Filtrowanie plików CSV

Mariusz Nogala

21 luty 2016

1. PLIKI PROJEKTU

Pliki źródłowe:

mn_alg.cpp

mn_csv.cpp

mn_defs.cpp

mn_fastbuf.cpp

mn_int64.cpp

mn_reader.cpp

mn_vect.cpp

mn_alg.h

mn_csv.h

mn_defs.h

mn_fastbuf.h

mn_int64.h

mn_reader.h

mn_vect.h

Pliki do programu testowego:

create_sample_csv.cpp (opcjonalnie)

mn_czas.h

test.cpp

Pliki wsadowe do kompilacji:

kompilacja_msvc.bat

kompilacja_gnu.bat

kompilacja_borland.bat

kompilacja_linux.sh

2. TEST

Przykładowy test został wykonany na komputerze DELL z 20 procesorami:

Target: x86_64-redhat-linux

Thread model: posix

gcc version 4.4.6 20110731 (Red Hat 4.4.6-3) (GCC)

Przykładowy plik "input.csv" (około 6 GB) został wygenerowany za pomocą programu create_sample_csv.cpp.

Liczba wierszy źródłowych: 90 000 000

Rozmiar pliku: 6 098 997 309 bajtów

Liczba kolumn w wierszach: 11

Zastosowany warunek filtrowania:

const char* Filter = "((k1 > 600 and k2 = 446) or k3 = 999) or (k5 = 2*k6 + 1 and k8 >= k9)";

Liczba wierszy wynikowych: 20 069

Czasy przetwarzania:

1 wątek: 279 sek

2 wątki: 168 sek

3 wątki: 119 sek

4 wątki: 99 sek

8 wątków: 64 sek

12 wątków: 55 sek

16 wątków: 51 sek

20 wątków: 48 sek

3. INTERFEJS PROGRAMISTYCZNY

Jedyna wyjściowa funkcja modułu to **csvfilter**. Posiada ona 12 parametrów wywołania. Zwraca 1 gdy jest sukces, lub kod błędu (<= 0).

```
int csvfilter(const char* InputName, const char* ResultName,
const char* Filter, unsigned nThreads, int bOverWriteResult, int bTest,
my_int64* p_nInputRows, my_int64* p_nResultRows, char* ErrorBuf, unsigned
ErrorBufSize, char* AstBuf, unsigned AstBufSize);
```

Funkcja ta jest funkcją języka C (extern "C"), aby można ją było stosować zarówno z poziomu C++ jak i zwykłego C. W tej formie można ją też łatwo umieścić w bibliotece DLL.

Parametry funkcji:

1) const char* InputName (IN)

Nazwa pliku źródłowego.

2) const char* ResultName (IN, opcjonalny)

Nazwa pliku wynikowego.

Jeśli parametr ten jest NULL, lub wskazuje na pusty string, to wówczas funkcja nie tworzy pliku wynikowego, ale algorytm działa i wylicza liczbę wierszy wynikowych.

3) const char* Filter (IN)

Logiczny warunek filtrowania.

4) unsigned nThreads (IN)

Liczba wątków.

5) int bOverWriteResult (IN)

Parametr logiczny, która wymusza nadpisanie istniejącego już pliku wynikowego lub zwraca błąd, gdy plik o tej nazwie już istnieje.

6) int bTest (IN)

Jeśli ten parametr logiczny jest niezerowy, to wówczas dla pierwszego przetwarzanego wiersza wykonywany jest test wyliczenia warunku filtrowania. Jeśli test ten zakończy się błędem, funkcja przerywa działanie generując odpowiedni komunikat błędu. Przykładowo, błąd jest generowany jeśli warunek filtrowania operuje nieistniejącą w pliku nazwą kolumny. Jeśli ten parametr jest równy zero, wówczas algorytm działa, ale w przypadku błędu wiersz jest po prostu odrzucany, czyli warunek filtrowania zwraca 0.

7) my_int64* p_nInputRows (OUT, opcjonalny)

Wskaźnik do 64-bitowej zmiennej całkowitej, gdzie na wyjściu zostanie umieszczona liczba wierszy źródłowych.

8) my_int64* p_nResultRows (OUT, opcjonalny)

Wskaźnik do 64-bitowej zmiennej całkowitej, gdzie na wyjściu zostanie umieszczona liczba wierszy wynikowych.

9) char* ErrorBuf (OUT, opcjonalny)

Wskaźnik do bufora, gdzie zostanie zwrócony tekst błędu.

10) unsigned ErrorBufSize (IN, opcjonalny)

Rozmiar bufora ErrorBuf w bajtach.

11) char* AstBuf (OUT, opcjonalny)

Wskaźnik do bufora, gdzie zostanie umieszczony przetłumaczony już przez program warunek filtrowania. Jeśli program poprawnie przetłumaczył wejściowy warunek filtrowania Filter, to AstBuf powinien zawierać tekst podobny do Filter. Tekst ten może się trochę różnić od źródłowego pod względem liczby znaków spacji oraz nawiasów. Operatory logiczne (OR, AND, <>) są tu wyświetlane odpowiednio jako (||, &&, !=).

12) unsigned AstBufSize (IN, opcjonalny)

Rozmiar bufora AstBuf w bajtach.

4. OPIS DZIAŁANIA ALGORYTMU

1) Brak limitu dla rozmiaru pliku źródłowego (może być większy niż 2 GB). Liczba wierszy jest również praktycznie nieograniczona, ponieważ licznik wierszy jest 64-bitowy.

2) Minimalne zużycie pamięci.

3) Nagłówek z nazwami kolumn w pliku źródłowym jest opcjonalny. Jeśli nagłówek istnieje, zostanie on również umieszczony w pliku wynikowym.

UWAGA: Nazwy kolumn (wybranych lub wszystkich) mogą być umieszczane w znakach cudzysłowia. Mogą one wtedy zawierać znaki spacji oraz inne nietypowe znaki. Wówczas warunek filtrowania też powinien używać nazw tych kolumn w cudzysłowiach. Jeśli jednak nazwy kolumn w cudzysłowiach są proste (np. "kolumnaA", "kolumnaB"), to w warunku filtrowania nazwy mogą występować bez cudzysłowia.

4) Warunek logiczny filtrowania może używać następujących operatorów:

alternatywa: ||, OR, or, Or, oR

koniunkcja: &&, AND, and, And, ...

porównanie: >, <, >=, <=, =, == (to samo co "=") !=, <> (to samo co "!=")

arytmetyczne: +, -, *, /

słowo kluczowe: NULL

5) Algorytm obsługuje wartość NULL. Może ona występować w pliku CSV, gdy po przecinku zamiast wartości liczbowej jest od razu (lub po białych znakach) kolejny przecinek, lub znak nowej linii lub słowo kluczowe NULL.

UWAGA: Algorytm traktuje też jako NULL każdy inny tekst w kolumnie, którego nie da się przetłumaczyć na liczbę. Wartość NULL jest też generowana w przypadku błędu (np. dzielenie przez 0).

Ponadto algorytm pozwala stosować wyrażenia: "kolumna = NULL", "kolumna != NULL". Zwracają one takie same wartości jak odpowiednie SQL-owe wyrażenia: "IS NULL", "IS NOT NULL".

6) Liczba kolumn w wierszach nie musi być stała. Jeśli dany wiersz ma mniej kolumn, wówczas zmienna reprezentująca brakującą kolumnę będzie w warunku filtrowania miała wartość NULL. Warunek filtrowania może jednak w takim przypadku również zwracać wartość logiczną 1 dla takiego wiersza (w przypadku alternatywy).

7) Wszystkie operacje na liczbach wykonywane są po konwersji na typ NUMERIC o precyzji 18 i odpowiedniej skali. Stąd zastosowanie 64 bitowych liczb całkowitych w programie. Program unika konwersji do typu zmiennoprzecinkowego double. Dlatego nie ma tu żadnych zaokrągleń. Operatory porównania działają zatem precyzyjnie.

Cała arytmetyka (+,-,*,/) wykonywana jest również na typie int64 (moduł: mn_int64.cpp).

8) Wieloplatformowość zapewnia moduł mn_defs.cpp.

Windows:

wątki: _beginthread, sekcje krytyczne

typ my_int64: __int64

Linux:

wątki: biblioteka pthread, muteksy

typ my_int64: long long

5. MECHANIZMY ALGORYTMU

UWAGA: Kluczowym modułem programu jest plik **mn_alg.cpp**. Znajdują się tu algorytmy kompilacji i wykonania warunku filtrowania. Warunek jest kompilowany 1 raz na samym początku, a następnie wykonywany dla każdego wiersza źródłowego. Funkcja wykonania warunku jest REENTRANT, czyli wykonuje się niezależnie w każdym wątku, bez wzajemnego blokowania. Dlatego nie można w niej wywoływać funkcji, które zwykle nie są REENTRANT (np. funkcje dynamicznego przydziału pamięci).

1) Wersja jednowątkowa.

Algorytm działa tu sekwencyjnie bez tworzenia wątków potomnych. Plik jest czytany (klasa: READER, moduł: mn_reader.cpp) w paczkach 32-kB, filtrowany i od razu zapisywany do pliku wynikowego.

2) Wersja wielowątkowa.

Wszystkie wątki odczytują plik źródłowy za pomocą tego samego obiektu klasy READER w paczkach 32-kB. Odczyt pliku przez wątek jest umieszczony w obrębie muteksu (Windows: sekcja krytyczna). Klasa READER zapewnia, że dany wątek dostanie zawsze bufor z całkowitą liczbą wierszy. Jednocześnie obiekt READER tworzy i systematycznie powiększa (realloc) w pamięci mapę bitów dla pobieranych przez wątki wierszy. Ostatecznie mapa ta zajmuje w pamięci tyle bitów (nie bajtów !!!)

ile jest wierszy w pliku źródłowym. Bity te ustawione są początkowo na wartość 0. Każdy wątek filtruje w pętli pobrany przez siebie zbiór wierszy i zapamiętuje (64-bitowe) numery wierszy, które spełniają warunek filtrowania. Po wykonaniu pętli dla wierszy, wątek ustawia na wartość 1 bity w mapie obiektu READER dla tych wierszy, które spełniają warunek filtrowania. Dokonuje tego (również w ramach muteksu !!!) za pomocą funkcji SetMap obiektu READER.

Po zakończeniu pracy wątków potomnych (join), wątek główny w oparciu o utworzoną przez wątki mapę bitów dokonuje zapisu do pliku wynikowego tych wierszy, dla których bit w mapie jest równy 1. Jest to robione w ten sposb, że plik źródłowy jest ponownie czytany przez obiekt READER (porcjami 32-kB) i w oparciu o informacje w mapie bitów dany wiersz jest lub nie jest zapisywany do pliku wynikowego.

UWAGA: Wersja wielowątkowa czyta zatem plik źródłowy dwukrotnie - na początku i na końcu. Zyskiem jest oszczędność pamięci. Jedyny koszt to utworzenie mapy bitów dla wszystkich wierszy źródłowych. Sam odczyt pliku nie jest zaś wąskim gardłem algorytmu. Przykładowo odczyt pliku 50MB zajmuje 0.04 sek, a pliku 6GB około 5 sek.

6. DALSZY ROZWÓJ PROJEKTU

1) Lepsza kontrola błędów podczas kompilacji warunku filtrowania.

2) Przyspieszenie algorytmu wykonywania warunku filtrowania.