Network Programming Lab. 2

Food Ordering Simulation. Based on restaurant simulation.

Food Ordering Simulation

Going to the restaurant for a dinner is an experience which is not really possible to get during the pandemic. For now let's consider our restaurant business grows and adopts to new conditions.

The basic idea of food ordering simulation is to have multiple clients ordering food from multiple restaurants. Your goal is to adopt your restaurant implementation to be able to integrate with some "third" party services such as `Food Ordering` service.

For this simulation we will add 2 more components to the system.

1. Client service
2. Food Ordering a.k.a Restaurant Aggregator service.

Let's analyze each new component individually.

Client service

Client service represents component which simulates end clients of the system, which want just to order and pick up food from restaurant.
Client service have to generate end users orders and send orders to `Food Ordering` service. Client service consists of `clients` which will actually generates orders and represents abstraction of a real system client.

Client

Main work unit of client service is `Client`. Each `client` is a separate work unit which works completely separate and isolated with respect to other clients from client service.

Food Ordering / Restaurant Aggregator service.

`Food ordering` service is connection link between `clients` and `restaurant`. In order to order some foods, client communicates with `food ordering` services and post generated orders. `Food ordering` service receives orders requests from `clients` and post order to dedicated restaurant.

`Food ordering` main work unit is `Orders Manager` it implements communication between `restaurant` and `clients` in process of food ordering. `Order Manager` handles orders requests generated by `clients` and send received `orders` to dedicated `restaurants`.

One of the important feature of `food ordering` service is aggregation of data from all restaurants and clients.

System Design and Requirements

In this simulation we will have multiple clients communicating with `food ordering` service and it should communicate with multiple restaurants.

For this simulation we will use multiple restaurants, each restaurant being replica of same restaurant implementation but with different menu and configuration of number of tables, cooks and cooking apparatus.

Your setup have to include:

1. One `client service` with multiple work units (threads) of clients.

2. One `food ordering` service
3. Multiple restaurants. Each restaurant is represented by dinning hall and kitchen. Your simulation have to include at least 2 restaurants with various setup. **Recommended number of restaurants is 4.**

Let's analyze each new component individually.

Client Service

In `Client service` we have multiple independent work units which are `clients`.
Each client have to be implemented as a dedicated thread. Each client implements logic of generating new totally random order with random number of foods and random foods assigned to this order. Actually `client` is just a combination of `table` and `waiter` from `dinning hall`.

In order to know which restaurants are connected to our `Food ordering` system, each `client` first of all have to request from `Food ordering` service, data about available restaurants which will include each restaurant menu. Having all restaurants menu, `Client` generates random order and send generated order to `food ordering` service. `Client` can order from multiple restaurants, that means that `client` final order send to `Food ordering` service can include multiple restaurants orders.

Client order is represented by at least 1 restaurant order, but usually client should order from multiple restaurants. For each restaurant order `maximum wait time` that a client is willing to wait before order pick up, should be calculated by taking the item with the highest preparation-time from the restaurant order and multiply it by `1.8`.

As a response sending generated order to `Food ordering` service, `client` receives data which includes estimated preparation time for each order for a dedicated restaurant. `Client` waits for this time plus some additional realistic time coefficient and after this, performs request to restaurant `dinning hall` to pick up the order.

If order is not ready at the time `client` performs request to the restaurant `dinning hall` in order to pick up his order.
`Client` waits for the order the time specified in response from restaurant `dinning hall`.

Each `client` should be created and destroyed for each new order request. This means, `clients` have to be destroyed after they pick up their orders. New clients should be created instead of destroyed `cients`.

Food Ordering Service

`Food ordering` service, plays role of data aggregator. All restaurants have to register in `Food ordering` service and store multiple restaurants menu in order to provide each restaurant menu to `client`.

`Food ordering` service should be able to handle registration of new `restaurant` at any time of of the simulation. Also `Food ordering` service should be able to handle cases when some restaurant will fail and will not be able to prepare more orders or will totally shutdown.

`Food ordering` service workflow:

- receives `client` orders and send that orders to dedicated `restaurants`
- aggregates response from all restaurants into single response
- sends that response back to `client`.
- receives client order ratings and propagate ratings to the restaurants
- calculates simulation average rating based on restaurants average rating

`Food ordering` service is responsible for calculating simulation average rating. It should aggregate all restaurants ratings and calculates(updates) the average after each rating being submitted by the client.

Restaurant

`Restaurant` is represented by `kitchen` and `dinning hall` which was implemented in previous simulation. All communication with `Food ordering` service and `Clients` service is managed by `dinning hall`.

All communication with new components have to be implemented on new route `/v2`. On this route, `dinning hall` have to define new endpoints which will implement communication with `Food ordering` service and `Clients` service.

`Restaurant` initialize communication with `Food ordering` service by registering itself in `Food ordering` system by providing its menu in request body together with additional restaurant info.

`Dinning hall` receives orders from `Food ordering` service and sends that orders together with orders generate by `dinning hall` itself. Communication protocol between `dinning hall` and `kitchen` remains the same as it was previously implemented.

For orders received from `Food ordering` service, `dinning hall` have to provide as a response to request, estimated preparation time.

For orders received from `Food ordering` service, estimated waiting time is equal to:

`(A / B + C / D) * (E + F) / F`

where:

A = total preparing time for all foods from order which do not require cooking apparatus

B = total sum of cooks proficiencies

C = total preparing time for all foods from order which require cooking apparatus

D = number of cooking aparatus

E = total number of foods which are in waiting list for current moment of time. Foods from orders which was registered by the kitchen but was not yet prepared

F = number of foods from current order

After order will be prepared by `kitchen`, order should be send back to `dinning hall`. `Dinning hall` should keep order until it will be picked up by `client`.

Reputation system

For each restaurant order generated by client (don't forget client can create orders which includes suborders from multiple restaurants), after order is picked up from restaurant, you will have to calculate food ordering rating.

After picking up the order from some dedicated restaurant you have to stop order timer and calculate restaurant order total preparing time.

```
Order_total_preparing_time = order_pick_up_timestamp - order_serving_timestamp
```

As previous according to the preparation time we will calculate rating based on the table.

Time frame	>⭐
< max_wait	5
max_wait * 1.1	4
max_wait * 1.2	3
max_wait * 1.3	2

Time frame	> ⭐
max_wait * 1.4	1

> max_wait * 1.4 | 0

Client calculates rating for each restaurant order after picking up the order itself. After all orders are picked up, for all restaurants order ratings are recorded and posted to the `food ordering` service which propagates this rating to the dedicated restaurant.

At the end we are interested in simulation average rating.

When the `food ordering` service submits rating to the restaurant, as a response it receives restaurant average rating. Simulation average rating equals to the average of the all restaurants ratings.

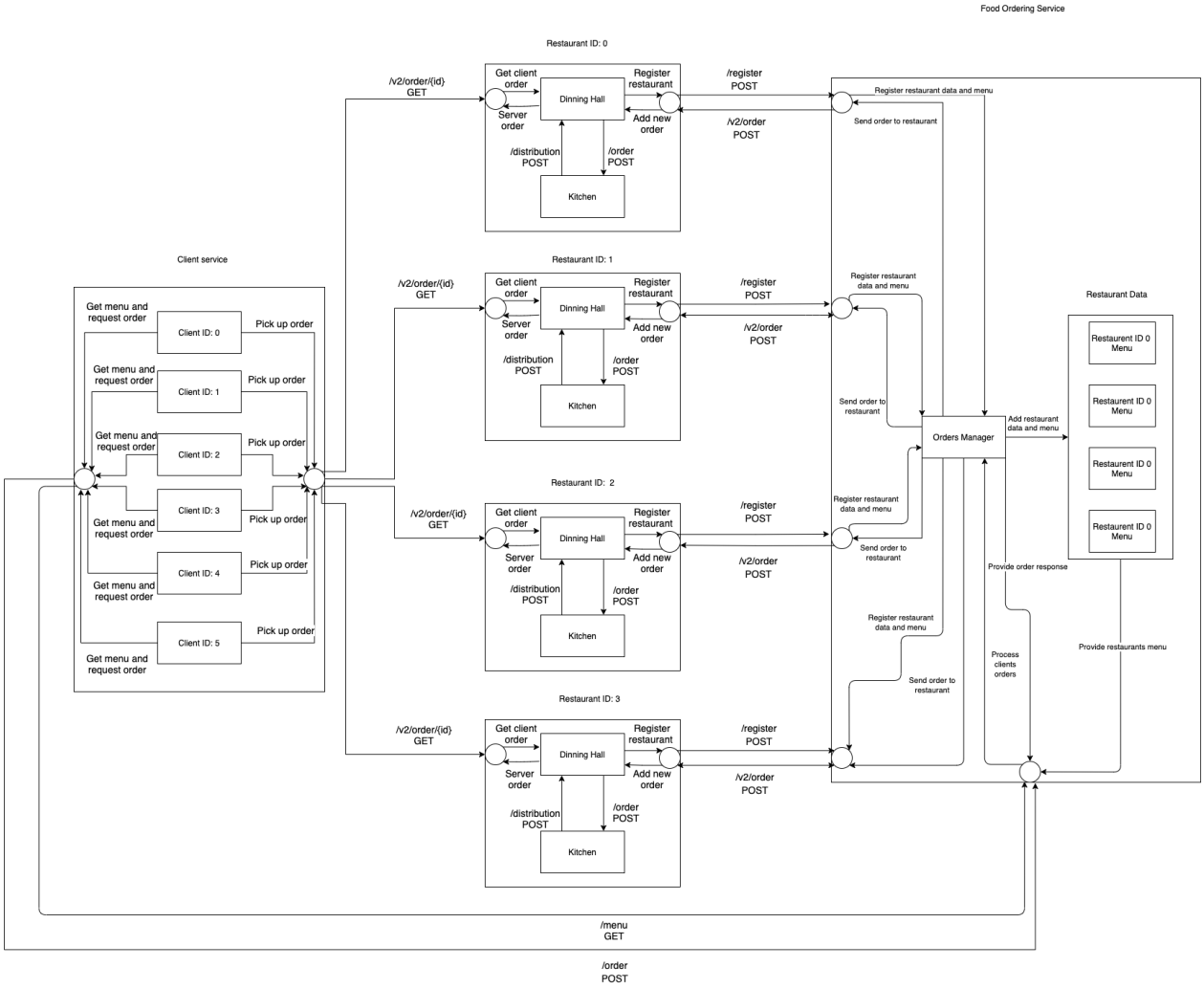
Ex: In case you have 2 restaurants:

Avg. Simulation Rating = (Restaurant_0_rating + Restaurant_1_rating) / N.

where

N = number of restaurants

System Architecture



Food ordering simulation workflow

1. All restaurants should register at `food ordering` service by performing request to related endpoint.
2. `Food ordering` service have to aggregate all restaurants data and store it. `Food ordering` **have to store it in memory not in a database**
3. Clients from `Clients service` perform requests to `Food ordering` service and receive aggregated restaurants data, which includes restaurant menu.
4. Clients pick up some restaurant, one or more and generate order(s) based on picked up restaurant(s) menu. Client randomly decides from which restaurants it want to order and generates orders for all selected restaurants. Order(s) should be randomly generated.
5. When some client creates one or more orders for one ore more restaurants. It sends order to `Food ordering` service. As a response client should receive info about received order in `Food ordering` service.
6. At receiving new `order` your `Food ordering` service should post that order to dedicated restaurants. Don't forget that client can order from multiple restaurants.
   As a response `Food ordering` service receive from restaurant info about order. In case when client orders from multiple restaurants, `Food ordering` service have to wait response from all restaurants and aggregate that data into single response send to a client.
7. `Food ordering` service is posting order to restaurant `dinning hall` and it posts orders which comes from `Food ordering` service together with other orders orders generated by `dinning hall` itself to the kitchen which actually prepares these orders.
8. Client from `Clients service` receives as a response from `Food ordering` service data about its order(s) and after dedicated time period mentioned in response from `Food ordering` service, it performs request to restaurant `dinning hall` in order to pick up order.
9. `Client` after picking up order have to calculate order rating and have to send rating or multiple ratings in case `client` have requested order from multiple restaurants. Order(s) ratings have to be posted on dedicated endpoint of `Food ordering` service.
   The dedicated restaurant order rating is propagated to the restaurant itself by `Food ordering` service and as a response it receives restaurant average rating. Based on all restaurants average ratings, `Food ordering` service updates simulation average rating.

Communication and messages format (Communication protocol)

Messages format for communication between `Dinning Hall` and `Kitchen` remains the same.

New components should follow next communication message format.

`Food ordering service`

```
Endpoint: /register
Method: POST
Payload:
{
  "restaurant_id": 1,
  "name": "McDonald's"
  "address": "localhost:8080"
  "menu_items": 5,
  "menu": [
    {
      "id": 1,
      "name": "pizza",
      "preparation-time": 20 ,
      "complexity": 2 ,
      "cooking-apparatus": "oven"
    },
    {
      "id": 2,
      "name": "Burger",
      "preparation-time": 15,
      "complexity": 1,
      "cooking-apparatus": "oven"
    },
```

```
    {
      "id": 3,
      "name": "Gyros",
      "preparation-time":     15,
      "complexity":     1 ,
      "cooking-apparatus":    null
    },
    {
      "id": 4,
      "name": "Waffles",
      "preparation-time": 10,
      "complexity": 1,
      "cooking-apparatus": "stove"
    },
    {
      "id": 5,
      "name": "salad",
      "preparation-time": 10 ,
      "complexity": 1 ,
      "cooking-apparatus": null
    }
  ],
  "rating": 3.7,
}
```

```
Endpoint: /menu
Method: GET
Payload:
{
  "restaurants": 4, // number of registered restaurants
  "restaurants_data": [
    {
      "name": "McDonald's"
      "menu_items": 5,
      "menu": [
        {
          "id": 1,
          "name": "pizza",
          "preparation-time": 20 ,
          "complexity": 2 ,
          "cooking-apparatus": "oven"
        },
        {
          "id": 2,
          "name": "Burger",
          "preparation-time": 15,
          "complexity": 1,
          "cooking-apparatus": "oven"
        },
        {
          "id": 3,
          "name": "Gyros",
          "preparation-time":     15,
          "complexity":     1 ,
          "cooking-apparatus":    null
        },
        {
          "id": 4,
          "name": "Waffles",
          "preparation-time": 10,
          "complexity": 1,
          "cooking-apparatus": "stove"
        },
        {
          "id": 5,
          "name": "salad",
          "preparation-time": 10 ,
          "complexity": 1 ,
          "cooking-apparatus": null
        }
      ],
      "rating": 3.7,
```

```
    },
    ...
  ]
}
```

```
Endpoint: /order
Method: POST
Payload:
{
  "client_id": 1,
  "orders": [
    {
      "restaurant_id": 1,
      "items": [ 1, 4, 2 ],
      "priority": 3,
      "max_wait": 45,
      "created_time": 1631453140 // UNIX timestamp
    },
    {
      "restaurant_id": 2,
      "items": [ 1, 5, 2, 3 ],
      "priority": 1,
      "max_wait": 68,
      "created_time": 1631453140 // UNIX timestamp
    }
    ...
  ]
}

Response:
{
  "order_id": 1,
  "orders": [
    {
      "restaurant_id": 1,
      "restaurant_address": "localhost:8080",
      "order_id": 1,
      "estimated_waiting_time": 75,
      "created_time": 1631453140 // UNIX timestamp,
      "registered_time": 1631454560 // UNIX timestamp
    },
    {
      "restaurant_id": 2,
      "restaurant_address": "localhost:8081",
      "order_id": 2,
      "estimated_waiting_time": 52,
      "created_time": 1631453140 // UNIX timestamp,
      "registered_time": 1631454560 // UNIX timestamp
    }
    ...
  ]
}
```

```
Endpoint: /rating
Method: POST
Payload:
{
  "client_id": 1,
  "order_id": 1,
  "orders": [
    {
      "restaurant_id": 1,
      "order_id": 1,
      "rating": 3,
      "estimated_waiting_time": 75,
      "waiting_time": 85
    },
    {
      "restaurant_id": 2,
```

```
      "order_id": 2,
      "rating": 4,
      "estimated_waiting_time": 50,
      "waiting_time": 55
    },
    ...
  ]
}

Response:
Payload: Empty response body
Status Code: 204
```

`Dinning Hall V2 routes`

```
Endpoint: /v2/order
Method: POST
Payload:
{
  "items": [ 1, 4, 2 ],
  "priority": 3,
  "max_wait": 45,
  "created_time": 1631453140 // UNIX timestamp
}

Response:
{
  "restaurant_id": 1,
  "order_id": 1,
  "estimated_waiting_time": 75,
  "created_time": 1631453140 // UNIX timestamp,
  "registered_time": 1631454560 // UNIX timestamp
}
```

```
Endpoint: /v2/order/{id}
Method: GET

In case order was not yet prepare
Response:
{
  "order_id": 1,
  "is_ready": false,
  "estimated_waiting_time": 15, // Estimated time until order will be prepared
  "priority": 3,
  "max_wait": 45,
  "created_time": 1631453140, // UNIX timestamp,
  "registered_time": 1631454560, // UNIX timestamp
  "prepared_time": 0, // UNIX timestamp
  "cooking_time": 0,
  "cooking_details": null
}

In case order is ready
Response:
{
  "order_id": 1,
  "is_ready": true,
  "estimated_waiting_time": 0,
  "priority": 3,
  "max_wait": 45,
  "created_time": 1631453140 // UNIX timestamp,
  "registered_time": 1631454560 // UNIX timestamp
  "prepared_time": 1631453140 // UNIX timestamp
  "cooking_time": 65
  "cooking_details": [
    {
      "food_id": 3,
      "cook_id": 1,
```

```
    },
    {
      "food_id": 4,
      "cook_id": 1,
    },
    {
      "food_id": 4,
      "cook_id": 2,
    },
    {
      "food_id": 2,
      "cook_id": 3,
    },
  ]
}
```

```
Endpoint: /v2/rating
Method: POST
Payload:
{
  "order_id": 1,
  "rating": 3,
  "estimated_waiting_time": 75,
  "waiting_time": 85
}

Response:
{
  "restaurant_id": 1,
  "restaurant_avg_rating": 4,44,
  "prepared_orders": 42 // Total number of orders preapred by resturant.
}
```

Test configuration

Your simulation will be tested using specified configuration. Based on this configuration your simulation will be marked.

Restaurants

In your simulation you should have at least 2 restaurants. Recommended number of restaurants is 4. Your restaurants should have self defined configuration based on next criteria.

Each restaurant should have his own menu based on provided foods. Just reduce for each restaurant its menu to some dedicated foods from previously provided list.

Dinning Hall

For all restaurantsIn dinning hall you have to have:

- 6 Tables
- 3 Waiters

Kitchen

In each restaurant you should have at least one cook of each rank and maximum 2 cooks for one of the ranks. Total proeficiency of cooks could not be bigger than 14. Recommended total proeficiency for all cooks is between 10-12.

Cooking apparatus number depends on number of foods in restaurant menu which requires cooking apparatus.

- Ovens = number of foods from restaurant menu which requires ovens / 3. If it is not integer number always round down to a integer number. But it could not be 0 in case cooking aparatus is required. Ex. 2.8 is 2 ovens in the kitcehn

- Stove = number of foods from restaurant menu which requires stove / 3. If it is not integer number always round down to a integer number. But it could not be 0 in case cooking aparatus is required. Ex. 2.8 is 2 stove in the kitcehn

Submission and Grading Policy

In order to present laboratory work you have to pass mandatory checkpoints. Each checkpoint could be passed only in one day. It is not possible to pass all checkpoints together. The idea of checkpoints is to present your work progress in time.

Without passing checkpoints is not possible to get the final grade

Checkpoints Submission

1. 40% Checkpoint -> 1 week

    i. Two additional repositories with initialized client and food ordering services. Don’t forget about Readme and all other configurations. New services should run in docker containers.
    ii. Whole cluster communication between containers should be configured. You have to prove that simulation cluster is running and all required containers communicate with each other. Some logs in std output will be enough.
    iii. Generation of dummy orders and sending that orders to food ordering from client service have to be implemented. Without taking into a count order pick up. Just having some multiple threads in client service which send some dummy orders to food ordering.
    iv. Initial logic of aggregating restaurants in food ordering service. All restaurants have to register in food ordering and food ordering service is able to post orders into some dedicated restaurant.

2. 70% Checkpoint -> 1 week

    i. Implementation of client service order generation based on provided menu from registered restaurants.
    ii. Implementation of food ordering service. It should include logic of sending orders to dedicated restaurants based on order received from client service.
    iii. Additional restaurants implementation optimisation. Improve logic of food preparing, having orders which comes from food ordering service and orders which comes from dinning hall.

3. 100% Checkpoint -> 1 week

    i. Implementation of orders pick up logic in clients service. Each client should perform request to dedicated restaurants and pick up requested orders. Implementation of clients dispose and re-creation logic in client service.
    ii. Implementation of client order rating. For each order picked up by client should be calculate rating and final client order rating should be submitted to food ordering service. Don't forget your simulation rating should be as big as possible.
    iii. Implementation of simulation average rating.
    iv. Optimisation of simulation. Adjustments in order to increase average simulation rating.

Grading

Your laboratory work grade will be calculated based on your checkpoints grades and your simulation average rating. It will be calculated according the formula.

Grade = 0.5 * (0.4 * 40% check grade + 0.3 * 70% check grade + 0.3 * 100% check grade) + simulation average rating

For each week being late you will be punished with -1 point.

Minimum acceptance criteria

In order to get the minimum acceptable mark, which is 5, you have to present a project which includes:

1. Three web servers which communicates over HTTP protocol between them.

2. First web server is producer which produces some data on multiple threads (more than 5) and it sends these data from multiple threds to the second web server.
3. Second server is aggregator which receives and aggregates data from first server and populates producer shared resources, a queue or stack with received data.
4. Second server also has multiple threads which extracts one by one elements from producer shared resource and is sending that extracted data element from second server to the third which is consumer.
5. Third server is consumer, which receives and consumes data from second server and populates shared resources, a queue or stack with received data.
6. Third server also has multiple threads which extracts one element from shared resource and is sending that extracted data element from it server to the second, aggregator server.
7. Second aggregator server is receiving data from third server and populates consumer shared resources, a queue or stack with received data. In aggregator server you should have 2 shared resources, a producer and consumer a queue or stack with received data.
8. Second server has multiple threads which extracts elements one by one from consumer shared resource and is sending that extracted data element from second server to the first, initial producer server.

Example implementation.

You have to create / reproduce your own and to not copy example.