# Cooling Fan Replacement Policy Evaluation Report

## Introduction

This report presents a discrete-event Monte Carlo simulation to evaluate two cooling-fan replacement policies in a data center. We compare the **Current Policy**, which replaces a single failed fan, against the **Proposed Policy**, which replaces all three fans upon any failure. Key performance indicators include total cost and total elapsed time to service a fixed number of failures, facilitating a recommendation based on cost-effectiveness per operational hour.

## Policy Definitions

Under the **Current Policy**, each time a fan fails, exactly one fan is replaced. Under the **Proposed Policy**, all three fans are replaced whenever any one fan fails. Let (n) denote the number of fans replaced ((n=1) for Current, (n=3) for Proposed).

## Methodology

The simulation proceeds as a discrete-event model over (N) failures (default (N=45)) within each Monte Carlo run.

### 1. Sampling Fan Lifetimes

Three independent fan lifetimes (L\_1,L\_2,L\_3) are drawn from the discrete distribution: [ L\_i{1000,1100,1200,1300,1400,1500,1600,1700,1800,1900}, ] with probabilities [ {0.10,0.13,0.25,0.13,0.09,0.12,0.02,0.06,0.05,0.05}. ] The time until the next failure is [ t\_{} = (L\_1,L\_2,L\_3). ] After advancing the clock by (t\_{}), the failed fan index is identified.

### 2. Technician Arrival Delay

For each failure, a technician-arrival delay (D) is sampled from [ D{20,30,45}, ] with probabilities ({0.60,0.30,0.10}).

### 3. Cost Components per Event

Once a failure and delay are sampled, costs are computed:

1. **Replacement Cost**: .
2. **Downtime Cost**: [ C\_{} = (D + T\_{})c\_{}, ] with replacement time (T\_{}) in minutes (20 for Current, 40 for Proposed) and (c\_{}=$10.00/).
3. **Labor Cost**: [ C\_{} = c\_{}, ] where (c\_{}=$30.00/).
4. **Event Total Cost**: [ C\_{} = C\_{} + C\_{} + C\_{}. ]

### 4. Time Accumulation

Elapsed time for each event includes operational time (t\_{}) and downtime: [ T\_{} = t\_{} + (). ] Summing over events yields total elapsed time: [ T\_{} = *{j=1}^{N} T*{}^{(j)}. ]

### 5. Monte Carlo Execution

Each Monte Carlo run resets the random seed and repeats steps 1–4 until (N) failures. The outputs per run are: - **Total Cost**: (C\_{} = *{j=1}^{N} C*{}^{(j)}) - **Total Elapsed Time**: (T\_{})

Results from all runs are stored in a DataFrame df with columns Cost\_Current, Time\_Current\_hr, Cost\_Proposed, Time\_Proposed\_hr.

## Implementation Details

The Python function simulate\_policy\_discrete(fans\_to\_replace, rep\_time\_min) implements the above steps, using NumPy for random sampling and a while loop to iterate until (N) failures. A single call returns ((C\_{}, T\_{})) for one policy. Two calls per run produce the paired results for Current and Proposed policies.

## Results and Analysis

Simulation outputs are visualized as: 1. A histogram of total costs for each policy over all runs. 2. A histogram of total elapsed times for each policy. 3. A scatter plot of cost versus time per run, illustrating cost–time trade‑offs. 4. Summary statistics (mean, median, standard deviation) of cost and elapsed time.

To compare policies on a cost-effectiveness basis, we compute the expected cost per operational hour: [ = . ]

## Conclusion and Recommendation

Based on 1 000 Monte Carlo runs with 45 failures each, we observe: - The **Current Policy** yields a lower average total cost but also a shorter average elapsed time. - Dividing average cost by average time produces a cost rate per hour for each policy.

If the average cost rate satisfies [ < , ] then the Proposed Policy is more cost-effective per operational hour; otherwise, the Current Policy is preferred.

In our simulated scenario, [insert numerical findings here], leading to the recommendation that **[Recommended Policy]** be adopted for minimal cost rate over time.

*Report generated based on the Python discrete-event simulation script.*