Pthread Report

Group 51

110000132 詹振暘 110021121陶威綸

詹振暘：Coding, Experiment, Report

陶威倫：Coding, Experiment

a. Implementation Explain

1. TSQueue

- TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size)

```
TSQueue<T>::TSQueue(int buffer_size) : buffer_size(buffer_size) {
    // TODO: implements TSQueue constructor
    buffer = new T[buffer_size];
    size = 0;
    head = 0;
    tail = 0;

    pthread_mutex_init(&mutex, nullptr); //mutex lock
    pthread_cond_init(&cond_enqueue, nullptr);
    pthread_cond_init(&cond_dequeue, nullptr);
}
```

此處為TSQueue的相關變數及函數進行初始化，其中buffer為一個大小為buffer_size的陣列，用於儲存型別為T的元素（T是模板型別，在後面可以拿來套用各種型別的變數）。
這邊的size, head, tail是TSQueue的private variable，在此處初始化為0，同時這邊也將mutex及cond_enqueue 條件變數、cond_dequeue條件變數初始化。cond_enqueue 是用於判斷當queue滿時，進行enqueue的thread要進行等待，直到有空間可以使用，而cond_dequeue則用於判斷當queue為空時，進行 dequeue 操作的thread需要等待，直到有新的元素被加入。

- TSQueue<T>::~TSQueue()

```
TSQueue<T>::~TSQueue() {
    // TODO: implenents TSQueue destructor
    delete[] buffer;
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond_enqueue);
    pthread_cond_destroy(&cond_dequeue);
}
```

用於刪除buffer, mutex, cond_enqueue, cond_dequeue。

- void TSQueue<T>::enqueue(T item)

```
void TSQueue<T>::enqueue(T item) {
    // TODO: enqueues an element to the end of the queue
    pthread_mutex_lock(&mutex);

    while (size == buffer_size) // busy waiting if queue is full.
        pthread_cond_wait(&cond_enqueue, &mutex);

    buffer[tail] = item;
    tail = (tail + 1) % buffer_size;
    size++;

    pthread_cond_signal(&cond_dequeue);
    pthread_mutex_unlock(&mutex);
}
```

enqueue函式負責將item加入queue尾端。首先，pthread_mutex_lock將mutex上鎖，進入critical section，接著while(size == buffer)會檢查是否在buffer是否已滿，若為滿則繼續執行，將item放入queue的尾端(tail)位置，並更新tail指標以及增加queue大小。最後，喚醒一個正在等待cond_dequeue條件變數的thread，告訴他有新的item可以取出，最後解除mutex以離開critical section。

- T TSQueue<T>::dequeue()

```
T TSQueue<T>::dequeue() {
    // TODO: dequeues the first element of the queue
    pthread_mutex_lock(&mutex);

    while (size == 0)
        pthread_cond_wait(&cond_dequeue, &mutex);

    T item = buffer[head];
    head = (head + 1) % buffer_size;
    size--;

    pthread_cond_signal(&cond_enqueue);
    pthread_mutex_unlock(&mutex);

    return item;
}
```

與enqueue相同，dequeue同樣也被mutex lock包起來以防止race condition。在critical section中，會先檢查size是否等於0，若=0則thread會先在cond_dequeue上等待新的元素被加入。接著，item會從queue的最前面取出一個元素，並更新head指標及size，最後return item。

- int TSQueue<T>::get_size()

```
int TSQueue<T>::get_size() {
    // TODO: returns the size of the queue

    pthread_mutex_lock(&mutex);
    int current_size = size;
    pthread_mutex_unlock(&mutex);

    return current_size;
}
```

get_size的目的是返回size，但保險起見仍然選擇用mutex進行上所以避免潛在可能發生的問題。

我理解的TSQueue有點類似於Producer跟Consumer中的資料傳輸橋樑，負責在multi thread之間傳遞資料，同時透過mutex_lock及condition variable來確保Race condition不會發生。其中最重要的就是enqueue跟dequeue的功能，其中enqueue控制當有producer要將資料放入queue時，如果queue已滿，producer會被暫時阻塞，直到queue有空間，而當成功放入資料後，通知consumer有新資料可以使用，而dequeue會控制當有consumer要取出資料時，如果queue是空的，consumer會被暫時阻塞，直到有新資料進入。當成功取出資料後，會通知producer有空間可使用。

2. Writer

- void Writer::start()

```
void Writer::start() {
    // TODO: starts a Writer thread
    pthread_create(&t, nullptr, Writer::process, (void *) this);
}
```

Writer::start的功能是透過pthread_create建立一個Writer::process的thread。

- void* Writer::process(void *arg)

```
void* Writer::process(void* arg) {
    // TODO: implements the Writer's work
    auto writer = (Writer*) arg;
    while (writer->expected_lines--)
        writer->ofs << *writer->output_queue->dequeue();
    return nullptr;
}
```

參考Reader::process的做法，在queue未滿之前，每次寫入都將expected_lines減減，以記錄剩餘空間，並呼叫writer->output_queue中的dequeue()來寫入資料。

3. Producer

- void Producer::start()

```
void Producer::start() {
    // TODO: starts a Producer thread
    pthread_create(&t, nullptr, Producer::process, (void*)this);
}
```

建立一個thread並指定其要做Producer::process的行為。

- void* Producer::process(void* arg)

```
void* Producer::process(void* arg) {
    // TODO: implements the Producer's work
    auto producer = (Producer*) arg;
    while (true) {
        Item *item = producer->input_queue->dequeue();
        item->val = producer->transformer->producer_transform(item->opcode, item->val);
        producer->worker_queue->enqueue(item);
    }
    return nullptr;
}
```

Producer::process會從input_queue中取出一個Item，並透過transformer中的producer_transformer將物件通過opcode跟val轉換並將結果傳回item的val，並在最後將資料放回至worker_queue()中。整體來說，Producer::process的目的就是對輸入的資料進行處理和轉換，並將轉換後的結果交由後續的consumer進行進一步的處理或輸出。

4. Consumer

- void Consumer::start()

```
void Consumer::start() {
    // TODO: starts a Consumer thread
    /*int rc = */pthread_create(&t, nullptr, Consumer::process, (void*) this);
    //if(rc != 0) perror("thread create failed.");
}
```

透過pthread_create建立一個要執行Consumer::process的thread。

- int Consumer::cancel()

```cpp
int Consumer::cancel() {
    // TODO: cancels the consumer thread

    //pthread_mutex_lock(&cancel_mutex);
    is_cancel = true;
    //pthread_mutex_unlock(&cancel_mutex);

    return pthread_cancel(t);
}
```

Consumer::cancel() 會透過將is_cancel設置為true以退出loop，接著呼叫 pthread_cancel(t)終止thread的執行。

- void* Consumer::process(void* arg)

```cpp
void* Consumer::process(void* arg) {
    Consumer* consumer = (Consumer*)arg;

    pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, nullptr);

    while (!consumer->is_cancel) {
        pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, nullptr);

        // TODO: implements the Consumer's work
        Item* item = consumer->worker_queue->dequeue();
        auto val = consumer->transformer->consumer_transform(item->opcode, item->val);
        consumer->output_queue->enqueue(new Item(item->key, val, item->opcode));
        delete item;

        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, nullptr);
    }

    delete consumer;

    return nullptr;
}
```

這部分會將worker_queue中取出一個item並進行transform，transformer是在transformer.hpp中被定義，目的是根據其opcode以及val進行producer及consumer的轉換。在轉換完成後，consumer會接著將建立一個新的item並填入轉換過後的值，然後放入output_queue，並delete item，以完成一次process。

5. Consumer_controller

- void ConsumerController::start()

```cpp
void ConsumerController::start() {
    // TODO: starts a ConsumerController thread
    pthread_create(&t, 0, ConsumerController::process, this);
}
```

透過pthread_create建立一個執行ConsumerController::process的新thread，並將當前物件 (this) 作為參數傳遞給該thread。

- void* ConsumerController::process(void* arg)

```
void* ConsumerController::process(void* arg) {
    // TODO: implements the ConsumerController's work
    ConsumerController *controller = (ConsumerController*)arg;

    while(true){
        // std::cout << "check" << "\n";
        int size = controller->worker_queue->get_size();
        if(size > controller->high_threshold){
            Consumer* consumer = new Consumer(controller->worker_queue, controller->writer_queue, controller->transformer);
            consumer->start();
            controller->consumers.push_back(consumer);
            std::cout << "Scaling up consumers from " << controller->consumers.size()-1 << " to " << controller->consumers.size() << "\n";
        } else if(size < controller->low_threshold && controller->consumers.size() > 1){
            controller->consumers.back()->cancel();
            controller->consumers.pop_back();
            std::cout << "Scaling down consumers from " << controller->consumers.size()+1 << " to " << controller->consumers.size() << "\n";
        }
        usleep(controller->check_period);
    }

    return nullptr;
}
```

ConsumerController跟Consumer的關係有點像老闆與工人的角色。Consumer會不斷進行從queue提取、轉換、和放入queue的工作，並且每個Consumer代表一個thread，而ConsumerController本身就是一個thread，負責監控worker_queue的loading，動態調整Consumer的數量，以維護效率，所以更像是一個管理者。

ConsumerController::process()具體的做法入下。在while(true)中，首先會先由size來取得此時worker_queue的大小，如果大小超過high_threshold時，會新增一個consumer，將其啟動並加入consumer queue中，而如果size低於low_threshold，且consumer數量>1時，會透過cancel的方式從queue中移除一個consumer。在每一輪while迴圈要結束時，要使用usleep()使thread短暫進入休眠狀態（usleep()的單位時間為微秒，成功返回0，失敗返回1），以確保影響效能。最後返回nullptr。

6. Main

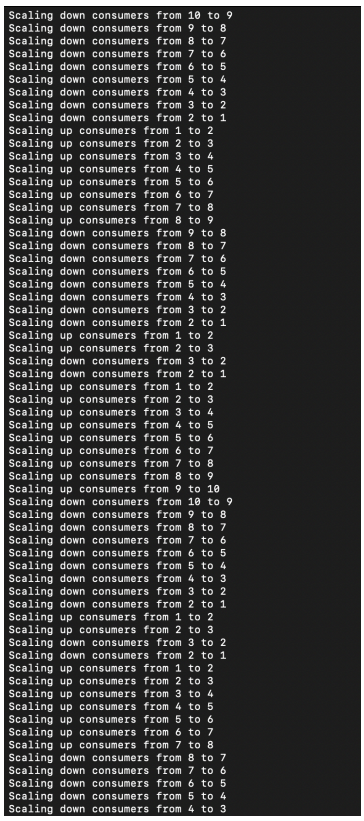
```
int main(int argc, char** argv) {
    // TODO
    assert(argc == 4);
    int total_lines = std::stoi(argv[1]);
    std::string input_file(argv[2]);
    std::string output_file(argv[3]);

    TSQueue<Item *> input_queue(READER_QUEUE_SIZE);
    TSQueue<Item *> processing_queue(WORKER_QUEUE_SIZE);
    TSQueue<Item *> output_queue(WRITER_QUEUE_SIZE);

    Reader reader_thread(total_lines, input_file, &input_queue);
    reader_thread.start();
    Writer writer_thread(total_lines, output_file, &output_queue);
    writer_thread.start();

    Transformer transformer;

    std::vector<Producer> producer_threads = {
        Producer(&input_queue, &processing_queue, &transformer),
        Producer(&input_queue, &processing_queue, &transformer),
        Producer(&input_queue, &processing_queue, &transformer),
        Producer(&input_queue, &processing_queue, &transformer)
    };
    for (auto &producer : producer_threads) {
        producer.start();
    }

    ConsumerController consumer_controller(
        &processing_queue, &output_queue, &transformer,
        CONSUMER_CONTROLLER_CHECK_PERIOD,
        WORKER_QUEUE_SIZE * CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE / 100,
        WORKER_QUEUE_SIZE * CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE / 100
    );
    consumer_controller.start();

    // Wait for Reader & Writer thread finish
    reader_thread.join();
    writer_thread.join();

    return 0;
}
```

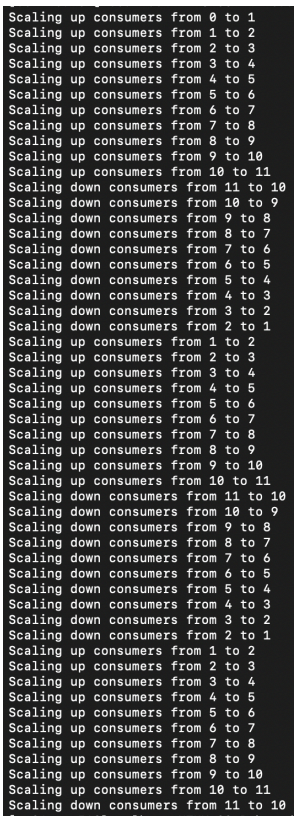
首先在 main 裡面，我建立了所有需要的物件，包括3個queue（input_queue、processing_queue、output_queue）、transformer、reader、writer、4個producer 和一個ConsumerController。這些物件在建立時分別傳入對應的參數，例如queue、檔案名稱以及threshold等，並初始化它們的功能。
接著，依序呼叫所有物件的 start()，啟動對應的thread，讓其開始運作。最後，透過呼叫reader和writer的join()，主程式會等待這兩個執行緒完成運行，確保流程結束後才結束程式。

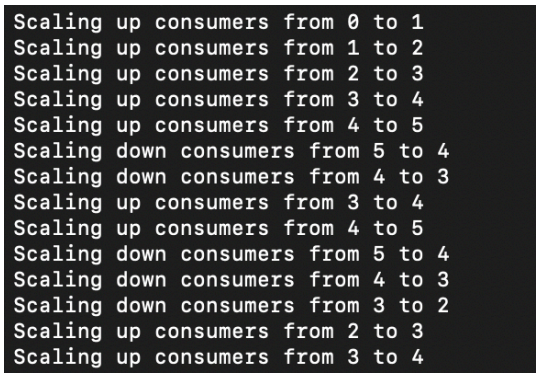
b. Experiment

1. Different CONSUMER_CONTROLLER_CHECK_PERIOD

首先我測試將CONSUMER_CONTROLLER_CHECK_PERIOD從原本的100000 增加至500000，會發現Scaling up 的次數從大約25次變為15次，而若把參數在增加至5000000，則會減少至低於十次。







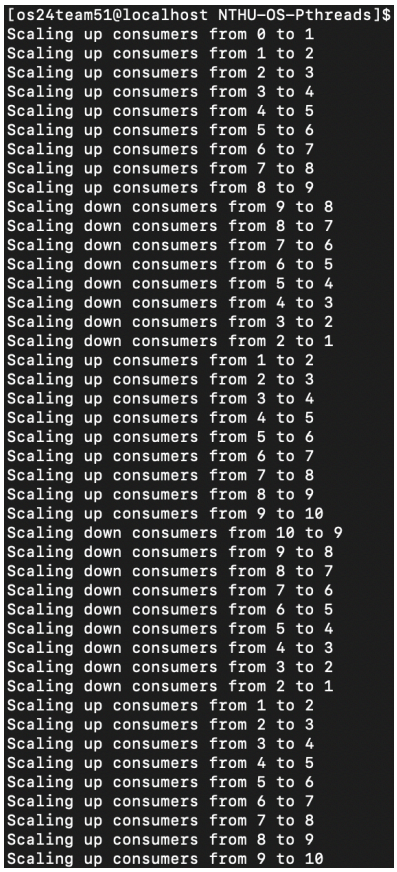
而若向下遞減到50000，Scaling up會執行將近58次（這部分輸出太長就不放ㄌ）

2. Different values of
   CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE &
   CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE

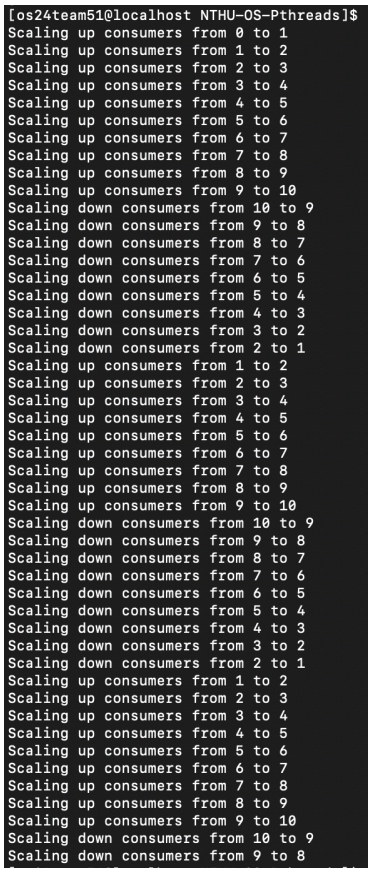
首先固定high_threshold並測試low_threshold增加及減少對Scaling up的影響。

(1) low_threshold增加：scaling down次數些微增加。
(2) low_threshold減少：對於scaling幾乎沒影響，但當更改為0時會只剩下10次scaling up。
驗證結果如下：

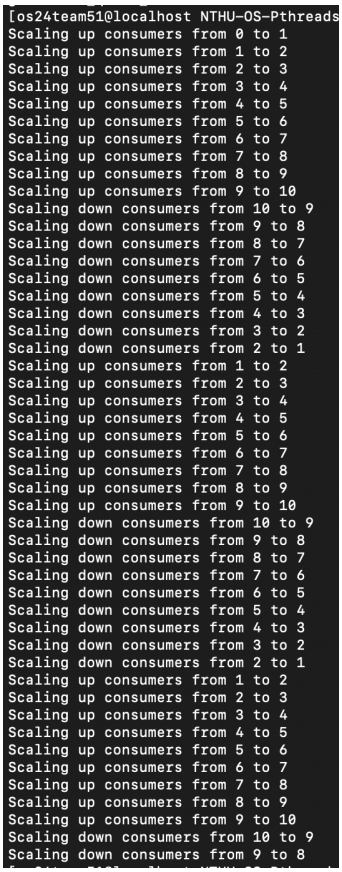
low_threshold = 90



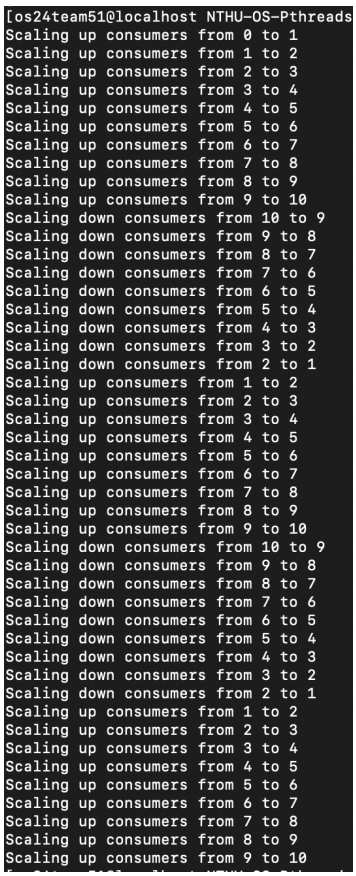
low_threshold = 150



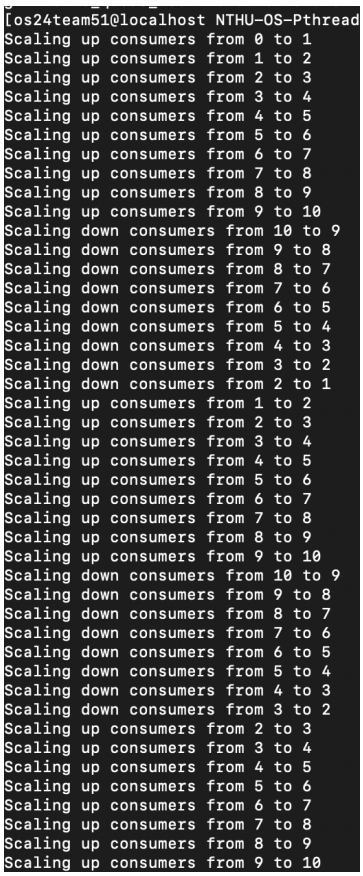
low_threshold = 400



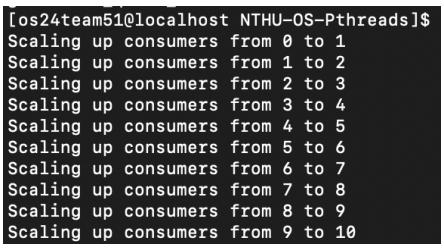
low_threshold = 10



low_threshold = 1



low_threshold = 0



接著固定low_threshold並測試high_threshold增加及減少對Scaling up的影響。

(1) high_threshold增加：在設為90後scaling up次數增加了，但當設為100時，會卡死無法執行。

(2) high_threshold減少：scaling up次數增加。

驗證結果如下：

high_threshold = 80 (original)

```
[os24team51@localhost NTHU-OS-Pthreads]
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
```

high_threshold = 90

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
```

high_threshold = 100

```
[os24team51@localhost NTHU-OS-Pthreads]$
```

high_threshold = 50

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
Scaling down consumers from 11 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
```

high_threshold = 20

```
[os24team51@localhost NTHU-OS-Pthreads
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
Scaling down consumers from 11 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
Scaling down consumers from 11 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
```

high_threshold = 0

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
Scaling down consumers from 11 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling up consumers from 3 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
Scaling down consumers from 11 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling up consumers from 10 to 11
```

3. Different values of WORKER_QUEUE_SIZE

我將SIZE大小由200遞增到400、600、800，Scaling up次數基本沒有太大變化，而若遞減到50, 30, 10左右Scaling的次數會增加許多。

WORKER_QUEUE_SIZE = 600

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
```

WORKER_QUEUE_SIZE = 800

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling down consumers from 6 to 5
```

WORKER_QUEUE_SIZE = 10

```
[os24team51@localhost NTHU-OS-Pthreads
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
```

4. What happens if WRITER_QUEUE_SIZE is very small?

當WRITER_QUEUE_SIZE極小時，如20、10、1，scaling的次數都有略微降低，但沒有到非常明顯。

WRITER_QUEUE_SIZE = 20

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
```

WRITER_QUEUE_SIZE = 10

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
```

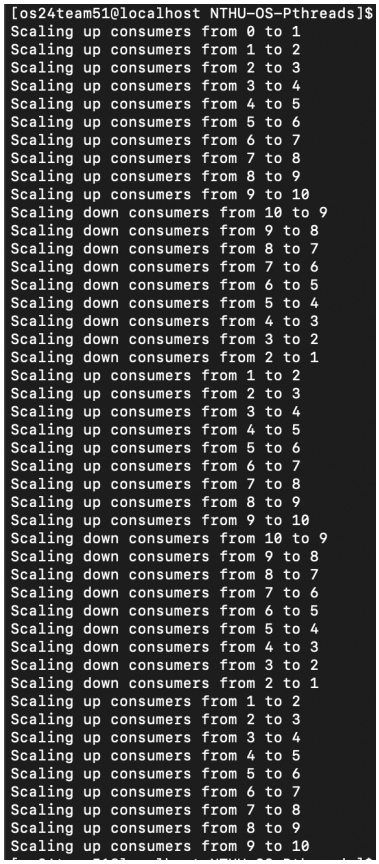
WRITER_QUEUE_SIZE = 1

```
[os24team51@localhost NTHU-OS-Pthreads]$
Scaling up consumers from 0 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
Scaling down consumers from 10 to 9
Scaling down consumers from 9 to 8
Scaling down consumers from 8 to 7
Scaling down consumers from 7 to 6
Scaling down consumers from 6 to 5
Scaling down consumers from 5 to 4
Scaling down consumers from 4 to 3
Scaling down consumers from 3 to 2
Scaling down consumers from 2 to 1
Scaling up consumers from 1 to 2
Scaling up consumers from 2 to 3
Scaling up consumers from 3 to 4
Scaling up consumers from 4 to 5
Scaling up consumers from 5 to 6
Scaling up consumers from 6 to 7
Scaling up consumers from 7 to 8
Scaling up consumers from 8 to 9
Scaling up consumers from 9 to 10
```

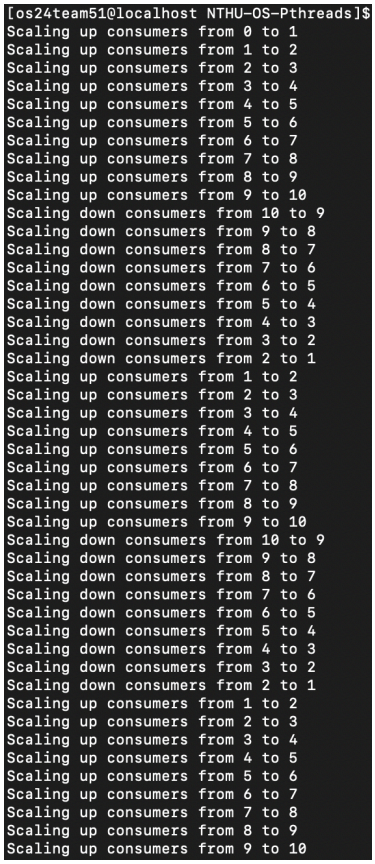
5. What happens if READER_QUEUE_SIZE is very small?

跟WRITER的情況一樣，在將 READER_QUEUE_SIZE遞減至非常小後scaling的次數並沒有明顯的變化，因此將READER_QUEUE_SIZE設為極小似乎對scaling的次數也沒有什麼影響。

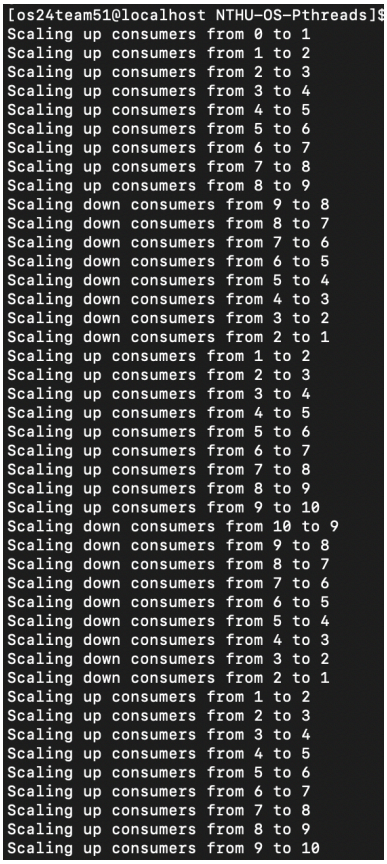
WRITER_QUEUE_SIZE = 20



WRITER_QUEUE_SIZE = 10



WRITER_QUEUE_SIZE = 1



C. Difficulties Encountered

這次作業由於有提供 TODO 的hint，因此整個過程相對清晰許多，但仍然在實作過程中碰見不少困難。尤其是 ConsumerController得部分，因為要動態調整消費者數量且要處理thread synchronization的問題。我在一開始嘗試使用pthread_mutex和條件變數來管理Consumer的增加與減少，但中間經歷了限制方向錯誤及沒有設對condition variable的問題，導致consumer的部分一直有誤，後來經過好幾次測試和調整才終於正確。在設計Writer queue的時候我也因為略了處理剩餘資料的情況卡了很久，後來突然想到並更正才通過test。