Harry Zhu

CS150

Project 2 Report

- Introduction

This project is designed to deal with a coffee shop trying to find the most ideal number of cashiers that they need to hire for max amount of profit. It is assumed that every cashier is hired with the same amount of daily salary, c; the shop opens everyday with the same hours, 6 am to 9 pm; at any point in time, the line should not be longer than eight times the number of the cashiers, as the overflown customers will be turned away; each customer is said to spend the amount of a random number in a specific range; it takes a random amount of time between a specific range to serve each customers; and the daily net profit of shop is the total revenue minus the daily cost of cashiers. Under such circumstances, the lab is to model how the number of cashiers hired changes while the number of cashiers vary, thus decide the number of cashiers that would maximize the coffee shop’s profit.

- Approach

Another purpose of the lab is to demonstrate event-based simulation design. Event-based simulation models operation of a system as a series of events rather than a continues timeline which would significantly improve the running time of our program as it does not simulate many during the day when nothing is happening in the coffee shop as oppose to time-based simulation.

In the simulation, a few data structures are used. The most important one is priority queue which is used for storing a few components in the simulation. Priority queue works as a sorted queue which works perfectly as a queue in the coffee shop storing all the events that are yet to happen, and cashiers both sorted by time. The other two data structures used are array list and array deque. Array list is used to store customers while array deque are used to store the busy cashiers with the Queue interface.

To module the coffee accurately, a few classes are created as objects that are to be in the shop during the simulation. The first one is customer class. The customer class takes three parameters in initialization. The first one is a String in the format of “hh:mm:ss” which represent its arrival time. It is broken up into three integers and stored as one integer for the arrival time of the customer. The second parameter takes in the amount of time that the customer takes to order in seconds, and the third parameter takes in the amount of money a customer spends ordering. The second class created in the simulation is an abstract class called Event. Event has two field variables, one called start Time and the other as customer. The Event class is used to model two main events happening in the shop, a customer arriving and a customer leaving the shop after ordering. Subsequently the event class has two subclasses, arrival and departure. Both arrival and departure store the time that the event occurs, and the customer that the event is associated to. However, departure class has one additional variable which is called counter. The counter stores the information about the cashier that the customer to which the object is associated ordered in, so that when departure is processed, it can find the appropriate cashier to process said event again. Another class essential class to the simulation is the cashier. As its name suggests, cashier class models single cashiers in the simulation. Each cashier class has three field variables. First one, time, represents the time each cashier starts. It is set to 6 am upon initialization. The second variable is counter which is assigned to each instance of cashier upon initialization and follows each cashier during the entire day. The last class would be simulation.

Simulation class models the entire coffee shop: how it takes input parameters; how it runs; how it outputs data for each simulation, and so forth. To correctly model how coffee shop run in a normal day, two groups of objects are necessary: *events* and *cashiers*, and each one of those objects stores its own time, which means all these objects need to be stored in priority queues. Before the day begins, all events for all customers that comes in this day are stored in one priority queue called *eventsIn*, while all cashiers are stored in another priority queue called *cashiers*. Another two queues, on the other hand, are also created. *CashiersBusy* is an array deque. It stores *cashier* objects just as the priority queue *cashiers*. *Line* is another priority queue, stores event objects just as *eventsIn*. At the very beginning of the day, one event is taken out of priority queue *eventsIn*. The program see it is an arrival objects, so subsequently, a cashier is taken out of the *cashiers* priority queue. Based on the customer stored in the arrival event, time when the customer is supposed to leave is calculated – if the customer arrives later than the time stored in cashier, time of departure equals to arrival time plus, else time equals to time stored in the cashier plus the ordering time of the customer – and put into both *departure* and *cashier*. At the same time, the amount that the customer ordered is added to the revenue, and the counter number of the *cashier* is also stored in the created *departure* event. After all the calculation, the *departure* object is put back into the priority queue *eventsIn*, while the *cashier* will be put into the array deque *cashiersBusy* and will not be taken out until the *departure* object just created are processed again.

As the day progresses, events of *departure* will start to appear when taking event out of the *eventsIn* priority queue. When an event is taken out of *eventsIn*, and identifies as an instance of *departure*, an entirely different process will take place. *Cashiers* will not come out of the priority queue *cashiers*, but *cashiersBusy* instead. The counter number of the *cashier* will be compared to the counter number stored in the *departure* event, and the said *cashier* will be put back into the *cashiersBusy* while a new one is taken out for comparison until a match is observed. Then the *cashier* with the correct counter number will be put back into the priority queue *cashiers*, and the *departure event* will be discarded.

During a particularly busy day, there will be time when all cashiers are busy, while the top of *eventsIn* is still an *arrival* event. That is when too people come into the shop and a line forms. The line priority queue is created specifically for this situation. Before every execution of taking an element out of *eventIn*, *cashiers* is checked to see whether there is cashier available. When there is not any, *arrival* events are taken out and put into the priority queue *line* until the next departure appears and checks a *cashier* back from the *cashiersBusy* array deque. In the event of the length of *line* exceeds the said limit, eight times the number of cashiers, *arrival* events will be removed from *eventsIn* without being put back anywhere, as the customer associated with the said arrival event will be identified as an overflow. Since *line* represents the actual line that forms in the coffee shop, it will have higher priority when taking *events* out over *eventsIn.*

After each day all cashiers should be back into the *cashiers* as all customer that started to order should already have departed the store. A set of data: profit, average waiting time, maximum waiting time, number of customers served, rate of overflow, and etcetera are then calculated for data analysis.

- Methods

To achieve the most accurate simulation for each number of cashiers, thirty different days are created for simulating the coffee shop. In each day, a set of simulations is initialized with cashier number from 1 to 20. Then *readFile* method for each simulation is called with the exactly same parameters. *ReadFile* method takes three parameters. The first one is a String for the input file that is to be read for the simulation. The second and third are two integers used as seeds for ordering time and money spent by each customer. Then the simulation is run.

During the simulation, a set of data is recorded and printed out into three .csv files: the profit for each day, the rate of overflow for each day, and the average waiting time for all customers each day.

- Data Analysis

The profit for each day of running the coffee shop depends on two key data assuming the set of customers who are to visit the coffee shop on that day are the same for each simulation. One is the overflow of customers. The number of customers that are served represents the revenue for each day. The more customers the coffee shop serves, the higher the revenue is going to be. Therefore, to maximize the profit for each day, it is ideal to have the least amount of overflow customers. The other key to having the highest profit is to have the least number of cashiers, as hiring each cashier takes quite a large amount of money compare to the money each customer brings in.

To express such logic in mathematical formula, it be can derive:

It is assumed that all the money that all customer that visit brings in is possible revenue, the cost for each cashier is c.

It is then assumed that the variable x represents the number of cashiers, overflow rate can be then expressed with f(x).

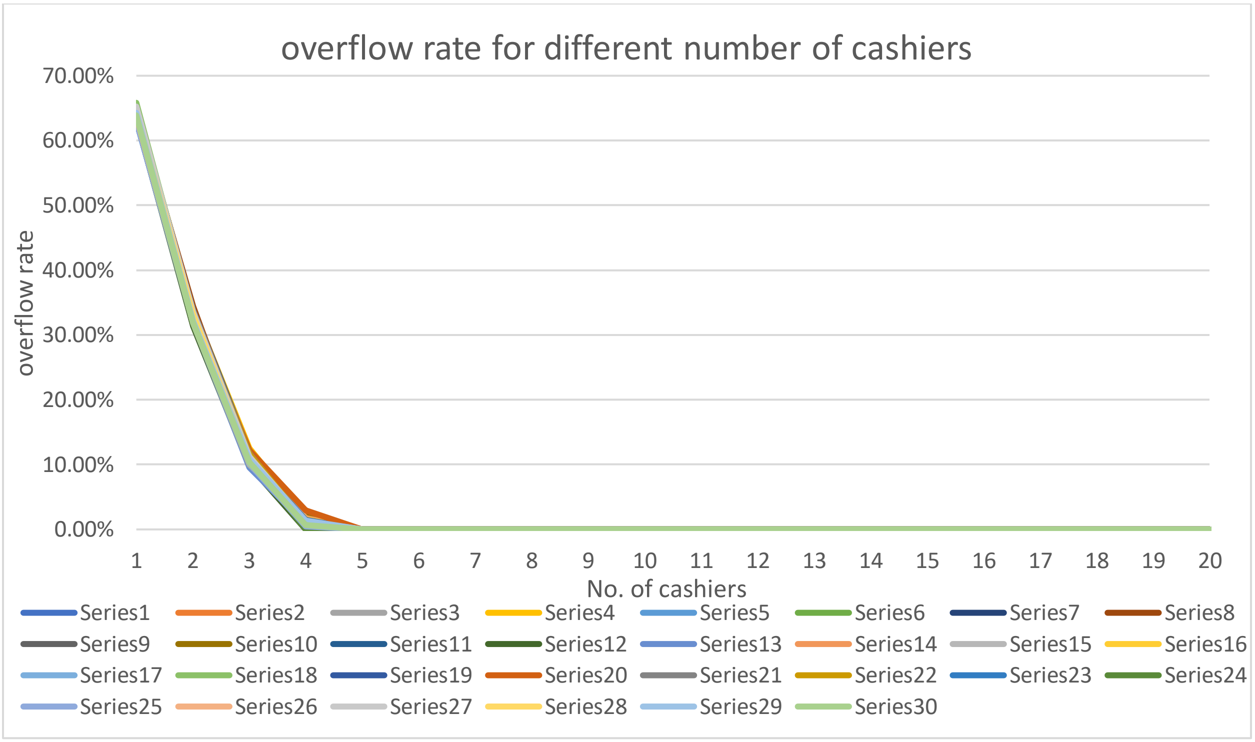
$$profit = possible\ revenue\left(1 - f(x)\right) - x * c$$

To find the c where profit maximize, we need to set the derivative of profit equals to 0:

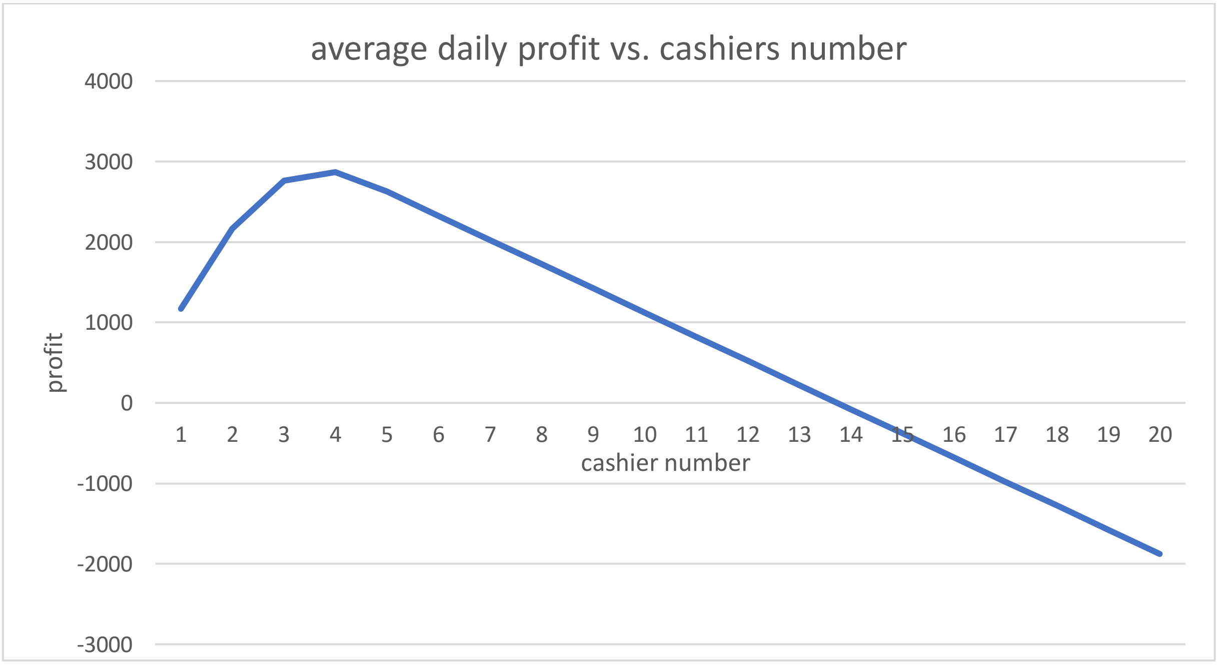
$$profit' = -possible\ revenue * \frac{df}{dx} - c$$

$$\frac{df}{dc} = -\frac{c}{possible\ revenue}$$

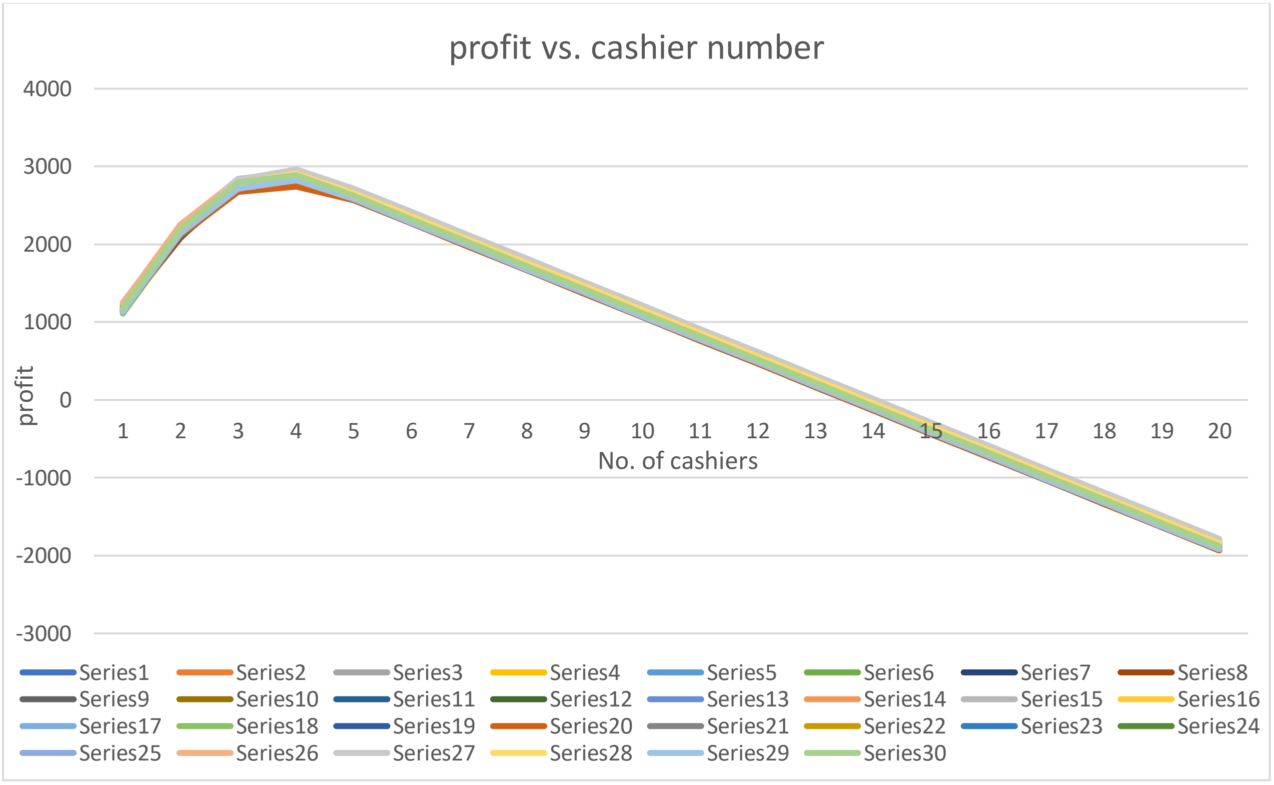
Although such equation only predicts the approximate location of the most ideal number of cashiers, it can still be calculated when the slope of the graph is around 0.2%. That is, just before the overflow rate entirely stop decreasing, the shop gets its highest profit.



From the graph above, it can be observed that when the coffee shop has four cashiers, the rate of overflow is the lowest right before it entirely stays at zero starting at when there are 5 cashiers. It cannot be 5 as there is no instance of simulation when 5 cashier produces overflow, which means that at c=5, the overflow rate has long done decreasing. We can then exclude 5 cashiers from being the most ideal number of cashiers.



The graph above shows the result of all simulations. It can be observed that the result matches the prediction, as 4 cashiers show the highest profit. It can also be observed that as soon as the overflow number reaches zero, profits starts decreasing linearly since there is no addition revenue is being created, and hiring more cashiers only takes money away from profit.



Conclusion

This project has clearly shown the advantage of event-based simulation as simulating every second for an entire day would take enormous amount of time compare to the two thousand events running each day. Such simulation process has made simulating such large amount of event happening, and multi-thread programming possible.