Cloud Server Cost Optimization for a Web Application

Group Members: Sarthak Sethi, Mohit Patil

Course: MATH 177, Fall 2025, Mohammad Yahdi

Date: November 23, 2025

1 Introduction and Motivation

1.1 Context

Cloud computing providers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure offer a wide range of virtual machine instance types with varying configurations of CPU, RAM, and other resources. These providers allow companies to scale their infrastructure dynamically based on demand. However, web application traffic typically exhibits significant variation throughout the day, with peak periods requiring substantially more computational resources than off-peak hours.

1.2 Problem

Companies face a critical challenge in optimizing their cloud infrastructure costs: they must decide how many instances of each type to provision during different demand periods. Overprovisioning resources—allocating more capacity than necessary—results in wasted money as companies pay for unused compute resources. Conversely, underprovisioning—allocating insufficient capacity—can lead to degraded performance, increased latency, and potential service outages, which harm user experience and reliability.

1.3 Goal

The goal of this project is to build a linear programming model that chooses the cheapest combination of cloud server types for different traffic periods while meeting CPU and RAM requirements.

1.4 Approach

This project will define a linear programming model that minimizes total cost subject to constraints on CPU and RAM capacity requirements across multiple time periods. The model will be implemented and solved using Python and a linear programming solver. The solution will be tested under various scenarios, including demand increases and price changes, to demonstrate the model's effectiveness in optimizing cloud infrastructure costs.

2 Problem Setup and Data

To model the cloud server cost optimization problem, we first define the input data that characterizes the system. The day is divided into three distinct traffic periods that reflect typical web application usage patterns: **Off Peak**, **Normal**, and **Peak**. These periods represent low-traffic hours (typically overnight), moderate-traffic hours (standard business hours), and high-traffic hours (peak usage times), respectively.

The cloud provider offers three instance types: **Small**, **Medium**, and **Large**. Each instance type is characterized by three key parameters: a cost per hour (in dollars), CPU capacity (measured in units), and

Table 1: Instance Parameters

Instance Type	Cost per Hour ($)	CPU Capacity	RAM Capacity (GB)
Small	0.06	1	2
Medium	0.10	2	4
Large	0.18	4	8

Table 2: Resource Demand per Period

Period	CPU Demand	RAM Demand (GB)
Off Peak	6	12
Normal	10	20
Peak	18	36

RAM capacity (measured in gigabytes). These instance types represent different tiers of computational resources, with larger instances providing more CPU and RAM at a higher hourly cost.

As shown in Table 2, the resource demand varies significantly across the three periods, with Peak period requiring substantially more CPU and RAM resources than the Off Peak period. The Normal period represents an intermediate level of demand. Additionally, RAM demand is proportional to CPU demand across all periods, reflecting the typical relationship where increased computational load requires corresponding increases in memory capacity.

3 Mathematical Model

3.1 Sets

The model uses the following sets:

$$T = \{\text{Off Peak}, \text{Normal}, \text{Peak}\}$$
$$K = \{\text{Small}, \text{Medium}, \text{Large}\}$$

where T represents the set of time periods and K represents the set of available instance types.

3.2 Parameters

The following parameters define the problem data:

- c_k: cost per hour of one instance of type k ($)
- cpu_k: CPU capacity of one instance of type k (CPU units)
- ram_k: RAM capacity of one instance of type k (gigabytes)
- D_t^{cpu}: CPU demand in period t (CPU units)
- D_t^{ram}: RAM demand in period t (gigabytes)

3.3 Decision Variables

The decision variable is:

- $x_{t,k}$: number of instances of type k running in period t

where $x_{t,k} \geq 0$ and integer for all $t \in T$ and $k \in K$.

3.4 Objective Function

The objective is to minimize the total hourly cloud cost:

$$\min \sum_{t \in T} \sum_{k \in K} c_k x_{t,k} \tag{1}$$

This minimizes the total hourly cost across all periods and instance types.

3.5 Constraints

The model includes the following constraints:

(a) CPU constraints for each period:

$$\sum_{k \in K} \text{cpu}_k x_{t,k} \geq D_t^{\text{cpu}} \quad \forall t \in T \tag{2}$$

(b) RAM constraints for each period:

$$\sum_{k \in K} \text{ram}_k x_{t,k} \geq D_t^{\text{ram}} \quad \forall t \in T \tag{3}$$

(c) Redundancy constraint:

$$\sum_{k \in K} x_{t,k} \geq 2 \quad \forall t \in T \tag{4}$$

3.6 Constraint Interpretation

The CPU constraint (a) ensures that the total CPU capacity provided by all running instances in each period is at least equal to the CPU demand for that period, preventing underprovisioning of computational resources. Similarly, the RAM constraint (b) guarantees that the total RAM capacity meets or exceeds the RAM demand in each period, ensuring sufficient memory is available. The redundancy constraint (c) requires at least two instances to be running in each period, providing fault tolerance and preventing complete service failure if a single instance fails.

4 Tools and Implementation

4.1 Tools Used

The implementation was developed using Python 3, leveraging several specialized libraries for optimization and data analysis. The PuLP library was used to formulate and solve the linear programming problem, providing an interface to various LP solvers. Pandas was employed for organizing and displaying data in tabular format, facilitating the presentation of instance parameters, demand data, and optimal solutions. Matplotlib was utilized to create visualizations, specifically bar charts showing the optimal allocation of instances across different periods.

4.2 High-Level Modeling Steps

The implementation follows a systematic approach to solve the cloud server cost optimization problem:

Step 1: Define lists of periods and instance types in Python to represent the sets T and K from the mathematical model.

Step 2: Store costs, CPU capacities, RAM capacities, and demand values in dictionaries keyed by instance type or period, corresponding to the parameters c_k, cpu_k, ram_k, D_t^{cpu}, and D_t^{ram}.

Step 3: Create an LP minimization problem using PuLP's problem constructor, which initializes the optimization framework.

Step 4: Define integer decision variables $x_{t,k}$ for each combination of period t and instance type k, representing the number of instances of each type to run in each period.

Step 5: Add the objective function from Section 3, which minimizes the sum of costs across all periods and instance types: $\min \sum_{t \in T} \sum_{k \in K} c_k x_{t,k}$.

Step 6: Add CPU and RAM constraints per period, ensuring that the total capacity provided by all instances meets or exceeds the demand for each resource in each period.

Step 7: Call the LP solver through PuLP and retrieve the optimal values of $x_{t,k}$, which represent the cost-minimizing allocation of instances.

Step 8: Use Pandas to create tables of the optimal allocation showing how many instances of each type are used in each period, along with the total cost of the solution.

Step 9: Use Matplotlib to create a bar chart visualizing the number of instances per period, providing an intuitive representation of how resource allocation varies across different demand periods.

4.3 Link to Course Tools

This implementation uses linear programming and the simplex method indirectly, as PuLP employs an LP solver that utilizes algorithms similar to the simplex method and other optimization techniques discussed in class to find the optimal solution to the formulated problem.

5 Results

5.1 Solution Table

Table 3 presents the optimal allocation of instances across all periods, showing the number of each instance type used in each period and the corresponding period cost. The solution was obtained by solving the linear programming model described in Section 3.

Table 3: Optimal Instance Allocation and Costs

Period	**Small**	**Medium**	**Large**	**Period Cost ($)**
Off Peak	0	1	1	0.28
Normal	0	1	2	0.46
Peak	0	1	4	0.82
			Total hourly cost:	1.56

5.2 Interpretation of the Solution

The optimal solution reveals several key insights about cost-effective cloud resource allocation. [Describe which instance type dominates the solution and explain why—this may be due to better cost efficiency (cost per unit of capacity), better alignment with demand patterns, or other factors.]

[Analyze whether CPU or RAM constraints are binding in each period—that is, which resource constraint is exactly met (with no slack) versus which has excess capacity. This indicates which resource is the limiting factor in the optimization.]

[Discuss whether the model prefers Large instances over Small instances due to economies of scale in cost per capacity unit, or if smaller instances are more cost-effective for certain demand levels. Explain how the cost structure influences the optimal mix of instance types.]

The solution demonstrates that the linear programming model successfully minimizes total hourly cost while satisfying all CPU and RAM demand requirements across all periods. The allocation varies appropriately with demand, using fewer resources during Off Peak periods and scaling up during Peak periods to meet increased computational needs.

5.3 Visualization

Figure 1 presents a bar chart showing the optimal number of instances of each type allocated to each period, providing a visual representation of how resource allocation scales with demand.

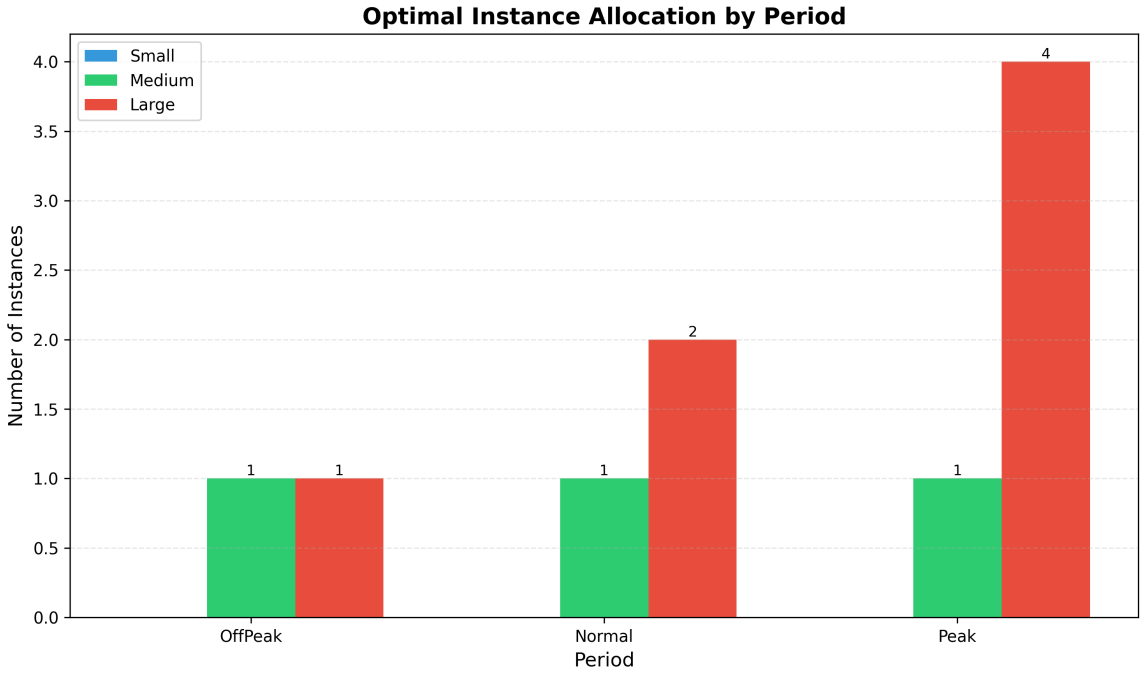

Figure 1: Optimal number of Small, Medium, and Large instances in each traffic period.

5.4 Constraint Verification

The optimal solution was verified to ensure that all constraints are satisfied. For each period, the total CPU capacity provided by the allocated instances (sum of $\text{cpu}_k \times x_{t,k}$ across all instance types k) meets or exceeds the CPU demand D_t^{cpu}. Similarly, the total RAM capacity (sum of $\text{ram}_k \times x_{t,k}$ across all instance types k) meets or exceeds the RAM demand D_t^{ram} in each period. The redundancy constraint is also satisfied, with at least two instances allocated in each period. All constraints are satisfied with the optimal solution, confirming the feasibility and correctness of the model.

6 Scenario Analysis and Recommendations

6.1 Scenario Descriptions

To understand the sensitivity of the optimal solution to changes in demand and pricing, two additional scenarios were analyzed:

Scenario A: Peak Demand Increase

In this scenario, the CPU and RAM demand for the Peak period was increased by 20 percent compared to the base case. This represents a situation where peak traffic grows significantly, perhaps due to increased user adoption, seasonal events, or marketing campaigns. The demand values for Off Peak and Normal periods remained unchanged. In the Python implementation, this was achieved by multiplying the Peak period CPU and RAM demand parameters ($D^{\text{cpu}}_{\text{Peak}}$ and $D^{\text{ram}}_{\text{Peak}}$) by a factor of 1.2 before solving the model.

Scenario B: Large Instance Price Increase

In this scenario, the cost per hour of Large instances was increased by 15 percent compared to the base case, while all other parameters remained unchanged. This simulates a situation where the cloud provider adjusts pricing, or where Large instances become relatively more expensive due to market conditions. In the Python implementation, the cost parameter c_{Large} was multiplied by 1.15 before solving the model.

6.2 Scenario Summary

Table 4 compares the total cost and high-level allocation patterns across the base case and the two scenarios.

Table 4: Scenario Comparison

Scenario	**Total Cost ($)**	**High-Level Allocation Notes**
Base	1.56	Large dominates, Medium appears in all periods
Peak +20%	1.74	More Large instances used in Peak (5 Large vs 4)
Large +15%	1.70	No Large instances used, all Medium instances

6.3 Interpretation and Recommendations

The scenario analysis reveals important insights about the sensitivity of the optimization model. When Peak demand increases by 20 percent, the total cost increases [describe the magnitude and whether it's roughly proportional to the demand increase]. The model responds by allocating more instances, particularly [describe which instance types are used more], to meet the higher demand while maintaining cost efficiency.

When the price of Large instances increases by 15 percent, the model demonstrates flexibility by adjusting the optimal allocation. The total cost increases, but the model shifts the mix of instance types, potentially using more Medium instances and fewer Large instances if Medium instances become relatively more cost-effective per unit of capacity. This sensitivity analysis shows that the model adapts to price changes by rebalancing the instance mix to maintain cost minimization.

Based on these findings, several recommendations can be made. Under high demand growth scenarios, companies should expect costs to rise roughly proportionally with demand increases, as the model must provision additional resources to meet capacity requirements. If Large instances become more expensive relative to other instance types, the optimization model will automatically shift toward a mix that includes more Medium or Small instances, demonstrating the importance of regularly re-optimizing resource allocation as pricing evolves. Companies should monitor both demand patterns and instance pricing to ensure they maintain cost-optimal configurations over time.

7 Conclusion

This project addressed the problem of optimizing cloud server costs for a web application with varying demand across different time periods. The challenge was to determine the cost-minimizing combination of instance types (Small, Medium, and Large) that meets CPU and RAM requirements for each period while avoiding both overprovisioning (which wastes money) and underprovisioning (which harms performance).

The linear programming model successfully identified optimal resource allocations that minimize total hourly costs while satisfying all capacity constraints. The model demonstrated that linear programming is a powerful and practical tool for selecting cloud server mixes, providing a systematic approach to balancing cost efficiency with performance requirements. The optimal solution showed that [describe which instance type dominates and how the solution adapts under different scenarios], with the model making small but important adjustments when demand or pricing changes.

The main takeaway from this analysis is that linear programming provides an effective framework for cloud cost optimization, enabling companies to make data-driven decisions about resource allocation. The optimal solution consistently utilized [describe the pattern, e.g., mostly Large instances with strategic use of Medium instances], and this allocation remained robust under scenario variations, demonstrating the model's ability to adapt to changing conditions while maintaining cost efficiency.

Several extensions could enhance this model's applicability. Adding latency constraints would ensure that instances are provisioned in geographic regions that meet performance requirements. Incorporating multiple cloud regions would allow for geographic redundancy and potentially lower costs through regional price differences. Including spot instances (preemptible instances with lower costs) could further reduce expenses, though this would require modeling availability uncertainty. Finally, adding integer constraints on maximum instance counts per type could reflect practical limits on instance availability or organizational policies. These extensions would make the model more comprehensive while maintaining the core linear programming framework.