2020_OS_Fall_HW3: Key-Value Stroages

學號 : F14076083
姓名 : 魏湧致
系級 : 資訊 111

開發環境 :

- 使用 Virtual Machine
- OS : Ubuntu 18.04.2 LTS
- CPU : Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 3
- Memory : 3.9 GiB
- Programming Language(version) : C++ 11

程式執行時間 :

- 1. PUT 約 4.2G 的資料(Key 為 uniform distribution 的 random)時間為 :
  426.389055422 seconds time elapsed
  107.080401000 seconds user
  25.838304000 seconds sys
  2. GET 1 分鐘約能處理 80000 個指令

程式開發與使用說明：

- **你是如何開發這支程式，程式在處理資料的流程及邏輯為何 :**

  PUT : 讀入要 PUT 的 key 與 value 後根據 key 的大小放入 buffer 中，buffer 的結構為 unordered_map，buffer 共分成 100000 個，當 buffer 的 size 達到設定的大小後再將各個 key, value 寫入對應的 tmp 檔中 (buffer [0]放到 1.tmp, buffer [1]放到 2tmp …)，將 key, value 放到 unordered_map 時會將已經存在的 key 刷新，但因為將 buffer 的放入 tmp 後會將該 buffer 清空，故 tmp 中可能會有重複的 key。

  GET : 讀入要 GET 的 key，先到 PUT 產生的 buffer 中找看看有沒有所要的 key，若有則 GET 結束，沒有的話則到對應的 tmp 檔中查找，在這過程中會將 tmp 檔讀入並放到 get_page(結構一樣為 unordered_map)中，在下次 GET 時若在 PUT buffer 內沒找到且 key 與上次的 GET 在同一 tmp，則直接到 get_page 內找，在不同 tmp 時才會去 tmp 找並將 get_page 刷新。

  SCAN : 讀入要 SCAN 的 key，先到對應的 tmp 找，接著再到 PUT buffer 內找。

- **你的程式該如何使用，請詳細說明執行的步驟：**

  Compile : g++ -o key_value_stroages key_value_stroages.cpp

  Run : ./key_value_stroages [data path]

效能分析報告：

- **效能優化過程說明：**

  Ver.1：將 PUT 的 key, value 直接放到一個 tmp 檔中，每次 GET 和 SCAN 都必須將檔案從頭掃到尾來取得最新的資料，PUT 速度很快但 GET 和 SCAN 效率非常差。

  Ver.2：將 PUT 的資料先放到 buffer 內，到達設定大小再放到 tmp 檔中，GET 先到 buffer 找，SCAN 一樣先到檔案找，再到 buffer 找，PUT 速度與 Ver.1 差不多，GET 若有在 buffer 找到則效率會快上許多，SCAN 則與 Ver.1 差不多。

  Ver.3 : PUT 的 buffer 的結構由 vector 改成 unordered_map，會將已經存在的 key 的 value 刷新，GET 在 buffer 內找的速度上升。

  Ver.4：分成多個 buffer 和 tmp，buffer 和 tmp 一一對應，將 buffer 和 tmp 的 size 縮小讓 GET 與 SCAN 的效率上升許多，尤其當在 buffer 找不到而必須到 tmp 找時不用讀取很大的檔案，會省下許多時間。

  Ver.5：加入 get_page，當 GET 在 PUT 的 buffer 找不到而去 tmp 找時，一併將檔案讀到 unorder_map 中，若在 buffer 找不到且上次 GET 的 key 與這次的 key 屬於同一 tmp 就直接到 get_page 找，這樣的方法若 GET 的 key 都很相近時就不用每次到 disk 找資料，在 memory 就能快速地找到，GET 效率大幅提升。

- **請說明你所設計的整理資料作法，並分析這些設計在我們存取資料的時候，如何能提供較佳的存取效率：**

  整理資料作法：同**程式開發說明**第一點的 PUT, GET 與上一點 Ver. 5 所述

  存取資料效率：加上 buffer 與 page 的目的是為了不用每次都要到 disk 存取資料，尤其當 GET, SCAN 連續的資料，只要 access memory 就好。

- **請觀察系統效能以及 OS 是如何服務我們的程式：**

  PUT 4G 資料的 perf stat 與執行時的 top 截圖如下，執行時間大約要 430 秒，CPU 使用率為 0.310，讀指令時因為要判斷 key 是要放在哪個 buffer，故 CPU 使用率較高，約為 90%，當 buffer 滿了之後開始寫檔，

CPU 使用率降為約 15%，而 GET 的 cpu 與 memory 使用率都很低。

```
Performance counter stats for './key_value_stroages ./1.input':

      13,1993.51 msec task-clock                #    0.310 CPUs utilized
         42,1947      context-switches          #    0.003 M/sec
               0      cpu-migrations            #    0.000 K/sec
         70,4065      page-faults               #    0.005 M/sec
  <not supported>      cycles
  <not supported>      instructions
  <not supported>      branches
  <not supported>      branch-misses

    426.389055422 seconds time elapsed

    107.080401000 seconds user
     25.838304000 seconds sys
```

```
top - 18:41:10 up 7 min,  1 user,  load average: 0.97, 0.63, 0.38
Tasks: 251 total,   4 running, 200 sleeping,   0 stopped,   0 zombie
%Cpu(s): 24.7 us,  5.6 sy,  0.0 ni, 62.7 id,  3.4 wa,  0.0 hi,  3.6 si,  0.0 st
KiB Mem :  4030480 total,   110976 free,  3201024 used,   718480 buff/cache
KiB Swap:  1942896 total,  1942116 free,      780 used.   483116 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4104 root      20   0 2132220 2.023g   3372 R  90.0 52.6   0:29.96 key_value_stroa
 3335 pd2       20   0 3767412 334012 135432 S   3.7  8.3   0:18.55 gnome-shell
   95 root      20   0       0      0      0 S   3.0  0.0   0:00.56 kswapd0
```

```
top - 18:42:38 up 9 min,  1 user,  load average: 1.86, 1.02, 0.54
Tasks: 252 total,   1 running, 204 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.3 us,  1.6 sy,  0.0 ni, 58.6 id, 32.9 wa,  0.0 hi,  3.7 si,  0.0 st
KiB Mem :  4030480 total,   102416 free,  3631068 used,   296996 buff/cache
KiB Swap:  1942896 total,  1816176 free,   126720 used.   118084 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4104 root      20   0 2671916 2.535g    764 D  14.6 66.0   0:45.60 key_value_stroa
   95 root      20   0       0      0      0 S   0.7  0.0   0:01.42 kswapd0
 3335 pd2       20   0 3781036 259760  47588 S   0.7  6.4   0:19.98 gnome-shell
```

結論：這次的作業是要撰寫一隻 Key-Value Storages 程式，能夠執行 PUT [key] [value], GET [key], SCAN [key1] [key2]這三個指令，經過幾次的優化後效率比起一開始好上許多，也透過這些過程初步了解到資料庫的運作架構。