A Cost Comparison of Quantity-based and Time-based Models in Shipment Consolidation Problem Using Simulation Approach

Batuhan Sarı & Bo Wei

September 1, 2025

Introduction

In this report, we compared two inventory control policies under a shipment consolidation constraint: the **Time-based Policy** and the **Quantity-ased Policy**. The comparison is made in terms of minimized cost ($) in different parameter scenarios. The Principal rule for this consolidation is that shipments are only triggered when Q* units are accumulated or in every predetermined T* days.

Problem Definition

A hypothetical logistics company wants to ship their loads in order to minimize the cost. The Company is able to ship their product in multiple units in a single shipment and trying to determine the best shipping policy (time-based vs. quantity-based) in terms of cost.

In addition, an order might be canceled because the waiting time (patience) is too long. If a cancellation occurs, it penalizes policies that fail to fulfill orders quickly, thus rewarding more responsive policies in cost evaluation.

$$\text{Cost} = F_c \times \text{Shipments} + V_c \times \text{Shipped Orders} + \omega \sum (W_s + W_c) + c \times \text{cancel}$$

Where,
F_c = Fixed cost of shipment
c = Fixed cost of cancellation
V_c = Per-unit transportation cost
ω_1 = Waiting cost per shipped unit per time
ω_2 = Waiting cost per cancelled unit per time
W_s = Waiting time of shipped orders
W_c = Waiting time of cancelled orders

Common Parameters

Both policies share the following key parameters:

- Simulation Duration: 1000 cycles
- Order Arrival: Poisson with $\lambda = 10$
- Cancellation: Poisson with $\mu = 0.05$
- Initial Inventory = 0 units

Policy Descriptions

4.1 Time-based Shipment Consolidation Policy

Under a time-based policy,consolidated shipments are released at periodic intervals. Orders that arrive between the release epochs are combined.

4.2 Quantity-based Shipment Consolidation Policy

Under a quantity-based policy, customer orders are held/combined until a target load, assuring scale economies, is accumulated.

Feature Comparison

Feature	**Quantity-based**	**Time-based**
Cleaning Logic	When inventory $\leq R$	Every T days
Order size	Fixed	Fixed
Responsiveness	High	Low
Implementation	Requires tracking of Inventory	No tracking (calendar)

Table 1: Comparison of Inventory Cleaning Policies

Conclusion

The quantity-based policy, being dynamic, is more responsive and can better adapt to variations in demand. However, under a consolidation rule (e.g., ship only when q* units are ready), the fixed-time policy may produce fewer shipments with higher volumes per shipment, potentially lowering the average shipment cost per day. The best choice depends on the balance between cost efficiency and responsiveness.

6.1 Cost Comparison

As can be seen in Figure 2, the quantity-based policy performs better than the time-based policy in terms of the average cost per day ($).

F_c	q^*	QP(q^*)	T^*	TP(T^*)	Cost Reduction (QP vs TP)
500	43	\$312.02	4.8	\$314.99	\$2.96
250	31	\$250.99	3.3	\$253.70	\$2.70
125	22	\$206.96	2.3	\$209.56	\$2.60
60	15	\$173.92	1.6	\$176.43	\$2.51
50	14	\$167.39	1.5	\$169.86	\$2.47
40	13	\$160.18	1.3	\$162.63	\$2.45
30	11	\$151.94	1.2	\$150.90	\$1.20

Table 2: The cost-optimal QP and TP with $\lambda = 10$, $\omega = 5$, and $c = 10$.

Conclusion

The quantity-based policy, being dynamic, is more responsive and can better adapt to variations in demand. However, under a consolidation rule (e.g., ship only when q* units are ready), the fixed-time policy may produce fewer shipments with higher volumes per shipment, potentially lowering the average shipment cost per day. The best choice depends on the balance between cost efficiency and responsiveness.

Appendices

8.1 Appendix A - MATLAB Codes

MATLAB code for QP

```
% === QUANTITY-BASED POLICY ===
clear; clc;

% --- Parameters ---
lambda = 1;                         % Order arrival rate
mu = .05;                           % Patience parameter (mean = 1/mu)
shipmentThreshold = 36;             % Ship when this many orders accumulate
fixedCostPerShipment = 60;
UnitVariableCost = 10;
w = 1;                              % Waiting cost per unit per day
c = 10;
days = 1000*shipmentThreshold / lambda; % Simulation duration

% --- Initialization ---
eventTime = 0;
shipmentTimes = [];
totalShipped = 0;
inventoryLevel = 0;
totalWaitingTime = 0;


orderCounter = 0;
orders = [];  % Each row: [ID, arrivalTime, cancelTime]


nonCanceledWaiting = [];
canceledWaiting = [];
canceledWaitingTimesPerShipment = [];
canceledWaitingTimes = [];
cancelTimes = [];

inventoryHistory = [];
inventoryTimestamps = [];

% Canceled Order ID tracking
canceledOrderIDsByShipment = {};   % Cell array of vectors
currentCycleCanceledIDs = [];


% First event times
nextArrival = -log(rand)/lambda;
nextCancel = inf;

% === EVENT-BASED SIMULATION ===
while eventTime < days
    % Determine next event
    if nextArrival < nextCancel
        % --- Arrival Event ---
        eventTime = nextArrival;
        if eventTime > days, break; end

        % Create order with unique ID
        orderCounter = orderCounter + 1;
        arrivalTime = eventTime;
        patience = -log(rand)/mu;
        cancelTime = arrivalTime + patience;
        orders = [orders; orderCounter, arrivalTime, cancelTime];

        % Schedule next arrival
        nextArrival = eventTime + (-log(rand)/lambda);

        % Update inventory
        inventoryLevel = inventoryLevel + 1;
        inventoryHistory(end+1) = inventoryLevel;
        inventoryTimestamps(end+1) = eventTime;

        % Update cancel
        if ~isempty(orders)
            [~, idx] = min(orders(:,3));
            nextCancel = orders(idx,3);
        end
    else
        % --- Cancellation Event ---
        eventTime = nextCancel;
        if eventTime > days, break; end

        % Cancel the earliest due order
        [~, idx] = min(orders(:,3));
        canceledOrderID = orders(idx,1);
        currentCycleCanceledIDs(end+1) = canceledOrderID;

        canceledWaitingTimes(end+1) = eventTime - orders(idx,2);
        canceledWaiting(end+1) = eventTime - orders(idx,2);
        cancelTimes(end+1) = eventTime;

        orders(idx,:) = [];
        inventoryLevel = inventoryLevel - 1;
        inventoryHistory(end+1) = inventoryLevel;
        inventoryTimestamps(end+1) = eventTime;

        if ~isempty(orders)
            [~, idx] = min(orders(:,3));
            nextCancel = orders(idx,3);
        else
            nextCancel = inf;
        end
    end

    % --- Shipment Trigger ---
    if inventoryLevel >= shipmentThreshold
        shipmentTimes(end+1) = eventTime;

        shippedNow = size(orders, 1);
        totalShipped = totalShipped + shippedNow;

        waitingTimes = eventTime - orders(:,2);  % orders(:,2) = arrivals
        totalWaitingTime = totalWaitingTime + sum(waitingTimes);
        nonCanceledWaiting(end+1) = sum(waitingTimes);

        % Save canceled order IDs and reset for next cycle
        canceledOrderIDsByShipment{end+1} = currentCycleCanceledIDs;
        canceledWaitingTimesPerShipment(end+1) = sum(canceledWaiting);
        currentCycleCanceledIDs = [];
        canceledWaiting = [];

        % Clear system
        orders = [];
        inventoryLevel = 0;
        nextCancel = inf;

        inventoryHistory(end+1) = inventoryLevel;
        inventoryTimestamps(end+1) = eventTime;
    end
end

% --- Cost Calculations ---
totalShipments = length(shipmentTimes);
totalCancel = length(canceledWaitingTimes);
variableCost = totalShipped * UnitVariableCost;
waitingCost = w * totalWaitingTime;
canceledWaitingCost = w * sum(canceledWaitingTimes);
totalCost = totalShipments * fixedCostPerShipment
          + variableCost + (waitingCost + canceledWaitingCost)
          + c*totalCancel;
costPerUnitPerDay = totalCost / days;


if totalShipments > 0
    costPerShipment = totalCost / totalShipments;
else
    costPerShipment = NaN;
end

% --- Output Report ---
fprintf('=== EVENT-BASED QUANTITY POLICY (FIXED) ===\n');
fprintf('Total Shipments: %d\n', totalShipments);
fprintf('Total Orders Arrived: %d\n', orderCounter);
fprintf('Total Waiting Time (unit-days): %.2f\n', totalWaitingTime);
fprintf('Variable Cost: $%.2f\n', variableCost);
fprintf('Waiting Cost: $%.2f\n', waitingCost);
fprintf('Canceled Orders Waiting Cost: $%.2f\n', canceledWaitingCost);
fprintf('Total Cost: $%.2f\n', totalCost);
fprintf('Average Cost per Unit per Day: $%.2f\n', costPerUnitPerDay);


% --- Plot 1: Inventory Level over Time ---
figure;
stairs(inventoryTimestamps, inventoryHistory, 'b-', 'LineWidth', 1.5);
hold on;
if ~isempty(shipmentTimes)
    stem(shipmentTimes, zeros(size(shipmentTimes)), 'r', 'filled', ...
        'LineStyle', 'none');
    legend('Inventory Level', 'Shipment Events');
else
    legend('Inventory Level');
end
xlabel('Time (Days)');
ylabel('Inventory Level');
title('Inventory Level Over Time');
grid on;

% --- Plot 2: Waiting Times per Shipment (Shipped vs. Canceled) ---
figure;
numShipments = length(nonCanceledWaiting);
if length(canceledWaitingTimesPerShipment) < numShipments
```

```
    canceledWaitingTimesPerShipment(end+1:numShipments) = 0;
end
bar(1:numShipments, [nonCanceledWaiting(:), ...
    canceledWaitingTimesPerShipment(:)], 'stacked');
xlabel('Shipment Number');
ylabel('Total Waiting Time');
legend('Shipped Waiting Time', 'Canceled Waiting Time');
title('Waiting Time per Shipment');
grid on;

% --- Plot 3: Canceled Order IDs per Shipment ---
figure;
numCanceled = cellfun(@length, canceledOrderIDsByShipment);
bar(1:length(numCanceled), numCanceled, 'FaceColor', ...
    [0.8500 0.3250 0.0980]);
xlabel('Shipment Number');
ylabel('Number of Canceled Orders');
title('Canceled Order IDs per Shipment');
ylim([0 max(numCanceled)+1]);
grid on;

% Annotate bars with actual canceled order IDs
for i = 1:length(canceledOrderIDsByShipment)
    ids = canceledOrderIDsByShipment{i};
   if isempty(ids)
        txt = 'None';
    else
        txt = strjoin(string(ids), ', ');
    end
    text(i, numCanceled(i) + 0.2, txt, 'HorizontalAlignment', ...
        'center', 'FontSize', 12, 'Rotation', 0);
end
```

MATLAB code for TP

```
% === Time-based Policy ===
clear; clc;

% --- Parameters ---
lambda = 10;
mu = 0.05;
T = 6.3;                                % Fixed shipment in terval
fixedCostPerShipment = 30;
variableCostPerUnit = 10;
w = 5;                                  % Waiting cost per unit per day
c = 10;
days = 1000*T;                          % # of cycles simulation runs


numRuns = 100;
Costs = zeros(1, numRuns);  % Store average cost per day for each run

% --- Initialization ---
eventTime = 0;
shipments = 0;
shipmentTimes = [];
unitsShipped = [];
inventoryLevel = 0;
inventoryHistory = [];
inventoryTimestamps = [];

totalWaitingTime = 0;
waitingTimePerShipment = [];
orderArrivalTimes = [];

totalOrders = 0;
nextShipmentTime = T;

% === Cancellation Setup ===
canceledWaitingTimes = [];
orderPatienceDeadlines = [];  % Each orders patience time is exponential
nextOrderTime = -log(rand) / lambda;
nextCancelTime = -log(rand) / mu;

% === EVENT-BASED SIMULATION ===
while eventTime < days
    % --- Cancel orders whose patience has expired (before any event) ---
    expiredIdx = find(orderPatienceDeadlines <= eventTime);
    for i = fliplr(expiredIdx)
        canceledWaitingTimes(end+1) = eventTime - orderArrivalTimes(i);
        orderArrivalTimes(i) = [];
        orderPatienceDeadlines(i) = [];
        inventoryLevel = inventoryLevel - 1;

        inventoryHistory(end+1) = inventoryLevel;
        inventoryTimestamps(end+1) = eventTime;
    end

    % Choose the next event (order or cancellation)
    if nextOrderTime <= nextCancelTime
        % Order arrives
        eventTime = nextOrderTime;
        nextOrderTime = eventTime + (-log(rand) / lambda);

        if eventTime > days
            break;
        end

        orderArrivalTimes = [orderArrivalTimes, eventTime];
        patience = -log(rand) / mu;  % <-- ADDED
        orderPatienceDeadlines = [orderPatienceDeadlines,
            eventTime + patience];  % <-- ADDED
        totalOrders = totalOrders + 1;
        inventoryLevel = inventoryLevel + 1;

        inventoryHistory(end+1) = inventoryLevel;
        inventoryTimestamps(end+1) = eventTime;

    else
        % Cancellation occurs
        eventTime = nextCancelTime;
        nextCancelTime = eventTime + (-log(rand) / mu);

        if eventTime > days
            break;
        end

        if ~isempty(orderArrivalTimes)
            canceledWaitingTimes(end+1) = eventTime - orderArrivalTimes(1);
            orderArrivalTimes(1) = [];
            orderPatienceDeadlines(1) = [];  % <-- ADDED
            inventoryLevel = inventoryLevel - 1;

            inventoryHistory(end+1) = inventoryLevel;
            inventoryTimestamps(end+1) = eventTime;
        end
    end

    % Check for shipment
    if eventTime >= nextShipmentTime
        % Cancel expired orders again before shipping (edge case)
        expiredIdx = find(orderPatienceDeadlines <= eventTime);
        for i = fliplr(expiredIdx)
            canceledWaitingTimes(end+1) = eventTime - orderArrivalTimes(i);
            orderArrivalTimes(i) = [];
            orderPatienceDeadlines(i) = [];
            inventoryLevel = inventoryLevel - 1;

            inventoryHistory(end+1) = inventoryLevel;
            inventoryTimestamps(end+1) = eventTime;
        end

        % All remaining orders are shipped now
        waitingTimes = eventTime - orderArrivalTimes;
        totalWaitingTime = totalWaitingTime + sum(waitingTimes);
        waitingTimePerShipment{end+1} = waitingTimes;
        cumulativeWaitingPerShipment = cellfun(@sum, ...
            waitingTimePerShipment);

        % Record shipment
        shipments = shipments + 1;
        shipmentTimes(end+1) = eventTime;
        unitsShipped(end+1) = length(orderArrivalTimes);

        % Reset
        orderArrivalTimes = [];
        orderPatienceDeadlines = [];  % <-- ADDED
        inventoryLevel = 0;

        inventoryHistory(end+1) = inventoryLevel;
        inventoryTimestamps(end+1) = eventTime;

        % Schedule next shipment
        nextShipmentTime = nextShipmentTime + T;
    end
end

% === Cost Calculations ===
totalUnitsShipped = sum(unitsShipped);
totalCancel = length(canceledWaitingTimes);
variableCost = totalUnitsShipped * variableCostPerUnit;
waitingCost = w * totalWaitingTime;
canceledWaitingCost = w * sum(canceledWaitingTimes);
totalCost = shipments * fixedCostPerShipment + variableCost + waitingCost
          + canceledWaitingCost + c * totalCancel;
averageCostPerDay = totalCost / days;

% === OUTPUT ===
fprintf('=== EVENT-BASED STATIC POLICY ===\n');
fprintf('Total Shipments: %d\n', shipments);
fprintf('Total Units Shipped: %.2f\n', totalUnitsShipped);
fprintf('Average Units in a Shipment: %.2f\n', mean(unitsShipped));
fprintf('Total Orders: %d\n', totalOrders);
fprintf('Canceled Orders: %d\n', length(canceledWaitingTimes));
fprintf('Canceled Waiting Time: %.2f\n', sum(canceledWaitingTimes));
fprintf('Total Waiting Time (unit-days): %.2f\n', totalWaitingTime);
fprintf('Variable Cost: $%.2f\n', variableCost);
fprintf('Waiting Cost: $%.2f\n', waitingCost);
fprintf('Canceled Waiting Cost: $%.2f\n', canceledWaitingCost);
fprintf('Total Cost: $%.2f\n', totalCost);
fprintf('Average Cost per Day: $%.2f\n', averageCostPerDay);


% === PLOTS ===
figure;
stairs(inventoryTimestamps, inventoryHistory, 'LineWidth', 1.5);
hold on;
stem(shipmentTimes, zeros(size(shipmentTimes)), ...
    'r', 'filled', 'LineStyle', 'none');
xlabel('Time (Days)');
ylabel('Inventory Level');
title('Inventory Level Over Time (Event-Based Simulation)');
legend('Inventory Level', 'Shipment Events');
grid on;

%figure;
plot(1:length(cumulativeWaitingPerShipment), cumulativeWaitingPerShipment, ...
    '-o', 'LineWidth', 1.5);
xlabel('Shipment Number');
ylabel('Cumulative Waiting Time (unit-days)');
title('Cumulative Waiting Time per Shipment');
grid on;

% === PLOT: Comparison of Waiting Times (Canceled vs Non-Canceled) ===
% Prepare vectors
numShipments = length(waitingTimePerShipment);
nonCanceledWaiting = cellfun(@sum, waitingTimePerShipment);
canceledWaiting = zeros(1, numShipments);

% Distribute canceled waiting times into shipment intervals (roughly)
if ~isempty(canceledWaitingTimes)
    cancelTimes = cumsum(canceledWaitingTimes);  % approximate times
    cancelBins = discretize(cancelTimes, [0, shipmentTimes]);
    for i = 1:numShipments
        canceledWaiting(i) = sum(canceledWaitingTimes(cancelBins == i));
    end
end
```

```
% Plot
figure;
bar(1:numShipments, [nonCanceledWaiting(:), canceledWaiting(:)], ...
    'stacked');
xlabel('Shipment Number');
ylabel('Total Waiting Time');
legend('Shipped Orders', 'Canceled Orders');
title('Comparison of Waiting Time per Shipment');
grid on;

% === Poisson Order Interarrival Generator ===
function samples = myPoisson(lambda, n)
samples = zeros(1, n);
for i = 1:n
    L = exp(-lambda);
    k = 0; p = 1;
    while p > L
        k = k + 1;
        p = p * rand();
    end
    samples(i) = k - 1;
end
end

% === Poisson Cancellation Interarrival Generator ===
function cancels = generateExpCancels(mu, n)
    cancels = -log(rand(1, n)) / mu;
end
```