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

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

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

May 21, 2025

Abstract

This study presents a series of optimization models for task assignment and resource allocation in data centers, aiming to minimize task completion time, energy consumption, and load imbalance. We developed and evaluated multiple models, including a nonlinear formulation that supports dynamic batching and a linear model that incorporates multi-resource constraints and energy-aware scheduling. Through extensive experiments on synthetic datasets, we observed that while the nonlinear model provides flexibility, it suffers from scalability issues due to computational complexity. In contrast, linearized models demonstrate stronger performance on larger instances, with reduced solver time and improved feasibility. Building on these insights, we introduced a fully linearized integrated model that combines dynamic batching with energy and resource management, offering the most robust results across various test conditions. Our analysis also highlights the importance of selecting an appropriate number of batches and the trade-offs it introduces in terms of model complexity and runtime. Additionally, we address practical challenges in experimentation, such as solver constraints on the NEOS platform and the need for automated job tracking. Future work will explore AI driven optimization techniques and apply the proposed models to real data from industry-scale data centers to further validate their applicability and effectiveness.

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. Mixed Integer Linear Programming (MILP) offers a mathematical approach to model and solve such optimization problems. This project focuses on formulating an MILP 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

2.1 Global Expansion 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 5,426 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 $7 million to $12 million per megawatt of commissioned IT load, depending on the facility's size and specifications. Hyperscale data centers developed by tech giants such as Amazon, Google, and Facebook can exceed $1 billion in construction costs. Operational expenses, including energy consumption, maintenance, staffing, and technological upgrades, can amount to millions of dollars annually. Notable projects like Google's $5.5 billion investment in its Council Bluffs, Iowa, data center and Microsoft's plans to invest approximately $80 billion in AI-enabled data centers during fiscal year 2025 underscore the significant financial commitment required. The construction timeline for a data center typically spans 12 to

36 months, influenced by factors such as the facility's scale, technological requirements, regulatory approvals, and site-specific challenges.

2.2 Data Centers and Cloud Computing: Challenges and Scheduling

Data centers form the essential backbone of the digital infrastructure, acting as global centers for secure data storage, internet connectivity, and digital activities. They facilitate pervasive applications across diverse domains, including consumer, scientific, and business sectors, often delivered through utility-oriented Cloud computing models based on pay-as-you-go principles. The demand for data processing, storage, and communication services provided by data centers is experiencing rapid growth. This growth is further accelerated by emerging data-intensive applications such as Artificial Intelligence (AI), the Internet of Things (IoT), Big Data analytics, and digital manufacturing. Data centers handle a wide variety of workloads, ranging from large-scale data-parallel applications common in enterprise computing, such as searching images or records and MapReduce jobs, to applications requiring real-time services and low latency, like web serving, search, interactive debugging, data streaming, and mobile applications. They also support high-performance computing tasks and event-triggered applications. These facilities must simultaneously manage workloads with vastly different requirements, from throughput-oriented batch processing (e.g., large-scale data analytics, indexing, video transcoding) to latency-sensitive continuous or real-time processing (e.g., web requests, financial transactions, interactive applications).

Operating at this scale and accommodating such diverse, dynamic workloads presents numerous significant challenges facing data centers today. One of the most prominent is their immense energy consumption, contributing substantially to high operational costs and significant carbon footprints. Energy efficiency has thus become a critical sustainability concern in the twenty-first century. Data centers can consume as much energy as 25,000 households, and the electricity sector, often reliant on fossil fuels, is a major source of CO_2 emissions, linking data centers directly to climate change concerns. A large portion of consumed electrical energy converts into heat, necessitating extensive cooling systems. The costs associated with cooling can be substantial, potentially annualizing to \$4–\$8 million for a medium-sized data center, and high temperatures can compromise system reliability and availability. Furthermore, dynamic workload fluctuations often lead to overprovisioning to ensure peak demand can be met, resulting in servers operating at low utilization levels. Servers with low utilization are energy inefficient on a per-computation basis because their substantial idle power loss (typically 10–70% of maximum power) is not offset by sufficient computational work. The challenge is compounded by the scale and heterogeneity of modern data center infrastructure and workloads. Ensuring reliable Quality of Service (QoS) and strict adherence to diverse Service Level Agreements (SLAs), sometimes with variable requirements, while simultaneously managing energy use and other costs, is a complex balancing act. Other challenges include significant communication and reconfiguration latency, maintaining system reliability and fault-tolerance in the face of frequent component failures, managing massive and diverse datasets, ensuring security and privacy, and increasingly, dealing with limited power availability which inhibits market growth.

To navigate these challenges, effective resource management and resource scheduling algorithms in data centers are paramount and represent active areas of research. The goal is often to improve efficiency, reduce costs, and ensure performance and reliability. Various optimization techniques in data centers and resource allocation models have been proposed, traditionally focusing on aspects like admission control, capacity allocation, load balancing, energy optimization, and QoS guarantees. Early work in energy management at the data center level focused on workload concentration and switching idle nodes off to minimize power consumption. Economic frameworks involving services "bidding" for resources have also been explored to balance usage cost and gained benefit. Control theory approaches, often using feedback loops, have been applied to coordinate power management policies. With the advent of virtualization, dynamic consolidation of Virtual Machines (VMs) onto a minimal set of physical nodes has become a productive technique for energy saving by allowing idle nodes to enter power-saving modes like sleep or hibernate. This approach leverages VM migration.

Cloud computing has become a transformative technology, revolutionizing the management and delivery of computational resources across various industries. These services are typically implemented within large-scale data centers, which house a vast number of servers, storage units, and networking infrastructure. As cloud infrastructure continues to grow and diversify, efficiently managing these resources and scheduling the numerous incoming jobs and tasks becomes a critical challenge.

The dynamic and heterogeneous nature of cloud environments presents significant complexities for scheduling algorithms. Key factors that influence scheduling decisions include the availability

and heterogeneity of resources, task priority, task demands, resource requirements, and dependencies among tasks. Cloud users often have specific QoS expectations, including constraints on task completion time and desired response times. Inefficient job scheduling can lead to excessive energy use, increasing operational costs and environmental impact. Thus, incorporating energy-efficient allocation strategies is a critical concern. Optimizing resource utilization, particularly by minimizing the number of active servers, is a key method for achieving energy savings. The complexity of these factors means that task scheduling in cloud computing systems is recognized as an NP-complete problem.

While traditional scheduling algorithms, such as First-Come-First-Serve or Round Robin, have been effective in simpler contexts, they often fail to meet the demanding and complex requirements of modern cloud infrastructures with their dynamic workloads and heterogeneous resources. These limitations necessitate the exploration and application of more advanced optimization techniques.

2.3 Core Objectives of Job Scheduling in Cloud Computing

Effective job scheduling mechanisms in cloud computing pursue a diverse yet interrelated set of objectives. Foremost is minimizing makespan, which shortens the total completion time of all tasks and boosts system throughput, an imperative for workloads with large task volumes (Kaur et al. 2022c; Raju et al. 2013). Hand in hand with this aim is maximizing resource utilization, ensuring that CPU, memory and storage are kept busy rather than idle, thereby driving down operational costs (Sharma and Sharma 2013; Loganathan and Mukherjee 2015). Schedulers must also safeguard fairness in resource allocation so that no user monopolizes shared infrastructure, a challenge in multi tenant settings with heterogeneous priorities (Wang et al. 2014b; Tang et al. 2020). Energy aware strategies complement these goals by curbing power draw through judicious scaling and elimination of idle periods (Hu et al. 2019, 2023), while SLA aware techniques keep task completion within agreed windows to avoid penalties and protect user satisfaction (Nayak et al. 2015; Islam et al. 2021). For latency sensitive domains such as online gaming or high frequency trading, reducing task latency is paramount (Memari et al. 2022; Aburukba et al. 2020). In distributed systems with heterogeneous resources, load balancing spreads work evenly to avert hotspots and bolster reliability (Wang et al. 2014a; Paul et al. 2011). Cost aware scheduling minimizes both allocation and data transfer expenses (Mansouri and Javidi 2020; Cheng et al. 2022a), while scalable algorithms accommodate rising workloads without sacrificing performance (Khallouli and Huang 2022; Alsaih et al. 2020). Robustness is maintained through fault tolerant rescheduling that mitigates hardware, network or software failures (Xu et al. 2023; Indhumathi et al. 2023). Data locality aware methods further cut congestion by assigning tasks near the data they require (Cheng et al. 2021; Mansouri and Javidi 2020). Dependency aware scheduling respects task precedence constraints to guarantee correct execution order and efficient usage (Chen et al. 2022b; Chung et al. 2020). Finally, multi objective optimization frameworks orchestrate trade offs among competing goals—such as jointly reducing makespan and enhancing fairness—using Pareto based techniques (Cui et al. 2023; Mohammadzadeh and Masdari 2023).

2.4 A Taxonomy of AI Driven Job Scheduling Techniques

Figure 9 in the recent survey (Tirmazi et al.) proposes a clear four branch taxonomy that helps position any data center scheduling approach on the broader AI landscape. At the top level it distinguishes methods that learn policies from data, those that search with AI inspired optimisation engines, classical heuristics enhanced by AI ideas, and finally hybrid frameworks that fuse AI with traditional exact algorithms. Machine learning based methods rely on statistical pattern discovery. They range from classic supervised pipelines that map workload features directly to scheduling decisions, to unsupervised clustering that groups tasks before allocation, through reinforcement learning agents that interact with the cluster in real time, and to modern deep learning models that ingest rich graphs or traces and output placements. AI based optimisation treats scheduling as a black box objective and uses advanced search engines to explore it. Evolutionary approaches mimic genetic evolution, swarm intelligence methods follow collective behaviour rules (ants, bees, particles), gradient based schemes exploit continuous relaxations when they exist, and other bio inspired variants borrow ideas from physics or ecology. AI based heuristic and meta heuristic techniques inject intelligence into well known constructive or local search heuristics. Some start with simple constructive rules but tune their parameters via learning; others use trajectory based meta heuristics such as simulated annealing or tabu search to escape local optima; population based meta heuristics—including genetic algorithms and differential evolution—maintain many candidate schedules and let them evolve. Hybrid AI based solutions combine the strengths of the

above categories. One prominent strand couples machine learning predictors with meta heuristics, using the former to guide or prune the latter's search. Another strand embeds AI modules inside classical scheduling algorithms—branch and bound or list scheduling—so that learned models provide branching priorities or dynamic cost estimates while the traditional core preserves worst case guarantees.

In the following section, we present four models along with various scenarios. We then select three models for evaluation using three 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:

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

Alternatively:

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

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

3.5 Fourth Model: Batch Scheduling with Multi-Resource Allocation

This model offers the best of both approaches by integrating a linearized version of Model 1, Scenario 3, and Model 3 into a single, compact formulation. It simultaneously enables dynamic batching and optimizes both resource utilization and energy efficiency. Specifically, the proposed optimization framework jointly addresses the scheduling of tasks in batches and the allocation of multiple resource types across a heterogeneous server pool. The objective is to minimize a composite cost function that incorporates weighted task completion times, server energy consumption, and resource load imbalance.

3.5.1 Objective Function

$$\min Z = \gamma \sum_{i \in T} w_i C_i + \alpha \sum_{j \in S} E_j + \beta \sum_{j \in S} \sum_{r \in R} L_{jr} \tag{16}$$

3.5.2 Constraints

$$\sum_{j \in S} \sum_{k \in K} x_{ijk} = 1 \qquad \forall i \in T \quad \text{(Task Assignment)} \tag{17}$$

$$D_k \geq p_{ij} \cdot x_{ijk} \qquad \forall i \in T, j \in S, k \in K \quad \text{(Batch Duration)} \tag{18}$$

$$s_{k+1} \geq s_k + D_k \qquad \forall k < N \quad \text{(Batch Sequencing)} \tag{19}$$

$$s_1 \geq 0 \qquad \text{(Batch Start Time)} \tag{20}$$

$$C\_batch_{ik} \geq s_k - M(1 - x_{ijk}) \qquad \forall i, j, k \quad \text{(Start Time Capture)} \tag{21}$$

$$C_i \geq C\_batch_{ik} + p_{ij} \cdot x_{ijk} \qquad \forall i, j, k \quad \text{(Completion Time)} \tag{22}$$

$$\sum_{i \in T} d_{ir} x_{ijk} \leq r_{jr} \cdot y_j \qquad \forall j, r, k \quad \text{(Resource Capacity)} \tag{23}$$

$$\sum_{i \in T} \sum_{k \in K} x_{ijk} \leq M y_j \qquad \forall j \quad \text{(Server Activation)} \tag{24}$$

$$E_j = e_j \cdot \sum_{i \in T} \sum_{k \in K} (p_{ij} + S\_time_{ij}) x_{ijk} \qquad \forall j \quad \text{(Energy Consumption)} \tag{25}$$

$$U_{jr} = \frac{\sum_{i \in T} \sum_{k \in K} d_{ir} x_{ijk}}{r_{jr}} \qquad \forall j, r \quad \text{(Utilization Ratio)} \tag{26}$$

$$L_{jr} \geq U_{jr} - \frac{\sum_{j' \in S} \sum_{i \in T} \sum_{k \in K} d_{ir} x_{ij'k}}{\text{sum\_r\_k}_r} \qquad \forall j, r \quad \text{(Load Imbalance Upper)} \tag{27}$$

$$L_{jr} \geq \frac{\sum_{j' \in S} \sum_{i \in T} \sum_{k \in K} d_{ir} x_{ij'k}}{\text{sum\_r\_k}_r} - U_{jr} \qquad \forall j, r \quad \text{(Load Imbalance Lower)} \tag{28}$$

3.5.3 Sets and Indices

- T : Set of tasks, indexed by i
- S : Set of servers, indexed by j
- $K = \{1, \ldots, N\}$: Set of batches, indexed by k
- R : Set of resource types, indexed by r

3.5.4 Parameters

- p_{ij} : Processing time of task i on server j
- S_time_{ij} : Setup time of task i on server j
- d_{ir} : Demand of resource r by task i
- r_{jr} : Capacity of resource r on server j
- e_j : Energy consumption rate of server j
- w_i : Priority weight of task i
- α, β, γ : Weighting coefficients for energy, load imbalance, and completion time in the objective function
- M : A sufficiently large constant used in big-M constraints
- $\text{sum\_r\_k}_r = \sum_{j \in S} r_{jr}$: Total capacity for resource r

3.5.5 Decision Variables

- $x_{ijk} \in \{0,1\}$: Equals 1 if task i is assigned to server j in batch k
- $s_k \geq 0$: Start time of batch k
- $D_k \geq 0$: Duration of batch k
- $C_i \geq 0$: Completion time of task i
- $C\_batch_{ik} \geq 0$: Start time of task i within batch k
- $y_j \in \{0,1\}$: Equals 1 if server j is active
- $E_j \geq 0$: Energy consumption of server j
- $U_{jr} \geq 0$: Utilization ratio of resource r on server j
- $L_{jr} \geq 0$: Load imbalance of resource r on server j

3.5.6 Ensuring Feasibility for Individual Tasks

To guarantee that each task can be assigned to at least one server, we first identify the maximum demand for each resource across all tasks. Specifically, for each resource $r \in R$, we compute

$$\text{maxDemand}[r] = \max_{t \in T} d[t, r].$$

We then assign server capacities such that for each server $j \in S$ and resource $r \in R$, the capacity $\text{cap}[j, r]$ is randomly selected from the interval

$$[\text{maxDemand}[r],\ \text{maxDemand}[r] + 6].$$

This ensures that every task t can be assigned to at least one server for every resource r, making the model feasible at the level of individual task-server assignments.

3.5.7 Determining the Number of Batches

To determine the number of batches N, we ensure that the total resource demand can be met across all servers and batches. For each resource $r \in R$, we compute:

$$\text{sum\_demand}[r] = \sum_{t \in T} d[t, r], \qquad \text{sum\_capacity}[r] = \sum_{j \in S} \text{cap}[j, r].$$

We then calculate the number of required batches for each resource as:

$$\text{neededBatches}_r = \left\lceil \frac{\text{sum\_demand}[r]}{\text{sum\_capacity}[r]} \right\rceil.$$

Finally, we define the number of batches N as:

$$N = \max\left(2,\ \max_{r \in R} \text{neededBatches}_r\right).$$

4 Assessment of Optimization Models

4.1 Overview of Experiments

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.

Finally, we analyzed the fourth model, which is linear and not only incorporates dynamic batching but also optimizes resource allocations and energy efficiency. This comprehensive formulation integrates the strengths of both Model One, Scenario Three, and Model Three, enabling flexible batch formation while accounting for heterogeneous server capabilities, energy consumption, and load balancing across multiple resources. By eliminating the non-linear constraint present in previous models, this version remains solver-friendly while offering a more realistic and scalable solution framework for complex scheduling environments.

We applied these optimization models to three distinct synthetic 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. We also discussed restrictions of applying model 4 on real dataset from Google data centers.

4.2 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

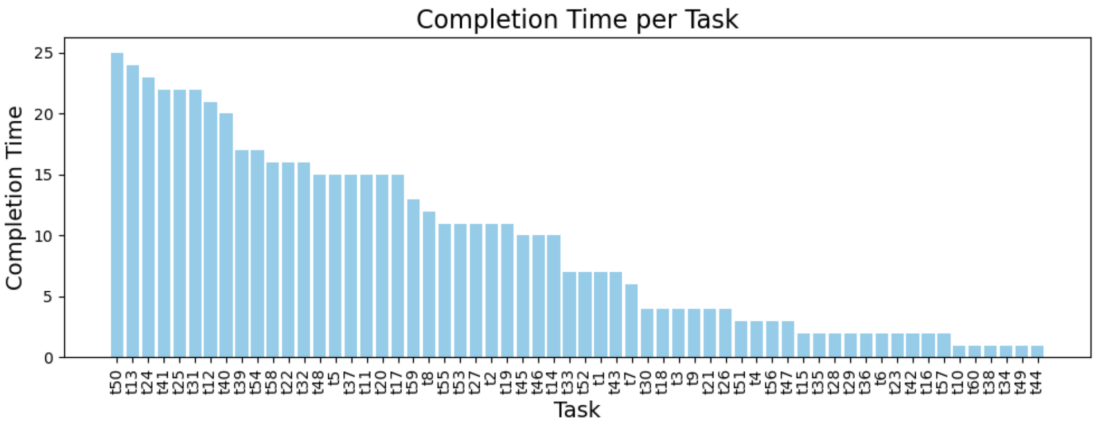

Figure 2: Completion time for each task

4.3 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.

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

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 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.

Figure 3: Distribution of tasks across batches

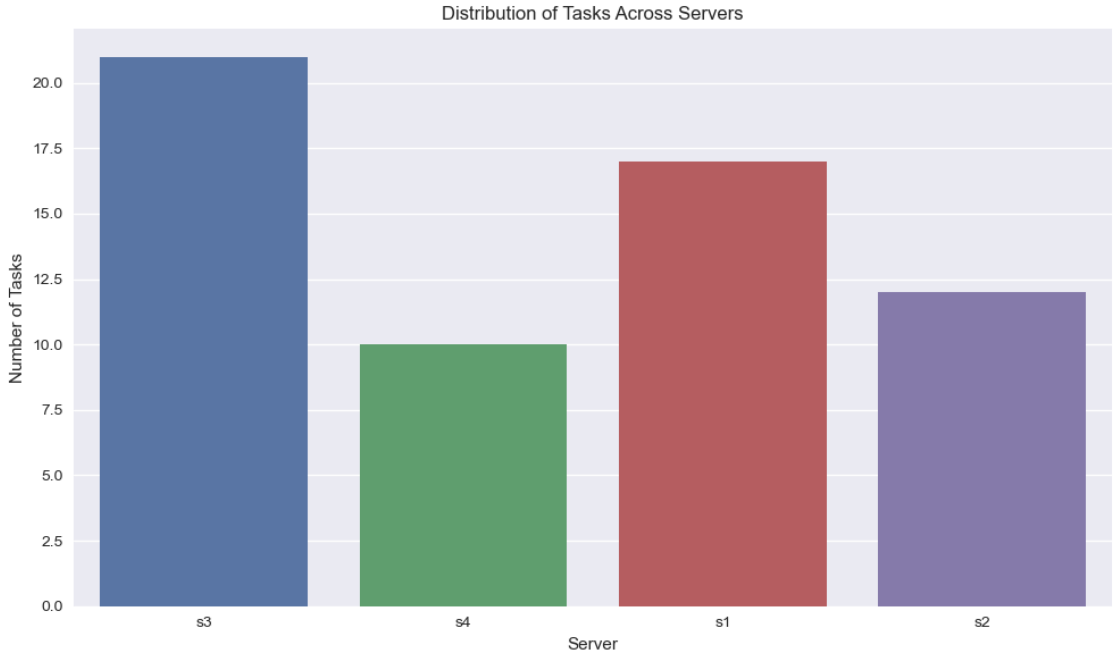

Figure 4: Distribution of tasks across servers

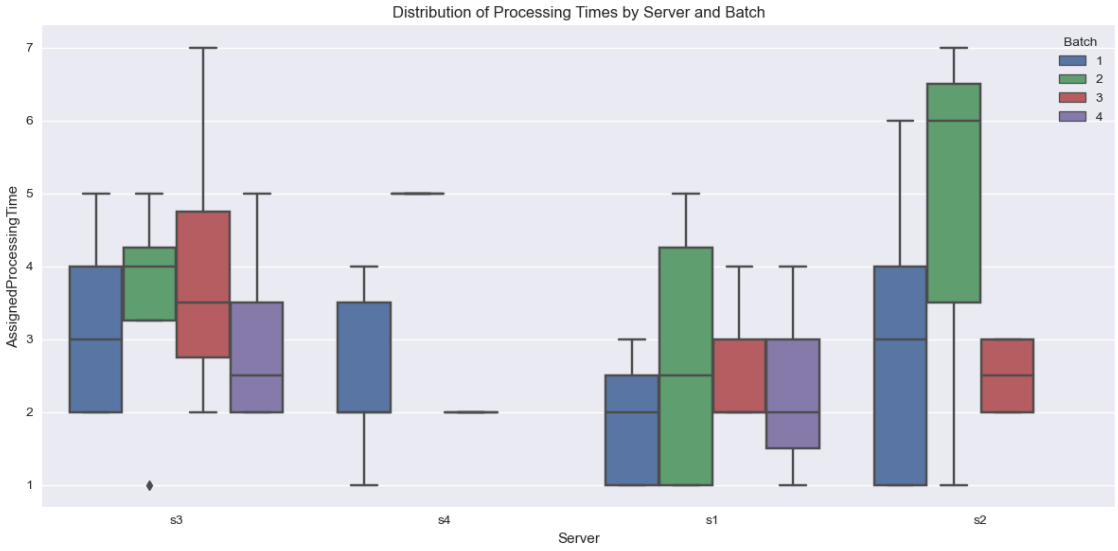

Figure 5: Distribution of assigned processing time by server and batch

The essential files associated with this model, including model3.mod, data3.dat, and job3.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, datagen3.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.

4.4 Third Experiment

We began by linearizing Model 1 (Scenario 3). Figure 9 compares the performance of non-linear version along with two linear versions of model 1. We ended up choosing linear test 2, which outperform others in terms of completion time, and number of iterations in a fixed amount of time.

To assess the performance of our proposed optimization framework, we conducted a series of computational experiments based on model 4 which is a linearized version of Model 1 (Scenario 3), subsequently combined with the resource allocation features of Model 3. This unified formulation integrates task scheduling and resource optimization through a weighted objective function, enabling a balanced trade-off between completion time and resource efficiency.

We chose to generate synthetic datasets programmatically in Python. These datasets facilitated controlled experimentation across various problem scales. In each experiment, we fixed the number of servers and their respective capacity levels. We then incrementally increased the number of tasks and the number of batches. The number of batches was computed using a heuristic to ensure that the total demand within each batch remained below the aggregate capacity of the servers. This approach ensured feasibility and realistic constraints within the synthetic workloads.

During early experimentation, we observed that the solver consistently returned an objective function value of zero for larger datasets. This output typically indicates that the model is infeasible under the current formulation or parameterization. To systematically investigate the source of this infeasibility, we designed a series of experiments using synthetic datasets with task counts ranging from 100 to 1000, in increments of 100. Each dataset was configured with three servers and two

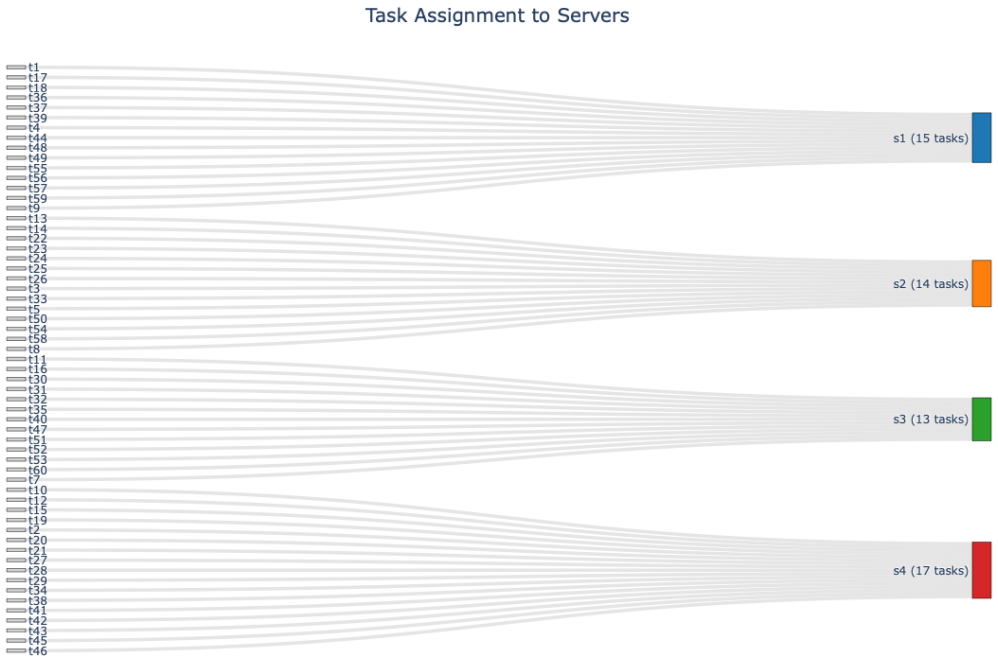

Figure 6: Task assignment to servers

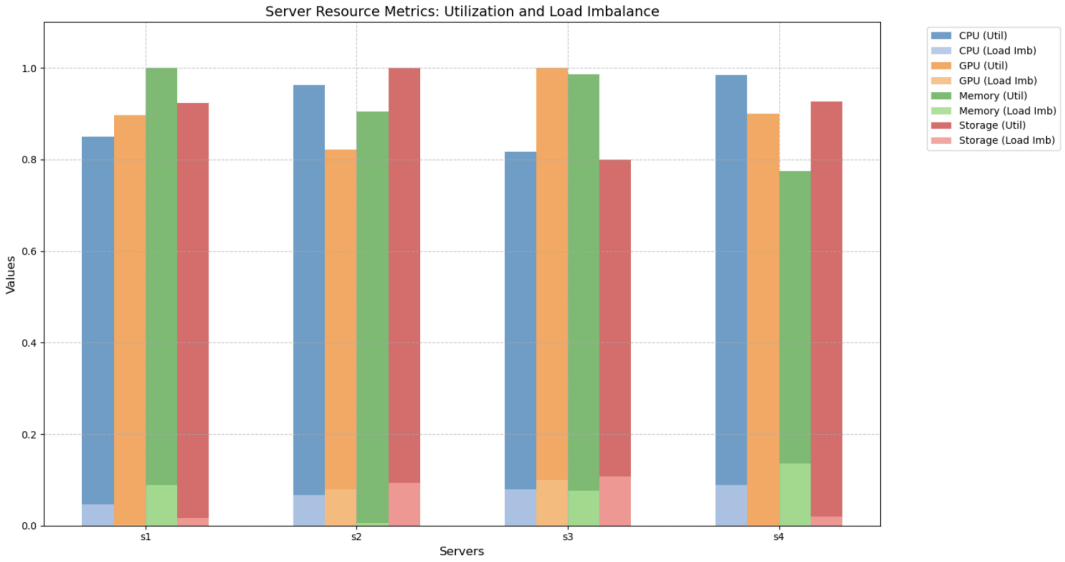

Figure 7: Server utilization and load imbalance for each server

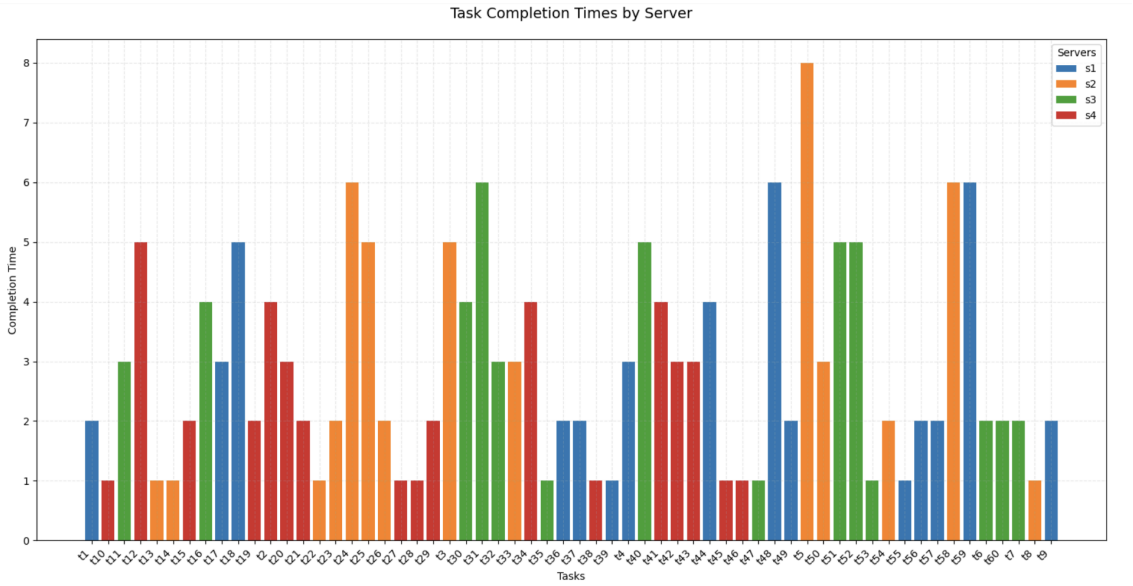

Figure 8: Completion time of assigned tasks on servers

resource types. The number of batches was not fixed but instead dynamically calculated to ensure that each batch's total resource demand remained within the aggregate capacity of the servers. We adopted 1000 seconds as the benchmark time limit for all subsequent experiments. We also employed Cplex as the optimizer for this model.

To investigate this behavior, we relaxed the integrality constraints, allowing binary variables to assume continuous values in the interval $[0, 1]$. In these relaxed scenarios, the model frequently found feasible solutions, suggesting that the core problem was not the formulation itself but the difficulty of solving the full MILP version at scale.

Our diagnostic experiments explored several model configurations and solver parameters aimed at restoring feasibility. First, we experimented with adjusting the value of the large constant M used in on of the constraints, which, if set improperly, can render the model either too loose or overly restrictive. We found that tuning this parameter had a marginal impact but was not the root cause of infeasibility.

Second, we increased the solver's time limit to allow more exhaustive exploration of the solution space. To better understand the trade-off between computation time and solution quality, we experimented with a range of solver time limits. The results indicated that, while modest improvements were observed with extended time limits, the majority of high-quality solutions were obtained within the first 1000 seconds. Additionally, this adjustment enabled the solver to make further progress in some instances, but did not consistently resolve the zero-objective issue.

Third, we developed and applied warm-start strategies to guide the solver using initial feasible solutions obtained from the relaxed model or from heuristic-based methods. While warm-starting significantly improved performance on mid-size instances, it was insufficient to guarantee convergence for larger datasets.

Fourth, we leveraged the solutionlimit option in AMPL to instruct the solver to stop upon finding the first feasible solution, rather than seeking optimality. This modification proved effective in several cases, confirming that feasible solutions existed but were difficult to find within default solver settings.

Next, We explored the effect of incrementally increasing the number of batches. We found that for large-scale datasets, expanding the number of batches often facilitated the discovery of feasible solutions. This supports our hypothesis that batching flexibility plays a critical role in solving MILP problems with high-dimensional task and resource configurations.

A key modeling challenge involved selecting an appropriate number of batches. Increasing the batch count generally improved feasibility by distributing task demand more evenly, thereby lowering the likelihood of violating server constraints. However, a larger number of batches also introduced more binary variables, which increased the model's complexity and computational burden. Figure 10 shows that after 600 tasks, estimated number of batches started to diverge from the feasible number of batches needed. Determining the optimal number of batches became a critical parameter tuning problem. While increasing the batch count often aided feasibility, it also posed trade-offs in terms of solver run-time and memory requirements.

Together, these diagnostic experiments provided critical insights into model behavior and solver limitations. They also guided the development of more robust formulations and solution strategies for handling large-scale data center scheduling problems.

All optimization runs were executed using the NEOS Server. While publicly accessible and widely used, Neos servers impose strict constraints on memory, disk space, execution time, and input file sizes. These limitations restricted our ability to solve large-scale problem instances. We carefully formatted and minimized input files to mitigate these issues, but many large problem instances were still unsolvable within NEOS's operational limits.

Managing experiments presented its own logistical difficulties. Each run required manual editing of AMPL files, submission to the NEOS server, and monitoring of the job's progress and outcome. In contrast to machine learning experimentation, where frameworks for automated job scheduling and experiment tracking are well developed, the field of optimization lacks widely adopted infrastructure for experiment management. To address this gap, we developed an application that automates job submission to NEOS and logs the outcomes, providing a more structured and reproducible experiment pipeline.

Despite the promising capabilities of the integrated MILP model, several methodological challenges persist. Unlike ensemble methods such as boosting in machine learning, there is currently no standard approach for combining multiple optimization models or learned heuristics in MILP frameworks. We are actively investigating the possibility of using learning-augmented optimization techniques, where surrogate models or trained policies guide solver decisions. Initial developments include the creation of an interface that sends structured optimization jobs to NEOS and archives the results for analysis. This hybrid method combining traditional MILP solvers with machine learning remains a focus of our ongoing work.

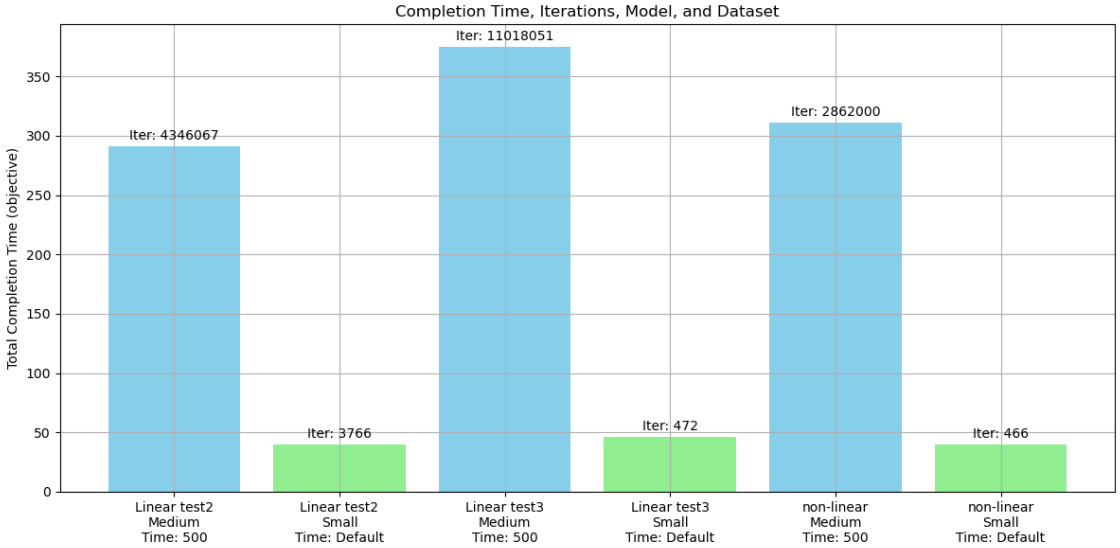

Figure 9: Comparison performance of different versions of model 1 scenario 3

Our experiments highlight several important trade-offs among the tested models. Model One, with its nonlinear constraints, enables dynamic batching and offers greater scheduling flexibility, which can be advantageous in environments with variable workloads. However, this added flexibility comes at the cost of significantly increased computational overhead, resulting in limited scalability and feasibility challenges for larger datasets. In contrast, Model Three adopts a fully linearized structure that integrates multiple resource types and energy considerations. This model not only promotes better resource utilization and sustainability but also scales more efficiently, making it suitable for larger problem instances under constrained computational resources. Building on these insights, Model Four offers a robust integration of both approaches. It retains the benefits of dynamic batching while maintaining linearity, allowing it to manage energy and multi-resource constraints effectively. Model Four consistently demonstrated higher feasibility, better objective values, and more stable performance across different time limits, positioning it as the most practical and scalable model tested.

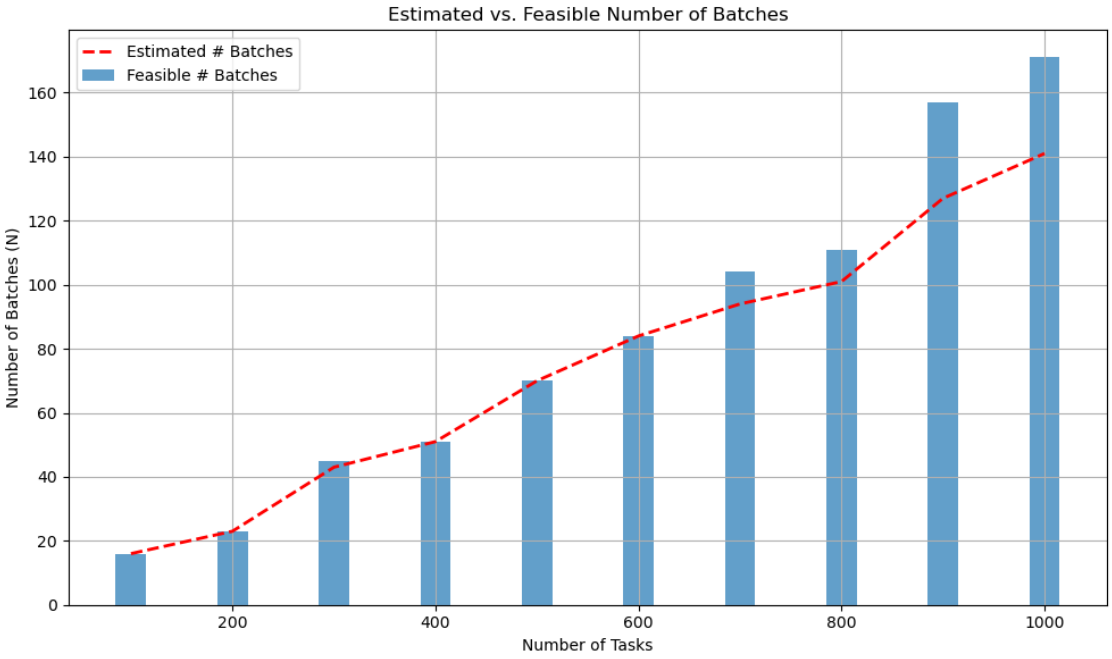

Figure 10: Estimated vs. Feasible Number of Batches

Table III: Summaray of Third Experiment

Parameter / # Tasks	100	200	300	400	500	600	700	800	900	1000 *
Binary Variables	4803	13803	40503	61203	105003	151203	218403	266403	423903	513003
Linear Variables	1747	4861	13905	20917	35655	51183	73723	89837	142529	172357
Equality Constraints	109	209	309	409	509	609	709	809	909	1009
Inequality Constraints	14526	41575	121829	183971	315504	454202	655942	799991	1272813	1540211
# Simplex Iterations	1.03E+07	4.5E+06	518601	254156	91510	103640	14	21	189755	1.7E+06
N (Feasible)	16	23	45	51	70	84	104	111	157	171
Z	5092	40781	130914	187971	339905	478905	679601	851925	1309770	1703840

4.5 Limitations of the Borg 2019 Trace for Full Model Evaluation

In 2011, Google released a one-month trace from its Borg cluster management system to help external researchers investigate large-scale scheduling behavior. Since then, hundreds of researchers have used this dataset to examine various aspects of cluster scheduling. However, questions remain about how workloads and scheduling strategies have changed over time. To address this, Google published a new dataset in 2019, which contains detailed job scheduling data from eight different Borg clusters (cells) throughout May 2019.

Table IV: Summary of Google Borg Trace (May 2019)

Attribute	**May 2019**
Duration	31 days
Clusters	8
Machines	96.4k
Machines per cluster	12.0k
Hardware platforms	7
Priority values	0–450
Alloc sets	Y
Job dependencies	Y
Batch queueing	Y
Vertical scaling	Y
Format	BigQuery tables

To adapt the 2019 Google Borg trace to our mixed-integer scheduling framework we first mapped each available trace column to the corresponding symbol in the model. Every task is

uniquely identified by the pair *(collection_id, instance_index)*, which therefore defines the set T. The field *machine_id* plays the same role for the server set S. Requested CPU and memory, stored in the loosely JSON-encoded field *resource_request*, provide the two-dimensional demand vector $d_{i,r}$. Because the public release strips machine-capacity information, provisional capacities $r_{j,r}$ were fixed only by requiring them to exceed the largest single-task demand observed; this choice guarantees feasibility but is not grounded in the trace itself. Borg's integer *priority* (0–450) serves as the weight w_i in the objective. Processing time $p_{i,j}$ was taken as the wall-clock interval *end_time* − *start_time* on the machine where each task ultimately finished, acknowledging that the trace contains no information about how long the same task would have run on any other host.

Several parameters required by the optimization remain without empirical support. The trace offers no direct or indirect measure of per-server energy consumption, so the coefficients e_j that scale the energy term cannot be populated from data. Because each task appears on at most one machine, the full task–server timing matrix $p_{i,j}$ is largely unobserved; only a single diagonal slice is available, leaving the majority of entries underspecified. Verified server capacities $r_{j,r}$ are also missing, as the capacity-defining protobuf attributes were removed during anonymisation. Additional resource classes such as disk bandwidth, network throughput, or accelerator slots are absent altogether, and there is no field that could serve as a setup or transition delay $S_{i,j}$. Finally, the trace provides no guidance on the relative economic value of completion time, energy, and load balance, so the scaling constants in the objective (α, β, γ) cannot be calibrated from measurement.

These omissions imply that the public Borg trace does not contain sufficient measured information to evaluate the full optimisation model on real data. Any numerical study performed directly on the trace would necessarily substitute external assumptions for the missing quantities, making the results a reflection of those assumptions rather than of the production-cluster behaviour that the trace records.

5 Leveraging Machine Learning to Accelerate MILP Based Batch Scheduling

Data driven models now act as powerful co pilots for exact MILP solvers such as CPLEX or Gurobi. While branch and bound is still required for global optimality, graph based neural networks can drastically shorten the search or, in time critical settings, generate high quality schedules outright. Three complementary roles have emerged. First, constructive policies rely on pointer network or Transformer architectures augmented with graph attention to capture task–server–resource relationships. Trained through reinforcement learning that rewards lower objective values, these policies sequentially emit assignment triples—task, server, batch—producing feasible schedules in milliseconds and supplying excellent warm starts for the MILP solver. Second, solver guidance networks embed a graph neural network inside the solver's callback. By imitating strong branching traces extracted from optimally solved instances, the model learns which branch to explore or prune, reducing the branch and bound tree by factors of two to ten while preserving optimality guarantees. Third, variable and constraint screening models use supervised GNN classifiers to flag unlikely assignment variables for early fixing or to suggest tighter cutting planes. By shrinking the model before optimisation begins, they lower memory use and node counts and can transform otherwise intractable large instances into solvable ones. A practical workflow starts by solving a representative set of small and medium instances of the MILP and recording optimal assignments, branching paths and dual signals. Each instance is then encoded as a heterogeneous graph with task, server and (optionally) batch nodes, while edge features carry processing times, demands and capacities. Supervised learning replicates optimal screening and branching choices, whereas reinforcement learning rolls out complete schedules and back propagates the weighted completion time, energy and imbalance cost to train constructive policies. Deployment is flexible: the constructive policy alone provides a rapid, near optimal schedule when that is sufficient, and when exact guarantees are mandatory, its schedule is passed to the solver as an initial incumbent while the guidance and screening models accelerate the remaining search. Several caveats remain. During decoding, infeasible moves must be masked or repaired to respect resource and batch constraints. Distribution shifts in tasks, servers or resources require re training because GNN generalization, although strong, is not unlimited. Finally, when regulatory or service level commitments demand rigorous optimality, the MILP solver should always verify and polish the machine learning proposal. By weaving these graph based techniques into the optimization pipeline, practitioners regularly achieve order of magnitude speed ups without sacrificing the robustness that makes MILP an attractive foundation for mission critical batch scheduling problems.

6 Conclusion

In this project, we developed and evaluated several optimization models aimed at improving task assignment and resource utilization in data center operations. Our focus included both nonlinear and linear formulations, with extensive testing conducted on synthetic datasets. Early experiments revealed that nonlinear constraints, while offering dynamic batching capabilities, significantly increased computational complexity and resulted in frequent memory and disk overflows on the NEOS optimization server. These scalability limitations made such models impractical for large-scale or real-time applications.

Among the tested solvers, FilMINT performed best under the first nonlinear scenario, though we faced uncertainty regarding whether performance limitations were due to the solver environment or the structure of the data. Conversely, the second, linearized model exhibited stronger scalability and fewer technical obstacles when applied to larger datasets.

Building on these insights, we developed a fully linearized Model Four that integrates the strengths of the earlier models. It supports dynamic batching and energy-aware multi-resource management while remaining computationally tractable. This model demonstrated consistent feasibility and improved solution quality, even under tight solver time limits, making it the most robust formulation in our study.

A central modeling challenge involved determining the number of batches. While increasing the batch count improved feasibility by distributing task demand more evenly, it also added binary variables, leading to higher computational overhead. As shown in Figure 10, beyond 600 tasks, the suggested number of batches increasingly diverged from the actual number needed, highlighting the need for better heuristics or adaptive strategies. This parameter tuning problem played a critical role in balancing solution quality with solver runtime and memory constraints.

All experiments were run using the NEOS Server for Optimization, which—despite being widely accessible—imposed strict limits on memory, disk space, execution time, and file size. These constraints restricted our ability to scale experiments, even with careful input formatting. Additionally, the lack of built-in tools for managing large experiment batches posed logistical challenges. Unlike in machine learning workflows, optimization lacks mature tools for experiment tracking and automation. To address this, we developed a prototype application for automated job submission and logging, facilitating more structured and reproducible experimentation.

Despite these advances, important challenges remain. While the idea of integrating boosting techniques from machine learning into optimization is promising, there is currently no general-purpose framework that supports this within a MILP context. Most existing solvers do not accommodate the kind of iterative, feedback-driven learning that boosting relies on. Although optimizing the branching rules or tuning the branch-and-bound process could theoretically improve performance, this was beyond the scope of our research. Instead, we began experimenting with ways to incorporate learning-based guidance into the modeling and solution process—such as using approximate models to guide variable selection or warm-start strategies. As part of this effort, we have developed a prototype tool for structuring, submitting, and tracking NEOS optimization jobs, laying the groundwork for more intelligent and automated experimentation in future work.

Future work will explore AI-driven continual learning that automatically adapts the machine-learning policies to changing workload patterns, as well as deeper integrations where the MILP solver can query the neural model on-the-fly for AI-generated branching choices and cut suggestions.

7 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.
- Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015, April). Large-scale cluster management at Google with Borg. In Proceedings of the tenth european conference on computer systems (pp. 1-17).
- Cheng, Yue & Anwar, Ali & Duan, Xuejing. (2018). Analyzing Alibaba's Co-located Datacenter Workloads. 292-297. 10.1109/BigData.2018.8622518.
- He, J. (2022). Decision scheduling for cloud computing tasks relying on solving large linear systems of equations. Computational Intelligence and Neuroscience, 2022(1), 3411959.
- Liu, Ning & Dong, Ziqian & Rojas-Cessa, Roberto. (2013). Task Scheduling and Server Provisioning for Energy-Efficient Cloud-Computing Data Centers. Proceedings - International Conference on Distributed Computing Systems. 226-231. 10.1109/ICDCSW.2013.68.
- Shi, K., Chen, S., Chen, S., Sun, Y., & Ding, Z. (2024, May). Energy-Aware Data Center Job Scheduling Scheme under Uncertain Environment. In 2024 IEEE/IAS 60th Industrial and Commercial Power Systems Technical Conference (I&CPS) (pp. 1-6). IEEE.
- Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., ... & Wilkes, J. (2020, April). Borg: the next generation. In Proceedings of the fifteenth European conference on computer systems (pp. 1-14).
- Zhou, Y., & Jiao, X. (2021). Knowledge-driven multi-objective evolutionary scheduling algorithm for cloud workflows. IEEE Access, 10, 2952-2962.
- Arunagiri, Ramathilagam & Kandasamy, Vijayalakshmi. (2016). A survey of scheduling algorithm in cloud computing environment. International Journal of Control Theory and Applications. 9. 137-145.
- Sanjalawe, Y., Al-E'mari, S., Fraihat, S., & Makhadmeh, S. (2025). AI-driven job scheduling in cloud computing: a comprehensive review. Artificial Intelligence Review, 58(7), 197.
- Durillo, J. J., & Prodan, R. (2014). Multi-objective workflow scheduling in Amazon EC2. Cluster computing, 17, 169-189.
- Pierson, J. M., Stolf, P., Sun, H., & Casanova, H. (2020). MILP formulations for spatio-temporal thermal-aware scheduling in Cloud and HPC datacenters. Cluster Computing, 23(2), 421-439.
- Dong, Z., Liu, N., & Rojas-Cessa, R. (2015). Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers. Journal of Cloud Computing, 4, 1-14.
- Scavuzzo, L., Aardal, K., Lodi, A., & Yorke-Smith, N. (2024). Machine learning augmented branch and bound for mixed integer linear programming. Mathematical Programming, 1-44.
- Sampaio, A. M., & Barbosa, J. G. (2020). Workflow scheduling with amazon EC2 spot instances: building reliable compute environments. Int. J. Mach. Learn. Comput, 10(1), 140-147.

- 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/`.

- HostingAdvice. (2025, April 22). *20 incredible data center statistics in 2025*. HostingAdvice.com. `https://www.hostingadvice.com/how-to/data-center-statistics/`

- Brightlio. (2025, April 18). *170 data center stats (April 2025)*. Brightlio. `https://brightlio.com/data-center-stats/`

- Dgtl Infra. (2025). *How much does it cost to build a data center?* DgtlInfra. `https://dgtlinfra.com/how-much-does-it-cost-to-build-a-data-center/`

- Google. (2025). *Council Bluffs, Iowa – Google data centers*. Google. `https://www.google.com/about/datacenters/locations/council-bluffs/`

- Reuters. (2025, January 3). *Microsoft plans to invest $80 billion on AI-enabled data centers in fiscal 2025*. Reuters. `https://www.reuters.com/technology/artificial-intelligence/microsoft-plans-spend-80-bln-ai-enabled-data-centers-fiscal-2025-cnbc-reports-2025-01-03/`

- Reuters. (2025, April 8). *Microsoft puts $1B data center builds on hold amid AI, tariff worries*. Reuters. `https://www.reuters.com/technology/microsoft-puts-1b-data-center-builds-hold-ai-tariff-worries-2025-04-08/`

- LinkedIn. (2025, February 28). *Building data centers: Timeline & innovation guide*. LinkedIn. `https://www.linkedin.com/pulse/designing-building-data-center-timeline-steps-hurdles-hztgf/`

- FMP Construction. (2025, March 29). *Data center construction: A step-by-step breakdown*. FMPConstruction.com. `https://www.fmpconstruction.com/blog/data-center-construction-steps/`

- Newmark. (2025, February 12). *2025 U.S. data center market outlook*. Newmark. `https://www.nmrk.com/insights/market-reports/2025-us-data-center-market-outlook`

- DatacenterDynamics. (2025, January 5). *Microsoft to spend $80bn on AI data centers in 2025*. DatacenterDynamics. `https://www.datacenterdynamics.com/en/news/microsoft-to-spend-80bn-on-ai-data-centers-in-2025/`

8 Appendix A, B, and C

- model 1 scenario 3 result

Table V: 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 V: 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

- All other files are located in this GitHub page:
  https://github.com/ai24-7/Math6940_projects_computational_mathematics/
  For further information, please contact author at ds177320@ohio.edu