北京航空航天大學
BEIHANG UNIVERSITY

实验报告

内容（名称）：队列模型(M/M/1)设计与仿真

院（系）名称	计算机学院
专业名称	计算机科学与技术
指导教师	宋晓
学号	18373584
姓名	甘天淳

2019 年 10 月

一、实验目的

应用M/M/1队列编程思想，模拟食堂单一窗口处的排队过程，熟悉离散事件推进方式、队列建立和提取方式。应用M/M/N队列编程思想，模拟食堂整体所有窗口处的排队过程，并对现实情况进行拟合，提出指导性意见。

二、数学模型描述

2.1 模型基本框架

在M/M/1队列模型中，接收 **平均到达间隔**(`averageArriveTime`)、 **平均服务时间** (`averageServeTime`)、**顾客总数目** (`customerNumber`)、 **队列最大长度** (`queueMaxLength`)为参数，模型主体为单个窗口和一个排队队列，其基本约定描述如下：

1. 顾客按照某种到达间隔数学期望为平均到达间隔的概率分布依次到达，到达时若窗口未被占用则占用窗口开始服务，若窗口被占用则进入排队队列进行排队并判断排队队列是否已满（即达到队列最大长度）。
2. 窗口按照某种服务时间数学期望为平均服务实际的概率分布花费时间进行服务，结束服务后若排队队列中仍有顾客在等待，则立即服务位于队列队首的顾客，若排队队列为空则结束自身占用状态。

在全过程中，维护一个全局的事件轴，将顾客到达、开始服务、结束服务三类事件统一管理，此事件轴按时间顺序排序，模型的仿真即为在轴上不断推进事件。仿真结束后给出 **顾客平均等待时间** 、 **队列平均顾客数** 、 **服务器利用率** 等衡量指标。

M/M/N队列模型将单窗口扩充为多窗口，增加 **窗口数量** (`windowNumber`)作为输入参数，将每一窗口的利用率均进行输出，而其基本约定与仿真推进方式等均与M/M/1队列模型无异。

事件管理方式及衡量指标的具体计算方式将在第三节中结合代码实现具体给出。

2.2 模型细节扩充

在基本框架的基础上，对细节做如下扩充及解释：

1. 顾客到达间隔按指数分布生成。符合生活常识与题述要求。
2. 服务时间按正态分布生成。由生活经验可以得出，单一业务的窗口（类似于食堂窗口）对于每个顾客的服务时间是近似相等的，方差较小，且左右震荡概率近似相等，因此我们用概率密度函数关于其数学期望对称的正态分布可以较好拟合随机的服务时间。
3. 采用下一事件时间推进的事件调度法进行仿真时间的推进。离散事件仿真的基本方法有事件调度法、活动扫描法和进程交互法三种基本方法。这三种方法核心都是事件按时间顺序发生（即按仿真时钟推进），但具体实现方式上有所不同。其中事件调度法将顾客到达和离开或完成服务当做事件，放入事件轴统一管理，维护队列长度和服务器状态作为全局参数，其事件调度核心在于每次把时钟推进到下一事件发生的时间，认为该事件发生并执行它。活动扫描法核心在于使用固定时间间隔和基于规则的方法来决定是否开始一个活动，主要方式为在每一个时间步长，检查条件，若为真则触发相应活动。这种方法写法简单，但是进行完全的活动扫描是费时费力的，数据量大时尤为如此，程序效率较低，一般不采用。进程交互法采用多个线程/进程，每个进程要处理实体在系统中流动时发生的所有事件（包括确定事件和条件事件），每个实体可能在进程中的不同时间点上被阻塞或延迟。这种方法思考难度与实现难度均较大，但其完全面向对象的特性使得在场景较为复杂时能够通过工程化方法生成代码。在本实验中，采用实现难度不大，仿真效率较高的事件调度法进行仿真。

三、编程实现与调试过程

由于M/M/N模型完全可以在M/M/1模型的基础上进行增量开发，因此直接实现M/M/N模型，通过将输入的窗口数量固定为 `1` 来进行M/M/1模型的仿真。使用脚本语言 `python` ，利用其面向对象的特性进行仿真程序的编写。源代码将以附录形式呈现，以下分三部分对代码实现与运行结果做出解释。

3.1 数据结构与组织形式

下一事件时间推进的事件调度法核心在于维护事件队列与排队队列，在实现上我们分别使用优先队列 `eventList` 和普通队列 `waitingQueue` 进行模拟，此外我们需要一个记录每个窗口占用状态的列表 `windows` ，记录每个顾客到达时间与离开时间的列表 `arriveTimeList` 和 `leaveTimeList` ，记录每个窗口上一次开始服务的时间和总服务时间的列表 `serveBeginTimeList` 和 `serveTimeList` ，此外我们需要维护全局变量 `lastTime` 和 `totalTime` 分别记录上一事件与当前事件的发生时间，其声明与初始化如下：

```
averageArriveTime = 0
averageServeTime = 0
customerNumber = 0
queueMaxLength = 0
windowNumber = 0                                    # 五个输入参数

eventList = queue.PriorityQueue()                   # 事件队列
waitingQueue = queue.Queue(queueMaxLength)          # 顾客等待队列
windows = []                                        # 记录窗口占用情况
arriveTimeList = []                                 # 记录顾客到达系统时间
leaveTimeList = []                                  # 记录顾客离开系统时间
serveBeginTimeList = []                             # 记录窗口上一服务开始时间
serveTimeList = []                                  # 记录窗口总服务时间
lastTime = 0                                        # 记录上一事件发生时间
totalTime = 0                                       # 记录当前事件发生时间
queueArea = 0                                       # 队列面积，作为计算平均队列人数的中间变量
```

下一事件时间推进的事件调度法需要将所有事件统一管理，因此将事件封装成 `Event` 对象，其中记录事件的顾客编号、窗口编号、事件类型、发生时间等信息，对外提供获取私有变量的接口与向日志中输出事件信息的接口，具体代码及解释如下：

```
class Event :
    def __init__(self, newCustomerId, newWindowId, newEventType, newOccurTime) :
                                                    # 初始化函数，接收各参数形成事件
        self.customerId = newCustomerId
        self.windowId = newWindowId
        self.eventType = newEventType               # 事件类型，约定顾客到达为 1，开始服务为2，
                                                    # 结束服务为3
        self.occurTime = newOccurTime

    def __lt__(self, other) :                       # 比较函数，时间优先，使得该类能作为优先队列的成员
        if (self.occurTime == other.occurTime) :
            return self.customerId < other.customerId
        else :
            return self.occurTime < other.occurTime

    def getCustomerId(self) :
```

```
        return self.customerId

    def getWindowId(self) :
        return self.windowId

    def getEventType(self) :
        return self.eventType

    def getOccurTime(self) :                    # 获取四个私有变量的对外接口
        return self.occurTime

    def output(self) :                          # 输出日志接口，根据事件类型分别输出不同内容
        sentence = ""
        sentence = sentence + "Time: " + str(round(self.occurTime, 2)) + ", "
        if (self.eventType == 1) :
            sentence = sentence + str(self.customerId) + " arrived"
        if (self.eventType == 2) :
            sentence = sentence + str(self.customerId) + " began to be served at window " + str(self.windowId)
        if (self.eventType == 3) :
            sentence = sentence + str(self.customerId) + " left window " + str(self.windowId)
        ui.textBrowser.append(sentence)
```

3.2 主要函数解释

程序接受五个输入参数后依次进行初始化、仿真、输出结果三个步骤，每个步骤中的主要函数解释如下：

生成顾客到达时间：关于顾客到达时间的生成与事件管理有以下约定：开始仿真前随机生成顾客人数个到达间隔，计算各顾客到达时间，生成若干到达事件并一次性加入事件队列中。具体生成方式上，利用 `numpy` 库进行指数分布随机变量的生成。生成顾客到达事件的函数及解释如下：

```
def produceCustomers(averageArriveTime, number) :
    tmpTime = 0
    tmpCustomerId = 0
    for i in range(customerNumber) :
        arriveInterval = numpy.random.exponential(averageArriveTime)  # 生成指数分布的间隔
        tmpTime = tmpTime + arriveInterval
        tmpCustomerId = tmpCustomerId + 1
        eventList.put(Event(tmpCustomerId, -1, 1, tmpTime))           # 形成事件加入队列
        arriveTimeList.append(tmpTime)                                # 维护到达时间列表
```

生成服务时间：利用 `numpy` 库生成符合正态分布的服务时间并返回。生成服务时间的函数及解释如下：

```
def serveTime() :
    normal = 0
    while (normal <= 0) :                                   # 对于异常数据重新生成
        normal = numpy.random.normal() + averageServeTime   # 生成期望为平均服务时间的
                                                            # 正态分布随机变量
    return normal
```

寻找空闲窗口：遍历所有窗口并返回第一个空闲窗口编号，若所有窗口均被占用则返回 `-1` 。具体函数及解释如下：

```
def getWaitingWindow() :
    for i in range(len(windows)) :
        if (windows[i] == 0) :          # 为0表示未被占用，直接返回其窗口编号
            return i
    return -1                           # 未找到未被占用的窗口，返回 -1
```

仿真：此函数是实现下一事件时间推进的事件调度法的主要函数。仿真的具体推进方式如下：当事件队列非空时，取出队首（即发生时间最小）事件，将当前时间推进到该事件的时间（即认为该事件正在发生），更新各窗口占用面积与顾客排队队列面积，维护各中间变量，向日志中打印事件内容，并根据事件类型做以下操作：

1. 若事件类型为 `1` （顾客到达事件），则调用 `getWaitingWindow()` 寻找一个空闲窗口，若找到则向事件队列中加入一个类型为 `2` 的事件（开始服务事件），其时间即为当前事件时间；若没有找到空闲窗口，则将该顾客编号加入顾客等待队列中并判断排队队列人数是否已达上限。
2. 若事件类型为 `2` （开始服务事件），则更改该窗口状态为占用，向事件队列中加入一个类型为 `3` 的事件（结束服务事件），其时间为当前事件时间加上调用 `serveTime()` 所得的随机服务时间。
3. 若事件类型为 `3` （结束服务事件），则更改该窗口状态为空闲，若顾客等待队列不为空则取出其队首顾客编号，向事件队列中加入一个类型为 `2` 的事件（开始服务事件），其时间为当前事件时间。

仿真具体函数及解释如下：

```
def simulate() :
    global lastTime, totalTime, queueArea
    while (not eventList.empty()) :          # 循环执行直到事件队列为空
        flush()                              # 刷新页面
        nextEvent = eventList.get()          # 取出事件队列队首事件
        lastTime = totalTime
        totalTime = nextEvent.getOccurTime()
        queueArea = queueArea + waitingQueue.qsize() * (totalTime - lastTime)
                                             # 维护全局变量与中间变量
        nextEvent.output()                   # 向日志中输出事件具体内容
        if (nextEvent.getEventType() == 1) :# 若为顾客到达事件
            tmpCustomerId = nextEvent.getCustomerId()
            tmpTime = nextEvent.getOccurTime()
            window = getWaitingWindow()      # 寻找空闲窗口
            if (window == -1) :              # 未找到空闲窗口
                if (waitingQueue.qsize() >= queueMaxLength) :
                    ui.textBrowser.append("Warning: The queue has reached the maximum number of people.\nThe simulation has stopped")
                    return                   # 顾客排队队列已满则停止仿真
                waitingQueue.put(tmpCustomerId)
```

```
                                        # 未满则加入排队队列
            else :                      # 存在空闲窗口
                eventList.put(Event(tmpCustomerId, window, 2, tmpTime))
                                        # 新增开始服务事件
        elif (nextEvent.getEventType() == 2) :
                                        # 若为开始服务事件
            tmpCustomerId = nextEvent.getCustomerId()
            tmpTime = nextEvent.getOccurTime()
            tmpWindow = nextEvent.getWindowId()
            windows[tmpWindow] = 1
            serveBeginTimeList[tmpWindow] = tmpTime
                                        # 维护全局变量与中间变量
            eventList.put(Event(tmpCustomerId, tmpWindow, 3,
tmpTime+serveTime()))
                                        # 新增结束服务事件
        elif (nextEvent.getEventType() == 3) :
                                        # 若为结束服务事件
            tmpWindow = nextEvent.getWindowId()
            tmpTime = nextEvent.getOccurTime()
            leaveTimeList.append(nextEvent.getOccurTime())
            serveTimeList[tmpWindow] = serveTimeList[tmpWindow] + tmpTime -
serveBeginTimeList[tmpWindow]
                                        # 维护全局变量与中间变量
            if (waitingQueue.empty()) :
                windows[tmpWindow] = 0      # 顾客排队队列为空则更改窗口状态
            else :
                tmpCustomerId = waitingQueue.get()
                eventList.put(Event(tmpCustomerId, tmpWindow, 2, tmpTime))
                                        # 顾客排队队列不为空则新增开始服务事件
    ui.textBrowser.append("Successfully done the simulation")
```

其余代码大多与前端 `ui` 直接相关，故在此不再赘述。

3.3 仿真结果与解释

该程序完成了仿真的基本功能并实现了实时显示队列长度等内容的附加功能，仿真进行中（图1）、仿真正常结束（图2）、仿真异常退出（图3）的程序界面截图分别如下所示：

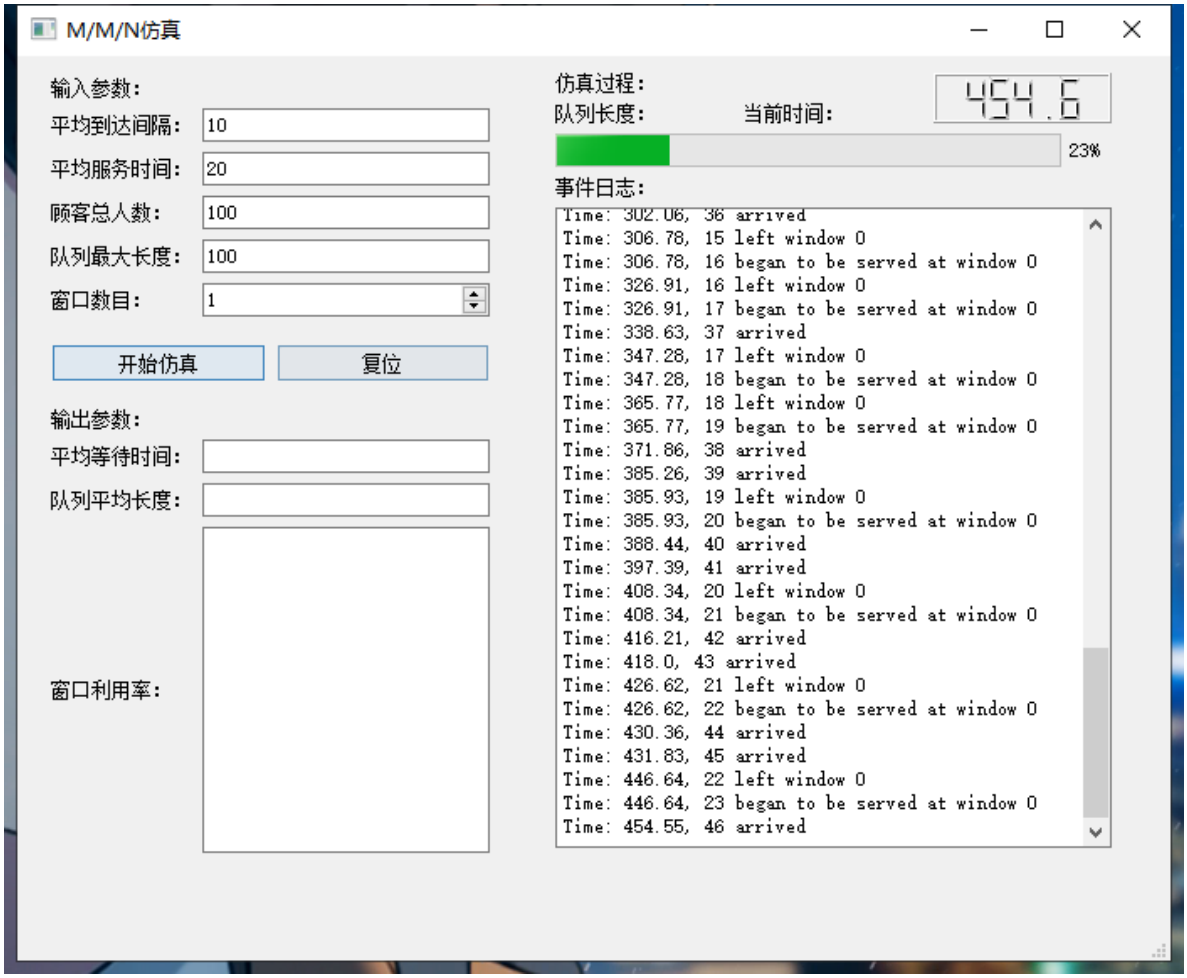

图1 程序正在运行

图2 仿真正常结束

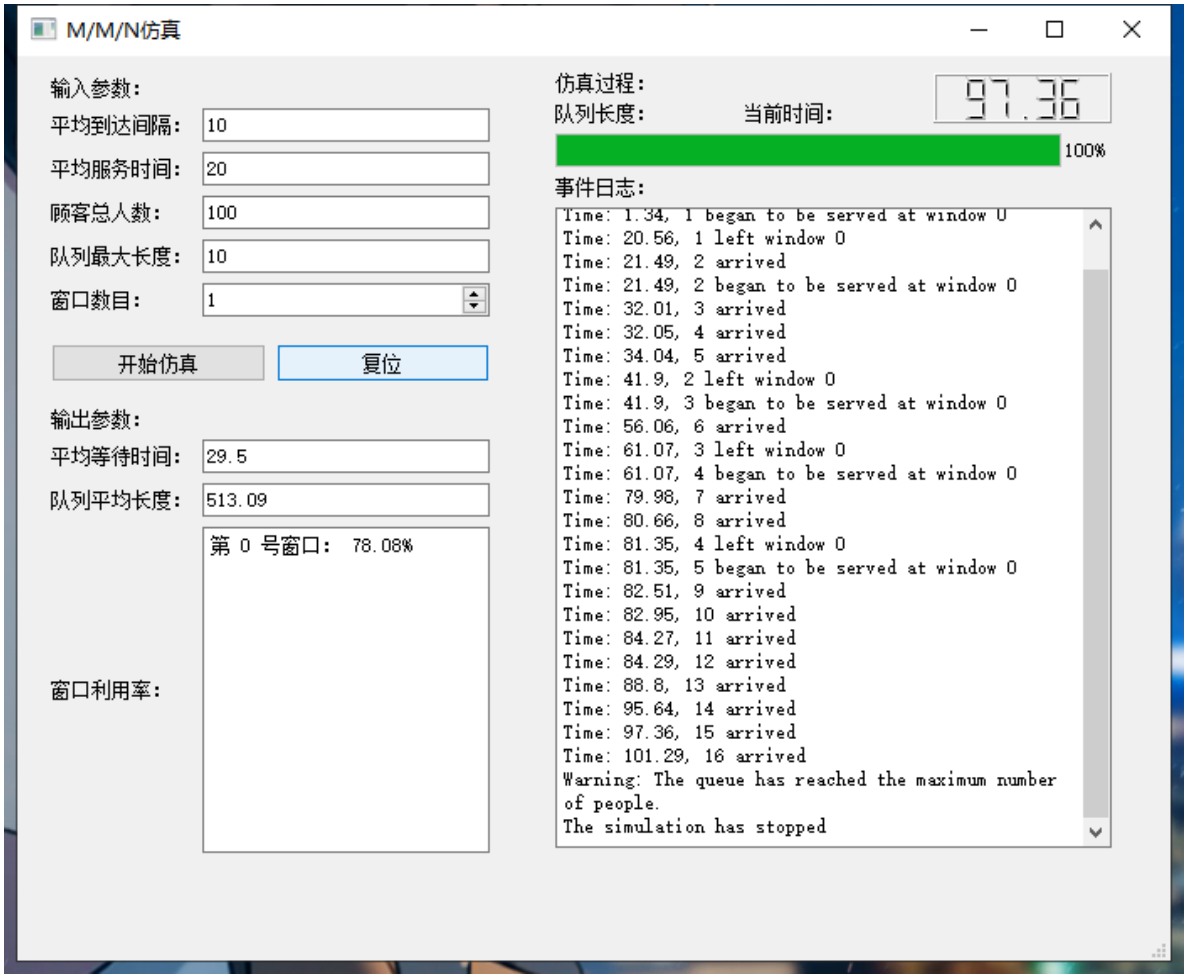

图3 仿真异常退出

为探究输入参数对仿真结果的影响，我们需要进行多次仿真。现给出数值为 `10，20，30，...，100` 的平均到达间隔与平均服务时间，在单双窗口情况下组合形成200组输入数据，为使得不同组别结果有差异，选取较大的顾客总人数，固定为 `1000` 人，每组数据分别进行五次仿真取结果平均值，共进行 `1000` 次仿真，得到如下的仿真表（由于篇幅限制，仅展示部分，完整的仿真表见电子版附表2）：

输入参数			仿真结果					仿真结果均值			
窗口数量	平均到达间隔	平均服务时间	仿真总耗时	平均等待时间	平均队列长度	0号窗口利用率	1号窗口利用率	平均等待时间	平均队列长度	0号窗口利用率	1号窗口利用率
1	10	10	10146.042452436788	102.68	9.13	0.9865					
1	10	10	10150.12672129486	242.81	32.07	0.9856					
1	10	10	10557.084784767034	59.81	35.55	0.9502		149.558	41.286	0.97224	
1	10	10	9957.917503178243	272.76	64.08	0.9982					
1	10	10	10637.634769823597	69.73	65.6	0.9407					
1	10	20	20050.192104233476	5016.11	283.98	0.9994					
1	10	20	20008.00005928775	5015.66	534.26	0.9998					
1	10	20	19978.985323339948	5098.73	789.24	0.9999		5102.414	790.698	0.99942	
1	10	20	20006.809516009384	5111.47	1042.63	0.9994					
1	10	20	20032.43561641676	5270.1	1303.38	0.9986					
1	10	30	29950.280961989025	10193.37	1211.12	0.9999					
1	10	30	30038.523853326467	10031.64	1540.52	0.9998					
1	10	30	30032.925828078107	9760.43	1864.8	0.9982		9858.698	1861.204	0.99938	
1	10	30	29978.502255324216	9520.39	2184.76	0.9997					
1	10	30	30043.482636274395	9787.66	2504.82	0.9993					
1	10	40	40029.473642619996	15117.77	2256.62	0.9992					
1	10	40	39965.28195720147	15026.75	2635.24	1.0					
1	10	40	39980.83382248431	14873.4	3005.22	1.0		14980.506	3004.116	0.99982	
1	10	40	40053.62805033405	14926.71	3371.43	1.0					
1	10	40	39966.23594792863	14957.9	3752.07	0.9999					
1	10	50	49976.21675568721	19943.51	3398.61	1.0					
1	10	50	50039.1247905715	20167.76	3796.37	0.9998					
1	10	50	50027.734413020415	20120.87	4198.43	1.0		20110.046	4199.464	0.9999	
1	10	50	49994.062914958005	20026.84	4600.85	0.9999					
1	10	50	50020.65138175121	20291.25	5003.06	0.9998					
1	10	60	59994.7322573696	25332.03	4592.54	0.9998					
1	10	60	59945.209130587165	24905.88	5010.81	0.9998					
1	10	60	59994.39533332036	25124.52	5424.48	1.0		25131.674	5425.61	0.99978	
1	10	60	60049.90382574382	25296.86	5839.73	0.9996					
1	10	60	59997.652328189135	24999.08	6260.49	0.9997					
1	10	70	69980.01249295061	29870.81	5793.3	0.9999					
1	10	70	70026.08899552395	29830.5	6214.48	0.9999					
1	10	70	70024.30083759494	29818.67	6639.47	0.9999		29931.954	6642.43	0.99988	
1	10	70	70037.67660703695	29959.54	7064.97	0.9999					
1	10	70	69990.5604491355	30180.25	7499.93	0.9998					
1	10	80	79965.68187708431	35229.64	7003.93	0.9999					
1	10	80	80002.15732742583	34811.18	7434.86	1.0					
1	10	80	79931.63768538191	35082.44	7879.33	1.0		35086.488	7874.194	0.99996	
1	10	80	80046.86693936575	35153.36	8306.15	1.0					
1	10	80	80025.20916185816	35155.82	8746.7	0.9999					
1	10	90	90008.84010668678	40043.59	8220.42	0.9998					

图4 仿真结果表（部分）

将所得的 200 组数据绘制三维图像，如下所示：

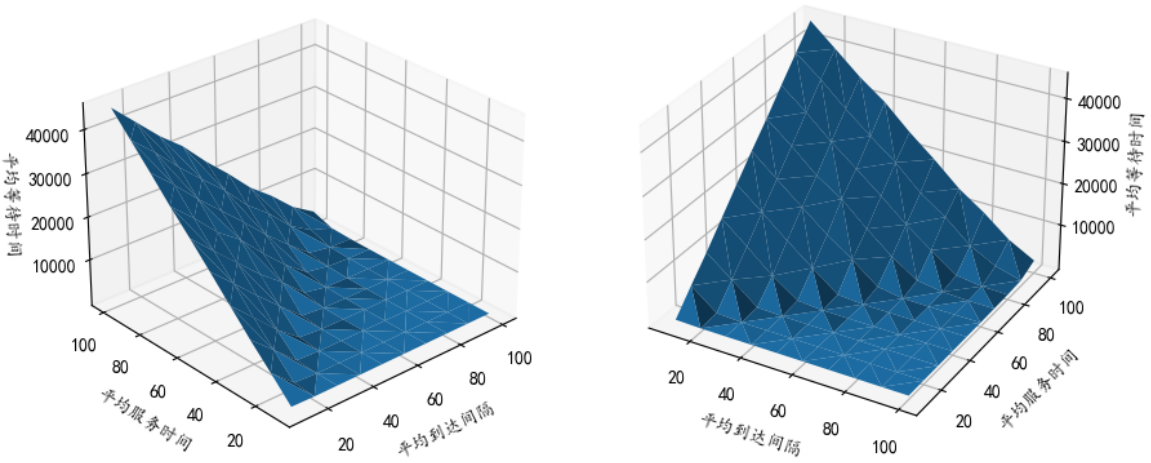

图5 两种视角下的平均等待时间与输入参数的关系图（单窗口）

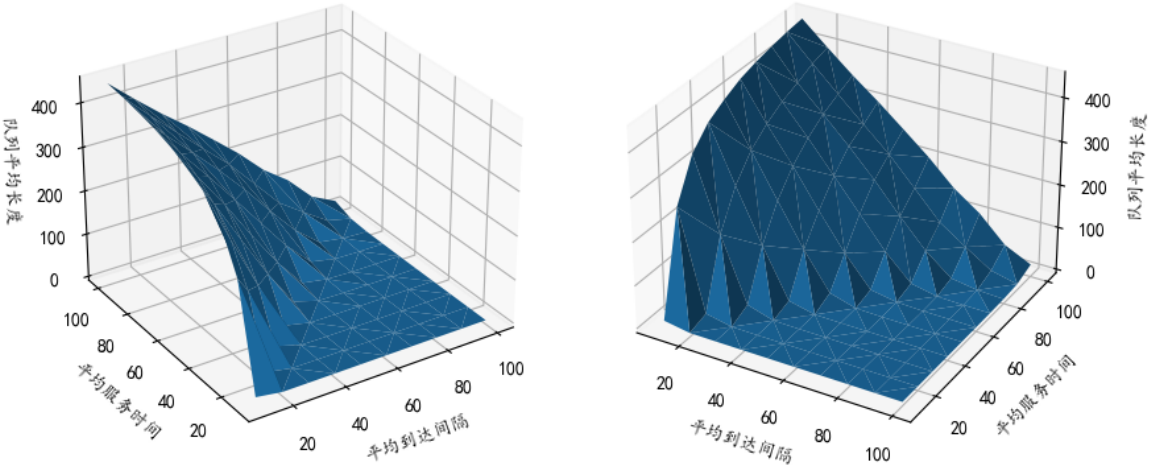

图6 两种视角下的队列平均长度与输入参数的关系图（单窗口）

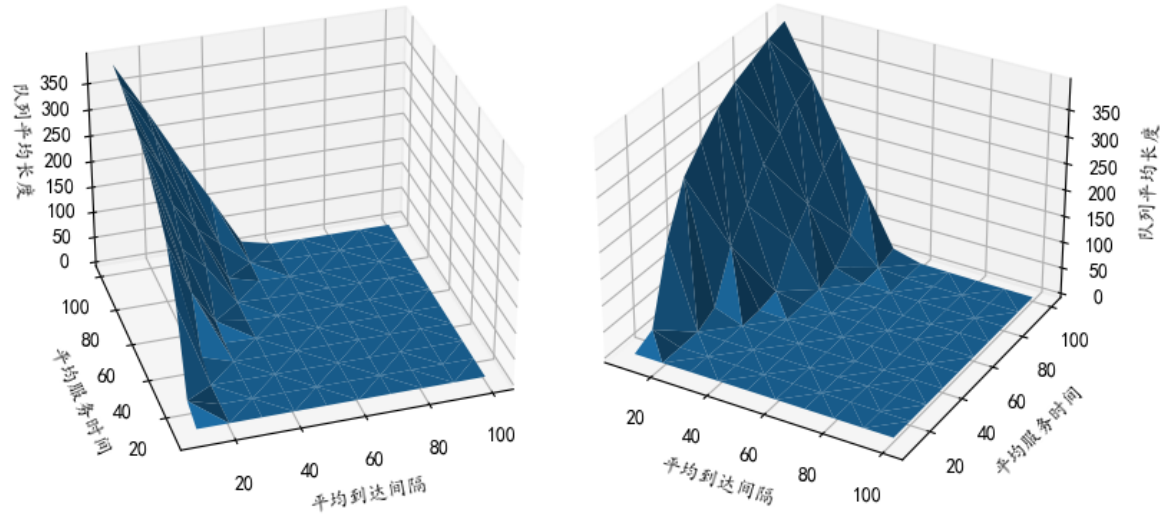

图7 两种视角下的队列平均长度与输入参数的关系图（双窗口）

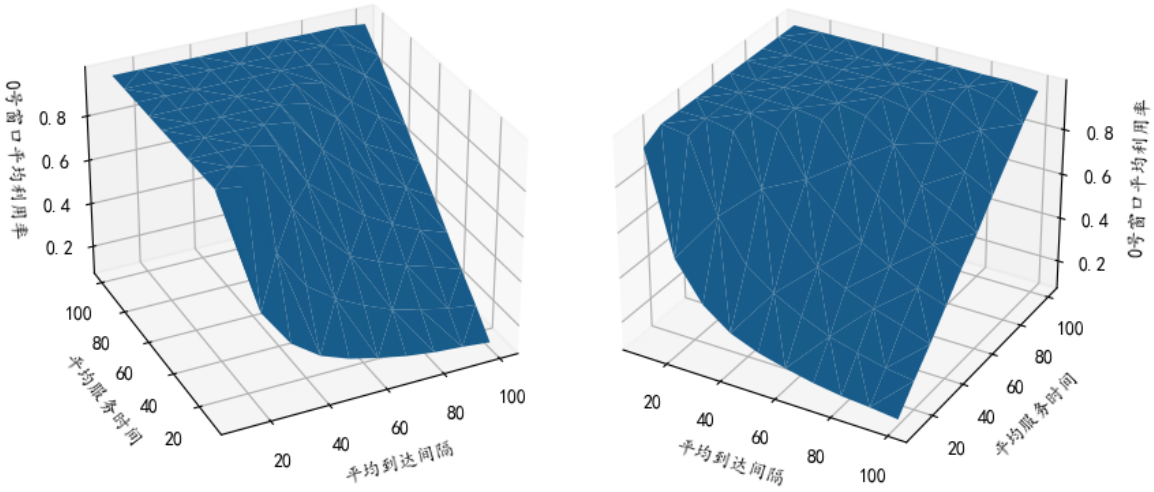

图8 两种视角下的0号窗口利用率与输入参数的关系图（单窗口）

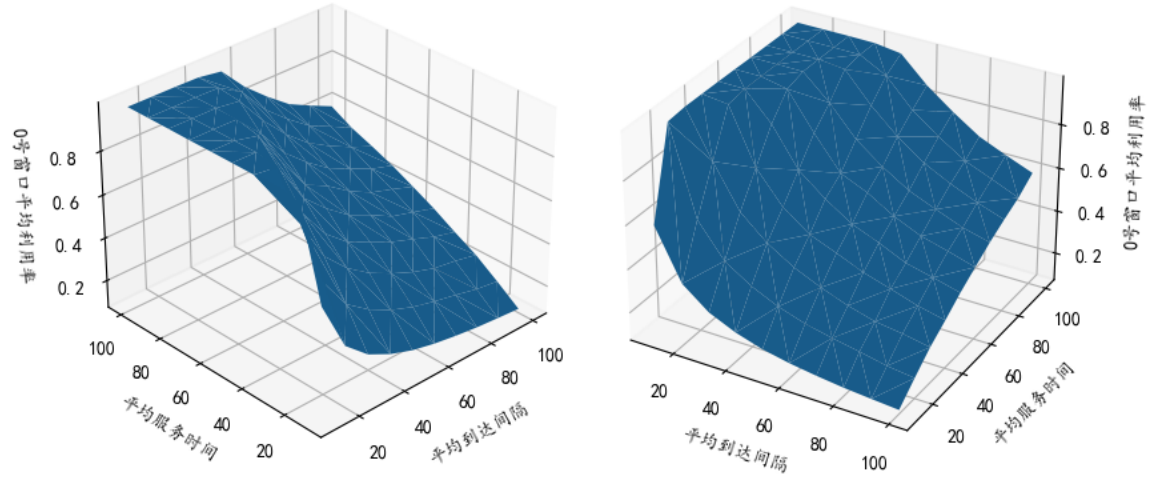

图9 两种视角下的0号窗口利用率与输入参数的关系图（双窗口）

以上结果与图像具有如下特点：

1. 总体而言，平均到达间隔越短，平均服务时间越长，则平均等待时间、队列平均长度、窗口利用率均越高，反之则均越低。
2. 单窗口下，平均到达间隔与平均服务时间近似相等时，平均等待时间、平均队列长度均近似于0，窗口利用率近似于100%。双窗口下，平均到达间隔近似于平均服务时间的一半时，平均等待时间、平均队列长度均近似于0，窗口利用率近似于100%。
3. 增加一个窗口会使得平均服务时间与平均到达间隔相等时的窗口利用率减半。
4. 平均到达间隔不变时，随着平均服务时间的延长，平均等待时间近似于线性增长，队列平均长度增长的斜率呈下降趋势，类似 `1nx` 图像。

以上特点恰与现实生活中食堂窗口排队的情况吻合，这说明仿真实验取得了良好的成果，通过对其结果的分析能够对现实场景提出指导性意见。

四、与现实的联系与窗口优化策略

以北航新北区食堂午高峰场景作为分析对象，结合上文中的仿真结果进行分析。

4.1 数据获取

新北区食堂午高峰（ `11:25 ~ 12:00` ）就餐人群往往是在北区住宿且上午第四节课有课的学生，大致估计为 `2000` 人，由于所有人同时下课，往往在十分钟内同时到达食堂，因此可认为其平均到达间隔极短，通过调查得出平均到达间隔约为 `1s` ，平均服务时间约为 `20s` ，窗口数量约为 `25` 个。

4.2 模型仿真

利用M/M/N队列模型仿真程序进行仿真，其中一次结果如下所示：

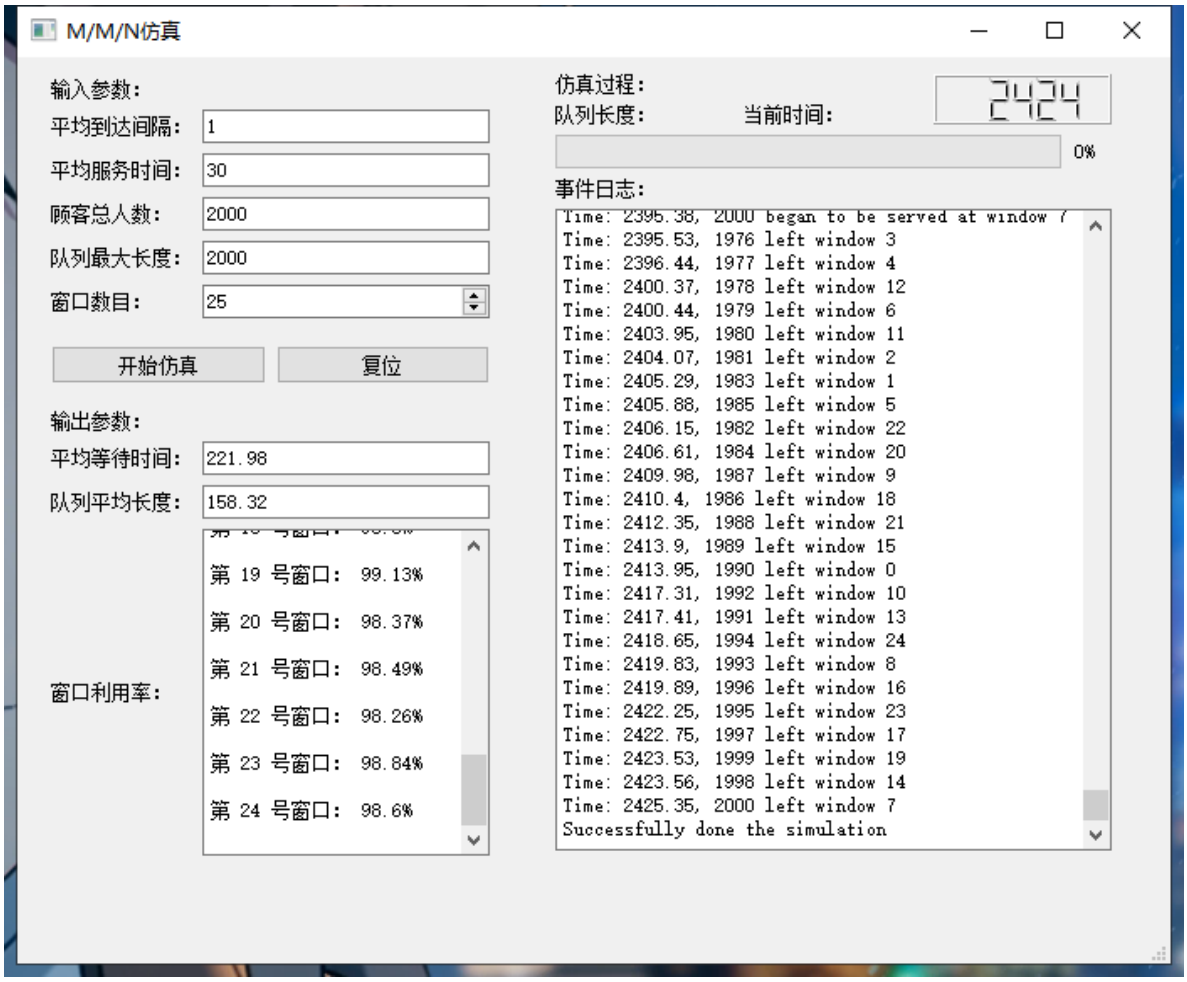

图10 实际场景仿真的一次结果

进行多次仿真，取结果均值如下所示：

仿真结果				仿真结果均值		
仿真总耗时	平均等待时间	平均队列长度	窗口平均利用率	平均等待时间	平均队列长度	窗口平均利用率
2424	221.98	158.32	98.84%	224.118	160.004	98.88%
2421	236.59	170.58	98.91%			
2422	228.06	163.41	98.86%			
2425	205.48	144.61	98.97%			
2432	228.48	163.1	98.84%			

则有平均等待时间 $t = 224.118s = 3.73min$ ，平均窗口队列长度 $l = 160.004/25 = 6.4$人 ，此数据符合实际场景的一般感受。

假设顾客平均到达时间间隔不变，略微修改其他参数，做仿真结果如下所示：

输入参数		仿真结果	
窗口数量	平均服务时间	平均等待时间	平均队列长度
20	30	583.02	365.52
30	30	48.61	18.22
25	25	51.78	26.39
25	35	423.75	275.29

4.3 改进措施

由上述仿真结果可以看出，增大窗口数量、缩短平均服务时间均可以减小平均等待时间与平均队列长度，且仿真结果对于窗口数量的变化更加敏感，因此提出若干改进措施如下：

1. 适当增加窗口数量，可以增设潮汐窗口，即该窗口仅在高峰时期开放，可以节省部分人力成本。
2. 缩短平均服务时间，可以考虑优化窗口菜品摆放结构，或是采用流水线供给菜品。
3. 扩大平均到达间隔，可以考虑增加线上点单，到食堂自取的新方式，将点单高峰期开始时刻提前，缓解窗口压力。

需要指出的是，由于食堂多楼层、多窗口、多队列的特性以及队伍长度会影响新到达顾客的选择（排队或离开）的因素，上述仿真与分析不一定能契合真实场景。若需要提出具体的改进措施，则需要建立更为精细的模型，综合实地考察等多种方式进行建模，才能达到更好的效果。

附录1：仿真程序的后端源代码

代码已开源，链接 https://github.com/gottfriede/M-M-N-simulation

```
import queue
import numpy
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
from functools import partial

import mmnUi

class Event :
    def __init__(self, newCustomerId, newWindowId, newEventType, newOccurTime) :
        self.customerId = newCustomerId
        self.windowId = newWindowId
        self.eventType = newEventType
        self.occurTime = newOccurTime

    def __lt__(self, other) :
        if (self.occurTime == other.occurTime) :
            return self.customerId < other.customerId
        else :
            return self.occurTime < other.occurTime

    def getCustomerId(self) :
        return self.customerId

    def getWindowId(self) :
        return self.windowId

    def getEventType(self) :
        return self.eventType

    def getOccurTime(self) :
        return self.occurTime

    def output(self) :
        sentence = ""
        sentence = sentence + "Time: " + str(round(self.occurTime, 2)) + ", "
        if (self.eventType == 1) :
            sentence = sentence + str(self.customerId) + " arrived"
        if (self.eventType == 2) :
            sentence = sentence + str(self.customerId) + " began to be served at window " + str(self.windowId)
        if (self.eventType == 3) :
            sentence = sentence + str(self.customerId) + " left window " + str(self.windowId)
        ui.textBrowser.append(sentence)

averageArriveTime = 0
averageServeTime = 0
```

```
customerNumber = 0
queueMaxLength = 0
windowNumber = 0

eventList = queue.PriorityQueue()
waitingQueue = queue.Queue(queueMaxLength)
windows = []
arriveTimeList = []
leaveTimeList = []
serveBeginTimeList = []
serveTimeList = []
lastTime = 0
totalTime = 0
queueArea = 0

def init() :
    for i in range(windowNumber) :
        windows.append(0)
        serveBeginTimeList.append(0)
        serveTimeList.append(0)
    produceCustomers(averageArriveTime, customerNumber)

def produceCustomers(averageArriveTime, number) :
    tmpTime = 0
    tmpCustomerId = 0
    for i in range(customerNumber) :
        arriveInterval = numpy.random.exponential(averageArriveTime)
        tmpTime = tmpTime + arriveInterval
        tmpCustomerId = tmpCustomerId + 1
        eventList.put(Event(tmpCustomerId, -1, 1, tmpTime))
        arriveTimeList.append(tmpTime)

def serveTime() :
    normal = 0
    while (normal <= 0) :
        normal = numpy.random.normal() + averageServeTime
    return normal

def getWaitingWindow() :
    for i in range(len(windows)) :
        if (windows[i] == 0) :
            return i
    return -1

def simulate() :
    global lastTime, totalTime, queueArea
    while (not eventList.empty()) :
        flush()
        nextEvent = eventList.get()
        lastTime = totalTime
        totalTime = nextEvent.getOccurTime()
        queueArea = queueArea + waitingQueue.qsize() * (totalTime -
lastTime)
        nextEvent.output()
        if (nextEvent.getEventType() == 1) :
            tmpCustomerId = nextEvent.getCustomerId()
            tmpTime = nextEvent.getOccurTime()
            window = getWaitingWindow()
```

```
            if (window == -1) :
                if (waitingQueue.qsize() >= queueMaxLength) :
                    ui.textBrowser.append("Warning: The queue has reached the maximum number of people.\nThe simulation has stopped")
                    return
                waitingQueue.put(tmpCustomerId)
            else :
                eventList.put(Event(tmpCustomerId, window, 2, tmpTime))
        elif (nextEvent.getEventType() == 2) :
            tmpCustomerId = nextEvent.getCustomerId()
            tmpTime = nextEvent.getOccurTime()
            tmpWindow = nextEvent.getWindowId()
            windows[tmpWindow] = 1
            serveBeginTimeList[tmpWindow] = tmpTime
            eventList.put(Event(tmpCustomerId, tmpWindow, 3, tmpTime+serveTime()))
        elif (nextEvent.getEventType() == 3) :
            tmpWindow = nextEvent.getWindowId()
            tmpTime = nextEvent.getOccurTime()
            leaveTimeList.append(nextEvent.getOccurTime())
            serveTimeList[tmpWindow] = serveTimeList[tmpWindow] + tmpTime - serveBeginTimeList[tmpWindow]
            if (waitingQueue.empty()) :
                windows[tmpWindow] = 0
            else :
                tmpCustomerId = waitingQueue.get()
                eventList.put(Event(tmpCustomerId, tmpWindow, 2, tmpTime))
    ui.textBrowser.append("Successfully done the simulation")

def flush() :
    ui.lcdNumber.display(round(totalTime, 2))
    ui.progressBar.setValue(waitingQueue.qsize())
    QApplication.processEvents()

def output() :
    ans = 0
    if (len(leaveTimeList) != 0) :
        for i in range(len(leaveTimeList)) :
            ans += leaveTimeList[i] - arriveTimeList[i]
        ans = ans / len(leaveTimeList)
    ui.lineEdit_5.setText(str(round(ans, 2)))
    ans = []
    for i in range(len(windows)) :
        ui.textBrowser_2.append("第 " + str(i) + " 号窗口:  " + str(round(serveTimeList[i] / totalTime * 100, 2)) + "%\n")
    ans = queueArea / totalTime
    ui.lineEdit_6.setText(str(round(ans, 2)))

def convert(ui) :
    global averageArriveTime, averageServeTime, customerNumber, queueMaxLength, windowNumber
    averageArriveTime = int(ui.lineEdit_3.text())
    averageServeTime = int(ui.lineEdit_4.text())
    customerNumber = int(ui.lineEdit_2.text())
    queueMaxLength = int(ui.lineEdit.text())
    windowNumber = int(ui.spinBox.text())
    ui.progressBar.setMaximum(queueMaxLength)
    init()
```

```
157     simulate()
158     output()
159 
160 def reset(ui) :
161     ui.lineEdit_5.setText("")
162     ui.lineEdit_6.setText("")
163     ui.textBrowser_2.setText("")
164     ui.textBrowser.setText("")
165     ui.lcdNumber.display(0)
166     windows.clear()
167     arriveTimeList.clear()
168     leaveTimeList.clear()
169     serveBeginTimeList.clear()
170     serveTimeList.clear()
171     lastTime = 0
172     totalTime = 0
173     queueArea = 0
174 
175 if __name__ == '__main__' :
176     app = QApplication(sys.argv)
177     MainWindow = QMainWindow()
178     ui = mmnUi.Ui_MainWindow()
179     ui.setupUi(MainWindow)
180     MainWindow.show()
181     ui.pushButton.clicked.connect(partial(convert, ui))
182     ui.pushButton_3.clicked.connect(partial(reset, ui))
183     sys.exit(app.exec_())
184 
```

附表2：千次仿真的具体结果

限于纸张，未将此表放入纸质版中，详见电子版文件 `附表2-千次仿真具体结果.xlsx`