Optimizing Resource Allocation and Time Completion in Data Centers Using Integer Linear Programming

Daniel Safavisohi[1] and Dr. Melkonian[2]

[1]Graduate Student, Mathematics Department, Ohio University
[2]Associate Professor, Mathematics Department, Ohio University

December 13, 2024

Abstract

This study presents optimization models for task assignment and resource allocation in data centers, with a focus on minimizing task completion time, energy consumption, and load imbalance. Two distinct models are developed: one leveraging batching with dynamic sizes and durations, and another incorporating multi-resource allocation and energy-aware scheduling. Experimental evaluations on synthetic datasets demonstrate that the first model effectively assigns tasks to servers within predefined batches, while the second model optimizes task prioritization and load balancing across resources. The results highlight the computational challenges associated with nonlinear constraints and the advantages of linearized models for scalability and efficiency. Future work includes integrating the strengths of both models and applying them to real-world datasets from industry leaders to enhance their applicability and performance in large-scale environments.

1 Introduction

In the era of big data and cloud computing, data centers play a pivotal role in processing and storing vast amounts of information. Efficient resource allocation and minimizing task completion time are critical for optimizing performance and reducing operational costs in data centers. Integer Linear Programming (ILP) offers a mathematical approach to model and solve such optimization problems. This project focuses on formulating an ILP model to optimize resource allocation and task scheduling in data centers, aiming to minimize total completion time while adhering to resource constraints.

2 Overview of Data Centers

Data centers are rapidly expanding across the globe to meet the escalating demand for digital services. The United States leads this growth, hosting approximately 2,670 data centers—the highest number of any country—contributing to a global total of over 8,000 facilities spread across regions like Europe, Asia-Pacific, and Latin America. Constructing and operating these centers involve substantial investments; building costs can range from $ 10 million for small to medium-sized facilities to over $1 billion for hyperscale data centers developed by tech giants such as Amazon, Google, and Facebook. Operational expenses, including energy consumption, maintenance, staffing, and technological upgrades, can amount to millions of dollars annually. Notable projects like Google's $2.5 billion data center in Iowa and Microsoft's investments exceeding $1 billion in various locations underscore the significant financial commitment required. The construction timeline for a data center typically spans 18 to 24 months, influenced by factors such as the facility's scale, technological requirements, regulatory approvals, and site-specific challenges.

2.1 Types of Projects Handled by Data Centers

Data centers are the backbone of modern digital infrastructure, supporting a wide array of projects critical to today's business operations and services. They provide the essential foundation for cloud computing platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform, enabling scalable and flexible computing resources. In the realm of big data and analytics,

data centers process massive datasets for industries such as finance, healthcare, and retail, driving informed decision-making and strategic insights. They are instrumental in powering artificial intelligence (AI) and machine learning (ML) applications, hosting the computational resources required to train complex models. Additionally, data centers facilitate content delivery networks (CDNs), ensuring the rapid and efficient global distribution of media content and web applications. They manage data from the Internet of Things (IoT), overseeing interconnected devices and sensors used in smart cities, industrial automation, and consumer electronics. In financial services, data centers support high-frequency trading platforms, online banking, and transaction processing systems, maintaining the robustness and security of financial transactions. Furthermore, they run critical enterprise applications like Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Supply Chain Management (SCM) systems, underpinning the operational efficiency of businesses worldwide.

2.2 Main Challenges Facing Data Centers Today

Data centers today face several significant challenges that impact their efficiency, cost-effectiveness, and sustainability. Energy consumption and environmental impact are major concerns, with data centers accounting for about 1 percent of global electricity use. This has led to a concerted effort to reduce carbon footprints through the adoption of renewable energy sources and more efficient technologies. Heat dissipation and cooling present another critical issue; managing the heat generated by high-density computing equipment is essential, and traditional cooling methods are energy-intensive. Innovations like liquid cooling and free-air cooling are being explored to address this. Cybersecurity threats have become increasingly sophisticated, necessitating robust security measures to protect sensitive data and ensure compliance with data protection regulations. The need for scalability and flexibility is a constant challenge due to rapid technological advancements, requiring data centers to implement modular designs and scalable infrastructure. Regulatory compliance adds another layer of complexity, as data centers must navigate complex regulations related to data sovereignty, privacy laws like GDPR, and industry-specific compliance standards. Lastly, supply chain disruptions caused by global events such as pandemics and geopolitical tensions can disrupt the supply of critical components, affecting both construction and maintenance.

2.3 Optimization Techniques in Data Centers

To enhance resource utilization and minimize completion times, data centers employ various optimization techniques that are essential for efficient operations. Load balancing is a fundamental strategy that distributes workloads evenly across servers, preventing both overloading and underutilization, which in turn enhances performance and reliability. Dynamic resource allocation further refines this process by adjusting resources in real-time based on current workload demands, thereby improving efficiency and responsiveness to fluctuating needs. Another critical technique is task consolidation, which groups smaller tasks to run concurrently on the same server, freeing up resources and reducing overall energy consumption. Additionally, data centers leverage predictive analytics, utilizing historical data and machine learning algorithms to forecast workload patterns. This allows for proactive adjustments in resource allocation, ensuring that resources are optimally used and ready to meet future demands.

2.4 Resource Scheduling Algorithms in Data Centers

Data centers utilize various resource scheduling algorithms to optimize task allocation and enhance overall efficiency. First-Come, First-Served (FCFS) processes tasks strictly in the order they arrive. While this method is straightforward, it may not be optimal for resource utilization or for handling time-sensitive tasks. Shortest Job First (SJF) prioritizes tasks with the shortest expected execution time, effectively reducing the average waiting time for all tasks. However, this can lead to the starvation of longer tasks if shorter ones continue to arrive. Priority Scheduling assigns tasks based on predetermined priority levels, ensuring that critical tasks receive immediate attention. This approach, though, can result in lower-priority tasks being neglected. Lastly, the Round Robin (RR) algorithm allocates fixed time slices to each task in a cyclic order, promoting fairness by giving all tasks equal opportunity to utilize resources. This method can increase overhead due to the frequent context switching between tasks, potentially impacting performance.

2.5 Batch Processing vs. Continuous Processing

Understanding the nature of computational tasks is essential for effective resource allocation in data centers, where tasks are generally categorized into batch processing and continuous processing. Batch processing involves executing a series of tasks collectively without manual intervention, making it suitable for non-interactive, time-insensitive tasks such as data analysis, report generation, and large-scale computations. This approach offers advantages like resource efficiency—allowing scheduling during off-peak hours to optimize utilization—and cost reduction by timing operations to coincide with lower energy rates or peak availability of renewable energy sources. It also simplifies management by reducing the complexity of real-time resource allocation. However, batch processing is not suitable for tasks requiring immediate results due to latency tolerance and requires careful scheduling to maximize resource utilization without impacting other operations.

In contrast, continuous processing handles tasks that require immediate processing, such as real-time data analytics, online transaction processing, and streaming services. Its advantages include low latency, providing immediate responses essential for user-facing applications; scalability through dynamic resource allocation to handle fluctuating workloads; and high availability to ensure services remain accessible at all times. The considerations for continuous processing involve higher resource demands, as it requires constant resource availability, potentially increasing operational costs. It also necessitates complex management with sophisticated scheduling and monitoring systems to maintain optimal performance.

2.6 Energy Management in Data Centers

Energy consumption constitutes a significant operational cost for data centers and has considerable environmental implications, making effective energy management strategies crucial for sustainable operations. A fundamental metric for assessing data center energy efficiency is Power Usage Effectiveness (PUE), defined as the ratio of total facility energy consumption to the energy consumed by IT equipment alone; a PUE value closer to 1 indicates higher efficiency. To optimize energy usage, data centers employ several techniques. Dynamic Voltage and Frequency Scaling (DVFS) adjusts the voltage and frequency of processors based on workload demands, reducing energy consumption during periods of low activity. Server virtualization consolidates multiple virtual servers onto a single physical server, optimizing hardware utilization and reducing the number of active physical servers required. Implementing efficient cooling systems further enhances energy efficiency: free cooling utilizes external environmental conditions, such as cool air or water, to lessen reliance on energy-intensive cooling systems, while hot/cold aisle containment organizes server racks to separate hot and cold air flows, improving cooling efficiency. Additionally, integrating renewable energy sources like solar or wind power reduces the carbon footprint and dependence on non-renewable energy sources. Effective workload management also plays a pivotal role in energy efficiency. Energy-aware scheduling incorporates energy consumption metrics into scheduling algorithms to balance performance with energy usage. Workload shifting transfers computational tasks to data centers in regions with lower energy costs or cooler climates, leveraging geographical advantages. Lastly, idle resource management powers down or places idle servers into low-power states to minimize unnecessary energy consumption. Together, these strategies contribute to more sustainable and cost-effective data center operations.

In the following section, we present three models along with various scenarios. We then select two models for evaluation using two synthetic datasets. Lastly, we conclude the study and provide appendices for additional analysis.

3 Models

3.1 First Model

The primary objective is to minimize the total completion time of tasks while efficiently allocating resources. The ILP model will focus on the following objective function:

$$\sum_{i \in T} C_i, \quad \forall i \in T$$

Decision Variables

Let:

x_{ij} be a binary decision variable where:

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is assigned to server } j, \\ 0, & \text{otherwise.} \end{cases}$$

C_i is a variable and represents the completion time of task i.

Parameters

- T is the set of tasks, $i \in T$
- S is the set of servers, $j \in S$
- p_{ij} is the processing time required to complete task i on server j. This is a fixed parameter based on server capabilities and task requirements. For instance, we have an estimate that task A takes 5 hours on my laptop (Server 1) but only 1 hour on a powerful university PC (Server 2).
- r_j is the resource capacity of server j
- d_i is the resource demand of task i

3.2 Scenarios and Constraints

Scenario 1: Single Resource Type

In this scenario, we consider a data center where tasks require a single type of resource, such as CPU or GPU.

Constraints

Assignment Constraint: Each task must be assigned to exactly one server.

$$\sum_{j \in S} x_{ij} = 1, \quad \forall i \in T$$

Resource Capacity Constraint: The total resource demand on each server must not exceed its capacity.

$$\sum_{i \in T} d_i x_{ij} \leq r_j, \quad \forall j \in S$$

Completion Time Calculation: The completion time for each task is determined by its processing time on the assigned server.

$$C_i = \sum_{j \in S} p_{ij} x_{ij}, \quad \forall i \in T$$

Scenario 2: Multiple Resource Types

Here, tasks require multiple types of resources (such as CPU, memory, and storage).

Additional Parameters

- R is the set of resource types, $k \in R$
- d_{ik} is the demand of resource k by task i
- r_{jk} is the capacity of resource k on server j

New Constraints

Resource Capacity Constraints: For each resource type, the total demand must not exceed the server's capacity.

$$\sum_{i \in T} d_{ik} x_{ij} \leq r_{jk}, \quad \forall j \in S, \forall k \in R$$

Scenario 3: Batch Scheduling with Dynamic Batch Sizes and Durations

This model allows batches to have variable sizes and durations, optimizing the schedule based on task requirements. The model has a non-linear constraint (constraint 6).

Sets and Indices

- T: Set of tasks, indexed by i.
- S: Set of servers, indexed by j.
- K: Set of batches (cycles), indexed by k.

Parameters

- p_{ij}: Processing time required to complete task i on server j.
- d_i: Resource demand of task i.
- r_j: Resource capacity of server j.

Decision Variables

- $x_{ijk} \in \{0, 1\}$: Binary variable equal to 1 if task i is assigned to server j in batch k; 0 otherwise.
- $s_k \geq 0$: Continuous variable representing the start time of batch k.
- $D_k \geq 0$: Continuous variable representing the duration of batch k.
- $C_i \geq 0$: Continuous variable representing the completion time of task i.

Objective Function

Minimize the total completion times of all tasks:

$$\min \quad \sum_{i \in T} C_i \tag{1}$$

Constraints

1. Assignment Constraint Each task must be assigned to exactly one server in one batch:

$$\sum_{j \in S} \sum_{k \in K} x_{ijk} = 1, \quad \forall i \in T \tag{2}$$

2. Resource Capacity Constraints For each server in each batch, the total resource demand must not exceed its capacity:

$$\sum_{i \in T} d_i x_{ijk} \leq r_j, \quad \forall j \in S, \ \forall k \in K \tag{3}$$

3. Batch Duration Constraints The duration of each batch must cover the processing times of the tasks assigned to it:

$$D_k \geq p_{ij} x_{ijk}, \quad \forall i \in T, \ \forall j \in S, \ \forall k \in K \tag{4}$$

4. Batch Sequencing Constraints Batches are processed sequentially; the start time of the next batch begins after the previous one ends:

$$s_{k+1} \geq s_k + D_k, \quad \forall k \in K \tag{5}$$

5. Batch Start Time The first batch starts at or after time zero:

$$s_1 \geq 0 \tag{6}$$

6. Completion Time Calculation (non-linear) The completion time of each task is the start time of its batch plus its processing time:

$$C_i = \sum_{j \in S} \sum_{k \in K} (s_k + p_{ij}) \, x_{ijk}, \quad \forall i \in T \tag{7}$$

Scenario 4: Batch Scheduling with Fixed Batch Sizes and Durations

This model uses predetermined batch sizes and durations, providing a structured scheduling framework.

Additional Parameters

- D_k: Fixed duration of batch k (given).
- m: Maximum number of tasks per batch (fixed batch size).

Decision Variables

- $x_{ijk} \in \{0, 1\}$: Binary variable equal to 1 if task i is assigned to server j in batch k; 0 otherwise.
- $s_k \geq 0$: Continuous variable representing the start time of batch k.
- $C_i \geq 0$: Continuous variable representing the completion time of task i.

Objective Function

Minimize the total completion times of all tasks:

$$\min \quad \sum_{i \in T} C_i \tag{8}$$

Constraints

1. Assignment Constraint Each task must be assigned to exactly one server in one batch:

$$\sum_{j \in S} \sum_{k \in K} x_{ijk} = 1, \quad \forall i \in T \tag{9}$$

2. Resource Capacity Constraints For each server in each batch, the total resource demand must not exceed its capacity:

$$\sum_{i \in T} d_i x_{ijk} \leq r_j, \quad \forall j \in S, \ \forall k \in K \tag{10}$$

3. Batch Duration Constraints The processing time of any task assigned to a batch must not exceed the fixed duration of that batch:

$$p_{ij} x_{ijk} \leq D_k, \quad \forall i \in T, \ \forall j \in S, \ \forall k \in K \tag{11}$$

4. Batch Size Constraints The number of tasks assigned to each batch must not exceed the maximum batch size m:

$$\sum_{i \in T} \sum_{j \in S} x_{ijk} \leq m, \quad \forall k \in K \tag{12}$$

5. Batch Sequencing Constraints Batches are processed sequentially; the start time of the next batch begins after the fixed duration of the current batch:

$$s_{k+1} \geq s_k + D_k, \quad \forall k \in K \tag{13}$$

6. Batch Start Time The first batch starts at or after time zero:

$$s_1 \geq 0 \tag{14}$$

7. Completion Time Calculation The completion time of each task is the start time of its batch plus its processing time:

$$C_i = \sum_{j \in S} \sum_{k \in K} (s_k + p_{ij}) \, x_{ijk}, \quad \forall i \in T \tag{15}$$

3.3 Second Model

The original objective function aims to minimize the total completion time of tasks. We can augment this by adding:

- **Energy Consumption Minimization**: Reduce the total energy consumption of servers.
- **Load Balancing**: Distribute tasks evenly across servers to prevent overloading.
- **Task Prioritization**: Prioritize critical tasks by assigning weights.

The new objective function becomes:

$$\text{Minimize} \sum_{i \in T} w_i C_i + \alpha \sum_{j \in S} E_j + \beta \sum_{j \in S} L_j$$

Where:

- w_i is the priority weight of task i.
- E_j is the energy consumption of server j.
- L_j is a load balancing term for server j.
- α and β are scaling coefficients.

New Decision Variables and Parameters

Decision Variables

- y_j is a binary variable where:

$$y_j = \begin{cases} 1, & \text{if server } j \text{ is active,} \\ 0, & \text{otherwise.} \end{cases}$$

Parameters

- e_j is the energy consumption rate of server j.
- w_i is the priority weight of task i.
- M is a large constant used in constraints.

Additional Constraints

Server Activation Constraint: A server must be activated if any task is assigned to it.

$$\sum_{i \in T} x_{ij} \leq M y_j, \quad \forall j \in S$$

Load Balancing Constraints: Ensure that the number of tasks assigned to each server is within a certain range.

$$L_j = \left| \sum_{i \in T} x_{ij} - \frac{|T|}{|S|} \right|, \quad \forall j \in S$$

Task Precedence Constraints: Some tasks must be completed before others can start.

$$C_i + S_{ij} \leq C_k, \quad \forall (i,k) \in P$$

Where:

- P is the set of task pairs with precedence relations.
- S_{ij} is the setup time between tasks i and k.

Time Window Constraints: Tasks must start and finish within specific time windows.

$$s_i \leq C_i \leq f_i, \quad \forall i \in T$$

Where:

- s_i is the earliest start time of task i.
- f_i is the latest finish time of task i.

3.4 Third Model

To create a more accurate and efficient resource allocation model for data centers, we can include additional separate load balancing terms for each critical resource, incorporating utilization ratios. This enhancement allows us to balance the load of each resource type (such as CPU, GPU, Memory, Storage) across servers, ensuring that no single resource becomes a bottleneck. The updated model builds upon the original objective function, which aims to minimize the total completion time of tasks, energy consumption, and load imbalance.

New Objective Function

The new objective function is:

$$\text{Minimize } Z = \sum_{i \in T} w_i \sum_{j \in S} p_{i,j} x_{i,j} + \alpha \sum_{j \in S} e_j \left(\sum_{i \in T} (p_{i,j} + S_{i,j}) x_{i,j} \right) + \beta \sum_{j \in S} \sum_{k \in R} L_{j,k}$$

To better show it:

$$\text{Minimize } Z = \sum_{i \in T} w_i C_i + \alpha \sum_{j \in S} E_j + \beta \sum_{j \in S} \sum_{k \in R} L_{j,k}$$

Where:

- w_i: Priority weight of task i.
- C_i: Completion time of task i.
- E_j: Energy consumption of server j.
- e_j: is the energy consumption rate of server j
- p_{ij}: Processing time required to complete task i on server j.
- $L_{j,k}$: Load imbalance term for server j for each resource k in the set of resources R.
- S_{ij}: is the setup time between tasks i and k.
- α, β: Scaling coefficients.
- T: Set of tasks.
- S: Set of servers.
- R: Set of resources.

Definitions

- **Resources (R)**: The set of key resources (GPU, CPU, Memory, Storage).
- **Utilization Ratio ($U_{j,k}$)**: The utilization of resource k on server j.
- **Average Utilization ($U_{\mathbf{avg},k}$)**: The average utilization of resource k across all servers.
- **Load Imbalance ($L_{j,k}$)**: The absolute difference between $U_{j,k}$ and $U_{\text{avg},k}$.

Parameters

- $d_{i,k}$: Demand of resource k by task i.
- $r_{j,k}$: Capacity of resource k on server j.
- e_j: Energy consumption rate of server j.
- $p_{i,j}$: Processing time of task i on server j.
- w_i: Priority weight of task i.
- M: A large constant for server activation constraints.
- α, β: Scaling coefficients.

Decision Variables

- $x_{i,j} \in \{0,1\}$: Assignment of task i to server j.
- $y_j \in \{0,1\}$: Server activation indicator.
- $C_i \geq 0$: Completion time of task i.
- $E_j \geq 0$: Energy consumption of server j.
- $U_{j,k} \geq 0$: Utilization of resource k on server j.
- $L_{j,k} \geq 0$: Load imbalance of resource k on server j.

Constraints

1. Assignment Constraint

Each task must be assigned to exactly one server:

$$\sum_{j \in S} x_{i,j} = 1, \quad \forall i \in T$$

2. Server Activation Constraint

A server must be activated if any task is assigned to it:

$$\sum_{i \in T} x_{i,j} \leq M y_j, \quad \forall j \in S$$

3. Resource Capacity Constraints

For each resource k, the total demand on a server cannot exceed its capacity:

$$\sum_{i \in T} d_{i,k} x_{i,j} \leq r_{j,k} y_j, \quad \forall j \in S, \forall k \in R$$

4. Completion Time Calculation

The completion time for each task is determined by its processing time on the assigned server:

$$C_i = \sum_{j \in S} p_{i,j} x_{i,j}, \quad \forall i \in T$$

5. Energy Consumption Calculation

The total energy consumption of a server over the period processing a set of sequential tasks (setup time included):

$$E_j = e_j \sum_{i \in T} (p_{i,j} + S_{i,j}) x_{i,j}, \quad \forall j \in S$$

6. Utilization Ratio Calculation

Calculate the utilization of each resource on each server:

$$U_{j,k} = \frac{\sum_{i \in T} d_{i,k} x_{i,j}}{r_{j,k}}, \quad \forall j \in S, \forall k \in R$$

7. Average Utilization Calculation

Compute the average utilization for each resource across all servers:

$$U_{\text{avg},k} = \frac{\sum_{j \in S} \sum_{i \in T} d_{i,k} x_{i,j}}{\sum_{j \in S} r_{j,k}}, \quad \forall k \in R$$

8. Load Imbalance Constraints

Calculate the load imbalance for each resource on each server:

$$L_{j,k} \geq |U_{j,k} - U_{\text{avg},k}|, \quad \forall j \in S, \forall k \in R$$

Alternatively:

$$L_{j,k} \geq U_{j,k} - U_{\text{avg},k}, \quad \forall j \in S, \forall k \in R$$

$$L_{j,k} \geq U_{\text{avg},k} - U_{j,k}, \quad \forall j \in S, \forall k \in R$$

4 Assessment of Optimization Models

In this study, we examined Model One, Scenario Three to evaluate the impact of batching on the optimal completion time for each server. Our objective was to determine whether the model could effectively assign tasks with varying demands to servers with different resource capacities across a set of predetermined batches. In this scenario, both the duration and size of each batch are unknown and are determined by the model itself. The model's sole issue is a non-linear constraint, which imposes certain limitations on solver selection.

Additionally, we analyzed Model Three to investigate how a diverse range of resources, energy efficiency, and load balancing influence the model's task assignment strategy. This model operates without the complexities associated with batching, allowing us to assess task assignments in a more straightforward context.

We applied these optimization models to two distinct datasets to evaluate their effectiveness in enhancing data center operations. Each model requires a specific dataset, and given the computational resources available for this project, we carefully considered the size of each dataset accordingly.

First Experiment

The dataset for Model One, Scenario Three comprises 60 tasks, 4 servers, and 4 batches. Each server's capacity is a randomly assigned integer between 20 and 30, while each task's demand is a randomly assigned value between 3 and 8. Predicted processing time of task on each server is also provided. This setup allows the model to determine the optimal assignment of tasks to servers within the specified batches.

For this specific application, each solver presents its own advantages and disadvantages. Through a trial-and-error approach, we ultimately selected the solver that offered the lowest computation time. The solver chosen for this model is FilMINT, which is based on the LP/NLP algorithm developed by Quesada and Grossmann and implemented within a branch-and-cut framework. FilMINT was developed by a group of scientists, Kumar Abhishek, Sven Leyffer, and Jeff Linderoth. We utilized NEOS Server to run these experiments.

Here is a brief summary of model 1 scenario 3 statistics.

Table I: Model 1 Scenario 3 Statistics

Parameter	**Value**
Number of Constraints	1,040
Number of Variables	1,029
Number of Continuous Variables	69
Number of Binary Variables	960

The FilMINT solver employed a robust Branch and Cut methodology integrated with Sequential Quadratic Programming (filterSQP) to efficiently identify the optimal solution for the given Mixed-Integer Nonlinear Programming (MINLP) problem. Initially, the solver performed presolve operations to eliminate redundant constraints, thereby simplifying the problem structure. FilMINT strategically handled the 60 nonlinear constraints by solving a single NLP relaxation, leveraging filterSQP to manage nonlinearity while maintaining a linear objective function for enhanced computational efficiency. The solver generated seven lifted knapsack covers as cutting planes to tighten the feasible region, effectively reducing the solution space without extensive cut generation. Additionally, active primal heuristics and bound improvement techniques facilitated rapid convergence

by quickly identifying feasible solutions and refining objective bounds. Notably, the Branch and Bound process concluded at the root node with a tree depth of zero, indicating that the optimal solution was attained without further branching. Overall, FilMINT demonstrated high efficiency by solving the problem within 0.44 seconds, underscoring its capability to handle large-scale MINLPs through a combination of advanced preprocessing, selective cut generation, and effective heuristic strategies.

The files included in Appendix A are model1.mod, data1.dat, and job1.run, all written in AMPL. Additionally, the data generator file, written in Python (datagen1.py), is attached in Appendix A for further analysis.

The results show that the model works fine and correctly assign tasks in various batches to each server based on the optimum processing time. The result table is shown in Figure 1. The complete table is attached in Appendix A. Figure 2 is also provided to show the completion time for each task. By allowing the model to determine batch sizes and durations dynamically, we observe that tasks are grouped in batches that minimize idle times and reduce total completion time. Figure 3 shows the distribution of tasks across batches, highlighting how the model leverages batching to enhance scheduling efficiency. The model considers the varying processing times of tasks on different servers, assigning tasks to servers where they can be completed more quickly. This strategic assignment contributes to the minimization of total completion time, as depicted in Figure 5, which shows the assigned processing time by server and batch.

Task	t1	t10	t11	t12	t13	t14	t15	t16	t17	t18	...	t55	t56	t57	t58	t59	t6	t60	t7	t8	t9
Server	s3	s4	s3	s1	s1	s3	s3	s4	s4	s4	...	s1	s1	s1	s2	s2	s3	s1	s2	s2	s2
Batch	2	1	3	4	4	2	1	1	3	1	...	2	1	1	3	2	1	1	1	2	1

Figure 1: A sample of tasks assigned to each server in each batch

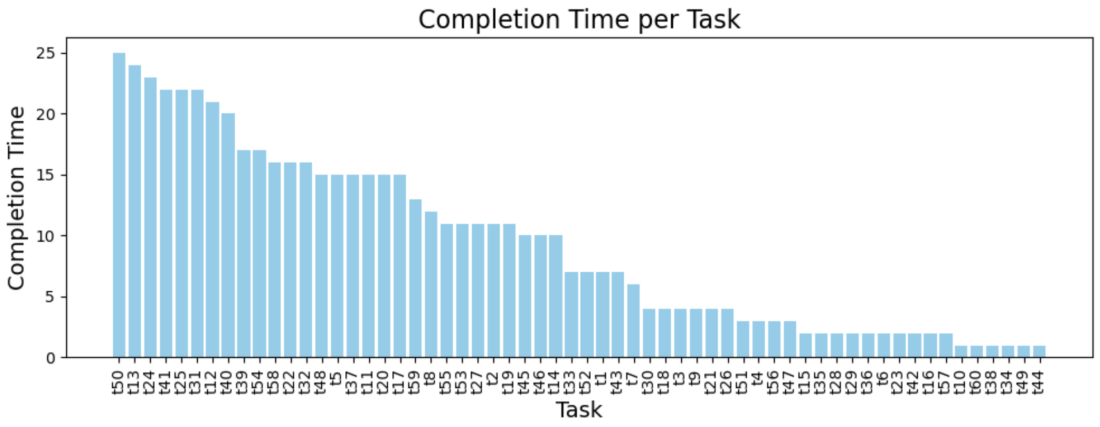

Figure 2: Completion time for each task

Second Experiment

The dataset for Model 3 includes 60 tasks, 4 servers, and 4 resources: CPU, GPU, Memory, and Storage. The processing time for tasks is randomly assigned as an integer between 1 and 10, while setup time ranges from 0 to 3. The resource demand for each task ranges from 12 to 25, and the available resources for each server fall between 50 and 100. Task priority weights are randomly assigned as integers between 1 and 5. Additionally, the energy consumption rate for each server is randomly selected within the range of 5 to 15.

To address this model, we employed Cplex which is a well known model for solving mixed integer linear programing. We used IBM ILOG CPLEX Optimizer provided by NEOS to implement this experiment. A summary of model 3 statistics is provided in table 2.

The CPLEX solver version 22.1.1.0 efficiently tackled the Mixed-Integer Programming (MIP) problem by utilizing its advanced multi-threaded capabilities, employing four threads to enhance computational performance. The solver implemented a primal simplex algorithm, executing 154 MIP simplex iterations to methodically explore the feasible region and optimize the objective

Figure 3: Distribution of tasks across batches

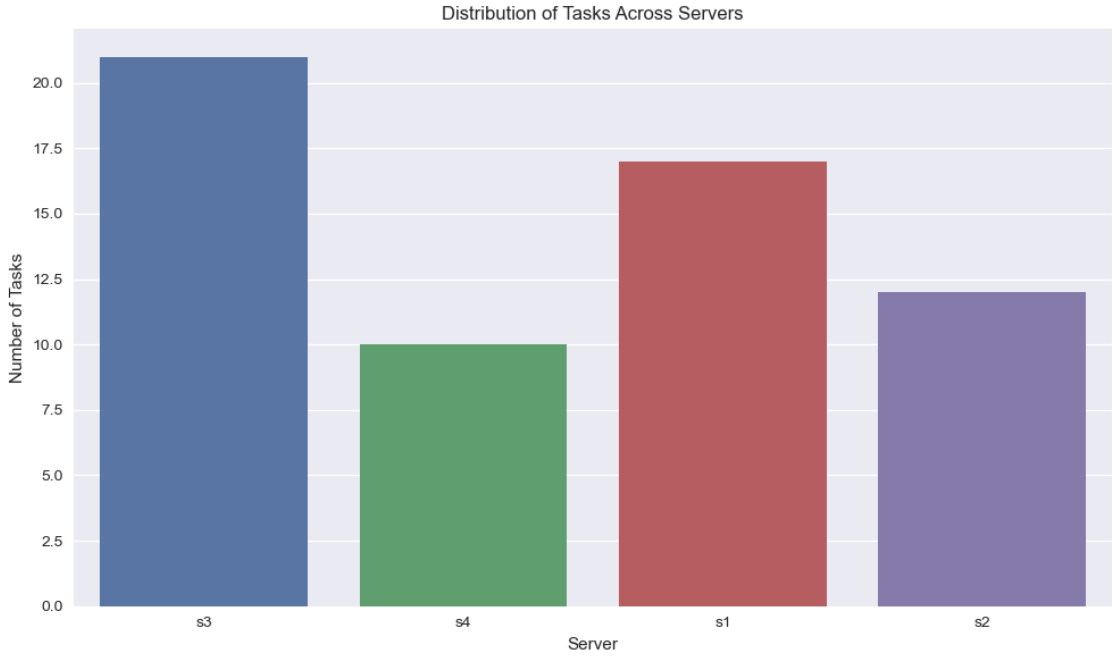

Figure 4: Distribution of tasks across servers

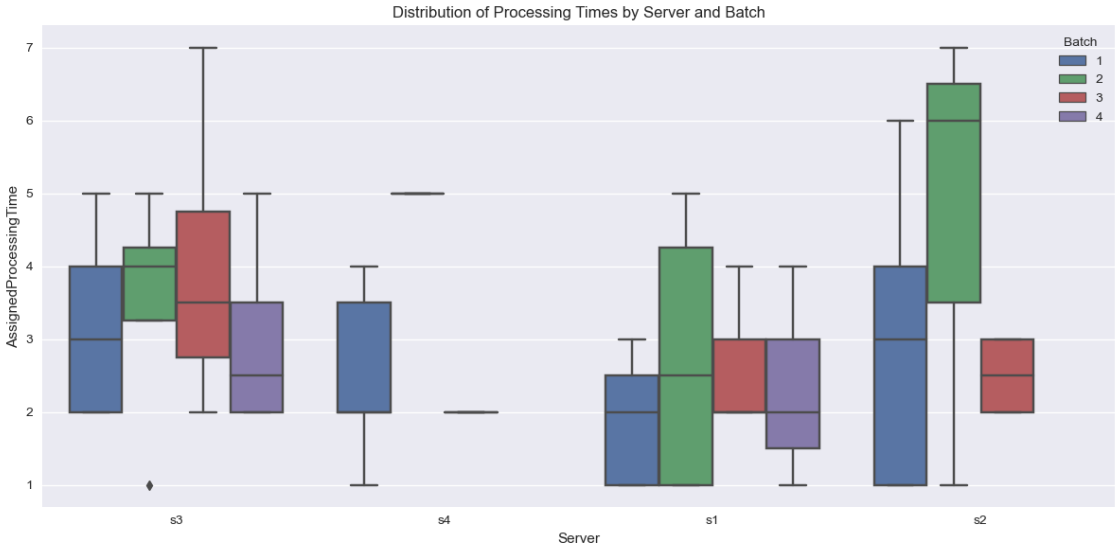

Figure 5: Distribution of assigned processing time by server and batch

Table II: Model 3 Statistics

Parameter	**Value**
Number of Equality Constraints	140
Number of Inequality Constraints	52
Number of Linear Variables	96
Number of Binary Variables	244

function, ultimately achieving an optimal integer solution with an objective value of 1728.301497 in less than two seconds. Notably, the optimization process concluded without initiating any branch-and-bound nodes, indicating that the linear programming relaxation of the problem was either inherently integer-feasible or that CPLEX's sophisticated presolving and cutting-plane techniques were sufficiently effective in identifying the optimal solution without the need for further branching. The essential files associated with this model, including model2.mod, data2.dat, and job2.run, are provided in Appendix B. These files are all written in AMPL, ensuring consistency and ease of use within the modeling environment. Additionally, we have included the Python-based data generator script, datagen2.py, in Appendix B to support further analysis and replication of our results. Figure 6 presents the results for Model 3, highlighting the outcomes of our optimization efforts and offering insights into the performance and efficiency of task assignments across servers based on task priorities. The model achieves a balanced utilization of critical resources (CPU, GPU, Memory, and Storage) across all servers. Figure 7 demonstrates that the load imbalance for each resource is minimized, preventing any single resource from becoming a bottleneck. By incorporating energy consumption into the objective function, the model assigns tasks in a way that reduces the total energy usage. Servers with lower energy consumption rates are preferred for tasks with higher processing times, leading to an overall reduction in energy expenditure. The inclusion of task priority weights ensures that critical tasks are prioritized in the scheduling process. This is reflected in Figure 8, where high-priority tasks are assigned to servers capable of completing them more efficiently.

Comparing the results from both experiments, several insights emerge:

- Model Complexity versus Scalability: The nonlinear constraints in Model One (constraint 6), while allowing for dynamic batching, increase computational complexity and limit scalability. In contrast, the linearized Model Three handles larger datasets more efficiently, making it more suitable for real-world applications where computational resources may be limited.

- Flexibility in Scheduling: Model One offers greater flexibility in scheduling through dynamic batching, which can be advantageous in environments with highly variable workloads. However, this flexibility comes at the cost of increased computational overhead.

- Comprehensive Resource Management: Model Three's incorporation of multiple resource

types and energy considerations provides a more holistic approach to resource management. This model effectively balances the utilization of different resources and aligns with sustainability objectives by minimizing energy consumption.

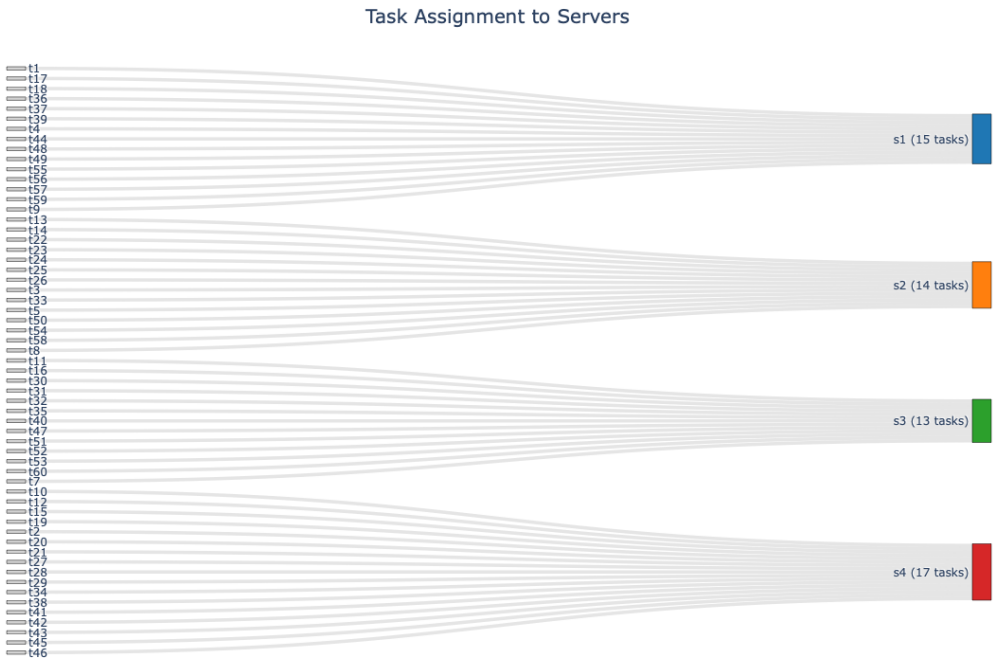

Figure 6: Task assignment to servers

5 Conclusion

In this project, we developed several optimization models aimed at enhancing task assignments within data center operations. Specifically, we proposed a couple of distinct models and applied two of them to synthetic datasets to evaluate their performance and effectiveness. Through these applications, we observed that the inherent nonlinearity within the models had a detrimental impact on both computation time and overall efficiency. The complex nonlinear constraints not only increased the computational burden, making the models less practical for larger datasets, but also resulted in out-of-disk or out-of-memory errors when scaling up, thereby further limiting their applicability in real-world, large-scale environments.

We analyzed various models designed for the first scenario, which involved nonlinear constraints. Our analysis revealed that the lack of proper documentation often creates confusion, limiting researchers' ability to fully leverage these models. Among the models tested, FilMINT demonstrated the best performance for the first scenario. However, we remain uncertain whether the limitations we encountered arose from the data generation process or from resource constraints on the NEOS server—particularly limited disk space for less commonly used solvers like FilMINT—which prevented us from testing larger datasets. In contrast, the model used in the second experiment, which was linear, posed fewer challenges when applied to large-scale datasets.

For future projects, we recommend adopting a strategic approach that combines the strengths of the two models explored. By linearizing these models, we can simplify complex relationships and reduce computational overhead, thereby improving both speed and efficiency. Linearized models are typically more tractable and faster to solve, making them suitable for real-time applications and larger datasets. Integrating the two models would address resource limitations on servers, enhance energy efficiency, and accommodate task prioritization. Additionally, distributing tasks across different batches would pave the way for more efficient models and improved overall performance.

Additionally, we propose applying these refined models to real-world datasets sourced from industry leaders such as Google and Microsoft. Utilizing actual data from these organizations will

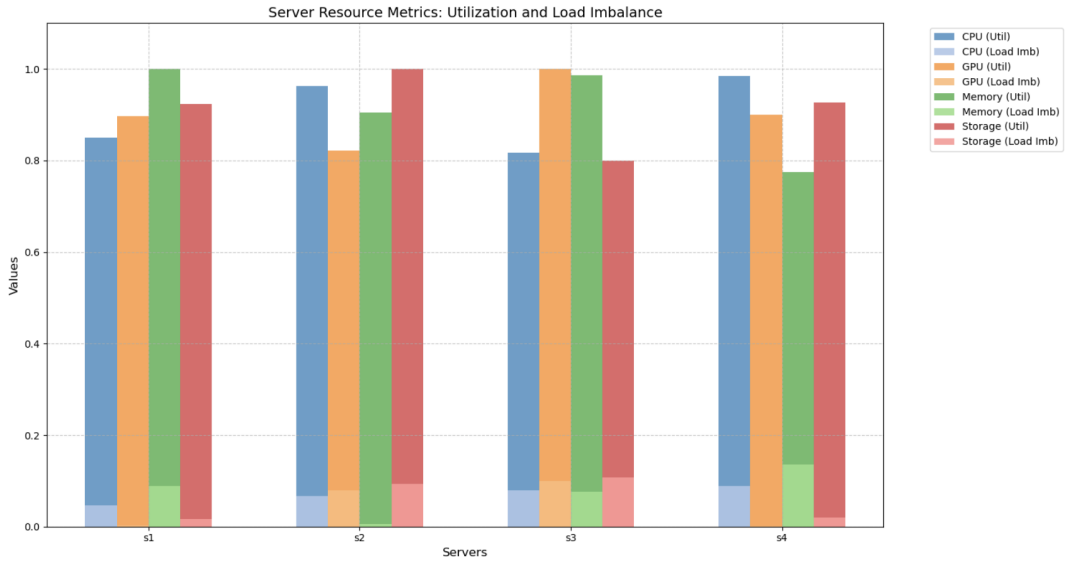

Figure 7: Server utilization and load imbalance for each server

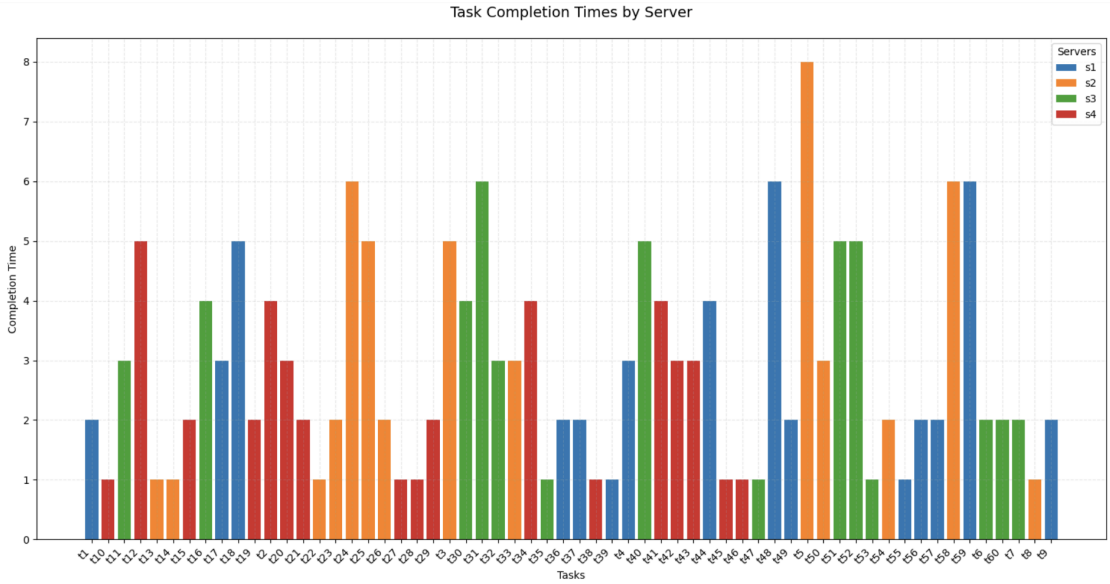

Figure 8: Completion time of assigned tasks on servers

provide a more accurate assessment of the models' applicability and performance in practical, large-scale environments. This real-world testing is essential for validating the models' effectiveness and ensuring they can meet the operational demands of modern data centers. Ultimately, this approach will contribute to the development of more robust and efficient task assignment solutions, leading to improved data center management and resource utilization.

6 References

- Barroso, L. A., Hölzle, U. (2009). The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers.
- Beloglazov, A., Abawajy, J., Buyya, R. (2012). "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing." Future Generation Computer Systems, 28(5), 755-768.
- Abhishek, K., Leyffer, S., and Linderoth, J. T. 2010. FilMINT: An Outer-Approximation-Based Solver for Nonlinear Mixed Integer Programs. INFORMS Journal on Computing 22: 555-567. DOI:10.1287/ijoc.1090.0373.
- Quesada, I. and I. E. Grossmann. 1992. An LP/NLP based branch-and-bound algorithm for convex MINLP optimization problems. Computers and Chemical Engineering 16: 937-947.
- IBM. 2023. IBM ILOG CPLEX Optimization Studio. Version 22.1.1.0. IBM. www.ibm.com/products/ilog-cplex-optimization-studio.
- Koomey, J. G. (2011). "Growth in data center electricity use 2005 to 2010." Analytics Press.
- Kliazovich, D., Bouvry, P., Khan, S. U. (2010). "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers." The Journal of Supercomputing, 62(3), 1263-1283.
- Garg, S. K., Yeo, C. S., Anandasivam, A., Buyya, R. (2011). "Energy-efficient scheduling of HPC applications in cloud computing environments." Computing, 91(9), 1199-1219.
- OpenAI. (2024). Generative Pre-trained Transformer (November 7 Version). Retrieved from https://chat.openai.com
- Anthropic. (2024). Claude [Large Language Model]. Retrieved from https://www.anthropic.com/claude
- Overleaf. (2024). [Online LaTeX Editor]. Retrieved from https://www.overleaf.com
- Google Inc. (2019). Google Cluster Data V3.
- Fourer, R., Gay, D. M., & Kernighan, B. W. (2002). AMPL: A Modeling Language for Mathematical Programming (2nd ed.). Duxbury Press/Brooks/Cole Publishing Company.
- AMPL Optimization Inc. (2024). AMPL [Mathematical Programming Software]. Retrieved from https://ampl.com
- NEOS Server. (2024). Wisconsin Institute for Discovery at the University of Wisconsin in Madison. Retrieved from https://neos-server.org
- Vidyarthi, D. P., & Bhattacharya, B. (2008). Scheduling in distributed computing systems: Analysis, design and models. Springer.
- Marinescu, D. C. (2013). Cloud computing: Theory and practice. Morgan Kaufmann.
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Retrieved from https://matplotlib.org/
- The pandas development team. (2023).Zenodo. https://doi.org/10.5281/zenodo.3509134
- Plotly Technologies Inc. (2023). Plotly: Open-source graphing library for Python. Retrieved from https://plotly.com/python/
- Microsoft. (2021). Azure Data Center Workload Dataset. Microsoft Research. Retrieved from https://github.com/Azure/azure-datacenter-workload-dataset
- GitHub. (2024). GitHub Copilot. GitHub. Retrieved from https://github.com/features/copilot
- CBRE Group, Inc. (2022). Data Center Solutions Market Update. Retrieved from `https://www.cbre.com/`.
- Uptime Institute. (2022). Global Data Center Survey. Retrieved from `https://uptimeinstitute.com/`.

- Cushman & Wakefield. (2022). Global Data Center Market Comparison. Retrieved from https://www.cushmanwakefield.com/.
- Statista. (2023). Number of data centers worldwide from 2015 to 2022, with forecasts until 2025. Retrieved from https://www.statista.com/.
- Turner & Townsend. (2022). Data Center Cost Index. Retrieved from https://www.turnerandtownsend.com/.
- Google. (2023). Google Data Centers. Retrieved from https://www.google.com/about/datacenters/.
- Microsoft. (2023). Inside Microsoft's Datacenters. Retrieved from https://www.microsoft.com/en-us/datacenters.
- Amazon Web Services (AWS). (2023). AWS Cloud Products. Retrieved from https://aws.amazon.com/products/.
- Microsoft Azure. (2023). Azure Products by Category. Retrieved from https://azure.microsoft.com/.
- Google Cloud. (2023). Google Cloud Services. Retrieved from https://cloud.google.com/products/.
- IDC (International Data Corporation). (2021). Worldwide Big Data and Analytics Software Forecast. Retrieved from https://www.idc.com/.
- Gartner. (2022). Emerging AI and ML Use Cases in Data Centers. Retrieved from https://www.gartner.com/.
- Akamai Technologies. (2023). Content Delivery Network (CDN) Services. Retrieved from https://www.akamai.com/.
- Cisco. (2023). Internet of Things (IoT). Retrieved from https://www.cisco.com/.
- Bank for International Settlements (BIS). (2021). Technology in Financial Services. Retrieved from https://www.bis.org/.
- International Energy Agency (IEA). (2021). Data Centres and Data Transmission Networks. Retrieved from https://www.iea.org/reports/data-centres-and-data-transmission-networks.
- Koomey, J. (2011). Growth in Data Center Electricity Use 2005 to 2010. Analytics Press. Retrieved from http://www.analyticspress.com/datacenters.html.
- ASHRAE Technical Committee 9.9. (2021). Thermal Guidelines for Data Processing Environments. Retrieved from https://www.ashrae.org/.
- ENISA (European Union Agency for Cybersecurity). (2022). Cyber Threat Landscape Report. Retrieved from https://www.enisa.europa.eu/.
- European Commission. (2020). General Data Protection Regulation (GDPR). Retrieved from https://gdpr.eu/.
- Supply Chain Management Review. (2021). Data Center Supply Chain Challenges. Retrieved from https://www.scmr.com/.
- Cardosa, M., Singh, A., Mirhoseini, A., & Bruno, J. (2009). Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. IEEE Cloud.
- Beloglazov, A., Buyya, R., Lee, Y. C., & Zomaya, A. (2011). A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. Advances in Computers, Elsevier.
- Gmach, D., Rolia, J., Cherkasova, L., & Kemper, A. (2009). Resource Pool Management: Reactive versus Proactive or Let's be Friends. IEEE Computer Society.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.

- Xu, J., & Fortes, J. A. B. (2010). Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. IEEE/ACM International Conference on Green Computing and Communications.

- Mishra, M., & Sahoo, A. (2011). On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach. IEEE Cloud.

- Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1), 107-113.

- Kumar, V., Grama, A., Gupta, A., & Karypis, G. (2003). Introduction to Parallel Computing. Addison-Wesley.

- The Green Grid. (2020). Green Grid Data Center Power Efficiency Metrics: PUE and DCiE. Retrieved from `https://www.thegreengrid.org/`.

- Fan, X., Weber, W.-D., & Barroso, L. A. (2007). Power Provisioning for a Warehouse-sized Computer. ACM SIGARCH Computer Architecture News, 35(2), 13-23.

- Beloglazov, A., & Buyya, R. (2012). Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. Concurrency and Computation: Practice and Experience, 24(13), 1397-1420.

- ASHRAE Datacom Series. (2015). Liquid Cooling Guidelines for Datacom Equipment Centers. American Society of Heating, Refrigerating and Air-Conditioning Engineers.

- Shehabi, A., Smith, S. J., Masanet, E., & Koomey, J. (2018). Data center growth in the United States: decoupling the demand for services from electricity use. Energy & Environmental Science, 11(3), 623-635.

- Google Sustainability. (2023). Our Data Centers: Efficiency and Sustainability. Retrieved from `https://sustainability.google/projects/data-centers/`.

7 Appendix A

- model 1 scenario 3 result

Table III: Model 1 Scenario 3 Results

Task	Demand	Server	Resource	Batch
t1	8	s3	30	2
t10	6	s4	28	1
t11	7	s3	30	3
t12	6	s1	24	4
t13	7	s1	24	4
t14	7	s3	30	2
t15	3	s3	30	1
t16	3	s4	28	1
t17	6	s4	28	3
t18	5	s4	28	1
t19	5	s4	28	2
t2	8	s4	28	2
t20	6	s2	22	3
t21	3	s3	30	1
t22	6	s3	30	3
t23	4	s4	28	1
t24	7	s3	30	4
t25	8	s1	24	4
t26	3	s3	30	1
t27	8	s4	28	2
t28	4	s4	28	1
t29	3	s3	30	1
t3	3	s4	28	1
t30	6	s3	30	1
t31	8	s3	30	4
t32	6	s2	22	3
t33	5	s1	24	2
t34	3	s1	24	1
t35	4	s3	30	1
t36	3	s3	30	1
t37	8	s1	24	3
t38	3	s2	22	1
t39	8	s3	30	3
t4	4	s2	22	1
t40	7	s3	30	3
t41	7	s3	30	4
t42	3	s1	24	1
t43	4	s2	22	2
t44	3	s1	24	1
t45	7	s1	24	2
t46	8	s3	30	2
t47	4	s1	24	1
t48	5	s2	22	3
t49	5	s2	22	1
t5	6	s1	24	3
t50	8	s3	30	4
t51	4	s3	30	1
t52	3	s1	24	2
t53	6	s3	30	2
t54	6	s1	24	3
t55	8	s1	24	2

Continued on next page

Table III: Model 1 Scenario 3 Results

Task	Demand	Server	Resource	Batch
t56	3	s1	24	1
t57	3	s1	24	1
t58	5	s2	22	3
t59	6	s2	22	2
t6	3	s3	30	1
t60	3	s1	24	1
t7	5	s2	22	1
t8	8	s2	22	2
t9	4	s2	22	1

- model1.mod

```
# Scenario 3
# Batch Scheduling with Dynamic Batch Sizes and Durations

# Sets and Indices
set T;  # Set of tasks
set S;  # Set of servers
param N integer > 0;  # Number of batches
set K;  # Set of batches

# Parameters
param p {T, S} >= 0;  # Processing time required to complete task i on server j
param d {T} >= 0;     # Resource demand of task i
param r {S} >= 0;     # Resource capacity of server j

# Decision Variables
var x {T, S, K} binary;  # 1 if task i is assigned to server j in batch k
var s {K} >= 0;          # Start time of batch k
var D {K} >= 0;          # Duration of batch k
var C {T} >= 0;          # Completion time of task i

# Objective Function
minimize TotalCompletionTime:
    sum {i in T} C[i];

# Constraints

# 1. Assignment Constraint
subject to Assignment {i in T}:
    sum {j in S, k in K} x[i,j,k] = 1;

# 2. Resource Capacity Constraints
subject to ResourceCapacity {j in S, k in K}:
    sum {i in T} d[i] * x[i,j,k] <= r[j];

# 3. Batch Duration Constraints
subject to BatchDuration {i in T, j in S, k in K}:
    D[k] >= p[i,j] * x[i,j,k];

# 4. Batch Sequencing Constraints
subject to BatchSequencing {k in K: k < N}:
    s[k+1] >= s[k] + D[k];

# 5. Batch Start Time
subject to BatchStartTime:
    s[1] >= 0;

# 6. Completion Time Calculation
subject to CompletionTime {i in T}:
    C[i] = sum {j in S, k in K} (s[k] + p[i,j]) * x[i,j,k];
```

- data1.dat

```
set T := t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17
    t18 t19 t20 t21 t22 t23 t24 t25 t26 t27 t28 t29 t30 t31 t32 t33
    t34 t35 t36 t37 t38 t39 t40 t41 t42 t43 t44 t45 t46 t47 t48 t49
    t50 t51 t52 t53 t54 t55 t56 t57 t58 t59 t60;
set S := s1 s2 s3 s4;
param N := 4;
set K := 1 2 3 4;

param p : s1 s2 s3 s4 :=
t1 7 7 1 5
t2 9 8 7 5
```

```
t3 8 6 10 4
t4 9 3 5 3
t5 2 10 5 9
t6 10 3 5 2
t7 2 6 8 9
t8 2 6 7 6
t9 10 4 9 8
t10 8 9 5 1
t11 9 1 2 7
t12 1 10 8 6
t13 4 6 2 4
t14 10 4 4 3
t15 9 8 2 2
t16 6 9 8 2
t17 5 9 5 2
t18 9 6 9 4
t19 10 9 10 5
t20 8 2 10 7
t21 6 10 4 5
t22 3 4 3 1
t23 10 5 8 2
t24 2 3 3 1
t25 2 9 7 9
t26 5 9 4 4
t27 10 7 10 5
t28 8 8 6 2
t29 6 10 2 8
t30 10 6 4 4
t31 1 5 2 4
t32 6 3 6 7
t33 1 2 3 4
t34 1 10 9 10
t35 2 1 2 4
t36 10 10 2 7
t37 2 6 2 1
t38 10 1 4 3
t39 2 8 4 1
t40 1 9 7 10
t41 2 5 2 4
t42 2 5 6 7
t43 3 1 9 8
t44 1 10 2 7
t45 4 5 6 8
t46 10 3 4 1
t47 3 3 6 9
t48 5 2 10 8
t49 3 1 8 7
t50 10 9 5 6
t51 7 5 3 9
t52 1 8 2 6
t53 1 9 5 3
t54 4 8 6 10
t55 5 6 10 10
t56 3 5 7 7
t57 2 1 10 4
t58 6 3 4 4
t59 8 7 10 7
t60 1 7 10 7
;

param d :=
t1 8
t2 8
t3 3
t4 4
t5 6
```

```
t6 3
t7 5
t8 8
t9 4
t10 6
t11 7
t12 6
t13 7
t14 7
t15 3
t16 3
t17 6
t18 5
t19 5
t20 6
t21 3
t22 6
t23 4
t24 7
t25 8
t26 3
t27 8
t28 4
t29 3
t30 6
t31 8
t32 6
t33 5
t34 3
t35 4
t36 3
t37 8
t38 3
t39 8
t40 7
t41 7
t42 3
t43 4
t44 3
t45 7
t46 8
t47 4
t48 5
t49 5
t50 8
t51 4
t52 3
t53 6
t54 6
t55 8
t56 3
t57 3
t58 5
t59 6
t60 3
;

param r :=
s1 24
s2 22
s3 30
s4 28
;
```

- job1.run

```
# Load the model file
#model model1_scenario3.mod;

# Load the data file
#data data1.dat;


# Solve the model
solve;

# Display the decision variables and parameters
display x;
display s;
display D;
display C;
```

- datagen1.py

```


import random
import math

# Set random seed for reproducibility
random.seed(0)

num_tasks = 60
num_servers = 4
#num_batches = 8
batch_duration = 10


tasks = [f"t{i}" for i in range(1, num_tasks + 1)]
servers = [f"s{j}" for j in range(1, num_servers + 1)]


# Generate processing times
processing_times = {}
for t in tasks:
    processing_times[t] = {}
    for s in servers:
        processing_times[t][s] = random.randint(1, 10)

# Generate resource demands
resource_demands = {t: random.randint(3, 8) for t in tasks}

# Generate resource capacities
resource_capacities = {s: random.randint(20, 30) for s in servers}

# Compute total processing time (minimum across servers for each task)
total_processing_time = 0
for t in tasks:
    min_p = min(processing_times[t][s] for s in servers)
    total_processing_time += min_p
# Compute total processing capacity per batch
processing_capacity_per_batch = len(servers) * batch_duration

# Estimate minimum number of batches based on processing time
min_batches_processing = math.ceil(total_processing_time /
    processing_capacity_per_batch)

# Compute total resource demand
total_resource_demand = sum(resource_demands[t] for t in tasks)

# Compute total resource capacity per batch
```

```
total_resource_capacity_per_batch = sum(resource_capacities[s] for s in servers)

# Estimate minimum number of batches based on resource capacities
min_batches_resource = math.ceil(total_resource_demand / total_resource_capacity_per_batch)

# Compute the overall minimum number of batches required
min_batches_required = max(min_batches_processing, min_batches_resource)

# Print the minimum number of batches required
#print(f"Minimum number of batches required: {min_batches_required}")

batches = [str(k) for k in range(1, min_batches_required + 1)]
# Write data to AMPL data file
with open('data_synthetic_dec1.dat', 'w') as f:
    # Write sets
    f.write('set T := ' + ' '.join(tasks) + ';\n')
    f.write('set S := ' + ' '.join(servers) + ';\n')
    f.write(f'param N := {min_batches_required};\n')
    f.write('set K := ' + ' '.join(batches) + ';\n\n')
    #f.write(f'param D0 := {batch_duration};\n')

    # Write processing times
    f.write('param p : ' + ' '.join(servers) + ' :=\n')
    for t in tasks:
        f.write(t + ' ')
        for s in servers:
            f.write(str(processing_times[t][s]) + ' ')
        f.write('\n')
    f.write(';\n\n')

    # Write resource demands
    f.write('param d :=\n')
    for t in tasks:
        f.write(t + ' ' + str(resource_demands[t]) + '\n')
    f.write(';\n\n')

    # Write resource capacities
    f.write('param r :=\n')
    for s in servers:
        f.write(s + ' ' + str(resource_capacities[s]) + '\n')
    f.write(';\n')
```

8 Appendix B

- model2.mod

```
# ----------------------------------------------------
# Optimization Model3
# Objective: Minimize weighted completion time, energy consumption, and load imbalance
# ----------------------------------------------------

# Sets
set T;      # Set of Tasks
set S;      # Set of Servers
set R;      # Set of Resources

# Parameters
param w {T};                    # Priority weight of task i
param e {S};                    # Energy consumption rate of server j
```

```
param p {T, S};                    # Processing time of task i on server j
param S_time {T, S};               # Setup time between task i and server j
param d {T, R};                    # Demand of resource k by task i
param r {S, R};                    # Capacity of resource k on server j
param alpha;                       # Scaling coefficient for energy consumption
param beta;                        # Scaling coefficient for load imbalance
param M;                           # Large constant for server activation constraints

# Precomputed Parameters
param sum_r_k {k in R} := sum {j in S} r[j, k];  # Sum of capacities for each resource

# Decision Variables
var x {T, S} binary;               # Assignment of task i to server j
var y {S} binary;                  # Server activation indicator
var C {T} >= 0;                    # Completion time of task i
var E {S} >= 0;                    # Energy consumption of server j
var U {S, R} >= 0;                 # Utilization of resource k on server j
var L {S, R} >= 0;                 # Load imbalance of resource k on server j

# Objective Function
minimize Z:
    sum {i in T} w[i] * C[i]
    + alpha * sum {j in S} E[j]
    + beta * sum {j in S, k in R} L[j, k];

# Constraints

# 1. Assignment Constraint
subject to Assignment {i in T}:
    sum {j in S} x[i, j] = 1;

# 2. Server Activation Constraint
subject to ServerActivation {j in S}:
    sum {i in T} x[i, j] <= M * y[j];

# 3. Resource Capacity Constraints
subject to ResourceCapacity {j in S, k in R}:
    sum {i in T} d[i, k] * x[i, j] <= r[j, k] * y[j];

# 4. Completion Time Calculation
subject to CompletionTime {i in T}:
    C[i] = sum {j in S} p[i, j] * x[i, j];

# 5. Energy Consumption Calculation
subject to EnergyConsumption {j in S}:
    E[j] = e[j] * sum {i in T} (p[i, j] + S_time[i, j]) * x[i, j];

# 6. Utilization Ratio Calculation
subject to UtilizationRatio {j in S, k in R}:
    U[j, k] = sum {i in T} d[i, k] * x[i, j] / r[j, k];

# 7. Load Imbalance Constraints
subject to LoadImbalanceUpper {j in S, k in R}:
    L[j, k] >= U[j, k] - (sum {j2 in S} sum {i in T} d[i, k] * x[i, j2] / sum_r_k[k]);

subject to LoadImbalanceLower {j in S, k in R}:
    L[j, k] >= (sum {j2 in S} sum {i in T} d[i, k] * x[i, j2] /
```

```
        sum_r_k[k]) - U[j, k];
```

- data2.dat

```
# ----------------------------------------------------
# Data File for model 3
# Generated by datagen2.py
# ----------------------------------------------------

# Sets
set T := t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17
    t18 t19 t20 t21 t22 t23 t24 t25 t26 t27 t28 t29 t30 t31 t32 t33
    t34 t35 t36 t37 t38 t39 t40 t41 t42 t43 t44 t45 t46 t47 t48 t49
    t50 t51 t52 t53 t54 t55 t56 t57 t58 t59 t60;
set S := s1 s2 s3 s4;
set R := GPU CPU Memory Storage;

# Parameters
# Priority weight of each task
param w :=
    t1 5
    t2 2
    t3 3
    t4 2
    t5 2
    t6 3
    t7 5
    t8 2
    t9 5
    t10 2
    t11 1
    t12 4
    t13 3
    t14 3
    t15 1
    t16 2
    t17 5
    t18 1
    t19 5
    t20 4
    t21 5
    t22 3
    t23 1
    t24 4
    t25 3
    t26 1
    t27 5
    t28 1
    t29 2
    t30 4
    t31 3
    t32 3
    t33 5
    t34 2
    t35 4
    t36 5
    t37 4
    t38 3
    t39 3
    t40 4
    t41 2
    t42 2
    t43 3
    t44 2
    t45 2
    t46 3
```

```
    t47 4
    t48 2
    t49 2
    t50 5
    t51 2
    t52 2
    t53 3
    t54 5
    t55 2
    t56 3
    t57 3
    t58 4
    t59 1
    t60 2
;

# Energy consumption rate of each server
param e :=
    s1 12
    s2 10
    s3 8
    s4 10
;

# Processing time of each task on each server
param p :=
    t1 s1 2
    t1 s2 10
    t1 s3 7
    t1 s4 5
    t2 s1 5
    t2 s2 9
    t2 s3 7
    t2 s4 4
    t3 s1 10
    t3 s2 5
    t3 s3 5
    t3 s4 5
    t4 s1 3
    t4 s2 5
    t4 s3 8
    t4 s4 10
    t5 s1 7
    t5 s2 8
    t5 s3 9
    t5 s4 7
    t6 s1 6
    t6 s2 8
    t6 s3 2
    t6 s4 6
    t7 s1 7
    t7 s2 5
    t7 s3 2
    t7 s4 6
    t8 s1 6
    t8 s2 1
    t8 s3 5
    t8 s4 7
    t9 s1 2
    t9 s2 3
    t9 s3 4
    t9 s4 5
    t10 s1 3
    t10 s2 2
    t10 s3 5
    t10 s4 1
```

```
    t11 s1 8
    t11 s2 2
    t11 s3 3
    t11 s4 3
    t12 s1 6
    t12 s2 7
    t12 s3 9
    t12 s4 5
    t13 s1 10
    t13 s2 1
    t13 s3 3
    t13 s4 4
    t14 s1 5
    t14 s2 1
    t14 s3 6
    t14 s4 2
    t15 s1 8
    t15 s2 4
    t15 s3 9
    t15 s4 2
    t16 s1 9
    t16 s2 6
    t16 s3 4
    t16 s4 7
    t17 s1 3
    t17 s2 10
    t17 s3 10
    t17 s4 10
    t18 s1 5
    t18 s2 8
    t18 s3 10
    t18 s4 10
    t19 s1 8
    t19 s2 3
    t19 s3 6
    t19 s4 2
    t20 s1 4
    t20 s2 10
    t20 s3 7
    t20 s4 3
    t21 s1 2
    t21 s2 7
    t21 s3 4
    t21 s4 2
    t22 s1 1
    t22 s2 1
    t22 s3 3
    t22 s4 8
    t23 s1 3
    t23 s2 2
    t23 s3 4
    t23 s4 6
    t24 s1 5
    t24 s2 6
    t24 s3 8
    t24 s4 6
    t25 s1 7
    t25 s2 5
    t25 s3 8
    t25 s4 9
    t26 s1 10
    t26 s2 2
    t26 s3 7
    t26 s4 6
    t27 s1 4
    t27 s2 4
```

```
    t27 s3 3
    t27 s4 1
    t28 s1 5
    t28 s2 3
    t28 s3 7
    t28 s4 1
    t29 s1 4
    t29 s2 4
    t29 s3 9
    t29 s4 2
    t30 s1 4
    t30 s2 8
    t30 s3 4
    t30 s4 7
    t31 s1 8
    t31 s2 5
    t31 s3 6
    t31 s4 10
    t32 s1 8
    t32 s2 7
    t32 s3 3
    t32 s4 5
    t33 s1 5
    t33 s2 3
    t33 s3 3
    t33 s4 9
    t34 s1 5
    t34 s2 8
    t34 s3 7
    t34 s4 4
    t35 s1 7
    t35 s2 3
    t35 s3 1
    t35 s4 2
    t36 s1 2
    t36 s2 3
    t36 s3 5
    t36 s4 5
    t37 s1 2
    t37 s2 8
    t37 s3 7
    t37 s4 7
    t38 s1 5
    t38 s2 8
    t38 s3 3
    t38 s4 1
    t39 s1 1
    t39 s2 6
    t39 s3 7
    t39 s4 9
    t40 s1 7
    t40 s2 9
    t40 s3 5
    t40 s4 7
    t41 s1 3
    t41 s2 9
    t41 s3 2
    t41 s4 4
    t42 s1 6
    t42 s2 7
    t42 s3 7
    t42 s4 3
    t43 s1 9
    t43 s2 8
    t43 s3 9
    t43 s4 3
```

```
t44 s1 4
t44 s2 4
t44 s3 5
t44 s4 6
t45 s1 7
t45 s2 1
t45 s3 1
t45 s4 1
t46 s1 4
t46 s2 5
t46 s3 3
t46 s4 1
t47 s1 8
t47 s2 4
t47 s3 1
t47 s4 6
t48 s1 6
t48 s2 6
t48 s3 7
t48 s4 9
t49 s1 2
t49 s2 5
t49 s3 7
t49 s4 3
t50 s1 4
t50 s2 3
t50 s3 5
t50 s4 9
t51 s1 9
t51 s2 7
t51 s3 5
t51 s4 9
t52 s1 10
t52 s2 5
t52 s3 5
t52 s4 8
t53 s1 9
t53 s2 10
t53 s3 1
t53 s4 10
t54 s1 1
t54 s2 2
t54 s3 10
t54 s4 1
t55 s1 1
t55 s2 6
t55 s3 8
t55 s4 5
t56 s1 2
t56 s2 9
t56 s3 8
t56 s4 4
t57 s1 2
t57 s2 6
t57 s3 6
t57 s4 7
t58 s1 10
t58 s2 6
t58 s3 8
t58 s4 8
t59 s1 6
t59 s2 9
t59 s3 5
t59 s4 10
t60 s1 10
t60 s2 8
```

```
    t60 s3 2
    t60 s4 6
;

# Setup time between tasks on servers
param S_time :=
    t1 s1 0
    t1 s2 2
    t1 s3 0
    t1 s4 0
    t2 s1 1
    t2 s2 3
    t2 s3 1
    t2 s4 3
    t3 s1 0
    t3 s2 0
    t3 s3 2
    t3 s4 2
    t4 s1 0
    t4 s2 0
    t4 s3 0
    t4 s4 2
    t5 s1 2
    t5 s2 1
    t5 s3 3
    t5 s4 3
    t6 s1 0
    t6 s2 1
    t6 s3 2
    t6 s4 0
    t7 s1 1
    t7 s2 0
    t7 s3 0
    t7 s4 1
    t8 s1 0
    t8 s2 1
    t8 s3 1
    t8 s4 2
    t9 s1 1
    t9 s2 1
    t9 s3 2
    t9 s4 3
    t10 s1 2
    t10 s2 3
    t10 s3 0
    t10 s4 2
    t11 s1 2
    t11 s2 2
    t11 s3 0
    t11 s4 3
    t12 s1 1
    t12 s2 3
    t12 s3 3
    t12 s4 1
    t13 s1 3
    t13 s2 0
    t13 s3 2
    t13 s4 2
    t14 s1 3
    t14 s2 0
    t14 s3 0
    t14 s4 0
    t15 s1 2
    t15 s2 3
    t15 s3 2
    t15 s4 0
```

```
t16 s1 1
t16 s2 3
t16 s3 1
t16 s4 0
t17 s1 0
t17 s2 3
t17 s3 2
t17 s4 2
t18 s1 0
t18 s2 1
t18 s3 2
t18 s4 3
t19 s1 1
t19 s2 2
t19 s3 1
t19 s4 1
t20 s1 3
t20 s2 1
t20 s3 0
t20 s4 2
t21 s1 3
t21 s2 1
t21 s3 2
t21 s4 0
t22 s1 2
t22 s2 0
t22 s3 0
t22 s4 3
t23 s1 2
t23 s2 0
t23 s3 2
t23 s4 3
t24 s1 1
t24 s2 1
t24 s3 0
t24 s4 2
t25 s1 3
t25 s2 2
t25 s3 1
t25 s4 1
t26 s1 0
t26 s2 3
t26 s3 2
t26 s4 3
t27 s1 0
t27 s2 1
t27 s3 1
t27 s4 3
t28 s1 2
t28 s2 0
t28 s3 2
t28 s4 2
t29 s1 3
t29 s2 1
t29 s3 2
t29 s4 3
t30 s1 2
t30 s2 3
t30 s3 2
t30 s4 2
t31 s1 1
t31 s2 3
t31 s3 0
t31 s4 0
t32 s1 3
t32 s2 3
```

```
    t32 s3 2
    t32 s4 0
    t33 s1 2
    t33 s2 0
    t33 s3 2
    t33 s4 2
    t34 s1 1
    t34 s2 1
    t34 s3 1
    t34 s4 0
    t35 s1 2
    t35 s2 2
    t35 s3 2
    t35 s4 2
    t36 s1 3
    t36 s2 3
    t36 s3 1
    t36 s4 2
    t37 s1 0
    t37 s2 3
    t37 s3 0
    t37 s4 2
    t38 s1 1
    t38 s2 1
    t38 s3 3
    t38 s4 0
    t39 s1 3
    t39 s2 1
    t39 s3 1
    t39 s4 1
    t40 s1 2
    t40 s2 3
    t40 s3 3
    t40 s4 1
    t41 s1 3
    t41 s2 0
    t41 s3 1
    t41 s4 0
    t42 s1 1
    t42 s2 3
    t42 s3 2
    t42 s4 2
    t43 s1 0
    t43 s2 0
    t43 s3 3
    t43 s4 0
    t44 s1 2
    t44 s2 3
    t44 s3 2
    t44 s4 3
    t45 s1 1
    t45 s2 0
    t45 s3 3
    t45 s4 3
    t46 s1 1
    t46 s2 0
    t46 s3 0
    t46 s4 2
    t47 s1 1
    t47 s2 3
    t47 s3 3
    t47 s4 0
    t48 s1 0
    t48 s2 0
    t48 s3 2
    t48 s4 0
```

```
    t49 s1 1
    t49 s2 2
    t49 s3 1
    t49 s4 2
    t50 s1 2
    t50 s2 2
    t50 s3 3
    t50 s4 3
    t51 s1 2
    t51 s2 2
    t51 s3 2
    t51 s4 2
    t52 s1 1
    t52 s2 0
    t52 s3 2
    t52 s4 0
    t53 s1 2
    t53 s2 3
    t53 s3 3
    t53 s4 1
    t54 s1 3
    t54 s2 1
    t54 s3 3
    t54 s4 0
    t55 s1 0
    t55 s2 3
    t55 s3 0
    t55 s4 3
    t56 s1 1
    t56 s2 2
    t56 s3 0
    t56 s4 1
    t57 s1 2
    t57 s2 1
    t57 s3 3
    t57 s4 0
    t58 s1 1
    t58 s2 3
    t58 s3 0
    t58 s4 1
    t59 s1 1
    t59 s2 0
    t59 s3 2
    t59 s4 3
    t60 s1 3
    t60 s2 0
    t60 s3 1
    t60 s4 3
;

# Demand of each resource by each task
param d :=
    t1 GPU 1
    t1 CPU 3
    t1 Memory 4
    t1 Storage 5
    t2 GPU 4
    t2 CPU 1
    t2 Memory 1
    t2 Storage 4
    t3 GPU 7
    t3 CPU 8
    t3 Memory 1
    t3 Storage 6
    t4 GPU 9
    t4 CPU 2
```

```
t4 Memory 1
t4 Storage 3
t5 GPU 8
t5 CPU 4
t5 Memory 1
t5 Storage 3
t6 GPU 8
t6 CPU 5
t6 Memory 4
t6 Storage 1
t7 GPU 9
t7 CPU 4
t7 Memory 5
t7 Storage 3
t8 GPU 8
t8 CPU 1
t8 Memory 9
t8 Storage 8
t9 GPU 2
t9 CPU 4
t9 Memory 8
t9 Storage 8
t10 GPU 9
t10 CPU 1
t10 Memory 9
t10 Storage 3
t11 GPU 7
t11 CPU 5
t11 Memory 2
t11 Storage 9
t12 GPU 9
t12 CPU 1
t12 Memory 7
t12 Storage 2
t13 GPU 5
t13 CPU 7
t13 Memory 8
t13 Storage 5
t14 GPU 4
t14 CPU 1
t14 Memory 8
t14 Storage 1
t15 GPU 1
t15 CPU 3
t15 Memory 1
t15 Storage 8
t16 GPU 1
t16 CPU 5
t16 Memory 1
t16 Storage 1
t17 GPU 1
t17 CPU 8
t17 Memory 3
t17 Storage 4
t18 GPU 2
t18 CPU 1
t18 Memory 10
t18 Storage 7
t19 GPU 7
t19 CPU 6
t19 Memory 4
t19 Storage 9
t20 GPU 2
t20 CPU 9
t20 Memory 8
t20 Storage 3
```

```
    t21 GPU 5
    t21 CPU 2
    t21 Memory 1
    t21 Storage 1
    t22 GPU 7
    t22 CPU 7
    t22 Memory 6
    t22 Storage 3
    t23 GPU 4
    t23 CPU 3
    t23 Memory 9
    t23 Storage 1
    t24 GPU 8
    t24 CPU 4
    t24 Memory 4
    t24 Storage 1
    t25 GPU 2
    t25 CPU 9
    t25 Memory 9
    t25 Storage 9
    t26 GPU 2
    t26 CPU 8
    t26 Memory 2
    t26 Storage 7
    t27 GPU 5
    t27 CPU 4
    t27 Memory 9
    t27 Storage 6
    t28 GPU 2
    t28 CPU 3
    t28 Memory 1
    t28 Storage 8
    t29 GPU 6
    t29 CPU 1
    t29 Memory 1
    t29 Storage 3
    t30 GPU 9
    t30 CPU 1
    t30 Memory 8
    t30 Storage 4
    t31 GPU 2
    t31 CPU 7
    t31 Memory 10
    t31 Storage 6
    t32 GPU 1
    t32 CPU 7
    t32 Memory 4
    t32 Storage 2
    t33 GPU 1
    t33 CPU 10
    t33 Memory 1
    t33 Storage 3
    t34 GPU 9
    t34 CPU 10
    t34 Memory 8
    t34 Storage 1
    t35 GPU 4
    t35 CPU 9
    t35 Memory 6
    t35 Storage 6
    t36 GPU 4
    t36 CPU 6
    t36 Memory 6
    t36 Storage 2
    t37 GPU 8
    t37 CPU 8
```

```
    t37 Memory 4
    t37 Storage 5
    t38 GPU 1
    t38 CPU 10
    t38 Memory 7
    t38 Storage 1
    t39 GPU 2
    t39 CPU 9
    t39 Memory 8
    t39 Storage 2
    t40 GPU 9
    t40 CPU 3
    t40 Memory 1
    t40 Storage 8
    t41 GPU 9
    t41 CPU 1
    t41 Memory 3
    t41 Storage 9
    t42 GPU 3
    t42 CPU 5
    t42 Memory 1
    t42 Storage 1
    t43 GPU 7
    t43 CPU 4
    t43 Memory 3
    t43 Storage 9
    t44 GPU 9
    t44 CPU 1
    t44 Memory 9
    t44 Storage 3
    t45 GPU 9
    t45 CPU 1
    t45 Memory 8
    t45 Storage 7
    t46 GPU 2
    t46 CPU 3
    t46 Memory 4
    t46 Storage 1
    t47 GPU 3
    t47 CPU 1
    t47 Memory 8
    t47 Storage 8
    t48 GPU 9
    t48 CPU 9
    t48 Memory 5
    t48 Storage 7
    t49 GPU 3
    t49 CPU 9
    t49 Memory 9
    t49 Storage 2
    t50 GPU 1
    t50 CPU 1
    t50 Memory 7
    t50 Storage 5
    t51 GPU 7
    t51 CPU 3
    t51 Memory 9
    t51 Storage 2
    t52 GPU 1
    t52 CPU 9
    t52 Memory 5
    t52 Storage 2
    t53 GPU 5
    t53 CPU 5
    t53 Memory 5
    t53 Storage 2
```

```
    t54 GPU 5
    t54 CPU 9
    t54 Memory 1
    t54 Storage 1
    t55 GPU 2
    t55 CPU 1
    t55 Memory 8
    t55 Storage 7
    t56 GPU 5
    t56 CPU 9
    t56 Memory 7
    t56 Storage 5
    t57 GPU 5
    t57 CPU 8
    t57 Memory 4
    t57 Storage 6
    t58 GPU 7
    t58 CPU 5
    t58 Memory 1
    t58 Storage 2
    t59 GPU 8
    t59 CPU 7
    t59 Memory 2
    t59 Storage 7
    t60 GPU 4
    t60 CPU 3
    t60 Memory 7
    t60 Storage 6
;

# Capacity of each resource on each server
param r :=
    s1 GPU 78
    s1 CPU 100
    s1 Memory 88
    s1 Storage 79
    s2 GPU 84
    s2 CPU 80
    s2 Memory 74
    s2 Storage 55
    s3 GPU 70
    s3 CPU 82
    s3 Memory 76
    s3 Storage 75
    s4 GPU 100
    s4 CPU 66
    s4 Memory 98
    s4 Storage 82
;

# Scaling coefficients and large constant
param alpha := 0.5;
param beta  := 0.3;
param M     := 1000;

‘
```

- job2.run

```
# Load the model file
#model model2.mod;

# Load the data file
#data data2.dat;


```

```
# Solve the model
solve;

# Display the decision variables and parameters
display x;
display C;
display E;
display U;
display L;
```

- datagen2.py

```
import random

def generate_ampl_data(num_tasks=60, num_servers=4,
    output_file='dataset_model3_nov30.dat'):
    # Define sets
    tasks = [f't{i}' for i in range(1, num_tasks + 1)]
    servers = [f's{j}' for j in range(1, num_servers + 1)]
    resources = ['GPU', 'CPU', 'Memory', 'Storage']

    # Initialize data structures
    w = {}
    e = {}
    p = {}
    S_time = {}
    d = {}
    r = {}

    # Priority weights
    for task in tasks:
        w[task] = random.randint(1, 5)

    # Energy consumption rates
    for server in servers:
        e[server] = random.randint(5, 15)

    # Resource capacities
    r = {server: {} for server in servers}
    for server in servers:
        for resource in resources:
            r[server][resource] = random.randint(50, 100)

    # Resource demands
    d = {task: {} for task in tasks}
    for task in tasks:
        for resource in resources:
            # Generate demands that are significantly less than
                capacities
            # This helps ensure that tasks can fit on servers
            max_demand = min(r[server][resource] for server in
                servers) // 4
            d[task][resource] = random.randint(1, max_demand)

    # Processing times and setup times
    p = {}
    S_time = {}
    for task in tasks:
        p[task] = {}
        S_time[task] = {}
        for server in servers:
            p[task][server] = random.randint(1, 10)
            S_time[task][server] = random.randint(0, 3)

    # Feasibility Checks
    # 1. Total Demand     Total Capacity for each resource
```

```
    total_demand = {resource: 0 for resource in resources}
    total_capacity = {resource: 0 for resource in resources}
    for resource in resources:
        total_demand[resource] = sum(d[task][resource] for task in
            tasks)
        total_capacity[resource] = sum(r[server][resource] for server
            in servers)
        if total_demand[resource] > total_capacity[resource]:
            print(f"Adjusting␣demands␣for␣resource␣{resource}␣to␣
                ensure␣feasibility.")
            # Scale down demands proportionally
            scaling_factor = total_capacity[resource] /
                total_demand[resource]
            for task in tasks:
                d[task][resource] = max(1, int(d[task][resource] *
                    scaling_factor))
            # Recalculate total demand
            total_demand[resource] = sum(d[task][resource] for task
                in tasks)

    # 2. Ensure each task can be assigned to at least one server
    for task in tasks:
        assignable = False
        for server in servers:
            can_assign = all(d[task][resource] <= r[server][resource]
                for resource in resources)
            if can_assign:
                assignable = True
                break
        if not assignable:
            print(f"Adjusting␣demands␣for␣task␣{task}␣to␣ensure␣it␣
                can␣be␣assigned.")
            # Adjust demands to fit the smallest capacity server
            for resource in resources:
                min_capacity = min(r[server][resource] for server in
                    servers)
                d[task][resource] = min(d[task][resource],
                    min_capacity)

    # Write data to file
    with open(output_file, 'w') as f:
        f.write("#␣
            ------------------------------------------------------\n")
        f.write("#␣Data␣File␣for␣Task␣Assignment␣to␣Servers␣Model\n")
        f.write("#␣Generated␣by␣generate_data.py\n")
        f.write("#␣
            ------------------------------------------------------\n\n")

        # Sets
        f.write("#␣Sets\n")
        f.write(f"set␣T␣:=␣{'␣'.join(tasks)};\n")
        f.write(f"set␣S␣:=␣{'␣'.join(servers)};\n")
        f.write(f"set␣R␣:=␣{'␣'.join(resources)};\n\n")

        # Parameters

        # Priority weights
        f.write("#␣Parameters\n")
        f.write("#␣Priority␣weight␣of␣each␣task\n")
        f.write("param␣w␣:=\n")
        for task in tasks:
            f.write(f"␣␣␣␣{task}␣{w[task]}\n")
        f.write(";\n\n")

        # Energy consumption rates
        f.write("#␣Energy␣consumption␣rate␣of␣each␣server\n")
```

```
        f.write("param e :=\n")
        for server in servers:
            f.write(f"    {server} {e[server]}\n")
        f.write(";\n\n")

        # Processing times
        f.write("# Processing time of each task on each server\n")
        f.write("param p :=\n")
        for task in tasks:
            for server in servers:
                f.write(f"    {task} {server} {p[task][server]}\n")
        f.write(";\n\n")

        # Setup times
        f.write("# Setup time between tasks on servers\n")
        f.write("param S_time :=\n")
        for task in tasks:
            for server in servers:
                f.write(f"    {task} {server}
                    {S_time[task][server]}\n")
        f.write(";\n\n")

        # Resource demands
        f.write("# Demand of each resource by each task\n")
        f.write("param d :=\n")
        for task in tasks:
            for resource in resources:
                f.write(f"    {task} {resource}
                    {d[task][resource]}\n")
        f.write(";\n\n")

        # Resource capacities
        f.write("# Capacity of each resource on each server\n")
        f.write("param r :=\n")
        for server in servers:
            for resource in resources:
                f.write(f"    {server} {resource}
                    {r[server][resource]}\n")
        f.write(";\n\n")

        # Scaling coefficients and large constant
        f.write("# Scaling coefficients and large constant\n")
        f.write("param alpha := 0.5;\n")
        f.write("param beta  := 0.3;\n")
        f.write("param M     := 1000;\n\n")

        f.write("#
            ----------------------------------------------------\n")
        f.write("# End of Data File\n")
        f.write("#
            ----------------------------------------------------\n")

    print(f"Data file '{output_file}' generated successfully with
        {num_tasks} tasks and {num_servers} servers.")

if __name__ == "__main__":
    generate_ampl_data()
```