STUDIA INFORMATICA 2011
Volume 32 Number 2B (97)

Mariusz FRYDRYCH, Wojciech HORZELSKI
Wyższa Szkoła Informatyki w Łodzi,
Uniwersytet Łódzki, Wydział Matematyki i Informatyki

ZASTOSOWANIE MECHANIZMU CDB DO ROZWIJANIA ALIASÓW POCZTOWYCH

Streszczenie. W artykule przedstawiono system szybkiego rozwiązywania aliasów pocztowych oparty na mechanizmie Constant Database (CDB). Zaprezentowano strukturę bazy danych, wykorzystywane funkcje haszujące oraz implementację funkcjonującą w rzeczywistym systemie poczty elektronicznej. Wyniki przeprowadzonych testów pokazały imponującą wydajność takiego rozwiązania, nawet dla systemów o bardzo dużej ilości aliasów. Zastosowanie modyfikacji funkcji haszującej dodatkowo poprawiło osiągane wyniki.

Słowa kluczowe: poczta elektroniczna, Constant Database

USE OF CDB BATABASES FOR E-MAIL ALIASES TRANSLATION

Summary. The paper presents an electronic mail system where CDB concept was implemented for the alias translation. The paper shows database structure, used hash functions and implementation details. Conducted tests showed significant advantage of described solution over traditional approach. Modification of hashing function leads to even better results.

Keywords: electronic mail, Constant Database

1. Wstęp

Constant DataBase (CDB) to baza danych wykorzystująca dwupoziomowe tablice z haszowaniem dla łańcuchów znakowych [1]. Składa się ona z rekordów w postaci *key->value*. Wyszukiwanie polega na znalezieniu wartości *value* na podstawie klucza *key*, gdzie *value* jest wartością funkcji haszującej dla *key*. Klucze i wartości są łańcuchami tekstowymi (stringami). Całość bazy jest zapisana w jednym pliku o rozmiarze do 4 GB, w wewnętrznym, nie-

zależnym od architektury systemu operacyjnego formacie. Pomysł takiej bazy danych pochodzi od D.J. Bersteina [2].

CDB służy jedynie do wyszukiwania rekordów. Operacje dynamiczne typu zmiana, dodanie, usuniecie rekordów, realizowane są wyłącznie przez całkowite przebudowanie bazy od nowa (wymagane jest ponowne wyliczenie wartości funkcji haszującej). Jednak ta pozorna niedogodność jest opłacalna, bowiem wyszukiwanie jest bardzo szybkie, a przebudowanie całej bazy trwa zwykle krócej niż pojedyncze operacje typu INSERT, UPDATE i DELETE w znanych implementacjach SQL-baz, takich jak Postgresql, Sqlite czy MySql [3, 4].

Zastosowanie CDB spotykamy najczęściej w systemach poczty elektronicznej, głównie na serwerach SMTP, przy rozwiazywaniu i wyszukiwaniu stosownych danych, takich jak m.in. aliasy pocztowe, trasowanie dla domen pocztowych oraz znajdowanie zawartość przeróżnych list typu *black*, *white*, *spam* czy *ham* [5].

2. Struktura bazy CDB

Jądro bazy CDB stanowi kolekcja 256 tablic haszujących, które można uważać za 256-elementowy wektor list cyklicznych. Każdej z 256 tablic przyporządkowana jest lista cykliczna (nazywana cyklem) o długości $hashOfflen(h_0)$, gdzie h_0 jest numerem tablicy. Lista ta zawiera referencje do danych.

Operacja skracania (haszowania) odbywa się dwustopniowo, tzn. dla klucza *key* wyznaczamy parę liczb:

$$String \ni key \rightarrow (h_0, h_1) \in \{0, 1, 2, \ldots, 2^{24} - 1\} \times \{0, 1, 2, \ldots, 2^8 - 1\},$$

gdzie:

$$h_0 = hash(key) \ mod \ 256,$$

$$h_1 = hash(key)/256,$$

a *hash* jest odpowiednią funkcją haszującą:

$$\text{String} \ni key \rightarrow hash(key) \in \{0, 1, 2, \ldots, 2^{32} - 1\}.$$

Liczba h_0 jest indeksem (numerem) tablicy haszującej, natomiast:

$$h_1 \ mod \ hashOfflen(h_0)$$

jest numerem slotu tablicy h_0, od którego zaczyna się poszukiwanie klucza.

Odległością klucza *key* od jego skrótu (*hash*(*key*)) nazywamy liczbę całkowitą nieujemną, ze zbioru $\{0, 1, 2, \ldots, hashOfflen(h_0) - 1\}$, wyrażającą liczbę slotów w cyklu h_0, licząc od: $h_1 \ mod \ hashOfflen(h_0)$ do slotu odpowiadającego poszukiwanej wartości klucza *key*. Oczywiście czym mniejsza jest ta odległość, tym szybciej zostaje znaleziony klucz.

Przemyślana sekwencja rozmieszczenia metadanych bazy CDB w pliku (serializacja) skutkuje po pierwsze bardzo krótkim czasem potrzebnym do przebudowania bazy od nowa oraz błyskawicznym wyszukiwaniem kluczy.

2.1. Format wejściowy

Plik źródłowy, na podstawie którego buduje się bazę CDB, składa się z sekwencji rekordów, z których każdy ma następujący format:

+klen , *vlen* : *key−>value* \n

gdzie *klen*, *vlen* to odpowiednio, długość klucza i wartości, a \n to znak nowej linii. Koniec pliku zaznaczony jest dodatkowym pustym wierszem.

3. Funkcja haszująca

D. J. Bernstein używa w bazie CDB następującej funkcji haszującej DJB (kod w jęz. C):

```
unsigned int cdb_hash (
                    unsigned char *buf ,
                    unsigned int len
                      ) {
   unsigned int h ;
   h = 5381; /*liczba pierwsza */
   while ( len ) {
      --len;
                  h += (h << 5);
                  h ^= ( unsigned int ) *buf++;
             }
   return h;
 }
```

gdzie pierwszy parametr *buf* to łańcuch tekstowy do zahaszowania (odpowiada on wartości *key* w CDB), natomiast drugi parametr *len* to długość argumentu *buf*. W wyniku działania funkcji otrzymujemy liczbę całkowitą 32-bitowa (*unsigned int*), będącą haszem łańcucha *buf*.

Jak widać, funkcja *cdb_hash*() działa w następujący sposób: zaczynamy od liczby pierwszej p = 5381, mnożymy ją przez 33 (nie uwzględniając skończoności typu *unsigned int*), następnie do wyniku przykładamy operator różnicy symetrycznej XOR z pierwszym znakiem parametru *buf*. Powyższe czynności powtarzamy cyklicznie, aż do wyczerpania długości łańcucha *buf* (XOR-ujemy z kolejnymi znakami łańcucha buf).

4. Implementacja

Baza CDB została zastosowana do rozwiązywania aliasów pocztowych w systemie poczty elektronicznej wyższej uczelni, obsługującym wiele domen. Na serwerze pocztowym SMTP [6], będącym wymiennikiem poczty (wskazywanym przez rekord MX DNS-u), obsługiwane są:

- różne domeny pocztowe,
- aliasy dla każdej domeny z osobna,
- wczesne odrzucenie nieistniejącego adresata wewnątrz naszej korporacji.

Implementacja bazuje na serwerze *Exim*. Istotne fragmenty pliku konfiguracyjnego przedstawiono poniżej:

```
domainlist local_domains = @
domainlist relay_to_domains = /usr/local/etc/exim\
/virtual_domains
hostlist relay_from_hosts = localhost:212.151.155.19
...
begin acl
acl_check_rcpt:
...
drop message = Not in all-users
domains = +relay_to_domains
local_parts = ! cdb;/usr/local/etc/exim/\
virtual_domains.d/all-users.cdb
...
begin routers
...
# virtual domains
virtual:
driver = redirect
domains = dsearch;/usr/local/etc/exim\
/virtual_domains.d
data = ${lookup{$local_part}cdb{/usr/local/etc\
/exim/virtual_domains.d/$domain.cdb}\
{$value}{$local_part@abc.naszauczelnia.edu.pl}}
allow_defer = yes
allow_fail = yes
no_more
...
```

Plik *virtual_domains* w opisywanym zastosowaniu ma postać:

```
virtual:
driver = redirect
domains = dsearch;/usr/local/etc/exim\
/virtual_domains.d
data = ${lookup{$local_part}cdb{/usr/local/etc\
/exim/virtual_domains.d/$domain.cdb}\
{$value}{$local_part@abc.naszauczelnia.edu.pl}}
allow_defer = yes
allow_fail = yes
no_more
...
```

Do budowy bazy CDB wykorzystano opisaną wcześniej funkcję haszującą DJB oraz własną funkcję o analogicznej konstrukcji, różniącą się od oryginalnej początkową liczbą pierwszą. Zastosowano tu *czwartą liczbę Fermata*:

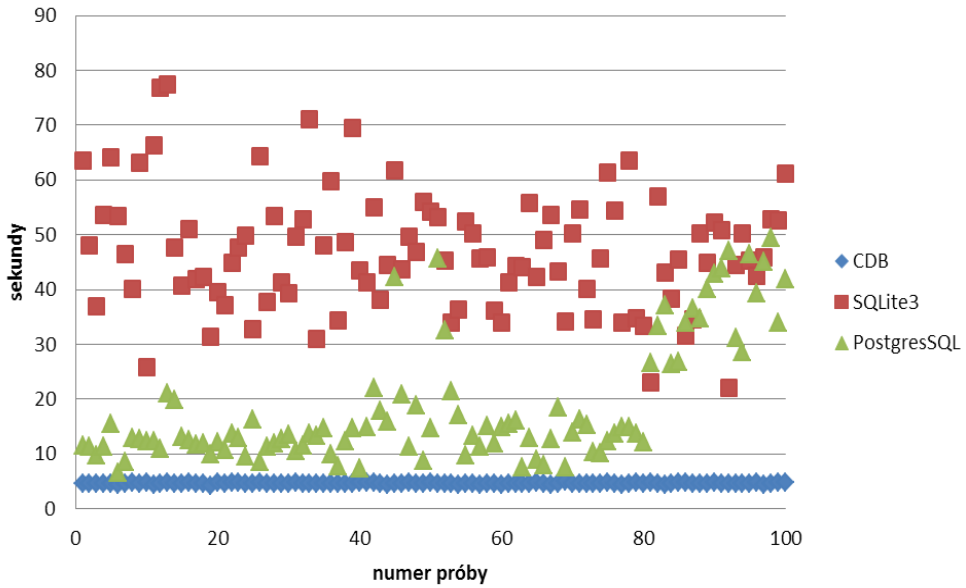
$$f_4 = 2^{2^4} + 1,$$

równą 65537 (jest to liczba pierwsza). Ta modyfikacja dała polepszenie haszowania pod względem liczby kolizji. DJB daje średnio 119 kolizji na milion kluczy, natomiast zastosowana modyfikacja daje 106 kolizji na milion kluczy.

5. Testy

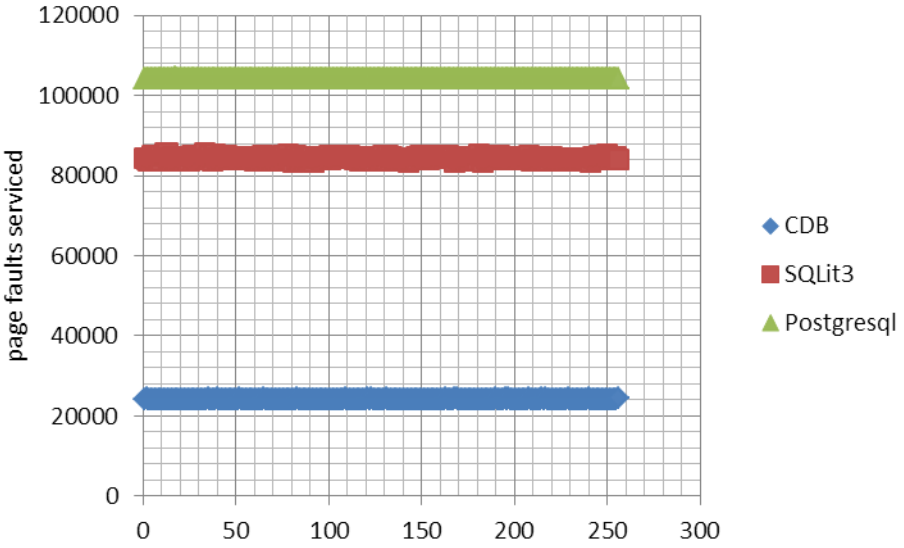
Dla zbadania wydajności zastosowanego rozwiązania przeprowadzono serię testów. Badania były prowadzone na komputerze Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66 GHz, z pamięcią 4 GB i czterema logicznymi (dwoma fizycznie) procesorami w architekturze 64-bitowej amd64, działającego pod kontrolą systemu operacyjnego OpenBSD Euler 4.8 GENERIC.MP#335 amd64. Testowano losowe bazy złożone z 25 milionów rekordów.

Czas odpowiedzi na zapytanie o 128 losowych rekordów testowanej bazy został zestawiony z analogicznym pytaniem do baz PostgersSQL oraz Sqlite3. Ze względu na bardzo duże czasy (różniące się o rząd wielkości), dla implementacji opartej na MySql, nie uwzględniono jej w dalszych zestawieniach. Wyniki przedstawione zostały na rys. 1.



Rys. 1. Porównanie czasu odpowiedzi baz CDB, Postgersql oraz Sqlite3
Fig. 1. The comparison of CDB, Postgresql and Sqlite3 response time

Zbadano też wykorzystanie zasobów systemowych. Wyniki dla parametru *ru_mjrflt* (liczba błędów stron pamięci dla kolejnych prób) przedstawia rys. 2.



Rys. 2. Porównanie liczby błędów stronicowania pamięci dla baz CDB, Postgresql oraz Sqlite3
Fig. 2. The comparison number of page faults CDB, Postgresql and Sqlite3 response time

Analogiczne wyniki osiągnięto dla pozostałych wskaźników wykorzystania zasobów systemowych (*ru_nvcsw*, *ru_nivcsw* itd.).

Badano też czas przebudowy bazy po jej zmianie (dodawaniu lub usuwaniu rekordów). W przypadku CDB wymagana jest ponowna budowa całego pliku. Dla 25 milionów rekordów czas ten wynosił 4 minuty i 37 sekund (średnia ze 100 prób). Analogiczna operacja dla pozostałych baz (w ich przypadku wymagane jest jedynie ponowne indeksowanie bazy) zajmowała znacznie dłuższy czas, który wynosił odpowiednio 14 godzin i 23 minuty dla Postgresql oraz 13 godzin i 19 minut dla Sqlite3. Testy prowadzane były przy wykorzystaniu indeksów o strukturze *B-tree*.

W czasie testów poszukiwano lepszej funkcji haszującej. Jej modyfikacja polegała na znacznym zwiększeniu wykorzystywanej w algorytmie liczby pierwszej. Podyktowane to zostało przekonaniem, że taka jej zamiana powoduje istotne powiększenie orbity przyjmowanych przez funkcję haszującą wartości. Oprócz standardowej funkcji (DJB) funkcji haszującej zastosowano również własną modyfikację opisaną wcześniej. Dla próby wielkości 1106852 rekordów uzyskano poprawę średniej ilości kolizji. Wyniki zestawiono w tabeli 1.

Tabela 1

Uśredniona ze 100 prób liczba kolizji kluczy dla bazy o rozmiarze 25 mln. rekordów i losowych kluczach

Funkcja haszująca	Ilość kolizji	Procent kolizji
zmodyfikowana	106	0,00957670944263551043
DJB	119	0,01075121154409080888

6. Wnioski

Przeprowadzone badania wykazały doskonałą sprawność opisywanego rozwiązania. Dodatkowo zmiana funkcji haszującej pozwala w niewielkim stopniu poprawić sprawność bazy CDB. Porównanie z innymi, klasycznymi podejściami (Postgresql, Sqlite czy MySql), wskazuje na zdecydowaną przewagę mechanizmu Constant DataBase do obsługi systemów pocztowych.

Zastosowana modyfikacja funkcji haszującej zmniejsza średnią ilość kolizji ze 119 dla DJB do 106 na milion rekordów. Wszystkie są kolizjami drugiego rzędu. W czasie testów nie udało się wygenerować kolizji trzeciego i wyższych rzędów.

BIBLIOGRAFIA

1. Askitis N., Sinha R.: Engineering scalable, cache and space efficient tries for strings. The VLDB Journal, Vol. 19, No. 5, 2010.
2. Berstein D. J. home page, http://cr.yp.to/djb.html.
3. Wood D.: Programming Internet Mail. O'Reilly, 1999.
4. Hughes L.: Internet e-mail Protocols, Standards and Implementation. Artech House Publishers, 1998.
5. Loshin P.: Essential Email Standards: RFCs and Protocols Made Practical. John Wiley and Sons, 1999.
6. Rhoton J.: Programmer's Guide to Internet Mail: SMTP, POP, IMAP, and LDAP. Digital Press, 1999.

Recenzent: Dr inż. Adam Duszeńko

Wpłynęło do Redakcji 31 stycznia 2011 r.

Abstract

The paper presents an electronic mail system which implements Constant Database concept for the aliases translation. CDB was implemented over the Exim server. Several tests were done using the original DJB hash function and slightly modified hash function. The modified function uses the fourth Fermat number as a seed. The conducted research revealed

a significant advantage of D.J. Berstein concepts over standard Postgresql, Sqlite or MySql approaches. Our modification decreases the average number of collisions form 119 to 106 for one million records.

Adresy

Mariusz FRYDRYCH: Wyższa Szkoła Informatyki w Łodzi, Rzgowska 17a, 93-008 Łodź, Uniwersytet Łódzki, Wydział Matematyki i Informatyki, ul. Banacha 22, 90-238 Łódź, Polska, frydrych@wsinf.edu.pl.
Wojciech HORZELSKI: Uniwersytet Łódzki, Wydział Matematyki i Informatyki, ul. Banacha 22, 90-238 Łódź, Polska, horzel@imul.uni.lodz.pl.