzy przesunięć: m[N_max][N_min]+=l
Przejdź na początek metody
```

gdzie: L_i oznacza obciążenie *i*-tego węzła równe $L(N_i)$; p – próg przeciążenia; m – macierz wynikową, będącą chwilową polityką podziału.

Jej zadaniem jest wyznaczanie najmocniej przeciążonego węzła, a następnie takie przeniesienie części jego obciążenia, aby nowe obciążenie było równe obciążeniu będącemu punktem odniesienia L_p. Obciążenie to wyznacza punkt, do którego dążą wszystkie węzły, natomiast osiągnięcie go oznacza uzyskanie stanu idealnego.

Operacja ta jest powtarzana tak długo, jak długo istnieją węzły, których obciążenie jest większe przynajmniej o wartość progu od obciążenia odniesienia. By móc sterować szybkością procesu wprowadzono współczynnik bezwładności (jego wartość w zakresie $\langle 0,1))$, którego zadaniem jest odpowiednie zmniejszanie „siły” algorytmu.

Wynikiem działania metody jest macierz przesunięć obciążenia m_{ij}, w której indeksami pól są identyfikatory poszczególnych węzłów. Dodatkowo, zachowana musi być zależność, że w każdej parze dwóch dowolnych węzłów, między którymi powinno zostać przeniesione obciążenie, tylko jeden z węzłów jest węzłem źródłowym (wysyłającym krotki, czyli zmniejszającym swe obciążenie), a drugi węzłem docelowym (otrzymującym krotki – zwiększającym swe obciążenie).

Metoda Dispatch

Niech N_i (dla i = *1..n*) oznacza kolekcję węzłów docelowych, do których mają zostać wysłane krotki z węzła źródłowego N_Z, natomiast P_i oznacza procentową wartość chwilowej strategii podziału dla N_i (stosowane typy strategii zostały opisane w rozdziale 3.5). $T(R_j)$ (dla $j = 1..m$) to liczba krotek w poszczególnych regionach R_j, które mają zostać wysłane z N_Z.

Celem metody jest takie rozdzielenie krotek wysyłanych z N_Z (których liczba jest równa $k = \sum_{j=1}^{m} T(R_j)$, aby do węzła N_i trafiła dokładnie $K_i = k \times P_i$ liczba krotek. Co więcej, podział powinien zostać przeprowadzony tak, aby nie doszło do sytuacji, w której wszystkie krotki danego regionu znalazły się w tylko jednym węźle.

Metoda Dispatch

Niech $k = \sum_{j=1}^{m} T(R_j)$,
Niech k_{left} stanowi liczbę krotek do rozdzielenia = k
Niech K_i stanowi liczbę krotek, które mają trafić do węzła N_i {metoda *CreateDistribution*}
Dla każdego regionu R_j
    Dla każdego węzła docelowego N_i
        Niech $t = \frac{T(R_j) \times K_i}{k}$
        Zapisz przydział krotek w macierzy wynikowej: $S_{ij} = t$
        Zmniejsz licznik krotek nierozdzielonych: $k_{left} = k_{left} - t$
Jeśli $k_{left} = 0$ wtedy zakończ działanie metody
  Dla każdego regionu R_j
    Niech r_{left} = liczba krotek nieprzydzielonych w regionie R_j
    Niech N_d = ostatni docelowy węzeł dla krotek z R_j
    Dla każdego węzła N_i
      N_d = cyklicznie następny węzeł względem N_d
      Niech n_{left} = liczba krotek, które mogą zostać wysłane do N_d
      Jeśli liczba krotek w N_d nie osiągnęła K_d wtedy
        $S_{dj} = min(n_{left}, r_{left})$
        $r_{left}t = min(n_{left}, r_{left})$.
      Jeśli $r_{left} = 0$ wtedy ostatni docelowy węzeł dla krotek z $R_j = N_d$

Początkowo wyznaczany jest podział wartości k względem kolekcji P – wykonuje to metoda *CreateDistribution*. W proponowanej implementacji metoda ta uwzględnia jedynie kolekcję P: $K_i = k \times P_i$

Następnie dla każdego regionu i każdego węzła wyznaczana jest wartość przesunięcia, która umieszczana jest w macierzy wynikowej S_{ij}

$$S_{ij} = \frac{T(R_j) \times K_i}{k} \tag{5}$$

Każde pole tej macierzy oznacza liczbę krotek z regionu R_j do przesłania z węzła N_Z do węzła N_i. Zastosowanie wzoru (5) nie gwarantuje podziału wszystkich krotek względem węzłów i regionów, dlatego też w drugiej części metody sprawdza się czy wszystkie krotki dla danego N_i zostały rozdzielone. Jeśli nie, to pobierany jest identyfikator węzła, do którego jako ostatniego wysłane zostały krotki z analizowanego regionu. Następnie pobierany jest (cyklicznie – zgodnie z zasadą działania algorytmu Round Robin) kolejny węzeł i przenoszona jest do niego odpowiednia liczba krotek. Fakt, że ostatnio wykorzystywany węzeł zostanie uznany za docelowy jako ostatni, ogranicza możliwość umieszczenia wszystkich krotek danego regionu w tym właśnie węźle.

3.5. Strategie podziału

Kluczowy dla działania całego systemu jest moduł przechowujący oraz zarządzający strategiami (ang. *policy*) podziału. Strategie te stanowią integralną część algorytmu i decydują o tym, w jaki sposób rozsyłane są dane (nowe lub już zawarte w systemie) pomiędzy węzłami. Rozróżnić można dwie strategie podziału: główną oraz chwilową.

Strategia główna decyduje o tym, w jaki sposób rozsyłane będą krotki dodawane do systemu. Suma wartości głównej strategii podziału dla poszczególnych węzłów powinna być stała w trakcie działania systemu.

Strategia chwilowa tworzona jest podczas przeprowadzania balansowania systemu (w trakcie wykonywania metody *Balance*) i służy jedynie do wyznaczenia sposobu przesłania krotek (metoda *Dispatch)*. Strategia chwilowa tworzona jest dla danego węzła i reprezentuje obciążenie, które powinno zostać przeniesione z danego węzła do węzłów, dla których istnieją wartości w strategii chwilowej. Ze względu na charakterystykę chwilowej strategii podziału suma wartości w niej znajdujących się nie musi być stała.

4. Testy

Na środowisko testowe składało się 11 komputerów stacjonarnych klasy PC. Każdy z nich wyposażony został w procesor Intel Core 2 Quad (4x2,66GHz) i 2GB pamięci RAM. Ponadto, komputery połączone zostały siecią klasy Ethernet tak, że każda jednostka w otoczeniu sieciowym ma dostęp do innych komputerów biorących udział w teście. Dodatkowo, na każdym komputerze dostępne są system operacyjny Windows XP oraz wirtualna maszyna JRE 1.6.

Zanim oceniono efektywność algorytmu ALBQ, przetestowano wpływ wartości poszczególnych parametrów na zachowanie algorytmu. Jako pierwszy zbadany został wpływ współczynnika bezwładności. Wyniki testów pokazują, że mniejsza wartość współczynnika bezwładności algorytmu prowadzi do szybszych, lecz mniej dokładnych procesów balansowania. Proces ten w znaczący sposób zwiększa czas przetwarzania zapytania, jednak konieczność przeprowadzenia mniej dokładnego wyrównania obciążeń węzłów (wysoki współczynnik bezwładności) sprawia, że szybkość balansowania jest większa. Czas balansowania zależy więc od liczby przesyłanych krotek. Dodatkowo, podczas pierwszego balansowania nawiązywane są połączenia pomiędzy węzłami, które utrzymywane są przez cały okres działania systemu.

Drugie doświadczenie miało na celu zbadanie wpływu wartości progu niezbalansowania na działanie systemu. W tym celu wykorzystano węzły o zróżnicowanej mocy obliczeniowej dla dwóch wartości progu: 2% oraz 5%.

Zgodnie z przewidywaniami, zmniejszenie progu skutkuje bardziej „rygorystycznym” balansowaniem systemu. Podczas tego doświadczenia nie zauważono negatywnych i nieocze-

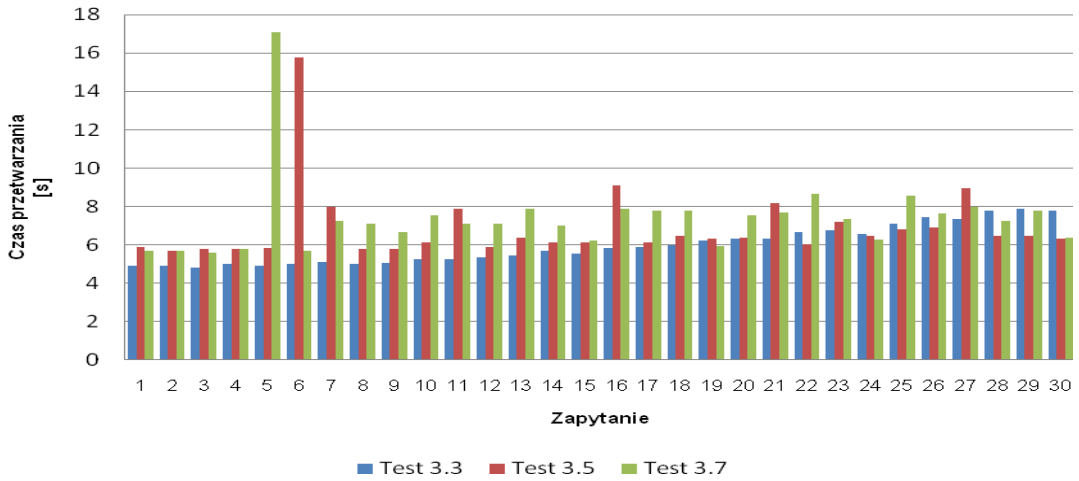
kiwanych wpływów zmiany badanego parametru. Natomiast w połączeniu z parametrem współczynnika bezwładności odnotowano, że czas osiągnięcia optymalnego stanu systemu wydłuża się wraz ze zmniejszeniem parametru progu niezbalansowania. Powód wydaje się oczywisty: potrzeba dokładniejszego zbalansowania przy tej samej szybkości tego procesu skutkuje zwiększeniem liczby zapytań, koniecznych do osiągnięcia wskazanego progu.

Druga seria doświadczeń miała na celu sprawdzenie zachowania systemu w sytuacji stopniowego zmniejszenia mocy obliczeniowej (wzrostu obciążenia) jednego z węzłów. Pozwoliło to na sprawdzenie reakcji algorytmu dla niewielkich zmian stanu systemu oraz wyznaczenie momentu, w którym algorytm umożliwia przetworzenie zapytania szybciej niż system bez balansowania.

W trakcie działania systemu zmieniane było obciążenie jednego węzła. Zmiana była stała – obciążenie zwiększało się o 5% co każde 3 zapytania.

Podczas działania systemu bez wykorzystania balansowania adaptacyjnego (rys. 1, test 3.3) czas przetwarzania zapytań stopniowo rośnie. Przyczyną jest spadek sumarycznej mocy obliczeniowej oraz osłabienie jednego z węzłów w stosunku do pozostałych.

Dla porównania przeprowadzono testy algorytmu adaptacyjnego balansowania w dwóch wariantach. W pierwszym (rys. 1, test 3.5) próg niezbalansowania był większy, ale jednocześnie współczynnik bezwładności algorytmu był mniejszy.



Rys. 1. Czas przetwarzania zapytań
Fig. 1. Query execution time

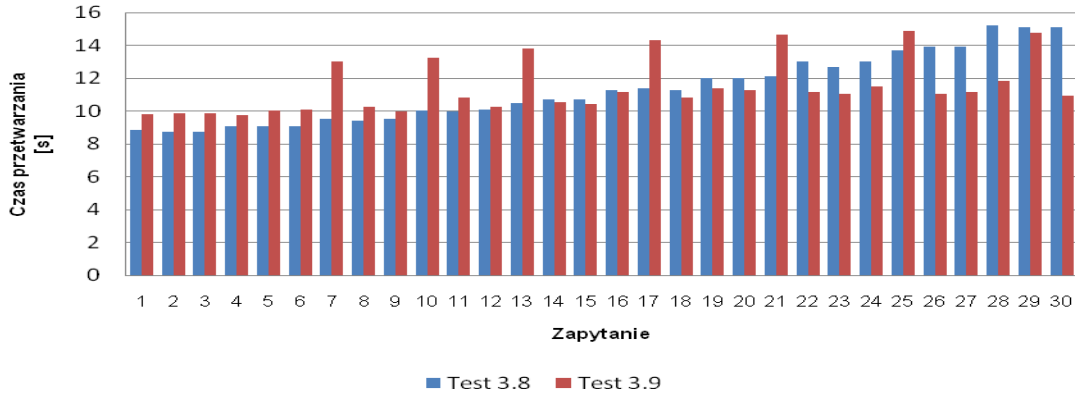
Efektem takiego doboru parametrów było wymuszenie przez algorytm ALBQ przeprowadzania rzadszych, lecz dokładniejszych procesów balansowania (odstępy między poszczególnymi operacjami balansowania były większe, jednak gdy system był już balansowany, algorytm starał się dość dokładnie wyrównać obciążenie wszystkich węzłów).

W drugim wariancie (rys. 1, test 3.7) próg został zmniejszony przy jednoczesnym zwiększeniu współczynnika bezwładności, w celu wymuszenia realizacji mniej dokładnych procesów balansowania, jednak sam proces miał być uruchamiany znacznie częściej.

Wyniki testów jasno pokazują, że czas potrzebny na zbalansowanie systemu w znaczący sposób wydłuża przetwarzanie zapytania. Widać również, że w przypadku systemu o podanej specyfikacji (500 tys. krotek, 5 węzłów o wskazanej mocy obliczeniowej) algorytm przyśpiesza przetwarzanie przy spadku mocy do 60%-65% początkowej wartości. Biorąc pod uwagę fakt, że obciążenie jednostki może być znacznie większe, wydaje się, że taki próg wydajności algorytmu ALBQ jest akceptowalny.

Kolejne testy miał na celu zbadanie działania systemu w takim samym scenariuszu, jak poprzednie, z tą różnicą, że tym razem w systemie znajdowało się 1 milion krotek.

Jako parametry systemu wybrano próg 2% oraz współczynnik bezwładności algorytmu 0,5. Są to wartości pośrednie pomiędzy wartościami badanymi w poprzednich testach. Porównanie wyników (rys. 2.) odbędzie się w stosunku do przypadku, w którym balansowanie adaptacyjne zostało wyłączone.



Rys. 2. Czas przetwarzania kolejnych zapytań
Fig. 2. Query execution time

Można zauważyć, że w porównaniu z poprzednim doświadczeniem wzrost czasu przetwarzania zapytania dla systemu bez algorytmu adaptacyjnego balansowania wzrasta bardziej dynamicznie. Spowodowane jest to oczywiście większą liczbą przetwarzanych krotek. Dla danego systemu stosowanie algorytmu okazuje się wydajne już przy spadku mocy obliczeniowej węzła do 80% wartości maksymalnej. Dodatkowo, przy spadku mocy obliczeniowej do 55% wartości maksymalnej, wykorzystanie algorytmu sprawia, że przetwarzanie zapytania wraz z balansowaniem systemu jest szybsze, niż wyznaczenie odpowiedzi w systemie niewyposażonym w badany algorytm.

Dodatkowe testy wpływu rozmiaru kolekcji danych na działanie algorytmu ALBQ potwierdzają, że skuteczność algorytmu zależy od średniej liczby krotek, przechowywanej przez węzły. Strata czasowa wynikająca z zastosowania algorytmu (konieczność analizowania stanu węzłów) jest tym mniejsza, im dłużej trwa właściwe wyznaczenie wyniku.

5. Wnioski

Opisany algorytm ALBQ w znacznym stopniu opiera się na analizie przedstawionej w [6]. Algorytm ma na celu balansowanie obciążenia węzłów przed właściwym wykonaniem zapytania, skierowanego do hurtowni danych. Podejście to służyć ma minimalizacji czasu potrzebnego do wyznaczenia odpowiedzi na zapytanie, jednak wymaga zastosowania pewnej heurystyki w szacowaniu wzrostu obciążenia, wynikającego z przetwarzania zapytania. Fakt ten sprawia, że w porównaniu z systemem niekorzystającym z algorytmu, czas przetwarzania zapytania w zbalansowanym systemie jest dłuższy (wydłużony o szacowanie oraz synchronizację pracy węzłów). Jednak, jak pokazały testy, wykorzystanie algorytmu okazuje się przydatne w sytuacji, w której węzły nie są w stanie w sposób ciągły zagwarantować maksymalnej mocy obliczeniowej.

Kolejną zaletą algorytmu jest fakt, że podczas procesu ładowania kolejnych danych do systemu (proces ETL) mogą one zostać rozdzielone pomiędzy istniejące węzły, korzystając z danych statystycznych, zebranych w trakcie działania systemu. Algorytm ALBQ nie zajmuje się tylko balansowaniem adaptacyjnym w chwili wykonywania zapytania, lecz także balansuje dane podczas procesu ETL.

Wydajność przedstawionego rozwiązania w dużej mierze zależy od doboru wartości parametrów. Jak pokazują testy, każdy z parametrów ma wpływ na sposób działania algorytmu. Dobór odpowiednich wartości utrudnia fakt, że dla różnych systemów (złożonych z różnej liczby węzłów oraz przechowujących różną ilość danych) dana wartość parametrów powoduje inne działanie systemu. W związku z tym, podczas tworzenia konfiguracji systemu, do grupy parametrów, których wartości należy ustawić, dodać trzeba atrybuty systemu oraz środowiska, w którym będzie on działał.

Wyniki testów pokazują, że zasadne jest stosowanie przedstawionego algorytmu ALBQ w rozproszonych przestrzenno-temporalnych hurtowniach danych, w których węzły uruchomione są na komputerach o zmiennej dostępnej mocy obliczeniowej. Ponadto, uzyskane wyniki pozwalają twierdzić, że dalszy rozwój algorytmu może jeszcze bardziej usprawnić proces balansowania obciążenia.

Można wyróżnić kilka dróg rozwoju testowej implementacji algorytmu ALBQ. Rozwój taki pozwoli na przeanalizowanie innych scenariuszy testowych niż omówione w ramach tego artykułu. Zmiany, które można wprowadzić to np.:

- umożliwienie dynamicznego dodawania i usuwania węzłów z systemu,
- uogólnienie algorytmu, by pracował na dowolnym schemacie danych,
- modyfikacja pracy tak, by możliwe było uruchomienie kilku zapytań jednocześnie.

BIBLIOGRAFIA

1. Gorawski M., Chechelski R.: Spatial Telemetric Data Warehouse Balancing Algorithm in Oracle9i/Java Environment. Intelligent Information Processing and Web Mining. 2005. s. 357÷365.
2. Gorawski M., Gorawski M.: Modified R-MVB Tree and BTV Algorithm Used in a Distributed Spatio-temporal Data Warehouse. PPAM 2007, s. 199÷208.
3. Bernardino J. R., Furtado P. S., Madeira H. C.: Approximate Query Answering Using Data Warehouse Striping. J. Intell. Inf. Syst. 19, 2002, s. 145÷167.
4. Gorawski M.: Architecture of Parallel Spatial Data Warehouse: Balancing Algorithm and Resumption of Data Extraction, Proceeding of the 2005 conference on Software Engineering: Evolution and Emerging Technologies. IOS Press 2005, s. 49÷59.
5. Kolsi N., Abdellatif A., Ghedira K.: Data warehouse access using multi-agent system. Distrib. Parallel Databases, 25, 2009, s. 29÷45.
6. Paton N., et al.: Autonomic query parallelization using non-dedicated computers: an evaluation of adaptivity options. The VLDB Journal, 18, 2009, s. 119÷140.
7. Bernardino J., Madeira H.: Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing.
8. Gorawski M.: Definiowanie schematów rozszerzonej gwiazdy kaskadowej, Wydawnictwo Naukowo-Techniczne 2007, s. 103÷114.
9. Papadias D., Kalnis P., Zhang J., Tao Y.: Efficient OLAP Operations in Spatial Data Warehouses. Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases. Springer-Verlag 2001, s. 443÷459.
10. Kambayashi Y., Winiwarter W., Arikawa M.: Introduction to Special Issue on Data Warehousing and Knowledge Discovery. J. Intell. Inf. Syst. 19, 2002, s. 143÷144.
11. Ganesan P., Bawa M., Garcia-Molina H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. Proceedings of the Thirtieth international conference on Very large data bases - Volume 30. VLDB Endowment 2004, s. 444÷455.
12. Gorawski M.: Zaawansowane hurtownie danych, Monografia, Studia Informatica 2010.
13. Gounaris A., Paton N. W., Fernandes A. A. A., Sakellariou R.: Self-monitoring query execution for adaptive query processing. Data Knowl. Eng. 51, 2004, s. 325÷348.
14. Stockinger K., Wu K., Shoshani A.: Strategies for processing ad hoc queries on large data warehouses. Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, ACM 2002, s. 72÷79.

Recenzent: Dr inż. Andrzej Sikorski

Wpłynęło do Redakcji 31 stycznia 2011 r.

Abstract

The article presents a new algorithm to conduct adaptive balancing of node load. Prepared algorithm should be useful in distributed spatial data warehouses, which are built using machines characterized by variable available CPU power.

The algorithm introduces new step during processing query, which goal is to predict the change of each node's load. The prediction is made by each node by estimating the cost of given query. The predicted nodes are normalized by server (node which purpose is storing statistics of data distribution and nodes' state) and if one's node load is about to exceed given threshold the server conducts system balancing. The overload of each node is recalculated to number of tuples, which should be transferred between nodes to change the system state. After determining the number of tuples to be moved from each node and sending this data to server, it determines the destination nodes for tuples. After moving data between nodes the query is processed on each node independently and the results are joined by server to create the answer to users query.

The behavior of the algorithm is parameter-dependent. The authors defined to parameters: the balancing inertia and the overload threshold. The article contains descriptions of tests describing the behavior of algorithm for different values of these parameters.

Moreover, there is description of tests showing the behavior of algorithm in the scenario, where the power of one of the nodes is variable.

Adresy

Michał GORAWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, m.gorawski@gmail.com.
Jakub ISAKOW: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, jakub.isakow@gmail.com.