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	18373528
姓名	杨凌华

2020 年 10 月

队列模型实验报告

一、 实验目的

应用 M/M/1 队列编程思想，模拟服务器服务顾客队列的过程，熟悉离散事件推进方式、队列建立和提取方式。

二、 数学模型（同学们需要写出主要用到的概率模型，事件推进的方法）

1、首先确定模型框架，即核心是创建一根事件轴和一支队列。先判定事件轴是否忙碌，是就根据时间先后顺序让顾客进入队列，否则推进事件。

事件轴：根据顾客总数 total、顾客平均到达时间、顾客平均服务时间生成 customer 序列，每个 customer 包括到达时间和服务开始时间两个成员变量，作为时间轴。

队列：创建一个 Queue<Customer>队列，用于记录等待被服务的顾客队列。

推进方法：采用时间异步模型，以事件（顾客的到达）作为事件推进的依据，变量 curTime 记录当前时间，直到下一事件发生时才对该变量进行更新，同时完成顾客的服务和离开，出队和入队。只有当顾客到达以及单个顾客服务结束时才对其进行更新，跳过了没有事件发生的时间段，提高了程序模拟仿真运行的效率。

2、扩充细节，比如：顾客到达间隔时间按指数分布生成；显示事件推进节点。

1）顾客到达时间间隔按指数分布生成：

指数分布，是描述泊松过程中的事件之间的时间的概率分布，即事件以恒定平均速率连续且独立地发生的过程。

其概率分布为 $F(x) = 1 - e^{-\lambda x}$，数学期望为 $1 / \lambda$，代表该随机分布的平均值。

因此顾客平均到达时间间隔为 $1 / \lambda$，此处设 x 为某次到达时间间隔，设 u 为其发生的概率，则有

$$u = 1 - e^{-\lambda x}$$

$$e^{-\lambda x} = 1 - u$$

$$-\lambda x = \ln(1-u)$$

$$x = -\frac{1}{\lambda}\ln(1-u)$$

因为概率 u ∈ [0, 1]，所以由对称性 1 - u ∈ [0, 1]，故有

$$x = -\frac{1}{\lambda}\ln(u),\ u \in [0, 1]$$

因此可以利用随机函数产生 u，进而求得相应分布的时间间隔 x，代码如下：

```
private double generateTime() {
    double u = Math.random();
    return (- Math.log(u) / lambda);
}
```

2）顾客服务时间的生成完全同上。

3）事件推进节点显示：

i）输入相关参数

```
请输入[平均到达时间]：
5
请输入[平均服务时间]：
10
请输入[顾客数目]：
500
请输入[队列最大长度]：
20
```

ii）对于每一个事件节点，显示相应的时间和具体的事件详情：

```
----------------------------------------事件流数据----------------------------------------
[1.41]: 顾客 1 到达了
[1.41]: 窗口 1 开始了 对顾客 1 的服务
[8.56]: 顾客 2 到达了
[6.70]: 窗口 1 结束了 对顾客 1 的服务
[8.56]: 窗口 1 开始了 对顾客 2 的服务
[36.89]: 顾客 3 到达了
[8.63]: 窗口 1 结束了 对顾客 2 的服务
[36.89]: 窗口 1 开始了 对顾客 3 的服务
[37.75]: 顾客 4 到达了
```

iii）对于每一个顾客，显示相应的顾客时间节点数据：

iv）最终统计数据显示

3、简述事件调度法、活动扫描法和进程交互法的异同。

事件调度法 Event Scheduling：关注事件以及它们如何影响系统状态。记录事件发生的过程，处理每个事件发生时系统状态变化的结果。

活动扫描法 Activity Scanning：关注活动（每一活动对应开始时间和结束时间），既使用调度时间也使用条件来选择事件列表中的下一事件。系统由成分组成，而成分包含着活动，这些活动的发生必须满足某些条件；每一个主动成分有一个相应的活动子例程；在仿真过程中，活动的发生时间也作为条件之一，而且是较之其他条件具有更高的优先权。

进程交互法 Process Interaction：关注实体以及它所经历的事件/活动序列。最为直观，但实现很复杂。采用进程描述系统，进程表按照时间顺序组织，将模型中的主动成分历经系统时所发生的事件进行组合。

三种方法的相同点：都是事件按时间顺序发生（即按仿真时钟推进）

三、 编程实现与调试过程（需要给出代码实现的主要函数及其对应的数学模型）

1、可以主要分析一下到达和离开的事件处理流程。

首先介绍几个主要的实体类：

```
public class Customer {
    public static double lambda;//用于生成到达时间的λ
    private static int cntId = 0;//顾客编号静态累加器
    private int id;//顾客编号
    private double arriveTime;//到达时间
    private double serveBeginTime;//开始服务时间
    private double finishTime;//服务结束时间

    public Customer(double lastTime) {
        cntId++;
        id = cntId;
        arriveTime = lastTime + generateTime();
    }
    ...
```

Customer 类封装了顾客 ID、到达时间、服务开始时间、服务结束时间，其中 arriveTime 通过 lambda 指数分布随机生成并累加而来。

```
public class Server {
    public static double lambda;//用于生成服务时间的λ
    private boolean busy = false;//当前是否工作
    private Customer curCustomer;//当前服务的顾客
    private double freeTime = 0.0;//下一个空闲时间点
    ...
```

Server 类封装了服务器当前工作状态、当前服务的顾客、下一个空闲时间点，其中 freeTime 是通过将当前服务顾客的开始服务时间点与通过指数分布随机生成的服务时间相加而得来。

顾客到达事件流程分析：

```
curTime = curCustomer.getArriveTime();
if (curTime > maxTime) {
    break;
}
//可能这个顾客来的比较晚，在其来之前已经服务了多名队列中的顾客，因此loop
while (true) {
    double dt = server.update(curTime);//服务完了则为服务时间，否则为0
    Statisticer.sumQueS += dt * queue.size();//若服务完了，则累加到队列面积上
    if (server.isBusy() || queue.isEmpty()) {
        break;
    }
    double nextStartTime = server.getFreeTime();
    server.serve(queue.poll(), nextStartTime);
}
```

当一个顾客到达时，就设定当前时间为顾客的到达时间，如果当前时间大于最大仿真时间的阈值，就退出结束仿真，然后进入循环，在循环中将队列中当前时间以前能够进行服务的顾客依次进行服务，当服务器正忙或者队列为空时就退出循环，进一步处理当前新来的顾客。

```java
//说明服务器忙，需要将新来的顾客加入队列
if (server.isBusy()) {
    //说明队列长度过大，顾客无法忍受，溜了
    if (queue.size() >= maxQue) {
        System.out.printf("[%.2f]", curTime);
        System.out.println(": 队列人数已满，顾客 " + curCustomer.getId() + " 离开了");
        lost++;
    }
    //队列长度在顾客的容忍范围之内，顾客进入队列
    else {
        queue.add(curCustomer);
        System.out.printf("[%.2f]", curTime);
        System.out.println(": 顾客 " + curCustomer.getId() + " 进入了队列");
    }
}
//说明队列为空，可以直接服务这个新来的顾客
else {
    server.serve(curCustomer, curTime);
}
```

对于新来的顾客，如果此时服务器正忙，则判断队列长度是否达到了 maxQue，若达到了则顾客会无法忍受，进而选择离开，若没有达到则顾客加入队列并等待。如果服务器空闲，则调用服务器的 serve 函数设定当前顾客为当前开始服务对象。

```java
//开始服务
public void serve(Customer customer, double curTime) {
    customer.setServeBeginTime(curTime);//设置顾客的服务开始时间
    Statisticer.sumWaitTime += curTime - customer.getArriveTime(); //累加总等待时间
    busy = true;//设置服务器为忙
    curCustomer = customer;//设置当前顾客
    freeTime = curTime + generateTime();
    customer.setFinishTime(freeTime);//设置当前顾客服务结束时间
    System.out.printf("[%.2f]", curTime);//输出
    System.out.println(": 窗口 " + id + " 开始了 对顾客 " + curCustomer.getId() + " 的服务");
}
```

serve 函数的主要操作就是处理当前时间节点并保存至相关记录变量，然后将服务器状态设置为正忙，最后输出当前事件相关信息。

顾客离开事件流程分析：

```java
//更新服务器的状态，判断是否服务完上一个顾客
public double update(double curTime) {
    double ret = 0;
    //若为真，则说明此时已经服务完上一个
    if (busy && freeTime < curTime) {
        busy = false;//将服务器状态设置为空闲
        double dtime = freeTime - curCustomer.getServeBeginTime();// 服务上一个顾客用时
        Statisticer.sumExcuTime += freeTime - curCustomer.getArriveTime();//顾客的总执行时间累加
        Statisticer.sumSerS += dtime;//服务器的总服务时间累加
        ret = dtime;
        Statisticer.cntCustomer++;//服务完的顾客+1
        System.out.printf("[%.2f]", freeTime);
        System.out.println(": 窗口 " + id + " 结束了 对顾客 " + curCustomer.getId() + " 的服务");
        curCustomer = null;
    }
    //若当前没有服务完上一个顾客，则返回0；若已经服务完了，则返回该顾客的服务时间
    return ret;
}
```

与 serve 函数相对应的，是 update 函数，它用于处理顾客结束服务并离开的操作。首先判断上一个服务的顾客是否完成了其需要的服务，也就是说当前时间是否达到了上一顾客的服务完成时间点（即 curTime > freeTime），如果达到了，则将服务器置为空闲，处理时间节点并更新相关记录统计变量，最后输出当前时间点的事件相关信息，并返回上一个顾客的净服务时长。如果当前服务器没有服务完上一个顾客，则返回 0。

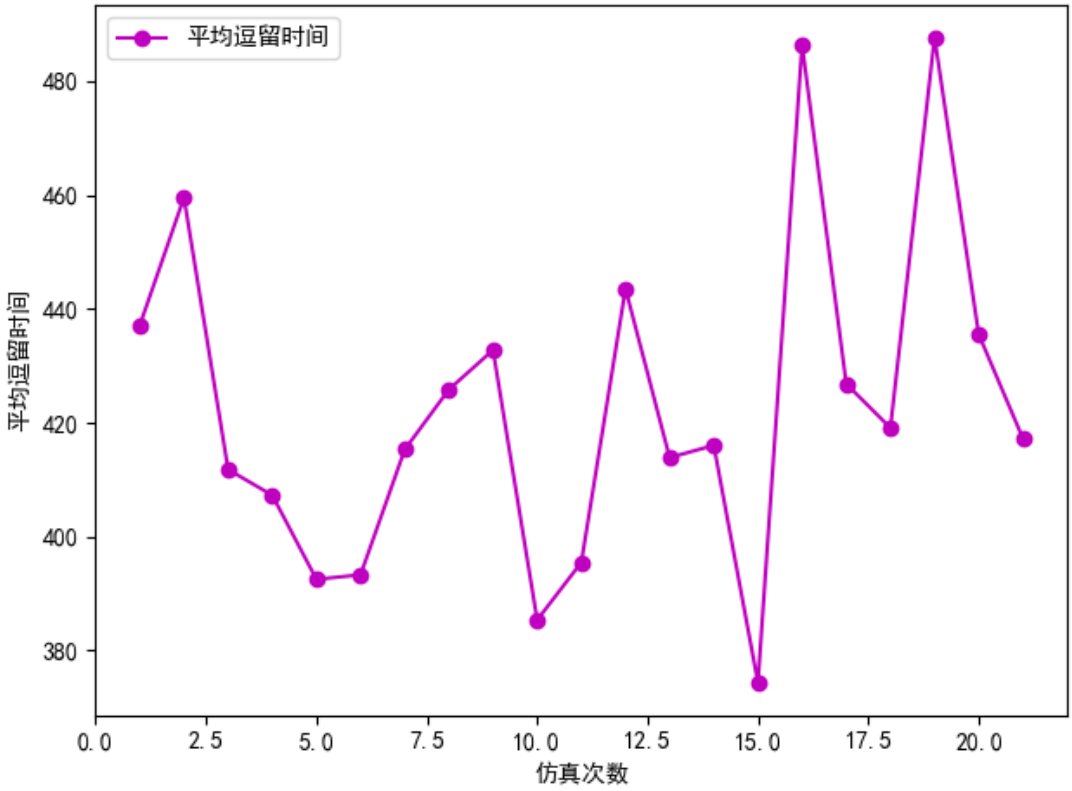
```java
//此时不会再有新顾客进来了，因此进入循环将队列中将现有的顾客全部服务完
while (!queue.isEmpty() || server.isBusy()) {
    double v = server.update(Double.POSITIVE_INFINITY);//无限等当前顾客服务完
    Statisticer.sumQueS += v * queue.size();
    curTime = server.getFreeTime();
    if (!queue.isEmpty()) {
        server.serve(queue.poll(), server.getFreeTime());
    }
}
```

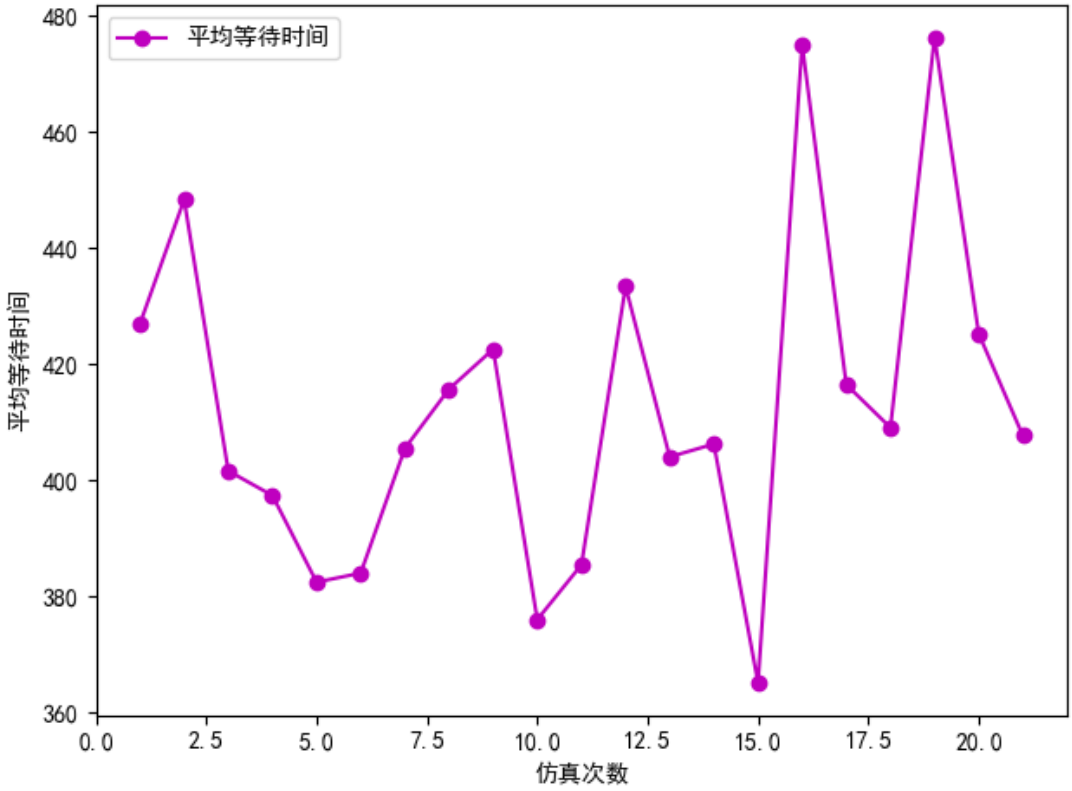
在依次处理完顾客序列中所有新来的顾客的到达时间之后，此时队列中可能还有顾客在等待，只是这个队列不会再继续增加了，因此直接依次按顺序服务完队列中剩下的顾客就 ok 了。

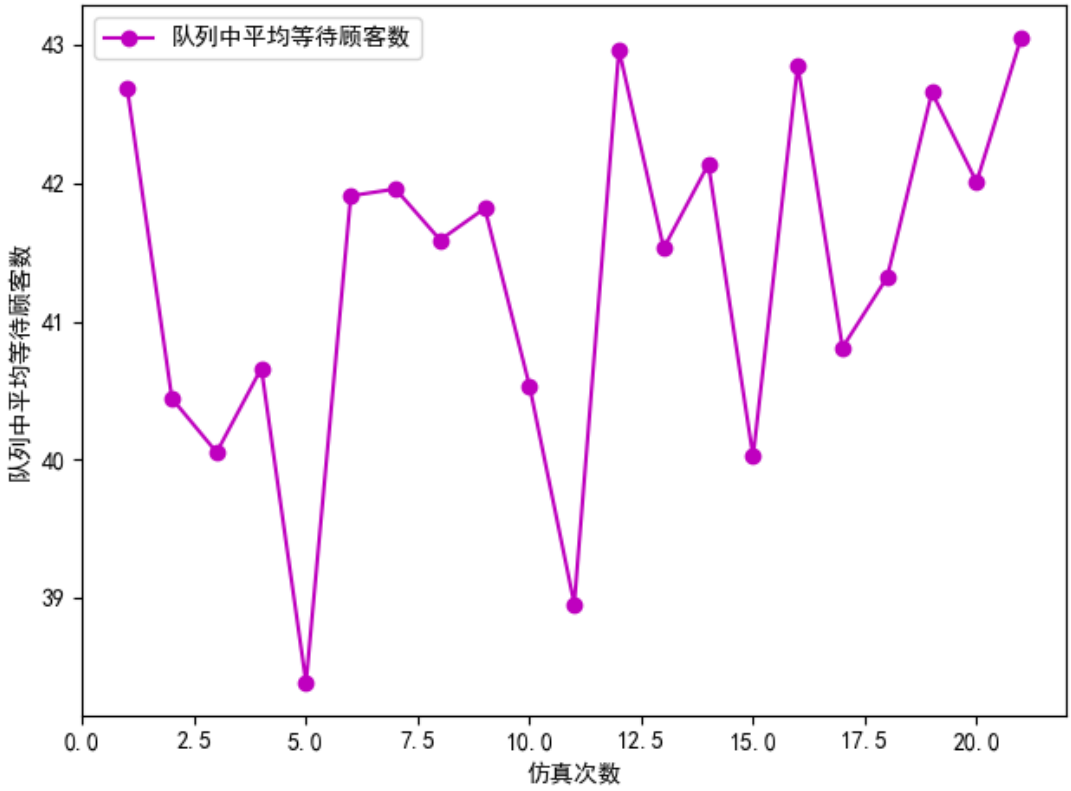
2、给出输入参数（平均到达时间，平均服务时间，顾客数目，队列最大长度）、输出参数（平均逗留时间；平均等待时间；队列中平均等待客户数；服务器利用率；顾客流失率）的具体值，可以用图或表的方式展示多次仿真的结果值。

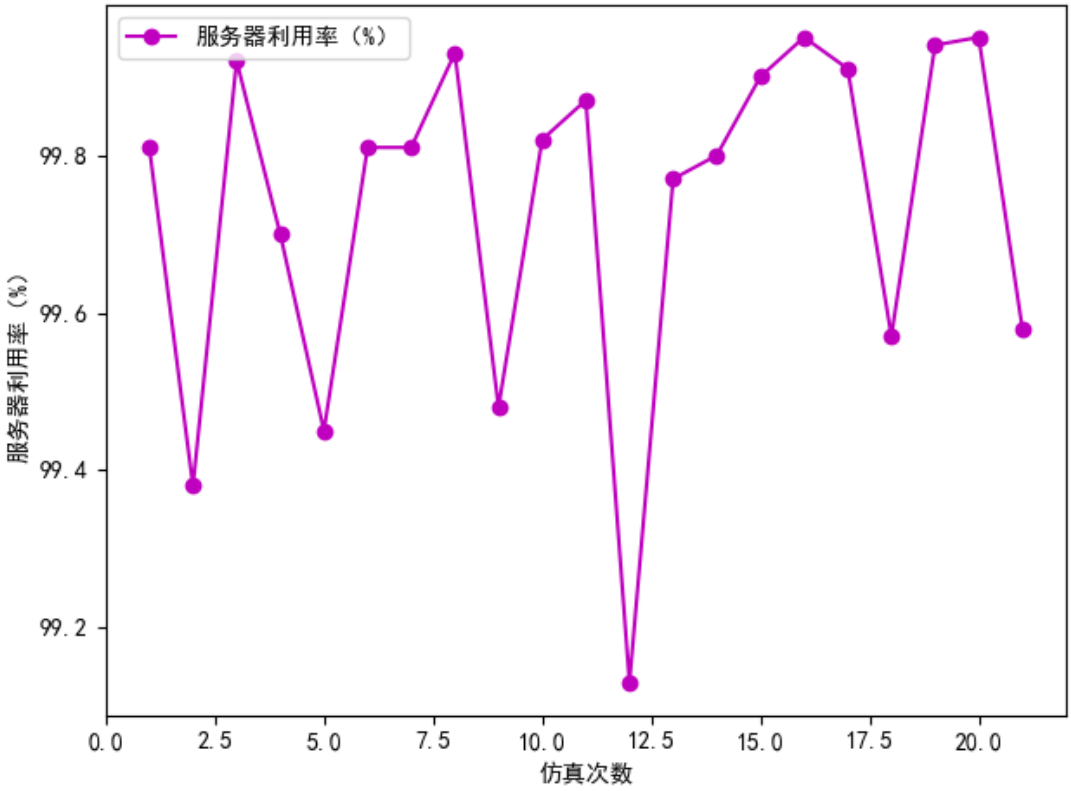
仿真一：模拟服务器压力较大的场景

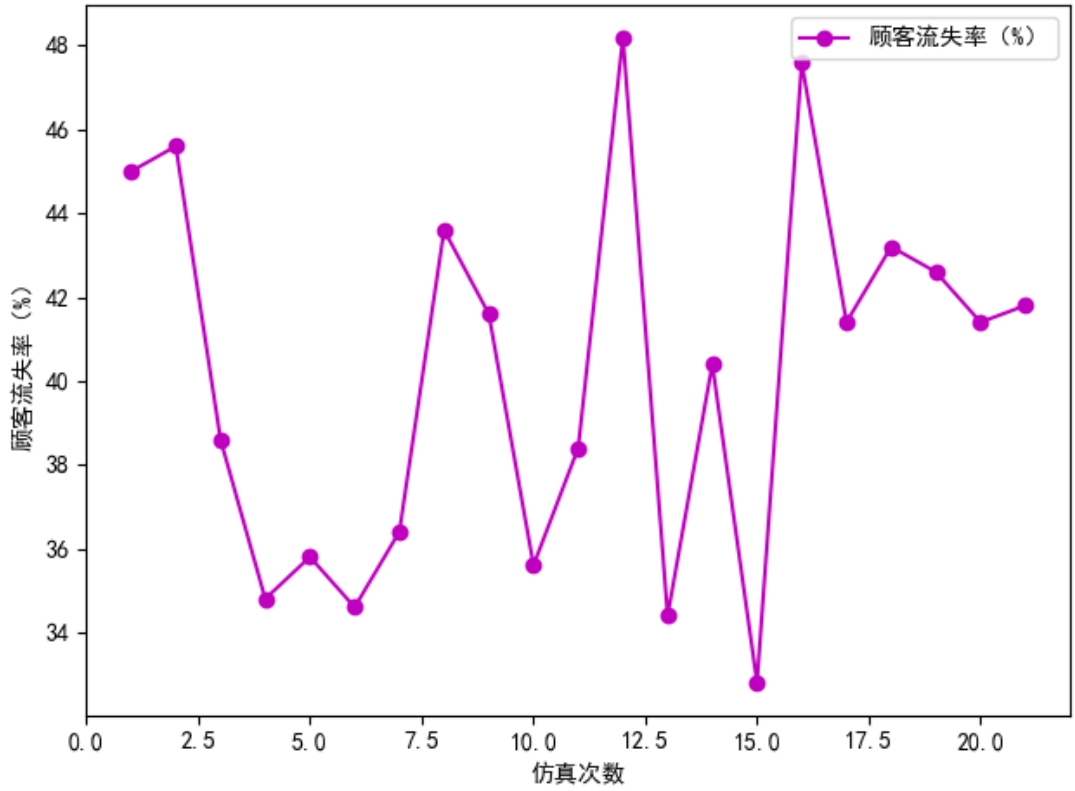
平均到达时间：5，平均服务时间：10，顾客数目：500，队列最大长度：50


平均逗留时间
480
460
440
420
400
380
平均逗留时间
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数


平均等待时间
480
460
440
420
400
380
360
平均等待时间
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数

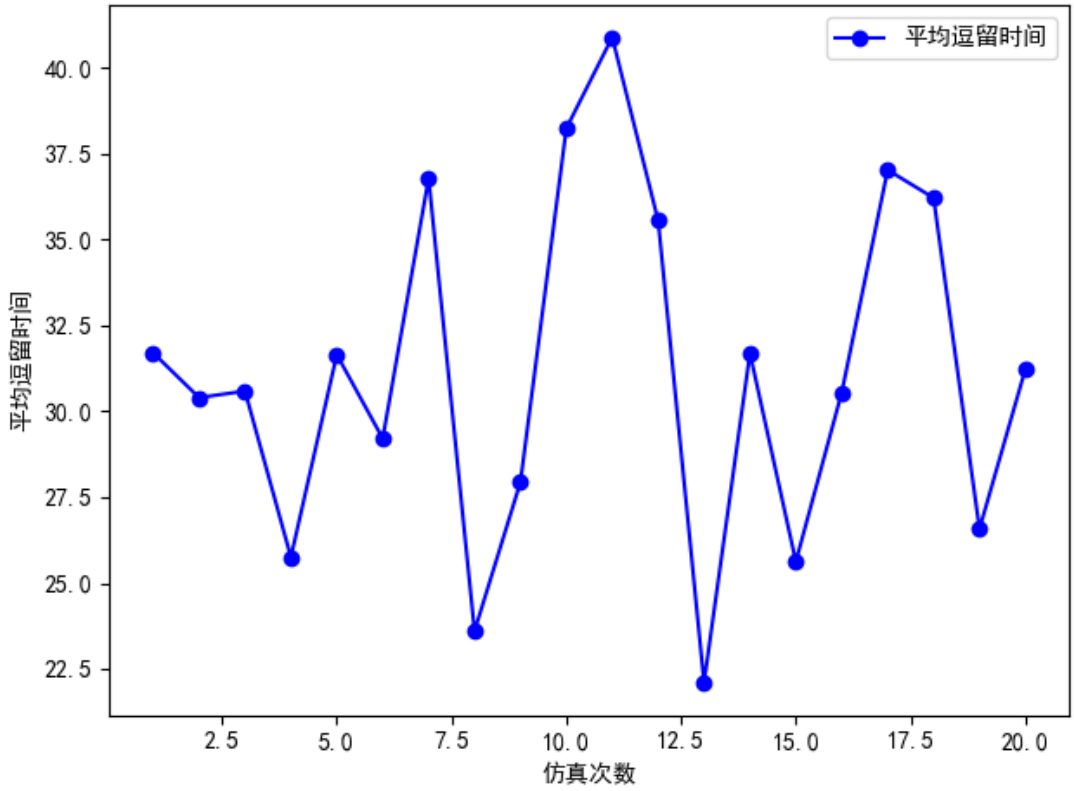

队列中平均等待顾客数
43
42
41
40
39
队列中平均等待顾客数
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数

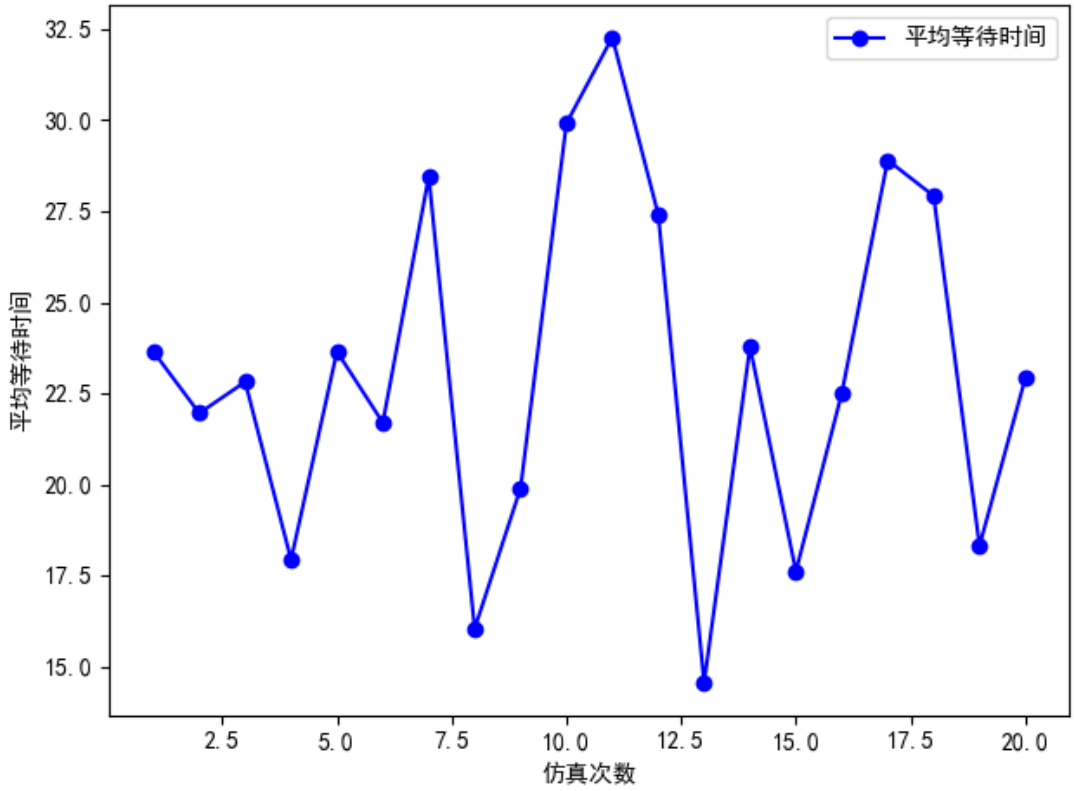

服务器利用率（%）
99.8
99.6
99.4
99.2
服务器利用率（%）
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数

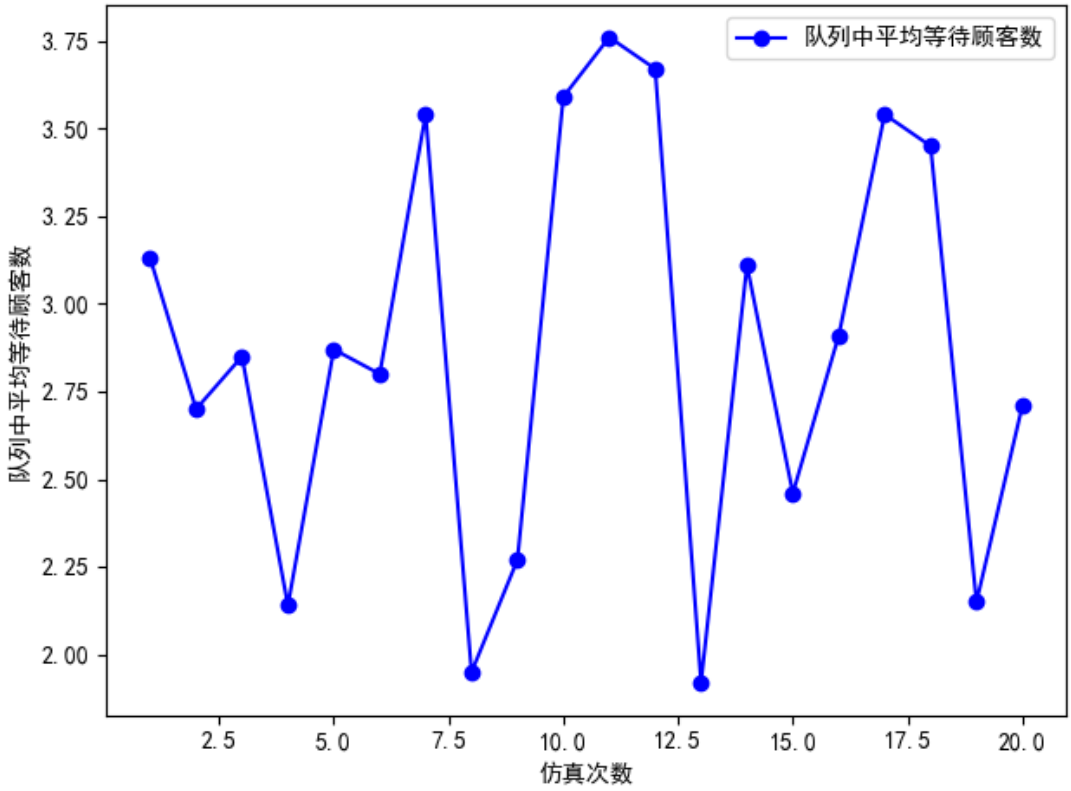


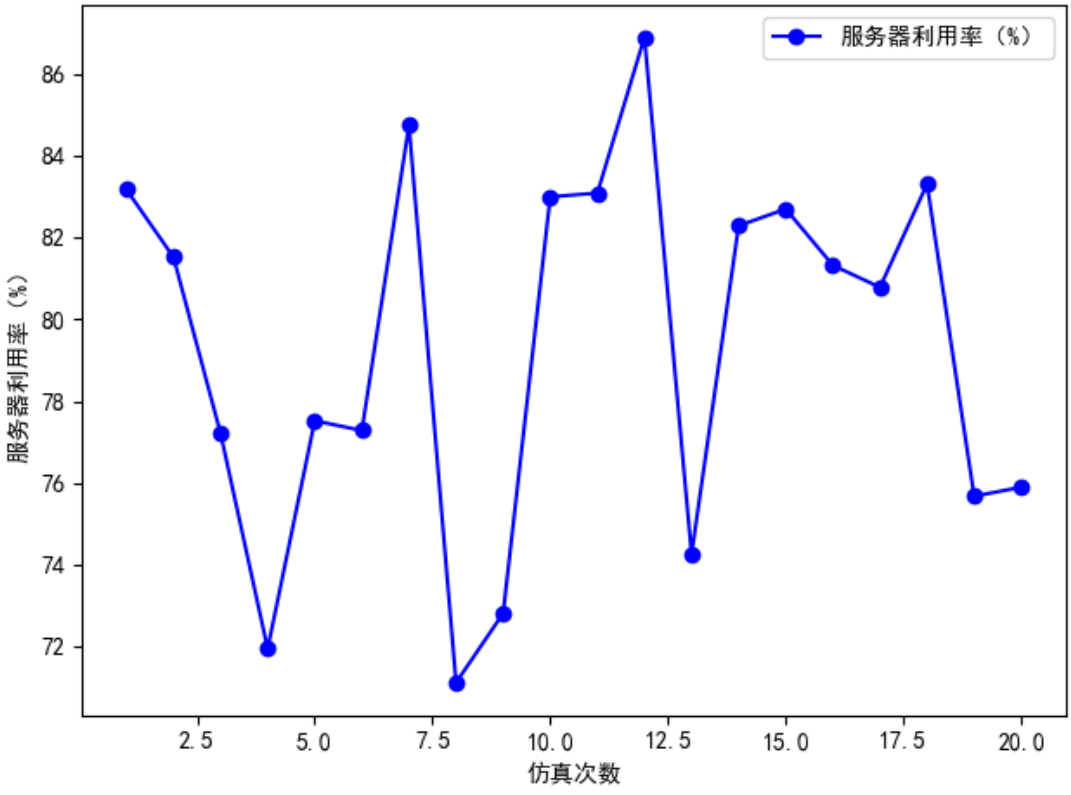
仿真二：模拟服务器压力较小的场景

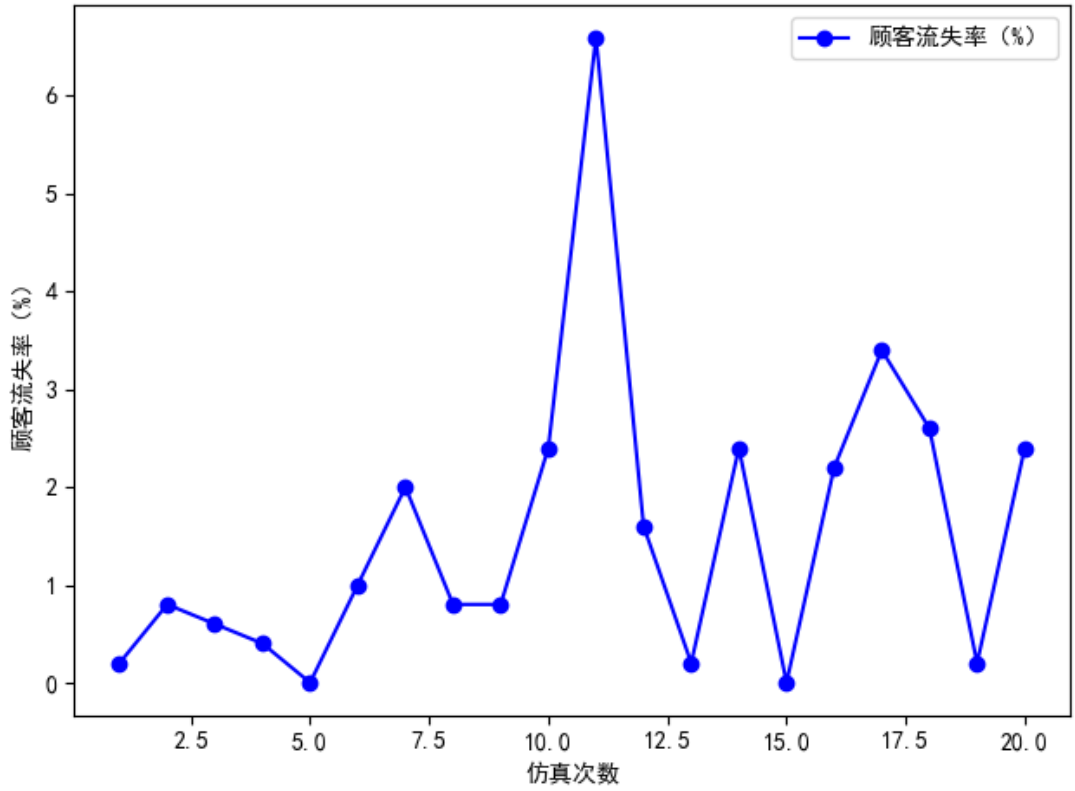
平均到达时间：10，平均服务时间：8，顾客数目：500，队列最大长度：10


平均逗留时间
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
平均逗留时间
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数


平均等待时间
32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0
平均等待时间
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数


队列中平均等待顾客数
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
队列中平均等待顾客数
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数


服务器利用率（%）
86
84
82
80
78
76
74
72
服务器利用率（%）
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
仿真次数



3、选做*：编程模拟或纯数学分析 M/M/n 模型。

(i)模型介绍与编程模拟

MMN 模型是 MM1 模型的拓展，主要区别在于服务器的数目增多，而每一个服务器内部的运行机制是一样的，就有点像上学期 OO 课的电梯调度，我在主函数里写一个调度算法，将新来的顾客加入到一个最优的窗口进行排队。这就势必要给所有窗口按照一定的标准进行排序，我采用的是队列长度最小的窗口具有最高优先级，对在 Server 类中实现 Comparable 接口，并覆写 compareTo 方法，然后利用 java 的 PriorityQueue 有序优先队列容器来存储每一个 Server，compareTo 函数的内容如下：

```java
@Override
public int compareTo(Server o) {
    if (this.getQueueLen() != o.getQueueLen())
        return Integer.compare(this.getQueueLen(), o.getQueueLen());

    if (!this.busy && !o.busy) {
        return Integer.compare(this.id, o.id);
    } else if (!this.busy && o.busy) {
        return -1;
    } else if (this.busy && !o.busy) {
        return 1;
    } else {
        int k = Double.compare(this.freeTime, o.freeTime);
        return k != 0 ? k : Integer.compare(this.id, o.id);
    }
}
```

大致思路就是：如果两个窗口队列长度不一样，则以队列短的为优先；如果队列长度一样，则以空闲的为优先，如果都空闲，则以服务窗口的 ID 递增顺序进行顾客分配。

因为 java 的 PriorityQueue 有序优先队列容器会自动将最优的窗口筛选至队列开头，因此每当新的顾客到达时，就取出队列开头的 server 并将顾客交给其进行服务，并修改 server 的相关状态，然后再加入 java 的 PriorityQueue 有序优先队列容器中，继续进行排序，以此直到所有顾客都到达完毕，每个服务器的内部实现细节基本和 MM1 一致，只是新增了相关统计数据的记录。

最终的实现效果如下：

```
请输入[平均到达时间]：
5
请输入[平均服务时间]：
100
请输入[顾客数目]：
500
请输入[队列最大长度]：
50
请输入[服务器数目]：
5
```

```
----------------------------------------服务器服务状况----------------------------------------
[服务器1] 利用率: 92.75%, 队列平均长度: 27.08, 共服务了 75 个顾客，分别是：
1 8 14 19 24 29 30 34 37 44 49 54 61 67 77 82 87 92 98 103 111 116 121 126 127 129 134 139 144 146 151 156 161 166 167 172 17
 240 246 253 259 264 272 277 282 287 297 302 308 313 322 327 345 389 413 418 421 435 481 489 493 494 495

[服务器2] 利用率: 99.76%, 队列平均长度: 27.40, 共服务了 74 个顾客，分别是：
2 6 11 13 18 23 26 32 39 42 47 50 55 58 64 69 70 76 81 86 91 96 101 105 114 120 125 132 137 142 148 153 158 164 170 175 181 1
  250 255 260 266 267 268 273 278 283 288 296 301 306 311 316 318 319 324 329 342 384 388 409 443

[服务器3] 利用率: 83.38%, 队列平均长度: 22.63, 共服务了 74 个顾客，分别是：
3 9 15 20 27 33 40 41 45 51 56 57 62 63 68 75 80 85 90 95 100 106 107 108 110 115 117 122 128 133 138 143 149 154 159 165 171
 242 243 248 252 257 258 263 265 269 274 279 284 289 290 291 298 303 309 314 323 328 383 423 473 487

[服务器4] 利用率: 89.50%, 队列平均长度: 27.24, 共服务了 68 个顾客，分别是：
4 7 12 17 22 25 31 36 43 48 53 60 66 72 74 79 84 89 94 97 102 109 113 119 124 131 136 141 147 152 157 163 169 174 180 186 191
 271 276 281 286 295 300 305 310 315 317 321 326 332 355 368 372 381 392 488

[服务器5] 利用率: 88.94%, 队列平均长度: 25.40, 共服务了 79 个顾客，分别是：
5 10 16 21 28 35 38 46 52 59 65 71 73 78 83 88 93 99 104 112 118 123 130 135 140 145 150 155 160 162 168 173 178 179 184 185
 249 251 256 261 270 275 280 285 292 293 294 299 304 307 312 320 325 331 337 341 370 373 375 385 426 454 475 490 491 492 497
```

```
----------------------------------------统计数据--
平均逗留时间： 3030.97
平均等待时间： 2925.63
队列中平均等待客户数： 25.95
服务器平均利用率： 90.87%
顾客流失率： 26.00%
```

四、 选做* 以二食堂（或你熟悉的某个队列场景）为例，给出初步的窗口优化策略

1、该场景的拓扑结构；

学校所有的食堂的窗口拓扑结构基本的都是分布式的，而且都是分散分布，数量还比较庞大，相互之间能够保持一定程度上的独立性，当某一个窗口局部出现故障停止供应之后，并不会对食堂的整体运作造成明显的影响，因此是具有很高可靠性的。

2、人流特点；

用餐高峰是（早餐 7:00-8:00, 午餐 11:00-12:30，晚餐 17:00-19:00），这些时间段人流量会激增，其他时间段人数都很少。

3、改进措施。

①在用餐高峰时间段增开窗口，在人数少时减少窗口的数量，并将主要的人力和物力运用到应对用餐高峰时间段内。

②给每个食堂增加多个分散的入口（当然疫情期间例外，人员需要单向流动），可以将进入食堂的通道增加，从而达到分流引流的效果。