ISE 762 Project

Numerical Simulation of Double Ended Queue

Ruoting Li and Xin Li

Department of Industrial and Systems Engineering
NC State University

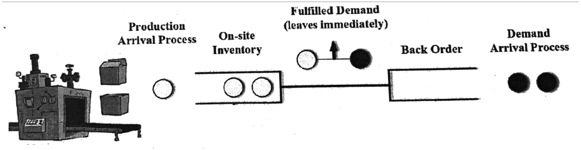
January 2, 2020

Outline

- Background and Motivation
- Main routine
- Numerical Simulation
- Discussions

Background and Motivation

Double Ended Queue (Background)



Orders:

- Order arrive occur according to a Poisson process with rate λ:
- Each arrival is fulfilled immediately if there are available products; otherwise, it will be backlogged, and wait for future fulfillment with FCFS;
- Backlogged orders are impatient with patience times $\sim$ i.i.d. Exp(θ_d)

Products:

- Each product is immediately matched with a backlogged order if there is any; otherwise, it joins the inventory queue and will be later fulfilled with FCFS;
- Products are perishable with lives $\sim$ i.i.d. Exp(θ_p)
- Products are produced according to Poisson process with rate
  - μ_H if the number of backlogged orders is $\geq$ a threshold $Q_d > 0$
  - μ_L if the number of inventory is $\geq$ a threshold $Q_d > 0$

Production system: A Double-Ended Queue Model

- Non-idling constraint: There cannot be positive inventories and backlogged orders simultaneously in the system.
- Variables:
  - Time variable t.
  - System state variable: (n, p)
    - ⋆ $n =$ queue length
      ($n^+ = max\{n, 0\}$ is on-hand inventory, $n^- = -min\{n, 0\}$ is No. of backlogs);
    - ⋆ $p \in \{L, H\}$ is production status
- Counter Variable:
  - $Q^+ =$ cumulative inventory level;
  - $Q^- =$ cumulative backlog level;
  - $N^+ =$ tot. No. of inventory wastage;
  - $N^- =$ tot. No. of order cancellations;
  - $N^m =$ tot. No. of matchings.
- Events:
  - a product is produced / perished;
  - an order is arrived / cancelled.
- Event list: $EL = \{t_{pa}, t_{pp}, t_{oa}, t_{oc}\}$

Main routine

Main routine

Initialize Simulation:

- set $t = 0$
- set $n = 0$ (initial queue length, no inventory or back orders)
- set $\lambda_{pa} = 2$ (initially at low production rate status; high rate = 10)
- set $\lambda_{oa} = 6$
- set perish_list = [] (perish time of product in inventory queue, when $n > 0$
- set patience_list = [] (cancellation time of back order in order queue, when $n < 0$
- set n_list = [] (record the queue length at each event time)
- set all_event_time_list = [] (sequentially record all event times)
- set $\lambda\_perish = 0.5$ (**product perishing rate**; set $\lambda\_perish \approx 0$ means extremely long perish time $\approx$ no perish)
- set $\lambda\_cancel = 0.5$ (**order cancellation rate**; set $\lambda\_cancel \approx 0$ means extremely long cancellation time $\approx$ no cancellation)
- set $a_{switch} = 10$ (a threshold number controlling when to switch production rate; whenever $n \geqslant a$, switch to $\lambda_{pa} = 2$; whenever $n \leqslant -a$, switch to $\lambda_{pa} = 10$)

Main routine... continued

Update system states: (There are 6 cases)

Case 1: $t_{pa} = min(EL)$ & $n \geqslant 0$ (an product arrival **sees no back order**)

1. n $\leftarrow$ n+1
2. t1 $\leftarrow$ t_{pa} (t1 records the latest event time)
3. $q_{plus} += (t1 - t) * (n - 1)$ (calculate cumulative inventory level)
4. If $n \geqslant a_{switch}$, then reset $\lambda_{pa} = 2$ (meet the inventory level, then switch to low production rate)
5. Generate $U \sim Unif[0, 1]$ and reset $t_{pa} = t1 - \frac{1}{\lambda_{pa}} \log(U)$ (gen. next product arrival)
6. Generate $U \sim Unif[0, 1]$ and reset $t_{perish} = t_{pa} - \frac{1}{\lambda_{pp}} \log(U)$ (gen. corresponding perish time)
7. perish_list.append(t_{perish}) (update perish list)
8. $EL[0] = t_{pa}, EL[1] = min(perish\_list)$ (update event list)

Main routine... continued

Case 2: $t_{pa} = min(EL)$ & $n < 0$ (an product arrival **sees back order**)

1. n $\leftarrow$ n+1
2. t1 $\leftarrow$ t_{pa} (t1 records the latest event time)
3. n_match $\leftarrow$ n_match+1 (counts an instantly satisfied pair)
4. $q_{minus} -= (t1 - t) * (n - 1)$ (calculate cumulative backorder level)
5. patience_list.pop(0) (delete the first back order's patience time since it's satisfied by an product arrival)
6. If patience_list==[], set EL[3] $= \infty$, else set EL[3] $= min(patience\_list)$
7. Generate $U \sim Unif[0, 1]$ and reset $t_{pa} = t1 - \frac{1}{\lambda_{pa}} \log(U)$ (gen. next product arrival)
8. $EL[0] = t_{pa}$ (update event list)

Main routine... continued

Case 3: $t_{oa} = min(EL)$ & $n \leqslant 0$ (an order arrival **sees no inventory**)

1. n $\leftarrow$ n-1
2. t1 $\leftarrow$ t_{oa} (t1 records the latest event time)
3. $q_{minus} -= (t1 - t) * (n+1)$ (calculate cumulative backorder level)
4. If $n \leqslant -a_{switch}$, then reset $\lambda_{pa} = 10$ (meet the backorder level, then switch to high production rate)
5. Generate $U \sim Unif[0,1]$ and reset $t_{oa} = t1 - \frac{1}{\lambda_{oa}} \log(U)$ (gen. next order arrival)
6. Generate $U \sim Unif[0,1]$ and reset $t_{cancel} = t_{oa} - \frac{1}{\lambda_{oc}} \log(U)$ (gen. corresponding order cancel time)
7. patience_list.append(t_{cancel}) (update patience list)
8. $EL[2] = t_{oa}, EL[3] = min(patience\_list)$ (update event list)

Main routine... continued

Case 4: $t_{oa} = min(EL)$ & $n > 0$ (an order arrival **sees available inventory**)

1. n $\leftarrow$ n-1
2. t1 $\leftarrow$ t_{oa} (t1 records the latest event time)
3. n_match $\leftarrow$ n_match+1 (counts an instantly satisfied pair)
4. $q_{plus} += (t1 - t) * (n + 1)$ (calculate cumulative inventory level)
5. perish_list.pop(0) (delete the first inventory product's perish time)
6. If perish_list==[], set EL[1] = ∞, else set EL[1] = $min(perish\_list)$
7. Generate $U \sim Unif[0, 1]$ and reset $t_{oa} = t1 - \frac{1}{\lambda_{oa}} \log(U)$ (gen. next order arrival)
8. $EL[2] = t_{oa}$ (update event list)

Main routine... continued

Case 5: $t_{pp} = min(EL)$ (**product perish;** happens only when $n > 0$)

1. n $\leftarrow$ n-1
2. t1 $\leftarrow$ t_{pp} (t1 records the latest event time)
3. n_plus $\leftarrow$ n_plus + 1
4. $q_{plus} += (t1 - t) * (n + 1)$ (calculate cumulative inventory level)
5. perish_list.remove(t_{pp}) (delete the perish time of the inventory product just perished)
6. If n==0, then $EL[1] = \infty$, else $EL[1] = min(perish\_list)$ (If after this perishing, inventory becomes empty, then next perish time becomes ∞

Main routine... continued

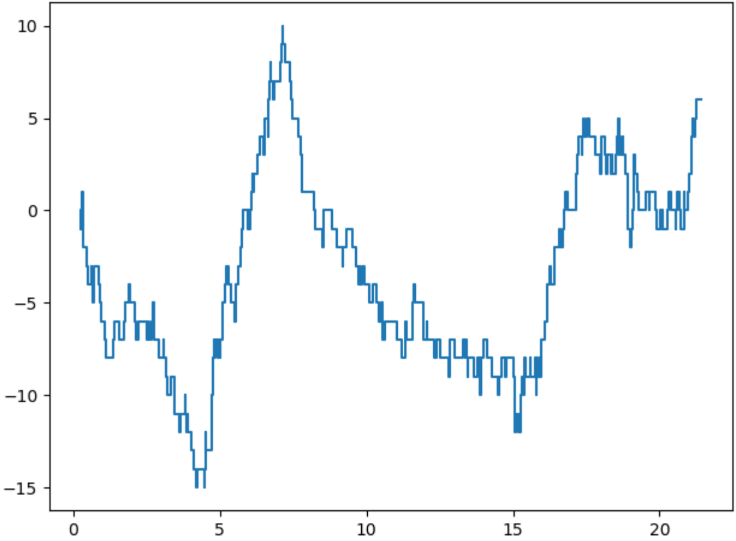
Case 6: $t_{oc} = min(EL)$ (**back order canceled;** happens only when $n < 0$)

1. n $\leftarrow$ n+1
2. t1 $\leftarrow$ t_{oc} (t1 records the latest event time)
3. n_minus $\leftarrow$ n_minus + 1
4. $q_{minus} -= (t1 - t) * (n - 1)$ (calculate cumulative backorder level)
5. patience_list.remove(t_{cancel}) (delete the patience time of the back order just cancelled)
6. If n==0, then $EL[3] = \infty$, else $EL[3] = min(patience\_list)$ (If after this cancellation, back order becomes empty, then next patience time becomes ∞

Numerical simulation

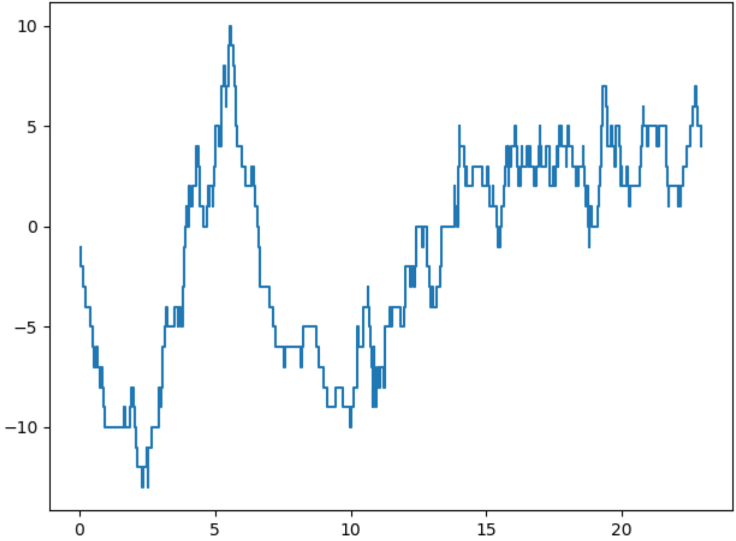
Numerical Simulation (With both abandonment)

$\lambda_{pa} = \{'Low' : 2, 'High' : 10\}$, $\lambda_{oa} = 6$, $\lambda_{perish} = \lambda_{cancel} = 0.5$, $a_{switch} = 10$



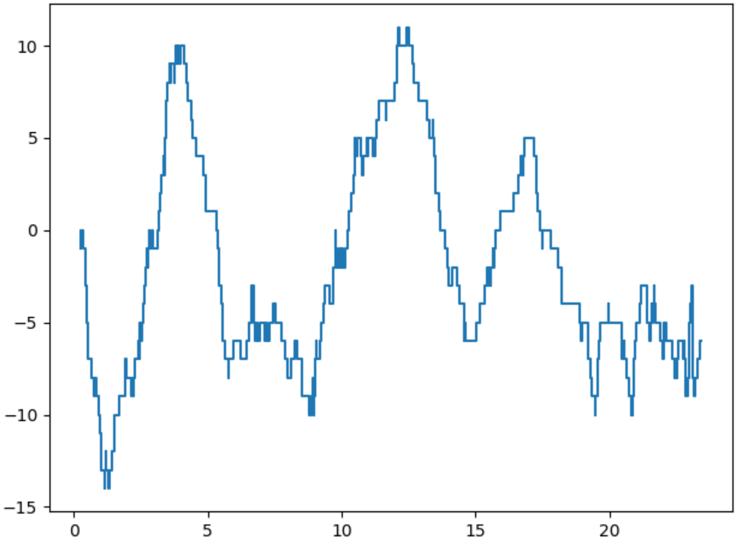
Numerical Simulation (Only perishing, no cancellation)

$\lambda_{pa} = \{'Low' : 2, 'High' : 10\}$, $\lambda_{oa} = 6$, $\lambda_{perish} = 0.5$, $\lambda_{cancel} = 0.001$, $a_{switch} = 10$



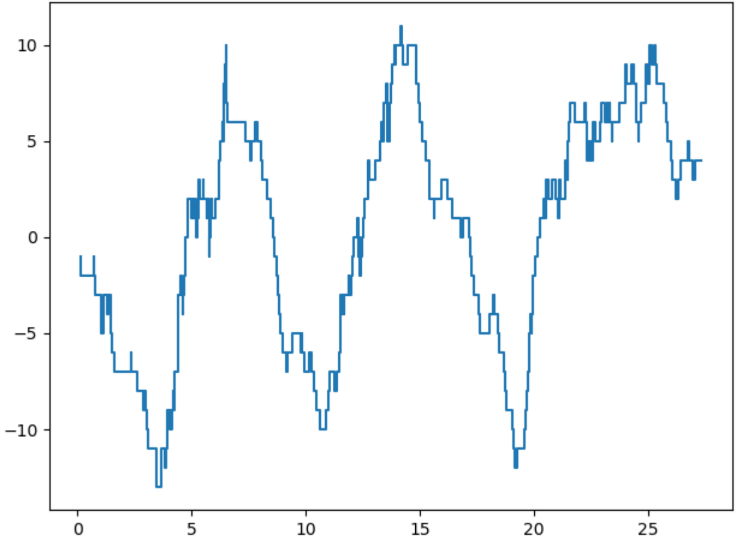
Numerical Simulation (No perishing, only cancellation)

$\lambda_{pa} = \{'Low' : 2, 'High' : 10\}$, $\lambda_{oa} = 6$, $\lambda_{perish} = 0.001$, $\lambda_{cancel} = 0.5$, $a_{switch} = 10$



Numerical Simulation (Without abandonment)

$\lambda_{pa} = \{'Low' : 2, 'High' : 10\}$, $\lambda_{oa} = 6$, $\lambda_{perish} = 0.001$, $\lambda_{cancel} = 0.001$, $a_{switch} = 10$



Discussions

Discussions

- More results to present (prop. of perished, prop. of cancelled, estimate cost)
- Inventory are retrieved based on FCFS (might be unrealistic in some cases)
- Available for downloading[1]

[1] https://github.com/xinge666666/Simulation-of-Double-Ended-Queue.git

All seasons ended

ISE-760 → ISE-761 → ISE-762

Any Questions?

Thank you!