# The Server's Best Response in M/M/s queueing systems

Daihui Lu

## Abstract

Although there have been a lot of literature trying to maximum the employers' profit or customers' satisfaction, only few literature works on behalf of the workers' benefits. In this project, we built a M/M/s system to study the best response of a server to maximize the server's profit. We assume that the queue is visible to the server and thus the server knows the queue's length and adjusts the work effort (high/low) accordingly. High/low effort takes more/less cost to the server. The server's payment is calculated by the number of customers that are processed in the system.

The server's net profit is analyzed in both a single queue system and a double queues system. In a single queue system, the simulation results turn out that the server's optimal choice is to choose low effort to process the customers when the queue's length is shorter than 5, otherwise use high effort. In a double queues system, the two servers have 6 possible strategies. By comparing the results of 6 strategies, we found that although the lower effort server always gets more profit than the other server in one simulation, the server's best response from the perspective of his/her most benefit, is to use varying effort. The server can only get most profit using low effort if the other server uses varying effort. Otherwise, varying effort always gives the server most benefit no matter which effort the other server chooses.

## 1 Introduction

There have been a lot of research on maximizing the workers' productivity or customers' satisfaction in service industry. For example, the workers' productivity could be affected by the queue's visibility. The feedback of the workload might be limited if the length of the queue is invisible or partially visible. Locke and Latham (1991) [1] claimed that the feedback has no motivational effect on performance. However, a number of empirical literature show that workers can strategically adjust their service rate in response to the workload. For example, Schultz et al. (1998) [2] conducted lab experiment and found that the feedback quality influences workers' motivation. Later, Kuntz et al. (2011) [3] found that if the workload increases from a low level, service outcomes are improved. Moreover, Shunko et al (2018) [4] provided the experimental evidence that poor visibility of the queue length slows down the server.

Despite the numerous research on maximizing the workers' productivity, only few literature working on the servers' benefit has been found. In this project, we will focus on maximizing the server's profit under the condition that the queue's length is visible, in which way the server could adjust the service rate as the queue's length changes. Then we will find the optimal response strategy for the server.

This paper is organized as follows: in Section 2, we will describe the problem statement in details; in Section 3, we will elaborate the algorithms implemented in the simulation; in Section 4, we will visualize the data and present the simulation results; finally, we will discuss the limitation and future work of this project in Section 5.

# 2 Problem Statement

We consider two common situations:

1. There is only a single queue with one server. The customers arrive at a Poisson rate $\lambda$. The first come, the first served.
2. There are two queues and two servers. The customers arrive at a Poisson rate $\lambda$. The new customer joins the shorter queue.

The server's total profit $P$ is calculated by payment minus the server's cost. The server could choose to use high effort (i.e., high service rate) or low effort (i.e., low service rate) when serving the customers. High effort takes shorter time while needs more cost; low effort takes longer time while needs less cost. Let $C_h$ ($C_l$) denote the cost of high (low) effort per customer. The server gets payment $p$ per customer. We assume that $N$ customers are processed in a certain period. Since the server could change service rate if the server sees the queue, we assume that the server uses high effort if the queue's length is greater than some number $n$. Once there are $n$ or less customers in the queue, the server uses low effort. Meanwhile, the customers behavior is also affected by the queue's length. If the queue's length reaches a limit, $l$, the new arriving customer will leave immediately.

The question is, how do we choose $n$ so that the server gets the most profit? We will analyze the net profit in each situation.

## 2.1 Single queue, single server

The single queue is a simple problem. The net profit the server gets is

$$P = pN - C_h N_h - C_l N_l,$$

where $N$ is the total number of customers, $N_h$ ($N_l$) is the number of customers served with high (low) effort. Note that $p > C_h > C_l$, thus the server is temped to use low effort since it reduces the cost. However, low effort might lead to a long queue, which results in the loss of customers. Therefore, the maximum profit the server could get is a trade off between the lower cost and more customers.

## 2.2 Two queues, two servers

In the situation of two queues, the two servers make a team. The net profit server $i$ gets is

$$P(i) = pN - C_h N_h(i) - C_l N_l(i),$$

where $N_h(i)$ is the number of customers served by server $i$ with high effort; $N_l(i)$ is the number of customers served by server $i$ with low effort; $N$ is the number of customers processed by the team, i.e., $N = \sum_i N(i)$. In this situation, the interaction between the two customers make the profit calculation more complicated. Both customers could choose to use high effort or low effort all the time, or they could vary the effort as the queue's length changes.

The situation may seem similar to prisoner's dilemma, but not exactly the same. We will show this in Sec. 4 with results.

# 3 Algorithms Implementation

We construct an M/M/s FIFO queue means a queue in a system with s server(s), where the arrivals are determined by a Poisson process and job service times have an exponential distribution.

To construct a basic M/M/s system, we need:

- CUSTOMER: a structure to store customer information: timestamp that indicates when the customer entries the system, the service time the customer gets, and customer ID;
- SERVER: a list of server id;
- CALENDAR: a heap, to store the sequence of event time and event type;
- QUEUE: queue(s) to store the sequence of customers waiting for being served;
- ADD_EVENT: A function to add an event (customer's arrival or departure) to the system;
- GET_EVENT: a function to get an event from the calendar;
- PROCESSOR: a function to process the event as scheduled by the calendar;

and some variables to store the information we want to track:

- current_Time: update the current time when an event is processed;
- Event type: ARRIVAL or DEPARTURE;
- Server's state: IDLE or BUSY.

Besides the basic functions in an M/M/s system, this project needs more:

- customer_Low: the number of customers processed by low effort;
- customer_High: the number of customers processed by high effort;
- queue_limit: once a queue reaches it's limit, the new arrivals will be ignored (the new customer directly leaves);
- bar: if the queue's length is smaller than the bar, the server works with low effort, otherwise high effort.

## 3.1 Algorithm for a single queue system

In a single queue, the system works as follows:

- Initially, current_Time is 0, $N_h = N_l = N = 0$. Queue is empty and server is IDLE. An arrival event is added to the calendar with its timestamp (0) and event type (ARRIVAL) by ADD_EVENT.
- Pop up the most recent event in the calendar by GET_EVENT, and process the event according to its type. Store the current time to CLOCK. The first event is the ARRIVAL of the first customer. Since now the server in still IDLE, we turn it to BUSY and add the DEPARTURE of this customer at time $t$ = current_Time + Exponential(service rate) to the calendar. (Note that Exponential(service rate) is the service time this customer will have if the server uses high effort. If the server decides to use low effort, which will be discussion soon, the service time doubles Exponential(service rate).) Next we add the ARRIVAL of next customer at $t$ = current_Time + Exponential(arrival rate). Thus the ARRIVAL and DEPARTURE flow is completed.
- For the following events, if the event type is ARRIVAL, we first check the state of the server. If server is IDLE, we turn it to BUSY and add the DEPARTURE of this customer to the calendar. Next we add the ARRIVAL of next customer at $t$ = current_Time + Exponential(service rate). If the server is BUSY, we need to check the current queue's length. If the length reaches the limit $l$, the customer leaves without further processing; if the length is below the limit, the new customer will be put at the end of the queue and wait for the service. Next we add the next ARRIVAL into the calendar.

- If the event type is DEPARTURE, we'll first check if there's still customers waiting in QUEUE. If no, change the server's state to IDLE; if yes, then we will arrange the departure of the first customer in QUEUE. Before that, we need to check the current queue's length: if the queue's length is shorter than server's bar, the server will use low effort (double the service time), and $N_l$ will add one; if not, the server will process this customer with high effort, and $N_h$ will add one; Next, we add the next departure event to the CALENDAR at $t$ = current_Time + Exponential(service rate).
- The above steps repeat until current_Time reaches the simulation length we set.

To find the optimal strategy for the server, we run simulations with multiple $n$. When the queue's length is greater than $n$, the server uses high effort, otherwise low effort. For each $n$, we run 30 simulations and take the average $N_h$ and $N_l$.

### 3.2 Algorithm for double queues system

In a double queues system, the system works similarly to the single queue system, while differs in a few places.

- Initially, current_Time is 0, $N_h(i) = N_l(i) = N = 0$ for $i = 0, 1$. Both queues is empty and servers are IDLE. An arrival event is added to the calendar with its timestamp (0) and event type (ARRIVAL) by ADD_EVENT.
- Pop up the most recent event in the calendar by GET_EVENT, and process the event according to its type. Store the current time to CLOCK. The first event is the ARRIVAL of the first customer. Every time a new customer arrives, the customer chooses to join the shorter queue. If the queues' lengths are equal, then the customer has equal possibility to choose queue 0 or 1.

  Since at $t = 0$, both queues are empty, the customer randomly chooses one queue. Then we turn the corresponding server to BUSY and add the DEPARTURE of this customer at time $t$ = current_Time + Exponential(service rate) to the calendar (again, Exponential(service rate) is the service time this customer will have if the server uses high effort. If the server decides to use low effort, which will be discussion soon, the service time doubles Exponential(service rate).) Next we add the ARRIVAL of next customer at $t$ = current_Time + Exponential(arrival rate). Thus the ARRIVAL and DEPARTURE flow is completed.
- For the following events, if the event type is ARRIVAL, we first check the state of the server. If server is IDLE, we turn it to BUSY and add the DEPARTURE of this customer to the calendar. Next we add the ARRIVAL of next customer at $t$ = current_Time + Exponential(service rate). If the server is BUSY, we need to check the current queue's length. If both queues' length reaches the limit $l$, the customer leaves without further processing; if either queue's length is below the limit, the new customer will be put at the end of this queue and wait for the service. Next we add the next ARRIVAL into the calendar.
- If the event type is DEPARTURE, we'll first check which server is responsible for the current DEPARTURE. If server $i$ is responsible, we then check queue $i$'s length. If queue $i$'s length is 0, change the server $i$'s state to IDLE; if not, we will arrange the departure of the first customer in queue $i$. The service time this customer get will be determined by queue $i$'s length again: if the length is shorter than server $i$'s bar, server $i$ will use low effort (double the service time), and $N_l(i)$ will add one; if not, server $i$ will process this customer with high effort, and $N_h(i)$ will add one. Next, we add the next departure at queue $i$ event to the CALENDAR at $t$ = current_Time + Exponential(service rate).
- The above steps repeat until current_Time reaches the simulation length we set.

Similarly, we run simulations with multiple $n$. When the queue's length is greater than $n$, the server uses high effort, otherwise low effort. Moreover, we would like to know the servers' profit

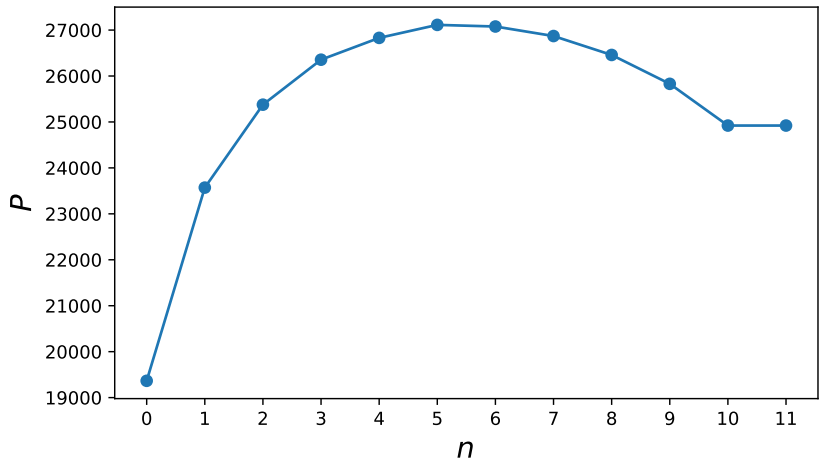


Figure 1: Net profit of the server in a single queue system

when they choose different strategies: both use low effort all the time, both use high effort all the time, one high effort one low effort all the time, and both servers use same $n$.

# 4 Results and Discussions

## 4.1 Single queue system

Let the arrival rate $\lambda = 1$ and service rate $\mu = 0.8$. The queue's limit is 10. We assume that the payment per customer $p$ is 10 USD, the cost of high effort $C_h$ is 5 USD, the cost of low effort $C_l$ is 2 USD.

To find the optimal threshold of high/low effort $n$, we run simulations with multiple $n$ from 0 to 11. $n = 0$ means that the server uses high effort all the time, and $n = 11$ means that the server uses low effort all the time since the queue's length never reaches 11.

The server's net profit is calculated by the formula in Sec. **??**. Results are summarized in Table 5 and plotted in Fig. 1. It is found that high-effort-all-the-time is the worst choice for the server – about 8000 USD loss compared to the maximum profit. The maximum profit is 27112.8 USD when $n = 5$. Therefore, we can conclude that the optimal strategy for the server in such a single queue system is to use low effort when the queue's length is shorter than 5, while high effort otherwise.

## 4.2 Double queues system

The service rate is still $\mu = 0.8$. The queue's limit is 10. Again, the payment per customer $p$ is 10 USD, the cost of high effort $C_h$ is 5 USD, the cost of low effort $C_l$ is 2 USD. One small difference from the single queue system is that, since there are two servers now, the system are able handle more customers, and thus the arrival rate is $\lambda = 0.5$.

In the double queues system, we need to consider the interaction between the two servers. We will compare the servers' net profit when they choose multiple strategies:

1. both servers uses high effort all the time;
2. both servers uses low effort all the time;
3. both servers vary their effort as the queue's length changes. $n$ is same for both server;

4. one server varies the effort as the queue's length changes, while the other always uses high effort;

5. one server varies the effort as the queue's length changes, while the other always uses low effort;

6. one server uses high effort and the other uses low effort all the time;

Strategy 1 and 2 are irrelevant to $n$. The simulation results turn out that both servers get similar net profit if either of the two strategies applies. If both servers use high effort (Strategy 1), both will get approximately 52050 USD; if both servers use low effort (Strategy 2), both will get approximately 56280 USD. This seems counter intuitive: one works with high effort, but gets less payoff. This is due to the fact that higher effort costs more. Both servers work with high efficiency, and quickly process all the customers in the queues at a higher cost. So the servers are idle for some period, during which they are not paid because their payment is determined by the number of processed customers.

Next, let's look at the effect of Strategy 3: both servers use low effort when the queue's length is shorter than $n$, and turns to high effort otherwise. The results are shown in Fig. 2. It is found that the two servers' net profits have similar trend. Theoretically, they should be exactly the same. However, due to the bias of random number generator, Server 1 gets slightly more profit than Server 0. The numbers in the plot are summarized in Table 5 in Appendix.

Clearly, the optimal choice if both servers vary effort is that $n = 4$: choose low effort when the queue's length is shorter than 4.

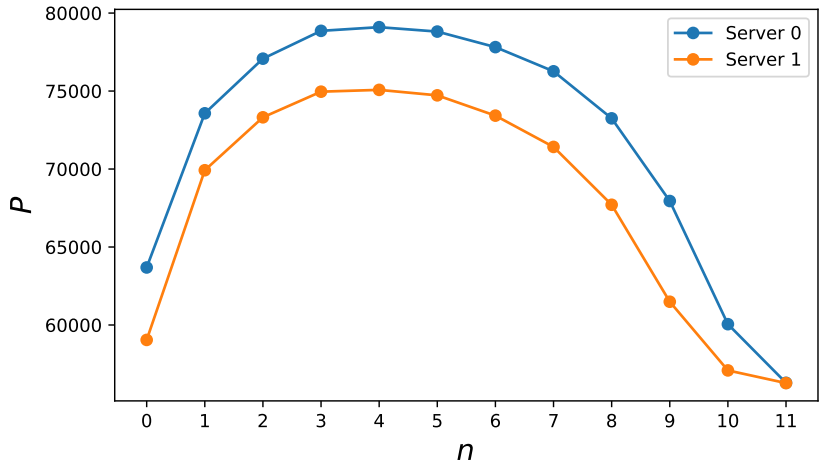


Figure 2: Net profit of the server in a double queues system: both servers vary effort as the queue's length changes

The results for Strategy 4 and 5 are shown in Fig. 3 and 4, respectively. In these two examples, Server 0 varies the effort while Server 1 sticks to high effort (Fig. 3) or low effort (Fig. 4). The results are interesting: Strategy 3 does not favor Server 1, who sticks to high effort all the time. As $n$ increases, Server 0 who uses low effort more frequently, gets more profit (over 8000 at $n \geq 5$), while Server 0 increases at the beginning and then starts decreasing at $n = 5$. The maximum profit Server 1 gets is around 65000, much less than Server 0. Strategy 4 is more beneficial to Server 1, who uses low effort all the time. The trend is opposite to Strategy 3: as $n$ increases, i.e., Server 0 uses low effort more frequently, Server 1 gets less profit. Strategy 3 and 4 shows that in a team, the "free-rider" (whoever uses low effort more frequently) gets more profit.

In Strategy 6, one uses high effort and other uses low effort. The high effort server gets 59080 USD and low effort server gets 81964.8 USD. Again, the free-rider gets more.

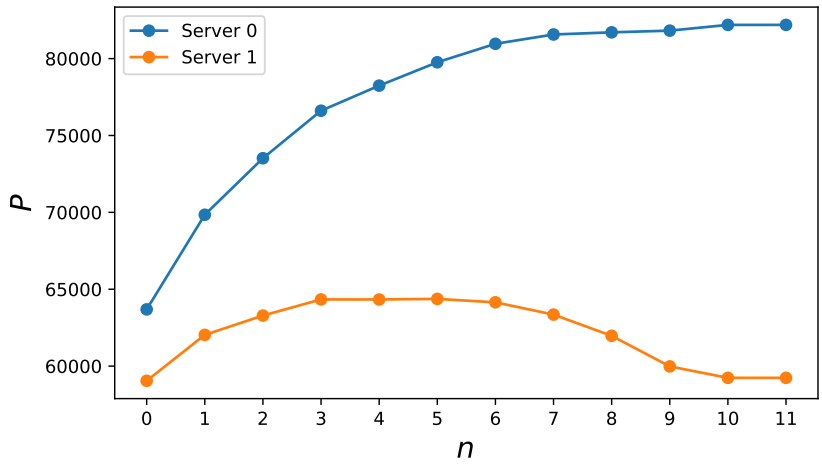


Figure 3: Net profit of the server in a double queues system: Server 0 adjusts effort as the queue's length changes, Server 1 uses high effort all the time

In the analysis of each strategy, the low effort server always gets more profit than the high effort server. It sounds like a classical prisoner's dilemma, but is it? We summarize the maximum profit that each server could get in every strategy in Table 1. From the perspective of an individual server, if the other server always takes low effort, this server's best response should be "varying" (changing effort at threshold $n$), since $61612 > 59080 > 56275$ (strategy 4, 2, 1, respectively); if the other server always takes high effort, this server's best response is again "varying" (changing effort at threshold $n$), since $82192 > 81965 > 52088$ (strategy 5, 2, 3, respectively); if the other server chooses to vary effort as queue's length changes, this server's best response is using low effort all the time and second best is " varying", since $83507 > 75076 > 64336$ (strategy 4, 6, 5, respectively). Therefore, Strategy 6, i.e., both servers vary effort during the service, is generally the best response for an individual server. Low effort is only the optimal choice when the other server uses varying effort, even though in a single round, the server who uses lower effort always gets more profit than the other server.

| Strategy | server 0 | server 1 | P(0) | P(1) |
|---|---|---|---|---|
| 1 | low | low | 56292 | 56275 |
| 2 | low | high | 81965 | 59080 |
| 3 | high | high | 52049 | 52088 |
| 4 | varying | low | 61612 | 83507 |
| 5 | varying | high | 82192 | 64336 |
| 6 | varying | varying | 77816 | 75076 |

Table 1: The maximum profit each server could get in each strategy

# 5 Conclusion

In this project, we built M/M/s systems to study the best response for the server to maximize the server's profit. The server's profit is analyzed in both a single queue system and a double queues system.

In a single queue system, the simulation results turn out that the server's optimal choice is to choose low effort to process the customers when the queue's length is shorter than 5, otherwise use high effort.

In a double queues system, the two servers have 6 possible strategies. By comparing the results

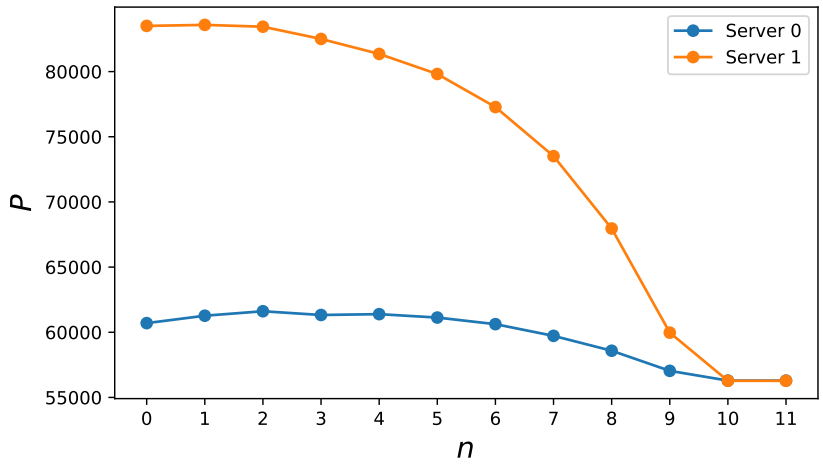


Figure 4: Net profit of the server in a double queues system: Server 0 adjusts effort as the queue's length changes, Server 1 uses low effort all the time

of 6 strategies, we found that although the lower effort server always gets more profit than the other server in one simulation, the server's best response from the perspective of his/her most benefit, the server should use varying effort. The server can only get most profit using low effort if the other server uses varying effort. Otherwise, varying effort always gives the server most benefit no matter which effort the other server chooses.

## Acknowledgement

Daihui Lu would like to thank Chen Wei for his insightful discussions on this project.

## References

[1] G. P. Latham, E. A. Locke, Self-regulation through goal setting, Organizational Behavior and Human Decision Processes 50 (2) (1991) 212–247. doi:10.1016/0749-5978(91)90021-K.

[2] K. L. Schultz, D. C. Juran, J. W. Boudreau, J. O. McClain, L. J. Thomas, Modeling and Worker Motivation in JIT Production Systems, Management Science 44 (12-part-1) (1998) 1595–1607. doi:10.1287/mnsc.44.12.1595.

[3] L. Kuntz, R. Mennicken, S. Scholtes, Stress on the Ward: Evidence of Safety Tipping Points in Hospitals, Management Science 61 (4) (2015) 754–771. doi:10.1287/mnsc.2014.1917.

[4] M. Shunko, J. Niederhoff, Y. Rosokha, Humans Are Not Machines: The Behavioral Impact of Queueing Design on Service Time, Management Science 64 (1) (2018) 453–473. doi:10.1287/mnsc.2016.2610.

# Appendix

| n | Net profit |
|---|---|
| 0 | 19365.5, |
| 1 | 23570.8, |
| 2 | 25376.3, |
| 3 | 26355.2, |
| 4 | 26829.9, |
| 5 | 27112.8, |
| 6 | 27077.5, |
| 7 | 26870.1, |
| 8 | 26458.3, |
| 9 | 25829.6, |
| 10 | 24921.6 |
| 11 | 24921.6 |

Table 2: Net profit of the server under multiple $n$ values in a single queue system

| n | Server 0 | Server 1 |
|---|---|---|
| 0 | 63688.266666666656, | 59043.33333333333, |
| 1 | 73572.83333333331, | 69921.96666666665, |
| 2 | 77076.66666666669, | 73315.93333333333, |
| 3 | 78860.23333333334, | 74959.96666666666, |
| 4 | 79095.79999999999, | 75075.59999999999, |
| 5 | 78813.2, | 74724.79999999999, |
| 6 | 77816.33333333333, | 73423.26666666666, |
| 7 | 76265.83333333333, | 71417.66666666666, |
| 8 | 73249.33333333334, | 67708.66666666667, |
| 9 | 67949.0, | 61491.299999999996, |
| 10 | 60052.33333333333, | 57088.26666666666, |
| 11 | 56291.8 | 56274.8 |

Table 3: Net profit of the server in a double queues system: both servers vary effort as the queue's length changes

| n | Server 0 | Server 1 |
|---|---|---|
| 0 | 60700.76666666665, | 83506.73333333331, |
| 1 | 61268.76666666668, | 83583.93333333335, |
| 2 | 61611.899999999994, | 83443.4, |
| 3 | 61325.03333333333, | 82505.86666666667, |
| 4 | 61389.63333333334, | 81356.26666666668, |
| 5 | 61128.13333333333, | 79814.86666666667, |
| 6 | 60622.033333333326, | 77279.46666666666, |
| 7 | 59723.899999999994, | 73513.4, |
| 8 | 58583.299999999996, | 67962.0, |
| 9 | 57040.03333333333, | 59971.066666666666, |
| 10 | 56291.8, | 56274.8, |
| 11 | 56291.8 | 56274.8 |

Table 4: The net profit of two servers when server 0 varies effort and server 1 uses low effort all the time

| n | Server 0 | Server 1 |
|---|---|---|
| 0 | 63688.266666666656, | 59043.33333333333, |
| 1 | 69843.36666666667, | 62030.333333333336, |
| 2 | 73520.13333333335, | 63286.66666666668, |
| 3 | 76610.06666666667, | 64335.833333333336, |
| 4 | 78241.36666666667, | 64333.33333333333, |
| 5 | 79760.29999999999, | 64371.0, |
| 6 | 80961.70000000001, | 64147.50000000001, |
| 7 | 81567.53333333334, | 63350.166666666664, |
| 8 | 81707.70000000001, | 61970.0, |
| 9 | 81818.0, | 59980.99999999999, |
| 10 | 82192.4, | 59237.0, |
| 11 | 82192.4 | 59237.0 |

Table 5: Net profit of the server in a double queues system: Server 0 adjusts effort as the queue's length changes, Server 1 uses high effort all the time