Dynamic Cloud Optimization

CS 524 Project Report - Spring 2025

Udhbhav Gupta

Introduction

Background

Most modern web applications are cloud based and AWS is one of the biggest cloud service providers. Cloud applications in AWS run on virtual machines called Elastic Compute Cloud (EC2) instances. AWS offers multiple types of EC2 instances with varying sizes and combinations of resources like CPU, Memory, network performance etc. Correspondingly, each instance type has a different hourly cost proportional to the amount and type of resources.

We will consider a cloud based web application that receives tasks and schedules them on EC2 instances for execution. For such an application, it is necessary to determine the most cost effective way to assign incoming workload of tasks to appropriate EC2 instances such that each instance has enough resources required by all the tasks assigned to it.

Further, for most web applications, the workload fluctuates throughout the day across time intervals. Consequently, for cost effectiveness, it becomes important to dynamically scale the EC2 instance allocation based on the workload. Inefficient instance allocation can lead to over-provisioning (wasting money) or under-provisioning (failing to meet performance requirements for the workload).

This project addresses the problem of optimally allocating and scaling cloud resources in response to time-varying workloads, using a Linear Integer optimization model that incorporates both task-instance assignment and instance scaling decisions over multiple time intervals.

Data

The data required for this modeling is a list of timestamped computational tasks along with their CPU and Memory requirements. To model the scaling of the instance resource

allocation across time periods the dataset should ideally be real and not simulated to reflect the variations in task volume and resource requirements across time intervals of a day.

Borg is Google's large-scale cluster management system, responsible for scheduling and running hundreds of thousands of jobs across thousands of machines. The Google Borg cluster trace contains anonymized, timestamped records of tasks scheduled on a Google compute cluster, including information such as CPU and memory usage, task duration, and submission times. This data is well-suited for the purposes of this project, as it allows for modeling time-varying workloads and simulating resource allocation decisions across multiple intervals.

The trace naturally is significantly large both in terms of amount and attributes of the task data and will has been preprocessed to extract tasks that fit within fixed-length time windows (e.g., 3-hour intervals), with CPU and memory requirements used as input parameters to the optimization model. Although the trace spans 29 days of tasks across 12.5k machines, to keep the optimization model computationally feasible, only a subset of this dataset will be used.

The EC2 instance information and corresponding costs have been obtained from the AWS website.

Problem Statement

You are given a time-varying workload of computational tasks, along with a set of available cloud virtual machine (VM) types. Your goal is to allocate tasks to instances of these VM types over time in a cost-efficient manner, ensuring that task resource requirements (CPU and memory) are satisfied, and that cloud usage costs (including startup costs) are minimized.

Inputs

1. Task data (for each time interval t = 1,…,8)
   Each time interval is 3 hours long and the 8 intervals collectively represent an entire day i.e. 0000 to 2400
   Note that each time interval can contain a different number of tasks. Each task must be assigned in the interval that it is received.

   For each time interval t, you are given a CSV file *tasks_t[t].csv* with:
   - cpu_cores: Number of CPU cores required by each task
   - mem_gb: Amount of memory (GB) required by each task

2. 12 EC2 instance types (cloud VM types)
   A CSV file *ec2_subset.csv* with:
   - Type: Instance name (e.g., t4g.nano)
   - vCPUs: Number of CPU cores available
   - Memory: Memory available (GB)
   - Cost: Cost per hour in $ of running the instance

In a time interval, any number of instances of any VM type can be spun up and used as long as each task for the interval is assigned to a specific instance. Tasks cannot be split across instances. Each instance can run multiple tasks as long as the total CPU and Memory requirements of the assigned tasks falls within the instance type's limits. In addition to the cost of running the instances for the entire time interval, there is an additional cost of starting up an instance. Assume that an instance startup takes 6 minutes and this startup cost is proportional to the corresponding hourly cost of the instance type. Also assume that before t=1, no instances of any VM type are running.

Your objective is to minimize the total cost of running the instances to execute the given tasks.

Assumptions

- Each task must be assigned to an instance in the time interval that it is received and completes within that interval.
- Once started at the beginning of the interval, all instances of all types run for the entire time interval.
- Both compute resources (CPU and Memory) cannot be shared across tasks.
- All instances only count towards the cost for startup and the duration that they are running any assigned tasks. There is no cost for terminating a running instance at the end of a time interval.

Mathematical Model (Across Time Intervals)

Sets

- $M = \{1\ (\text{t4g.nano}), 2\ (\text{t4g.medium}), 3\ (\text{t4g.xlarge}), 4\ (\text{r8g.large}), 5\ (\text{c8g.xlarge}), 6\ (\text{r8g.2xlarge}),$
  $7\ (\text{c8g.4xlarge}), 8\ (\text{c8g.8xlarge}), 9\ (\text{m8g.8xlarge}), 10\ (\text{r8g.8xlarge}), 11\ (\text{m8g.12xlarge}), 12\ (\text{c8g.16xlarge})\}$
  = Set of Machine Types
- $T = \{1\ (\text{12am--3am}),\ 2\ (\text{3am--6am}),\ 3\ (\text{6am--9am}),\ 4\ (\text{9am--12pm}),\ 5\ (\text{12pm--3pm}),$
  $6\ (\text{3pm--6pm}),\ 7\ (\text{6pm--9pm}),\ 8\ (\text{9pm--12am})\}$
  = Set of Time Intervals
- S_t = Set of given tasks arriving in interval $t \in T$
- $J_t = \{1, 2, \ldots, |S_t|\}$ = Index set of potential instances per machine type in interval $t \in T$

Decision Variables

- x_{ijkt} : 1 if task $k \in S_t$ is assigned to instance $j \in J_t$ of machine type $i \in M$ during interval $t \in T$, 0 otherwise
- y_{ijt} : 1 if instance $j \in J_t$ of machine type $i \in M$ is used in interval $t \in T$, 0 otherwise
- f_{it} : number of instances of machine type $i \in M$ active at the end of interval $t \in T$
- g_{it} : number of new instances of machine type $i \in M$ spun up in interval $t \in T$
- r_{it} : number of active instances of machine type $i \in M$ shut down at beginning of interval $t \in T$

Parameters

- c_i : cost per hour in $ofmachinetype$i \in M$
- p_i : CPU limit in vCPUs of machine type $i \in M$
- q_i : Memory limit in GiB of machine type $i \in M$
- m_k : CPU requirement in vCPUs of task $k \in S_t$
- n_k : Memory requirement in GiB of task $k \in S_t$
- W : size of time window in hrs for which tasks are executing
- $\alpha = 0.1$: instance startup time in hrs (6 min)
- $s_i = \alpha c_i$: startup cost in Dollars of an instance of machine type $i \in M$

Objective Function

$$\min \sum_{t \in T} \sum_{i \in M} (W c_i f_{it} + s_i g_{it})$$

Constraints

- $\sum_{k \in S_t} m_k x_{ijkt} \leq p_i\,, \quad \forall i \in M, j \in J_t, t \in T$ `(CPU constraint per instance per machine type)`
- $\sum_{k \in S_t} n_k x_{ijkt} \leq q_i\,, \quad \forall i \in M, j \in J_t, t \in T$ `(Memory constraint per instance per machine type)`
- $\sum_{i \in M} \sum_{j \in J_t} x_{ijkt} = 1\,, \quad \forall k \in S_t, t \in T$ `(each task can only be assigned to 1 specific instance)`
- $y_{ijt} \leq \sum_{k \in S_t} x_{ijkt} \leq |S_t|\, y_{ijt}\,, \quad \forall i \in M, j \in J_t, t \in T$ `(link x and y constraint)`
- $\sum_{j \in J_t} y_{ijt} = f_{it}\,, \quad \forall i \in M, t \in T$ `(active instance count relationship/constraint)`
- $f_{it} = f_{i(t-1)} + g_{it} - r_{it}\,, \quad \forall i \in M, t \in T$ `(flow balance constraint)`
- $r_{it} \leq f_{i(t-1)}\,, \quad \forall i \in M, t \in T$ `(shut down cannot exceed previous active)`
- $x_{ijkt} \in \{0, 1\}\,, \quad \forall i \in M, j \in J_t, k \in S_t, t \in T$
- $y_{ijt} \in \{0, 1\}\,, \quad \forall i \in M, j \in J_t, t \in T$
- $f_{it}, g_{it}, r_{it} \geq 0\,, \quad \forall i \in M, t \in T$
- $f_{i0} = 0\,, \quad \forall i \in M$

Solution (Across Time Intervals)

In [2]:
```
function check_optimality(m)
    stat = termination_status(m)
    if stat != MOI.OPTIMAL
        println("Solver did not find an optimal solution: $stat")
    end
end;
```

In [7]:
```
using CSV, DataFrames
```

```julia
# Read EC2 types
ec2_df = CSV.read("ec2_subset.csv", DataFrame)

# Read tasks (one CSV per time interval)
tasks_df = Dict()
for t in 1:8
    tasks_df[t] = CSV.read("tasks_t$(t).csv", DataFrame)
end

M = collect(1:nrow(ec2_df)) # machine types
println("Number of machine types: ", length(M))
machine_name = Dict(i => ec2_df.Type[i] for i in M)

c = Dict(i => ec2_df.Cost[i] for i in M) # cost per hour of machine type
p = Dict(i => ec2_df.vCPUs[i] for i in M) # CPU limit of machine type
q = Dict(i => ec2_df.Memory[i] for i in M) # Memory limit of machine type

W = 3 # time interval window size
α = 0.1 # instance startup time
s = Dict(i => α * c[i] for i in M) # startup cost in $ of machine type
T = collect(1:8) # time intervals

S = Dict() # Set of tasks in each time interval
J = Dict() # Set of instances in each time interval
m = Dict() # CPU req
n = Dict() # Mem req

for t in T
    df = tasks_df[t]
    S[t] = collect(1:nrow(df))  # Task indices
    J[t] = collect(1:nrow(df)) # Instance indices
    println("Number of tasks in time interval $t: ", length(S[t]))
    m[t] = Dict(k => df.cpu_cores[k] for k in S[t])
    n[t] = Dict(k => df.mem_gb[k] for k in S[t])
end;
```

```
Number of machine types: 12
Number of tasks in time interval 1: 27
Number of tasks in time interval 2: 18
Number of tasks in time interval 3: 20
Number of tasks in time interval 4: 22
Number of tasks in time interval 5: 27
Number of tasks in time interval 6: 32
Number of tasks in time interval 7: 24
Number of tasks in time interval 8: 34
```

In [ ]:

```julia
using JuMP, HiGHS
model = Model(HiGHS.Optimizer)

@variable(model, f[i in M, t in T] >= 0) # number of active instances of type i at end of interval t
@variable(model, g[i in M, t in T] >= 0) # number of instances of type i started at beginning of interval t
@variable(model, r[i in M, t in T] >= 0) # number of instances of type i terminated at beginning of interval t

# time varying binary variables
x = Dict()
y = Dict()

for t in T
    x[t] = @variable(model, [i in M, j in J[t], k in S[t]], Bin) # task k is assigned to instance j of type i in interval t
    y[t] = @variable(model, [i in M, j in J[t]], Bin) # instance j of type i is active in interval t
end

@objective(model, Min, sum(W*c[i]*f[i,t] + s[i]*g[i,t] for i in M, t in T))

for t in T, i in M, j in J[t]
    @constraint(model, sum(m[t][k] * x[t][i,j,k] for k in S[t]) <= p[i]) # CPU constraint
    @constraint(model, sum(n[t][k] * x[t][i,j,k] for k in S[t]) <= q[i]) # Memory constraint
end

for t in T, k in S[t]
    @constraint(model, sum(x[t][i,j,k] for i in M, j in J[t]) == 1) # each task is assigned to one instance
end

for t in T, i in M, j in J[t]
    @constraint(model, y[t][i,j] <= sum(x[t][i,j,k] for k in S[t])) # instance is active if it has tasks assigned
    @constraint(model, sum(x[t][i,j,k] for k in S[t]) <= y[t][i,j] * length(S[t])) # instance can only have tasks assigned if
end

for t in T, i in M
    @constraint(model, sum(y[t][i,j] for j in J[t]) == f[i,t]) # number of active instances at end of interval t
end

for i in M
    @constraint(model, f[i,1] == g[i,1] - r[i,1])  # number of active instances at beginning of interval 1

    for t in 2:length(T)
        @constraint(model, f[i,t] == f[i,t-1] + g[i,t] - r[i,t]) # flow balance
        @constraint(model, r[i,t] <= f[i,t-1]) # # of instances terminated cannot exceed # of instances at beginning of interv
```

```
    end
end

set_silent(model)
optimize!(model)
check_optimality(model)
```

```
Set parameter Username
Set parameter LicenseID to value 2650823
Academic license - for non-commercial use only - expires 2026-04-13
Solver did not find an optimal solution: MEMORY_LIMIT
```

In [ ]:

```
println("Minimized total cost: ", objective_value(model))
for t in T
    println("Time interval $t:")
    for i in M
        println("  Machine type $(machine_name[i]):")
        println("    Active instances: ", value(f[i,t]))
        println("    Started instances: ", value(g[i,t]))
        println("    Terminated instances: ", value(r[i,t]))
        for j in J[t]
            if value(y[t][i,j]) > 0.5
                println("      Instance $j is active")
                for k in S[t]
                    if value(x[t][i,j,k]) > 0.5
                        println("        Task $k is assigned to instance $j")
                    end
                end
            end
        end
    end
end
```

Mathematical Model (Single Time Interval)

Sets

- $M = \{1\ (\text{t4g.nano}), 2\ (\text{t4g.medium}), 3\ (\text{t4g.xlarge}), 4\ (\text{r8g.large}), 5\ (\text{c8g.xlarge}), 6\ (\text{r8g.2xlarge}),$
  $7\ (\text{c8g.4xlarge}), 8\ (\text{c8g.8xlarge}), 9\ (\text{m8g.8xlarge}), 10\ (\text{r8g.8xlarge}), 11\ (\text{m8g.12xlarge}), 12\ (\text{c8g.16xlarge})\}$
  = Set of Machine Types
- S = Set of Tasks
- $J = \{1, 2, \ldots, |S|\}$ = Index set of potential instances per machine type

Decision Variables

- x_{ijk} : 1 if task $k \in S$ is assigned to instance $j \in J$ of machine type $i \in M$, 0 otherwise
- y_{ij} : 1 if instance $j \in J$ of machine type $i \in M$ is used, 0 otherwise

Parameters

- c_i : cost per hour in $of machine type$i \in M$
- p_i : CPU limit in vCPUs of machine type $i \in M$
- q_i : Memory limit in GiB of machine type $i \in M$
- m_k : CPU requirement in vCPUs of task $k \in S$
- n_k : Memory requirement in GiB of task $k \in S$
- W : size of time window in hrs for which tasks are executing
- $\alpha = 0.1$: instance startup time in hrs (6 min)
- $s_i = \alpha c_i$: startup cost in Dollars of an instance of machine type $i \in M$

Objective Function

$$\min \sum_{i \in M} \sum_{j \in J} (W c_i y_{ij} + s_i y_{ij})$$

Constraints

- $\sum_{k \in S} m_k x_{ijk} \leq p_i\,, \quad \forall i \in M, j \in J$ `(CPU constraint per instance per machine type)`
- $\sum_{k \in S} n_k x_{ijk} \leq q_i\,, \quad \forall i \in M, j \in J$ `(Memory constraint per instance per machine type)`
- $\sum_{i \in M} \sum_{j \in J} x_{ijk} = 1\,, \quad \forall k \in S$ `(each task can only be assigned to 1 specific instance)`
- $y_{ij} \leq \sum_{k \in S} x_{ijk} \leq |S|\, y_{ij}\,, \quad \forall i \in M, j \in J$ `(link x and y constraint)`
- $x_{ijk} \in \{0, 1\}\,, \quad \forall i \in M, j \in J, k \in S$
- $y_{ij} \in \{0, 1\}\,, \quad \forall i \in M, j \in J$

Solution (Single Time Interval)

In [2]:
```
function check_optimality(m)
    stat = termination_status(m)
    if stat != MOI.OPTIMAL
        println("Solver did not find an optimal solution: $stat")
    end
end;
```

In [1]:
```
using CSV, DataFrames

# Read EC2 types
ec2_df = CSV.read("ec2_subset.csv", DataFrame)

M = collect(1:nrow(ec2_df)) # machine types
println("Number of machine types: ", length(M))
machine_name = Dict(i => ec2_df.Type[i] for i in M)

c = Dict(i => ec2_df.Cost[i] for i in M) # cost per hour of machine type
p = Dict(i => ec2_df.vCPUs[i] for i in M) # CPU limit of machine type
q = Dict(i => ec2_df.Memory[i] for i in M) # Memory limit of machine type
W = 3 # time interval window size
α = 0.1 # instance startup time
s = Dict(i => α * c[i] for i in M) # startup cost in $ of machine type
```

```
# Read tasks
tasks_df = CSV.read("tasks_t7.csv", DataFrame)
S = collect(1:nrow(tasks_df)) # task indices
J = collect(1:nrow(tasks_df)) # job indices
println("Number of tasks: ", length(S))
m = Dict(k => tasks_df.cpu_cores[k] for k in S) # CPU cores required by task k
n = Dict(k => tasks_df.mem_gb[k] for k in S); # Memory required by task k
```

```
Number of machine types: 12
Number of tasks: 24
```

In [6]:

```
using JuMP, HiGHS
model = Model(HiGHS.Optimizer)

@variable(model, x[i in M, j in J, k in S], Bin) # x[i,j,k] = 1 if task k is assigned to instance j of type i
@variable(model, y[i in M, j in J], Bin) # y[i,j] = 1 if instance j of type i is used

@objective(model, Min, sum(W*c[i]*y[i,j] + s[i]*y[i,j] for i in M, j in J)) # minimize cost

@constraint(model, [i in M, j in J], sum(m[k]*x[i,j,k] for k in S) <= p[i]) # CPU limit
@constraint(model, [i in M, j in J], sum(n[k]*x[i,j,k] for k in S) <= q[i]) # Memory limit

@constraint(model, [k in S], sum(x[i,j,k] for i in M, j in J) == 1) # each task is assigned to one instance

@constraint(model, [i in M, j in J], y[i,j] <= sum(x[i,j,k] for k in S)) # instance is active if it has tasks assigned
@constraint(model, [i in M, j in J], sum(x[i,j,k] for k in S) <= length(S)*y[i,j]) # instance can only have tasks assigned if

set_silent(model)
optimize!(model)
check_optimality(model)

println("Minimized total cost: ", objective_value(model))
for i in M
    println("Machine type: ", machine_name[i], ", CPU limit: ", p[i], ", Memory limit: ", q[i])
    println("  Total instances: ", sum(value(y[i, j]) for j in J))
    println("  Total assigned tasks: ", sum(value(x[i, j, k]) for j in J, k in S))
    for j in J
        if value(y[i, j]) > 0.5
            for k in S
                if value(x[i, j, k]) > 0.5
                    println("        Task $k (CPU: ", m[k] , ", Memory: ", n[k] ,") is assigned to instance $j")
                end
            end
        end
    end
end
```

```
Minimized total cost: 2.499839999999995
Machine type: t4g.nano, CPU limit: 2, Memory limit: 0.5
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: t4g.medium, CPU limit: 2, Memory limit: 4.0
  Total instances: 15.99999999999995
  Total assigned tasks: 15.999999999999927
        Task 21 (CPU: 2, Memory: 0.5088) is assigned to instance 1
        Task 20 (CPU: 2, Memory: 0.5088) is assigned to instance 2
        Task 11 (CPU: 1, Memory: 3.0544) is assigned to instance 3
        Task 2 (CPU: 1, Memory: 2.5472) is assigned to instance 4
        Task 4 (CPU: 1, Memory: 2.5472) is assigned to instance 5
        Task 24 (CPU: 2, Memory: 2.8) is assigned to instance 7
        Task 3 (CPU: 1, Memory: 2.5472) is assigned to instance 8
        Task 22 (CPU: 2, Memory: 1.22272) is assigned to instance 9
        Task 8 (CPU: 1, Memory: 2.5472) is assigned to instance 10
        Task 1 (CPU: 1, Memory: 2.5472) is assigned to instance 11
        Task 5 (CPU: 1, Memory: 2.5472) is assigned to instance 12
        Task 12 (CPU: 1, Memory: 3.3088) is assigned to instance 13
        Task 10 (CPU: 1, Memory: 2.6752) is assigned to instance 14
        Task 23 (CPU: 2, Memory: 2.5472) is assigned to instance 15
        Task 14 (CPU: 1, Memory: 3.8208) is assigned to instance 19
        Task 16 (CPU: 1, Memory: 3.8208) is assigned to instance 24
Machine type: t4g.xlarge, CPU limit: 4, Memory limit: 16.0
  Total instances: 2.0
  Total assigned tasks: 7.999999999999989
        Task 13 (CPU: 1, Memory: 3.8208) is assigned to instance 12
        Task 15 (CPU: 1, Memory: 3.8208) is assigned to instance 12
        Task 17 (CPU: 1, Memory: 4.0704) is assigned to instance 12
        Task 19 (CPU: 1, Memory: 4.0704) is assigned to instance 12
        Task 6 (CPU: 1, Memory: 2.5472) is assigned to instance 13
        Task 7 (CPU: 1, Memory: 2.5472) is assigned to instance 13
        Task 9 (CPU: 1, Memory: 2.5472) is assigned to instance 13
        Task 18 (CPU: 1, Memory: 4.0704) is assigned to instance 13
Machine type: r8g.large, CPU limit: 2, Memory limit: 16.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: c8g.xlarge, CPU limit: 4, Memory limit: 8.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: r8g.2xlarge, CPU limit: 8, Memory limit: 64.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: c8g.4xlarge, CPU limit: 16, Memory limit: 32.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: c8g.8xlarge, CPU limit: 32, Memory limit: 64.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: m8g.8xlarge, CPU limit: 32, Memory limit: 128.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: r8g.8xlarge, CPU limit: 32, Memory limit: 256.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: m8g.12xlarge, CPU limit: 48, Memory limit: 192.0
  Total instances: 0.0
  Total assigned tasks: 0.0
Machine type: c8g.16xlarge, CPU limit: 64, Memory limit: 128.0
  Total instances: 0.0
  Total assigned tasks: 0.0
```

In [ ]:

Discussion

Note: All data and model files can be found in this repository.

Data

- As mentioned previously, the Google Borg Trace dataset is too large for the scope of this project. To select tasks from the dataset I implemented a custom Extract Transform Load data pipeline in Apache Spark. The pipeline filtered task submission events within a 24-hour period, removed incomplete or irrelevant records, and selected the most compute-intensive task per job based on normalized CPU and memory usage. Tasks were binned into 3-hour windows, scaled to EC2-equivalent resource units, and the top 1% most resource-demanding tasks in each window were retained. The final datasets were partitioned and saved as CSV files, one per time window, for use in optimization modeling. The pipeline can be found here. The final task counts per window were

Time Period	Number of Tasks
1	27
2	18
3	20
4	22
5	27
6	32
7	24
8	34

- Of the 700 plus types of EC2 instances that AWS provides, 8 were selected for this project covering a variety of hardware types and cost ranges.

Modeling

As observed from the solution above, for multi time period modeling, the optimization model proved to be too large and complex for my PC to handle and it ran into a out-of-memory issue. I also tried running it remotely on a Computer Science Lab machine with better hardware but that too ran into the same error. Potentially using an even stronger

machine with more resources and considering some optimizations to the model itself to lower its complexity should allow the model to run.
Following the unsuccessful run of the multi time period model, as proof of concept of the project, I tweaked and simplified the model to only run for a single time period, as shown in the 2nd pair of Mathematical Model and Solution above. In the specific solution above, the model was run for the 7th time interval with 24 tasks.As indicated by the solution, the model balances between selecting instance types that fit the task resource requirements and controlling the total number of active instances to reduce cost. In this run, the model selected 16 **t4g.medium** instances (2 vCPU, 4 GiB memory) for 16 tasks and 2 **t4g.xlarge** instances (4 vCPU, 16 GiB memory) for 4 tasks each to satisfy all task requirements at a minimized total cost of approximately **2.50 USD** for the interval. No larger or more expensive instances were selected, which is intuitive given the scale of the task set and the absence of very large individual task requirements.

Insights from the Single Time Period Model

- **Instance Size Efficiency**: The model demonstrates an ability to avoid overprovisioning. For this particular task mix, smaller and mid-size instances (t4g.medium, t4g.xlarge) were more cost-effective than larger high-capacity instances, which remained unused.
- **Task Packing:** The assignment pattern shows that tasks with higher memory requirements (3–4 GiB) were efficiently grouped together into larger t4g.xlarge instances, while smaller tasks were allocated to medium instances. This reflects the bin-packing nature of the problem where tight packing improves utilization and reduces cost.
- **Feasibility and Scalability:** The successful run of the single-interval model validates that the ILP formulation and implementation are correct and can scale to small-to-moderate workloads. This provides a working baseline from which extensions can be developed including the multi-time period model.

Limitations

- **Static Task Duration Assumption:** The current model assumes that all tasks arrive and complete within the same time interval. In practice, tasks may span multiple intervals, introducing resource locking and scheduling complexities.
- **No Task Deadlines or Priorities:** All tasks are treated equally in terms of urgency and importance. Introducing task priorities or deadlines could improve relevance to real-world workloads.

- **No Network Bandwidth or Storage Constraints:** The model only considers CPU and Memory. In cloud systems, network bandwidth (I/O) or storage might be additional bottlenecks not modeled here.
- **Limited Cost Dynamics:** The single time period model cannot capture tradeoffs between short-term overprovisioning versus long-term savings.
- **Lack of Workload Elasticity:** In real-world scenarios tasks can arrive dynamically and could expect allocation for execution immediately instead of being batched into fixed time intervals.

Conclusion

This project explored the use of optimization modeling to allocate cloud resources efficiently by assigning tasks to EC2 instance types in a cost-effective manner. Using a filtered and preprocessed subset of the Google Borg Trace dataset, I formulated a Integer Linear Programming (ILP) model to minimize instance costs while satisfying task CPU and memory demands. Although the multi-time period model proved too large to solve with available computing resources, the single time period model successfully demonstrated that the approach can effectively balance task resource requirements with instance selection and cost minimization.

Looking ahead, a key direction for future work is to extrapolate this project into a live resource allocation system by integrating with AWS APIs. This system would dynamically monitor cloud workloads, invoke the optimization model to guide task placement and scaling decisions in near-real time, and automatically provision or deprovision instances based on model outputs. Besides this, future work can also address the limitations discussed in the previous section to model real life workflows more closely although that would likely entail a more complex model.

It is worth noting that, as evident by the unsuccessful run of the multi time period model, more complex models likely suffer from computational infeasibility for this problem at scale. This could be one of the reasons that most real-world schedulers and instance scaling systems are based on heuristics and algorithms rather than pure optimization models.