Customised Scheduler

GROUP 11 (UG)

A1809860 - LE, DUC GIA HUY
A1805637 - YUANXI, WANG
A1805276 - DANLI, MILLICENT JESSLYN

LINK

Introduction

The Setup

4 MAIN TECHNIQUES:

- Priority Queues
- Shortest Remaining Time
- Dynamic Round Robin
- First Come First Serve

PRIORITY QUEUE - SHORTEST REMAINING TIME

```
286                 // high priority customer will have 3 turn to access the game immediately
287                 if (turn < 3) {
288                     if (!queue_high_0.empty()) // is anyone waiting?
289                     {
290                         std::sort(queue_high_0.begin(), queue_high_0.end());
291                         current_id = queue_high_0[0].second;
292                         // int slotsremaining = queue_high_0[0].first;
293                         queue_high_0.pop_front();
294                         if (TIME_ALLOWANCE > customers[current_id].slots_remaining)
295                         {
296                             time_out = current_time + customers[current_id].slots_remaining;
297                         }
298                         else
299                         {
300                             time_out = current_time + TIME_ALLOWANCE;
301                         }
302                         customers[current_id].playing_since = current_time;
303                     } else if (!queue_low_1.empty())
304                     {
305                         std::sort(queue_low_1.begin(), queue_low_1.end());
306                         current_id = queue_low_1[0].second;
307                         queue_low_1.pop_front();
308                         if (TIME_ALLOWANCE > customers[current_id].slots_remaining)
309                         {
310                             time_out = current_time + customers[current_id].slots_remaining;
311                         }
312                         else
313                         {
314                             time_out = current_time + TIME_ALLOWANCE;
315                         }
316                         customers[current_id].playing_since = current_time;
317                     }
318                 }
```

```
319             else if (turn > 2) {
320                 if (!queue_low_1.empty()) // is anyone waiting?
321                 {
322                     std::sort(queue_low_1.begin(), queue_low_1.end());
323                     current_id = queue_low_1[0].second;
324                     queue_low_1.pop_front();
325                     if (TIME_ALLOWANCE > customers[current_id].slots_remaining)
326                     {
327                         time_out = current_time + customers[current_id].slots_remaining;
328                     }
329                     else
330                     {
331                         time_out = current_time + TIME_ALLOWANCE;
332                     }
333                     customers[current_id].playing_since = current_time;
334                 } else if (!queue_high_0.empty())
335                 {
336                     std::sort(queue_high_0.begin(), queue_high_0.end());
337                     current_id = queue_high_0[0].second;
338                     queue_high_0.pop_front();
339                     if (TIME_ALLOWANCE > customers[current_id].slots_remaining)
340                     {
341                         time_out = current_time + customers[current_id].slots_remaining;
342                     }
343                     else
344                     {
345                         time_out = current_time + TIME_ALLOWANCE;
346                     }
347                     customers[current_id].playing_since = current_time;
348                 }
349             }
350             turn++;
351             turn = turn % 5;
352         }
```

PRIORITY QUEUE - SHORTEST REMAINING TIME

CONFIGURATION OF THE QUEUES

```
216         // welcome newly arrived customers
217         while (!arrival_events.empty() && (current_time == arrival_events[0].event_time))
218         {
219             // initialqueue.push_back( std::make_pair(customers[arrival_events[0].customer_id].slots_remaining,arrival_events[0].customer_id) );
220             if (customers[arrival_events[0].customer_id].priority == 0){
221                 queue_high_0.push_back( std::make_pair(customers[arrival_events[0].customer_id].slots_remaining,arrival_events[0].customer_id) );
222             }
223             else if (customers[arrival_events[0].customer_id].priority == 1){
224                 queue_low_1.push_back( std::make_pair(customers[arrival_events[0].customer_id].slots_remaining,arrival_events[0].customer_id) );
225             }
226             arrival_events.pop_front();
227         }
```

ROUND ROBIN

```
250         // if machine is empty, schedule a new customer
251         if (current_id == -1)
252         {
253             TIME_ALLOWANCE = 4;
254             bool timesplayed0 = has0timesplayed(queue_high_0,customers);
255             bool timesplayed1 = has0timesplayed(queue_low_1,customers);
```

DYNAMIC ROUND ROBIN

International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-3, Issue-5, May 2017
ISSN: 2395-3470
www.ijseas.com

BEST TIME QUANTUM ROUND ROBIN CPU SCHEDULING ALGORITHM

Dolly Khokhar , Mr Ankur Kaushik

Meerut Institute Of Engineering And Technology, Meerut, India

Step 5 - Calculate the best time quantum

B.T.Q= [mean + median]/2

```
286                 else {
287                     TIME_ALLOWANCE = (med+mean)/2;
```

Reference: Khokhar & Kaushik, 2017, 'BEST TIME QUANTUM ROUND ROBIN CPU SCHEDULING ALGORITHM', International Journal of Scientific Engineering and Applied Science (IJSEAS), vol. 3, (5)

FIND MEAN

```cpp
// Function for calculating mean
double findMean(std::vector<int> a, int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    return (double)sum / (double)n;
}
```

FIND MEDIAN

```cpp
// https://www.geeksforgeeks.org/finding-median-of-unsorted-array-in-linear-time-using-c-stl/
double findMedian(std::vector<int> a, int n)
{
    // If size of the arr[] is even
    if (n % 2 == 0) {

        // Applying nth_element on n/2th index
        nth_element(a.begin(),
                    a.begin() + n / 2,
                    a.end());

        // Applying nth_element on (n-1)/2 th index
        nth_element(a.begin(),
                    a.begin() + (n - 1) / 2,
                    a.end());

        // Find the average of value at index N/2 and (N-1)/2
        return (double)(a[(n - 1) / 2]
                        + a[n / 2])
               / 2.0;
    }

    // If size of the arr[] is odd
    else {
        // Applying nth_element on n/2
        nth_element(a.begin(),
                    a.begin() + n / 2,
                    a.end());

        // Value at index (N/2)th is the median
        return (double)a[n / 2];
    }
}
```

FIRST COME FIRST SERVE

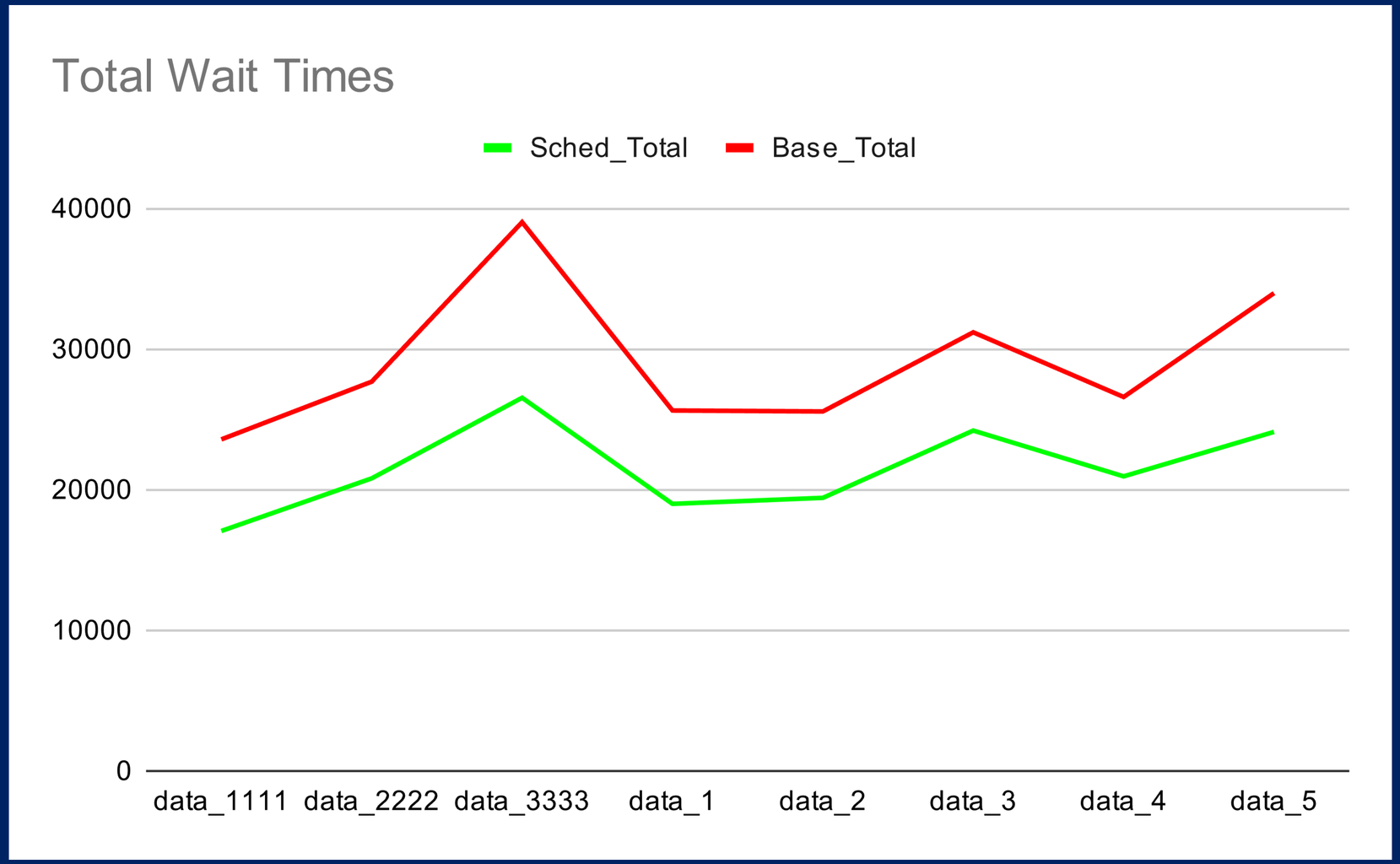
```
bool has0timesplayed( std::deque< std::pair<int,int> >queue, std::vector<Customer> cus){
    for (int i = 0; i < queue.size(); i++){
        if (cus[queue[i].second].timesplayed == 0){
            return true;
        }
    }
    return false;
}
```

RESULTS

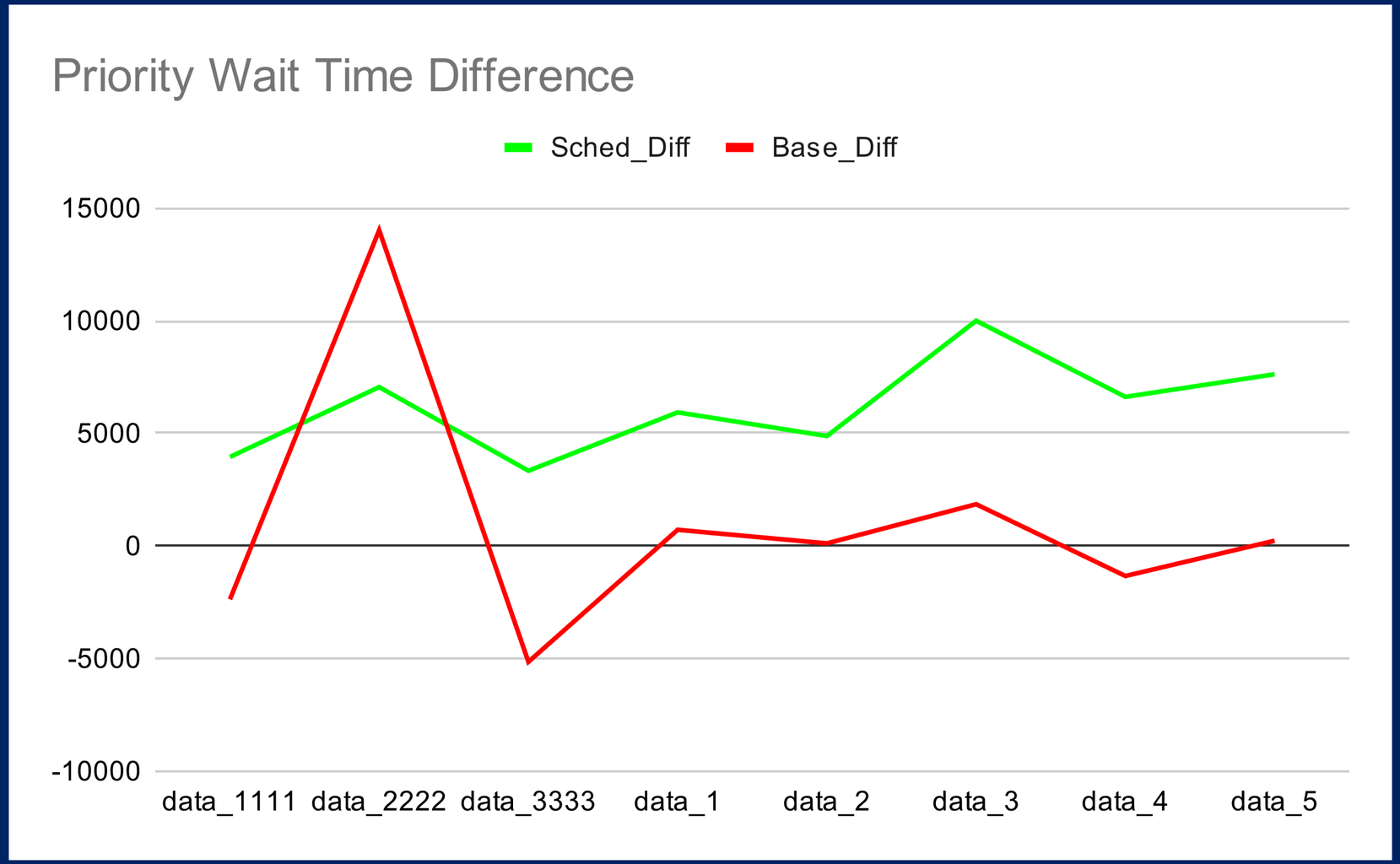
```
Scheduler Stats data_1111.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
6554 10511 17065 233 124
Baseline Stats data_1111.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
12972 10606 23578 266 184
==============================
Scheduler Stats data_2222.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
6874 13933 20807 232 137
Baseline Stats data_2222.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
13661 14023 27684 292 201
==============================
Scheduler Stats data_3333.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
11597 14942 26539 270 150
Baseline Stats data_3333.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
22084 16943 39027 354 225
==============================
Scheduler Stats data_1.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
6528 12465 18993 236 129
Baseline Stats data_1.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
12455 13178 25633 296 186
```

```
Scheduler Stats data_2.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
7267 12153 19420 212 137
Baseline Stats data_2.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
12722 12842 25564 293 184
==============================
Scheduler Stats data_3.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
7100 17105 24205 214 133
Baseline Stats data_3.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
14668 16526 31194 316 205
==============================
Scheduler Stats data_4.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
7162 13787 20949 210 136
Baseline Stats data_4.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
13961 12632 26593 286 194
==============================
Scheduler Stats data_5.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
8241 15870 24111 256 137
Baseline Stats data_5.txt
total_wait_0 total_wait_1 total_wait longest_response n_switches
16870 17109 33979 309 213
```

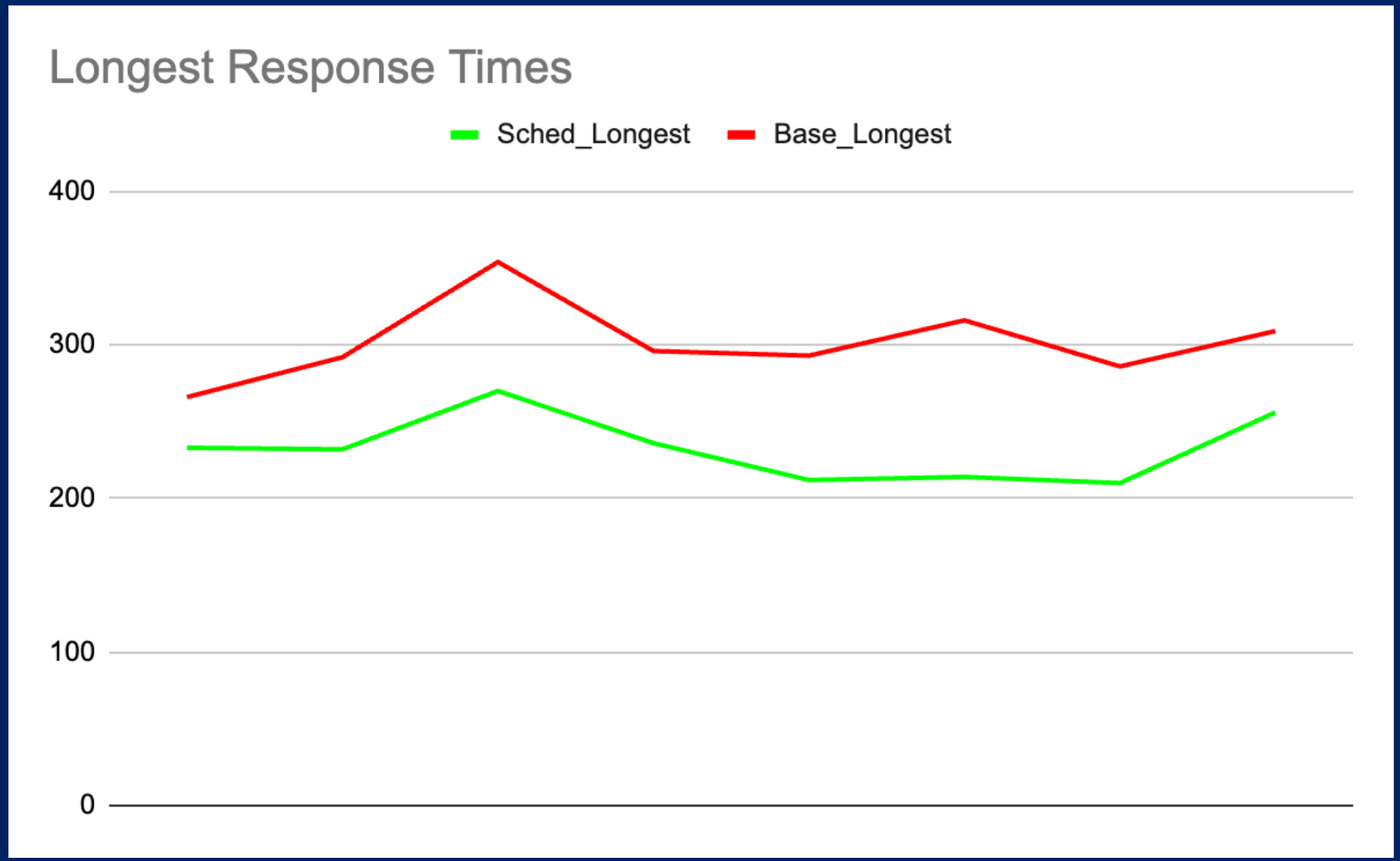
Results


Total Wait Times
Sched_Total
Base_Total
40000
30000
20000
10000
0
data_1111
data_2222
data_3333
data_1
data_2
data_3
data_4
data_5

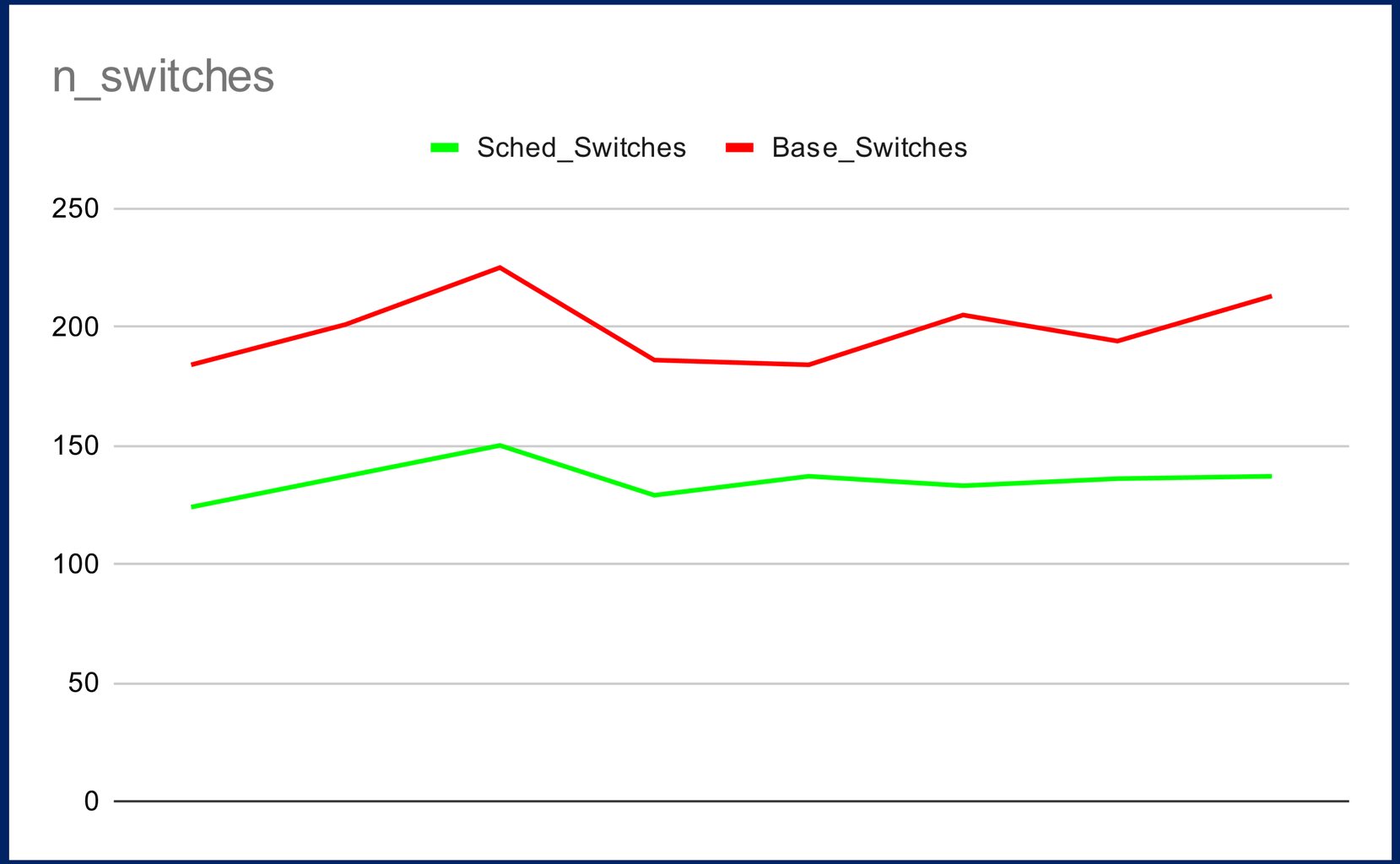
Results


Priority Wait Time Difference
Sched_Diff
Base_Diff
15000
10000
5000
0
-5000
-10000
data_1111
data_2222
data_3333
data_1
data_2
data_3
data_4
data_5

Results


Longest Response Times
Sched_Longest
Base_Longest
400
300
200
100
0

Results



CONCLUSION