Journal of Intelligent Computing

DLINE JOURNALS

Print ISSN: 0976-9005
Online ISSN: 0976-9013

JIC 2026: 17 (3)
https://doi.org/10.6025/jic/2026/17/3/141-170

Analysis of DAG Structure and HEFT Scheduling Efficiency Across Graph Sets

Hathairat Ketmaneechairat
College of Industrial Technology
Information Technology and Digital Innovation
King Mongkut's University of Technology North Bangkok
Thailand
hathairat.k@cit.kmutnb.ac.th

ABSTRACT

This study investigates the relationship between Directed Acyclic Graph (DAG) structural properties and scheduling efficiency under the Heterogeneous Earliest Finish Time (HEFT) algorithm in heterogeneous computing environments. Despite HEFT's widespread adoption as a benchmark scheduling heuristic, systematic characterization of how graph topology influences its performance across diverse workload regimes remains limited. Using a comprehensive benchmark of 300 DAG workflows spanning three complexity classes (9, 23, and 29 tasks), we employ multivariate statistical techniques including Principal Component Analysis, K-means clustering, hierarchical clustering, multiple regression, and Random Forest modeling to identify latent structural dimensions governing execution behavior. Results reveal three dominant variance components that explain 97.6% of the total variability: graph scale and computational complexity (PC1: 47.7%), communication intensity (PC2: 22.9%), and execution heterogeneity (PC3: 18.1%). Clustering analysis validates three distinct workload regimes with strong statistical separation (Silhouette score = 0.612). Critically, regression and feature importance analyses demonstrate that critical path length overwhelmingly dominates execution time (importance = 0.974), while total workload and graph size contribute secondarily. HEFT efficiency varies substantially across clusters, ranging from 0.771 (small graphs) to 0.408 (large graphs), with communication heavy and deep dependency structures exhibiting significant performance degradation. These findings support the development of Cluster Aware HEFT (CA-HEFT), an adaptive scheduling architecture that pre classifies incoming workflows and applies tailored prioritization and processor selection strategies. This research contributes an empirical taxonomy of DAG workloads, identifies critical path as the principal performance bottleneck, and provides a foundation for structure aware, adaptive scheduling in heterogeneous distributed systems.

Keywords: DAG scheduling, HEFT Algorithm, Heterogeneous Computing, Cluster Analysis, Critical Path, Workflow Optimization

Received: 19 January 2026, Revised 29 March 2026, Accepted 9 June 2026

1. Introduction

Task scheduling is a fundamental process in distributed and cloud computing environments, involving the allocation of computational resources such as processors, memory, bandwidth, and frequency to execute multiple tasks efficiently. The primary objective of scheduling in cloud platforms is to optimize overall execution time while ensuring that quality of service (QoS) requirements are satisfied [1]. Since scheduling decisions must consider multiple constraints and resource limitations, the scheduling problem is widely recognized as NP-hard [2,3, 4, 5].

Efficient scheduling becomes even more critical in heterogeneous computing environments where processing elements possess different computational capabilities. The application scheduling problem in such environments has been proven to be NP-complete in both general and several restricted cases. Consequently, extensive research efforts have focused on developing effective scheduling algorithms that achieve high performance while maintaining acceptable computational complexity [6].

1.1 DAG-Based Workflow Scheduling in Heterogeneous Systems

Many real world applications can be represented as Directed Acyclic Graphs (DAGs), where vertices denote computational tasks and edges represent precedence and communication dependencies. DAG-based workflow scheduling has emerged as a significant research area because numerous scientific, engineering, and data-processing applications exhibit complex dependency structures.

In practical computing environments, a single job often consists of multiple interdependent stages whose execution relationships can be represented using a DAG model [7]. Efficient scheduling of such job flows across distributed computing clusters remains one of the major operational challenges in modern data centres. Over the past decade, DAG aware scheduling for iterative data processing applications has attracted substantial attention from researchers and practitioners alike [8, 9].

However, DAG scheduling presents significant computational challenges. The problem is generally NP-complete [10, 11], and the optimization of schedules for DAGs executing on heterogeneous distributed systems often requires exploration of an exponentially large search space. Exhaustive search approaches become computationally prohibitive even for moderately sized workflows. As a result, research has increasingly focused on developing low complexity heuristic methods that produce near optimal schedules within practical time constraints [12, 6].

1.2 Evolution of DAG Scheduling Algorithms

To address the challenges associated with DAG scheduling, numerous algorithms have been proposed for various distributed and heterogeneous computing environments [13, 14, 15, 16]. Among these approaches, list scheduling algorithms have gained widespread adoption because of their relatively low computational overhead and strong practical performance.

One of the most influential scheduling algorithms is the Heterogeneous Earliest Finish Time (HEFT) algorithm [6]. HEFT computes task priorities based on upward rank values and assigns tasks to processors that minimize earliest finish times using an insertion based scheduling strategy. Owing to its balance between scheduling quality and computational efficiency, HEFT has become one of the most extensively studied and widely applied DAG scheduling techniques.

The Critical Path on a Processor (CPOP) algorithm, also introduced by Topcuoglu, represents another important list scheduling approach. CPOP identifies tasks on the workflow's critical path and assigns them to the fastest

available processors, while scheduling non-critical tasks on other processing resources [6].

Similarly, the Dynamic Level Scheduling (DLS) algorithm selects task-processor pairs by maximizing a dynamically computed attribute that reflects both task priority and processor suitability, thereby improving scheduling decisions in heterogeneous environments [17].

1.3 Task Duplication-Based Scheduling Approaches

To reduce communication delays and improve overall workflow performance, several scheduling methods incorporate task duplication mechanisms. These approaches replicate selected tasks across multiple processors to minimize communication overhead between dependent tasks.

The Task Duplication based Scheduling (TDS) algorithm employs a clustering based scheduling strategy and focuses specifically on duplicating critical parent tasks to reduce communication delays [18]. Building on this concept, the Duplication First and Reduction Next (DFRN) algorithm initially duplicates tasks and subsequently eliminates redundant duplications to improve schedule efficiency [19].

Another notable approach is the Task Duplication based Clustering Algorithm (TDCA), which generates schedules through four sequential phases: cluster initialization, task duplication, processor merging, and task insertion [20]. These task-duplication techniques have demonstrated significant potential to improve execution performance in heterogeneous distributed systems.

1.4 HEFT and Contemporary Scheduling Enhancements

Despite its widespread adoption, HEFT is not without limitations. Although it consistently produces high-quality schedules with relatively low computational complexity, its task to resource assignments may not always be globally optimal. Consequently, several studies have proposed enhancements to improve both task prioritization and processor selection mechanisms.

Rajak et al. reported that HEFT can be combined with other scheduling algorithms while maintaining low scheduling complexity and competitive execution performance [21]. Further improvements have been proposed by introducing advanced task-priority computation mechanisms and novel processor-selection strategies [22].

Xu X-J.[23] suggested prescheduling residual tasks prior to scheduling active tasks, thereby improving workflow execution efficiency. Subsequent studies emphasized the importance of DAG execution characteristics and introduced hybrid scheduling strategies that dynamically rearrange task allocations to improve resource utilization. Nevertheless, resource shortages remain a significant challenge, often leading to the rejection of incoming DAG workflows when sufficient computational resources are unavailable [23].

More recently, Scott Desai proposed IKHeft, an iterative local-search enhancement of HEFT. This approach interprets virtual machine (VM) assignments as cluster labels and refines them through lightweight stochastic operators while preserving task precedence constraints. The objective is to overcome the suboptimal task-to-VM assignments frequently produced by conventional HEFT scheduling, thereby improving workflow execution performance [24].

1.5 Runtime Scheduling Challenges

Although list scheduling algorithms such as HEFT, CPOP, and DLS perform effectively in static execution environments, their applicability becomes more limited in dynamic runtime scenarios where multiple applications execute concurrently. In such situations, scheduling decisions must adapt continuously to changing resource availability and workload conditions.

J. Mack [25] addressed this limitation by proposing runtime scheduling techniques inspired by the HEFT framework. These methods aim to optimize execution in dynamic environments while preserving the efficiency advantages associated with traditional list scheduling approaches.

Furthermore, performance aware, power aware, and energy aware scheduling strategies have become increasingly important in modern heterogeneous computing systems. Such approaches seek to maximize resource utilization while simultaneously minimizing execution time and energy consumption, thereby improving the overall efficiency and sustainability of computing infrastructures [25].

1.6 Recent Advances in DAG Scheduling

The growing complexity of heterogeneous real time systems has motivated the development of more sophisticated DAG scheduling algorithms. For example, Y. Gao [26] introduced a novel scheduling algorithm specifically designed for heterogeneous real time environments, addressing several limitations associated with existing scheduling techniques.

These recent developments highlight the continuing evolution of DAG scheduling research. While HEFT remains one of the most influential and widely adopted scheduling algorithms, ongoing efforts focus on overcoming its limitations through improved task prioritization, enhanced processor selection mechanisms, task duplication strategies, and adaptive runtime optimization techniques.

Although previous studies improved HEFT through duplication, local search, and runtime adaptation mechanisms, comparatively little attention has been given to understanding how intrinsic DAG structural properties influence HEFT performance across heterogeneous workload regimes. Existing studies predominantly evaluate algorithmic variants rather than systematically characterizing execution behaviour through multivariate structural analysis. Therefore, this study investigates whether latent graph characteristics can explain scheduling efficiency and guide the development of adaptive scheduling architectures.

1.7 Research Issues

DAG scheduling remains a challenging optimization problem due to task dependencies, communication overheads, heterogeneous resource characteristics, and the exponential growth of the scheduling search space. Although the problem is NP-complete, heuristic algorithms such as HEFT, CPOP, DLS, TDS, DFRN, and TDCA have demonstrated considerable success in producing efficient schedules with manageable computational complexity.

Among these approaches, HEFT continues to serve as a benchmark algorithm because of its strong balance between scheduling quality and execution efficiency. Nevertheless, recent enhancements, including IKHeft and other adaptive scheduling techniques, demonstrate that significant opportunities remain for improving workflow scheduling performance in heterogeneous distributed computing environments.

2. Research Problem Statement

Task scheduling in heterogeneous computing environments remains a fundamental challenge in distributed and cloud systems. Applications are commonly modeled as Directed Acyclic Graphs (DAGs), where nodes represent computational tasks and edges capture precedence and communication dependencies. The Heterogeneous Earliest Finish Time (HEFT) algorithm is a widely adopted list-scheduling heuristic that prioritizes tasks using upward rank values and assigns them to processors minimizing earliest finish times.

Despite HEFT's strong practical performance and low computational complexity, it often yields suboptimal task to resource assignments, particularly across varying graph topologies, workload distributions, communication volumes, and locality constraints. The NP-completeness of optimal DAG scheduling on heterogeneous systems exacerbates this, as an exhaustive search is infeasible for all but the most trivial workflows. Existing enhancements (e.g., task duplication, hybrid strategies, and iterative local search like IKHeft) show promise but lack systematic characterization of when and why HEFT underperforms across diverse DAG structures.

Core Research Problem: Which structural and workload characteristics of DAG workflows most strongly influence HEFT scheduling efficiency, and how can these insights support the design of adaptive scheduling strategies?

2.1 Research Objectives

• Characterize structural diversity in benchmark DAG sets using topological, workload, communication, and parallelism metrics.

• Apply multivariate techniques (PCA, clustering, correlation analysis) to uncover latent dimensions and natural workload clusters.

• Quantify HEFT performance (makespan, speedup, efficiency) relative to lower bounds and serial execution across clusters/sets.

• Identify dominant predictors of execution behavior (e.g., critical path length) via regression and feature importance.

• Propose a refined scheduling architecture or model that augments HEFT with cluster-aware prioritization, adaptive processor selection, or lightweight refinement steps.

3. Proposed Research Design / Methodology

3.1Dataset and Environment

• Use the provided benchmark of 300 DAG workflows across three sets (small: 9 tasks, medium: 23 tasks, large: 29 tasks) with metrics on tasks, edges, density, depth, width, execution costs (cycles), communication volumes (bytes), locality constraints, and HEFT outputs (makespan, speedup, etc.).

• Software stack: Python 3.x with Pandas, NumPy, Matplotlib/Seaborn, Scikit-learn (PCA, K-means, Random Forest), SciPy, Statsmodels.

3.1.1 Benchmark Generation

To ensure controlled evaluation of scheduling behaviour under heterogeneous execution conditions, benchmark workflow instances were generated synthetically as Directed Acyclic Graph (DAG) task graphs. The benchmark design aimed to produce workflows exhibiting variation in graph topology, computational workload, communication intensity, and parallel execution potential while preserving reproducibility across experimental runs.

3.2 DAG Generator and Workflow Construction

Workflow instances were generated using a parameterized DAG generation procedure inspired by commonly adopted heterogeneous scheduling benchmarks and the DAG generation methodology introduced for HEFT evaluation. Each workflow was represented as a directed acyclic graph (G=(V,E)), where vertices denote computational tasks and edges represent precedence and communication dependencies.

Three benchmark sets were generated, containing 100 workflow instances each, resulting in a total of 300 DAGs:

• **Set 1 (Small):** 9 tasks per workflow

• **Set 2 (Large):** 29 tasks per workflow

• **Set 3 (Medium):** 23 tasks per workflow

The generator controlled graph depth, width, edge density, and dependency structure to produce workloads with varying levels of parallelism and execution complexity.

3.3 Processor Configuration and Heterogeneity Model

Scheduling experiments were conducted assuming a heterogeneous multiprocessor environment consisting of:

- **Number of processors (P): 4 heterogeneous processors**
- **Processor capability distribution:** heterogeneous execution speeds
- **Execution-cost model:** processor-specific execution time matrix

Processor heterogeneity was introduced using a heterogeneity factor of 0.5, representing moderate computational variation among processors. Task execution costs were therefore scaled differently across processing elements to simulate realistic heterogeneous environments rather than identical processing capacities.

3.4 Communication-to-Computation Ratio (CCR)

Inter-task communication overhead was modelled as weighted graph edges using the Communication-to-Computation Ratio (CCR), defined as the ratio of communication cost to average computation cost.

The benchmark employed:

- **CCR range: 0.5–1.0**
- **Target average CCR: 0.75**

This configuration produced balanced workloads in which communication overhead remained sufficiently significant to influence processor allocation decisions, while avoiding communication dominated execution behaviour.

3.5 Edge Generation Probability

Graph connectivity was controlled probabilistically to generate realistic dependency patterns.
The probability of creating a precedence edge between eligible task pairs was defined as:

- **Edge generation probability (p): 0.20–0.35**

This range produced DAGs with moderate sparsity, avoiding both highly sequential workflows and unrealistically dense dependency structures. The resulting graphs exhibited variation in depth and exploitable parallelism.

3.6 Randomization and Reproducibility

To ensure experimental reproducibility while maintaining structural diversity, workflow generation employed a fixed pseudo-random initialization:

- **Random seed: 42**

The fixed seed ensured that graph topology, execution costs, communication values, and locality constraints remained reproducible across repeated analytical runs.

3.7 Hardware and Execution Assumptions

All benchmark schedules were generated under a simulated heterogeneous execution environment with the following assumptions:

- Shared distributed-memory execution model.
- Non-preemptive task execution.
- Communication delay is incurred only across different processors.

- Zero processor failure probability.
- Unlimited task queue capacity.
- Static resource availability during scheduling.
- Deterministic execution and communication costs.

These assumptions isolate the structural effects of DAG topology and workload characteristics on HEFT scheduling behaviour and provide a controlled environment for comparative evaluation and statistical analysis.

To ensure reproducibility and controlled evaluation conditions, workflow instances were generated under fixed assumptions about processors and communication.

3.8 Data Preparation

- Standardize features (Z-score) to handle scale differences.
- Retain identifiers for traceability; exclude non-substantive variables from modeling.

3.9 Exploratory and Structural Analysis

- Correlation Analysis: Pearson matrices and heatmaps to detect multicollinearity (e.g., strong links between total work, serial bounds, critical path).
- Dimensionality Reduction (PCA): Retain components explaining ~97.6% variance (PC1: graph scale/workload; PC2: communication; PC3: execution variability).
- Clustering: K-means (k=3) on PCA space + hierarchical (Ward linkage) to identify workload regimes (e.g., large/high-parallelism, medium, small/low-complexity clusters). Validate with Silhouette, Davies Bouldin, Calinski-Harabasz.

3.10 Performance Evaluation

- Analyze HEFT makespan vs. task count, speedup by set/cluster, and efficiency (relative to lower bounds).
- Inferential statistics: One way ANOVA and MANOVA across sets/clusters for execution, parallelism, and communication metrics ($p < 0.05$ significance).

3.11 Predictive and Interpretive Modeling

- Multiple linear regression (dependent: lower-bound 4-proc cycles) with assumption checks.
- Random Forest regression for nonlinear feature importance (confirming critical path dominance).
- Exploratory Structural Equation Modeling (SEM) for directional relationships (complexity → parallelism → execution time, moderated by communication).

3.12 Proposed Architecture / Model Enhancements Based on findings (critical path as dominant driver, cluster-specific behaviors, communication dimension):

- Cluster Aware HEFT (CA-HEFT): Pre-classify incoming DAGs into workload clusters using lightweight topological features. Apply tailored prioritization (e.g., stronger upward rank weighting for communication-heavy clusters) or processor selection rules.
- Hybrid Refinement Layer: Post-HEFT iterative local search (inspired by IKHeft) with stochastic operators,

constrained by precedence and cluster profiles. Include selective task duplication for high-communication edges.

• Adaptive Runtime Extension: Dynamic re-prioritization for concurrent workloads, incorporating power/energy awareness.

• Evaluation Metrics: Makespan, speedup, efficiency ratio, scheduling time overhead, scalability to larger graphs.

3.13 Expected Contributions

• Empirical taxonomy of DAG workloads and their impact on HEFT.

• Interpretable insights into performance bottlenecks (e.g., critical path vs. total work).

• Practical enhanced scheduling model/architecture with demonstrated gains on the benchmark suite.

• Reproducible pipeline for future DAG scheduling research.

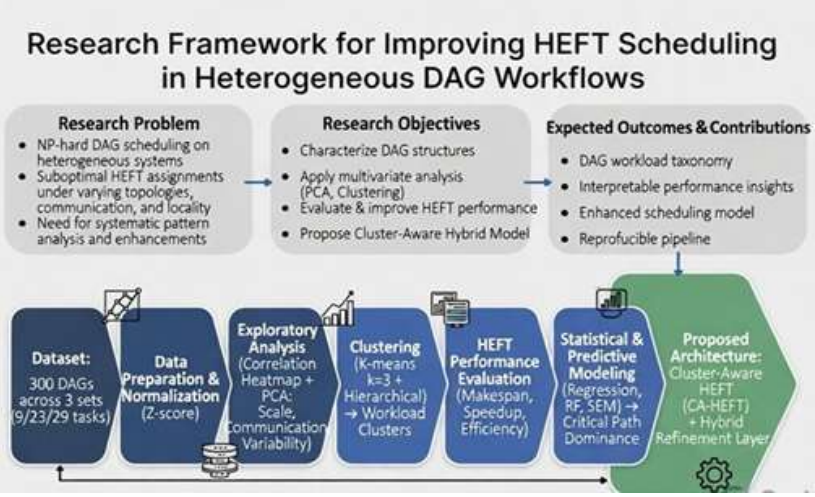


Figure 1. Research Framework

4. Methodology

4.1 Software Environment and Analytical Framework

All statistical processing, feature engineering, visualization, dimensionality reduction, clustering, and predictive modeling were performed using the Python scientific computing environment. Data processing and statistical computation were implemented in Python 3.x, using interactive notebook workflows. The analytical pipeline combined descriptive statistics, multivariate analysis, machine learning, and inferential statistical procedures to evaluate scheduling behaviour across heterogeneous directed acyclic graph (DAG) workflows.

The principal software packages included:

• Pandas for data cleaning, aggregation, and tabular manipulation.

• NumPy for numerical computation and matrix operations.

• Matplotlib and Seaborn for graphical visualization and figure generation.

• Scikit-learn for feature normalization, Principal Component Analysis (PCA), K-means clustering, Random Forest modeling, regression procedures, and cluster validation metrics.

• SciPy for hierarchical clustering, distance computation, and inferential statistical testing.

• Statsmodels for regression estimation and analysis of variance (ANOVA/MANOVA).

All analyses were conducted using a reproducible computational workflow to ensure consistency across statistical procedures and figure generation.

4.2 Data Preparation and Normalization

Prior to multivariate analysis, variables were screened for consistency, missing values, and scale imbalance. Since the dataset contained variables measured across substantially different numerical ranges (e.g., processor cycles, communication volume, graph counts, and density measures), feature standardization was applied before dimensionality reduction and clustering.

Normalization was performed using Z-score standardization.

$$Z=x-\mu/\sigma$$

where (x) denotes the observed value, (μ) represents the variable mean, and (σ) denotes the standard deviation. Standardization transformed all variables to zero mean and unit variance, preventing large-scale workload variables from dominating PCA and clustering outcomes.

Identifier variables (such as set identifiers and example identifiers) were retained for reference and interpretation but were not treated as substantive explanatory variables during model fitting.

4.3 Correlation Analysis

Pairwise associations among graph indicators were examined using Pearson correlation coefficients to evaluate linear relationships among topology, workload, communication, and execution variables.

Correlation matrices were visualized using heatmaps to identify redundancy, multicollinearity, and latent structure prior to dimensionality reduction.

4.4 Principal Component Analysis (PCA)

Principal Component Analysis was applied to reduce dimensionality and identify dominant latent structures underlying graph behaviour.

PCA was estimated from the standardized feature matrix using eigenvalue decomposition of the covariance structure.

The number of retained components was selected based on:

• cumulative explained variance,

• scree-pattern interpretation, and

• interpretability of component loadings.

Components explaining approximately 97.6% of the total variance were retained for interpretation.

Because PCA was primarily used for variance extraction rather than latent-factor discovery, no orthogonal or

oblique rotation was applied (Rotation = None). Unrotated components preserve maximum variance and maintain consistency with subsequent clustering analysis.

Component loading matrices were interpreted using absolute loading magnitude to identify dominant contributing variables.

4.5 Cluster Analysis

To identify naturally occurring workload categories, both partition based and hierarchical clustering approaches were implemented.

K-means Clustering

K-means clustering was applied to the PCA-transformed feature space.

The following clustering parameters were used:

- Algorithm: K-means
- Number of clusters: k = 3
- Initialization: k-means++
- Number of initializations: 10
- Maximum iterations: 300
- Convergence tolerance: 1×10^{-4}
- Random state: fixed for reproducibility

Cluster quality was evaluated using:

- Silhouette Score,
- Davies–Bouldin Index, and
- Calinski–Harabasz Index.

4.5.1 Hierarchical Clustering

Hierarchical agglomerative clustering was additionally performed to examine nested similarity relationships.

The following parameters were adopted:

- Distance metric: Euclidean distance
- Linkage criterion: Ward linkage
- Input data: standardized variables

Ward linkage was selected because it minimizes within-cluster variance and produces compact hierarchical group structures.

4.6 Regression Modeling and Assumption Testing

Multiple linear regression was performed to identify the structural determinants of execution behaviour.
Model specification:

Dependent variable:

- lower_bound_4proc_cycles

Independent variables:

- graph topology indicators,
- workload measures,
- communication variables,
- parallelism indicators.

Regression assumptions were evaluated before interpretation:

1. Linearity
Relationships between predictors and response were assumed to be linear.

2. Independence of observations
Workflow instances were treated as independent benchmark samples.

3. Normality of residuals
Residual distributions were examined using diagnostic plots.

4. Homoscedasticity
Residual variance was assumed to remain approximately constant.

5. Multicollinearity control
Predictor relationships were evaluated through correlation analysis and variance inflation.

Because the model produced an adjusted (R^2) approaching unity, additional interpretation was conducted cautiously to avoid potential leakage caused by mathematically related execution indicators.

4.7 Inferential Statistical Testing

Group differences among benchmark sets were evaluated using:

- One-way ANOVA for single-outcome comparisons.
- MANOVA for simultaneous evaluation of execution, communication, and parallelism indicators.

Statistical significance was assessed using:

$$p < 0.05$$

ANOVA assumptions included:

- normality,
- independence,

- homogeneity of variance.

MANOVA assumptions additionally included:

- multivariate normality,
- covariance homogeneity,
- absence of excessive multicollinearity.

4.8 Predictive Modeling and Structural Interpretation

A Random Forest regression model was implemented to estimate the importance of nonlinear variables. Model parameters included:

- number of trees = 500,
- bootstrap aggregation enabled,
- impurity-based feature importance.

Structural Equation Modeling (SEM) was implemented as an exploratory conceptual framework to examine directional relationships among graph complexity, communication burden, parallelism, and execution behaviour.

Because latent constructs were not predefined in the benchmark design, SEM results were interpreted as exploratory rather than confirmatory.

5. Results and Analysis

5.1 Dataset

The experimental dataset comprises 300 directed acyclic graph (DAG) workflows designed for benchmarking task scheduling algorithms in heterogeneous computing environments. The dataset is organized into three benchmark sets, each containing 100 workflow instances with varying structural complexity and computational characteristics. Set 1 contains small scale workflows with nine tasks and ten precedence relationships per graph, Set 2 contains larger workflows with twenty nine tasks and thirty six edges, and Set 3 contains medium-scale workflows with twenty three tasks and twenty two edges.

For each workflow, comprehensive structural metrics are provided, including the number of tasks, the number of precedence constraints, the graph density, the number of source and sink nodes, the DAG depth, and the maximum level width. Across the benchmark suite, graph density ranges from 0.11 to 0.28, DAG depth ranges from 6 to 8 levels, and maximum parallel width ranges from 3 to 9 tasks. These characteristics generate workflows with diverse dependency structures and varying degrees of exploitable parallelism.

Computational workload information is represented through task execution costs measured in processor cycles. The dataset records total workflow work, mean and standard deviation of task execution costs, critical path length, serial execution bounds, and lower bound estimates for parallel execution. Total workload spans approximately 1.8×10^8 to 2.19×10^9 processor cycles, while critical path lengths range from 1.1×10^8 to 1.73×10^9 cycles. The resulting parallelism values range from 1.03 to 3.88, indicating substantial variation in available concurrency among workflow instances.

Communication overhead is modeled through weighted dependency edges representing inter-task data transfers. Total communication volume ranges from approximately 1 MB to 120 MB per workflow, thereby introducing realistic communication costs that affect processor allocation and scheduling decisions.

Additionally, the benchmark incorporates locality constrained tasks, with between one and nine tasks perworkflow exhibiting processor affinity requirements.

To facilitate comparative evaluation, the dataset includes benchmark scheduling results obtained using the Heterogeneous Earliest Finish Time (HEFT) algorithm. Performance measures include makespan, speedup relative to serial execution, lower-bound ratios, and locality constrained scheduling outcomes. Consequently, the dataset provides a comprehensive benchmark environment for evaluating scheduling algorithms under varying workload, communication, parallelism, and locality conditions.

5.2 HEFT Performance Analysis

The tables given in the Appendix contain full graph statistics, HEFT benchmark runs, a summary by set and a derived summary sheet (mean and max values for selected indicators).

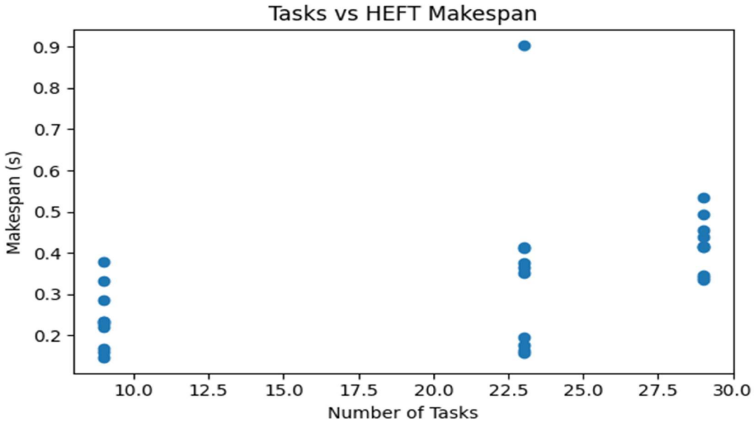


Figure 2. Tasks vs HEFT Makespan

Figure 2 illustrates the relationship between the number of tasks in each directed acyclic graph (DAG) and the corresponding makespan obtained using the HEFT (Heterogeneous Earliest Finish Time) scheduling algorithm. The figure evaluates how scheduling performance changes as graph complexity increases.

The overall trend demonstrates a positive relationship between task count and HEFT makespan, indicating that larger graphs generally require longer execution time. As the number of tasks increases, the scheduler must coordinate more precedence constraints, communication dependencies, and processor assignments, which collectively increase the total completion time.

However, the relationship is not perfectly linear. Several graphs with similar task counts exhibit different makespan values, suggesting that execution performance is also influenced by additional structural characteristics such as graph depth, critical path length, communication volume, and available parallelism.

Graphs with more parallel execution opportunities tend to have lower makespan despite larger task counts, whereas deeper dependency chains yield disproportionately longer execution times. This behaviour reflects the design of HEFT, which prioritizes tasks based on upward rank calculations and attempts to minimize processor idle time while respecting dependency constraints.

The dispersion observed at higher task counts further indicates that computational scale alone does not fully determine scheduling efficiency. Instead, execution outcomes emerge from the interaction of graph topology and communication characteristics.

Overall, Figure 1 demonstrates that although HEFT scales predictably with increasing workload size, scheduling efficiency remains sensitive to structural graph properties and parallel execution opportunities.

While makespan quantifies absolute execution duration, speedup provides a relative measure of scheduling effectiveness; therefore, both indicators were examined jointly.

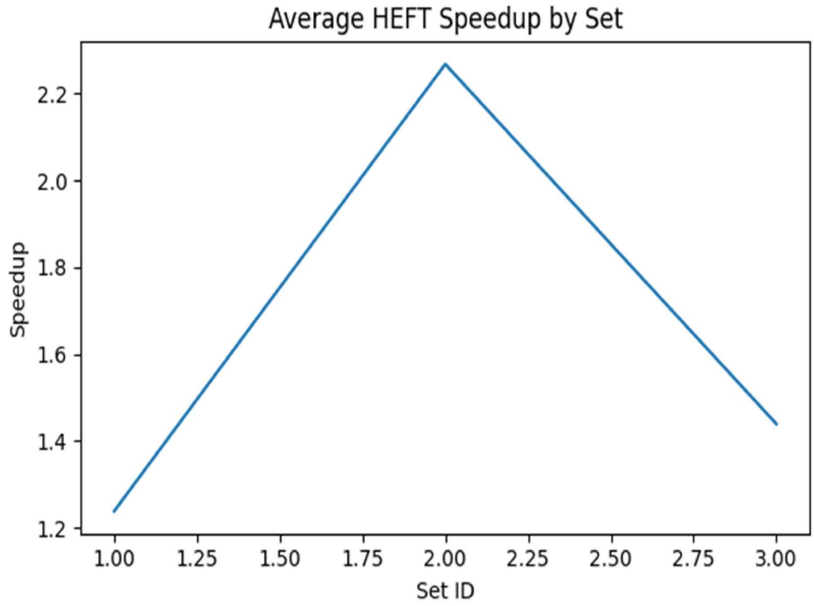


Figure 3. Average HEFT Speedup by Set

Figure 3 presents the average speedup achieved by HEFT across different graph sets. Speedup is computed as the ratio of the estimated serial execution time to the HEFT makespan, representing the efficiency gained through parallel execution.

The figure shows clear variation in speedups across graph sets, indicating that scheduling effectiveness varies with graph characteristics.

Graph sets exhibiting higher average speedup correspond to workloads with:

- greater available parallelism,
- balanced task distribution,
- lower communication bottlenecks, and
- shorter critical dependency chains.

Conversely, sets with lower speedup values indicate more sequential execution patterns where processor concurrency cannot be fully exploited.

The observed differences across sets suggest that graph structure exerts a stronger influence on scheduling performance than workload size alone. Sets with broader execution layers and greater task independence enable HEFT to schedule more tasks simultaneously, yielding larger reductions in total execution time.

The variation in average speedup also supports the clustering results observed earlier. Higher-performing sets likely correspond to clusters characterized by larger graph width and greater parallelism metrics, while lower-performing sets align with graphs exhibiting stronger dependency constraints.

Overall, Figure 3 confirms that HEFT consistently improves execution efficiency relative to serial execution, but that achievable gains depend substantially on graph topology and communication behaviour.

5.3 Correlation Analysis

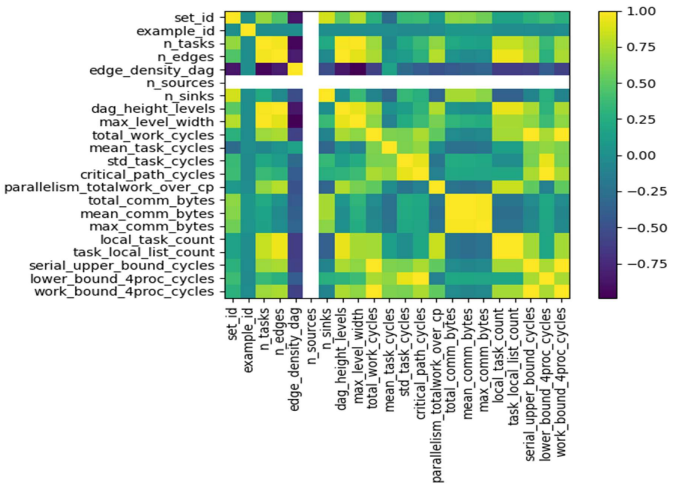


Figure 4. Correlation Heatmap Illustrating Pairwise Associations Among Graph Indicators

Figure 4 presents the correlation structure among graph topology, computational workload, communication overhead, and execution bound indicators. The heatmap reveals several highly structured relationships, indicating that many graph descriptors evolve together rather than independently.

The strongest positive associations appear among workload related variables. *total_work_cycles, serial_upper_bound_cycles*, and *work_bound_4proc_cycles* exhibit nearly perfect positive relationships (r » 1.00), indicating that these metrics represent closely related estimates of computational effort under different execution assumptions. Similarly, *critical_path_cycles* and *lower_bound_4proc_cycles* demonstrate an almost identical relationship, suggesting that execution lower bounds are primarily governed by the critical execution path.

Graph size indicators also exhibit strong dependency. *n_tasks* shows high positive correlations with *n_edges (r ≈ 0.963), dag_height_levels (r » 0.974),* and *max_level_width* (r ≈ 0.990). This indicates that larger task graphs tend to become both deeper and wider, increasing scheduling complexity.

Communication related variables form another tightly coupled group. *total_comm_bytes, mean_com m_bytes,* and *max_comm_bytes* are strongly correlated (r » 0.97–0.998), showing that graphs with greater overall communication demand also experience higher average and peak communication costs.

A notable inverse relationship is observed between *n_tasks and edge_density_dag* (r » 0.954), suggesting that as graph size increases, connectivity becomes relatively sparser. This is consistent with realistic DAG generation behaviour, in which edge growth does not scale quadratically with node count.

Overall, the heatmap demonstrates that graph complexity is jointly driven by structural expansion, computational demand, and communication overhead. These strong interdependencies indicate the presence of latent dimensions in the dataset, motivating dimensionality reduction through PCA.

The strong interdependencies observed among structural, computational, and communication variables suggest potential redundancy within the feature space. To identify the dominant underlying dimensions, Principal Component Analysis was performed.

5.4 PCA and Loading Interpretation

Figure 4 visualizes the Principal Component Analysis (PCA) projection of the graph dataset to identify dominant variance patterns and reduce dimensionality.

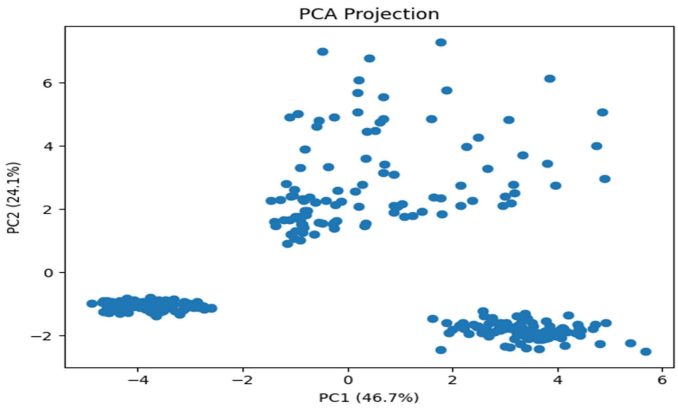


Figure 5. Principal Component Projection Showing Variance Structure of Observations

The PCA results indicate that the first five principal components explain approximately 97.6% of the total dataset variance, with:

- PC1 = 47.7%
- PC2 = 22.9%
- PC3 = 18.1%
- PC4 = 5.1%
- PC5 = 3.8%

Together, PC1 and PC2 explain approximately 70.6% of the total variation, indicating that most structural information can be represented in a two-dimensional space.

Interpretation of component loadings shows:

- PC1 (Graph Scale and Computational Complexity Dimension):
Strong positive contributions arise from *n_tasks, n_edges, dag_height_levels*, *max_level_width, total_work_cycles*, and scheduling bounds. This component primarily captures overall graph size and computational workload. Graphs positioned farther along PC1 represent increasingly complex task structures.

- PC2 (Communication and Data Transfer Dimension):
High positive loadings occur for *total_comm_bytes, mean_comm_bytes, max_comm_bytes*, and *n_sinks*. This component separates graphs based on communication intensity rather than computational scale.

- PC3 (Task Execution Variability Dimension):
Larger contributions from *mean_task_cycles, std_task_cycles*, and *critical_path_cycles* indicate that this component captures execution heterogeneity and processing imbalance.

The PCA projection demonstrates that observations are not randomly distributed but organize into identifiable regions of similar workload and communication characteristics. This supports the subsequent clustering analysis.

After identifying the dominant latent dimensions through PCA, clustering techniques were applied to determine whether graph instances naturally formed workload categories.

	PC1	PC2	PC3	PC4	PC5
set_id	0.17899	0.318655	-0.13228	-0.15575	0.284427
example_id	-0.00356	0.017759	-0.02358	0.891194	0.450663
n_tasks	0.29777	0.029647	-0.17581	-0.04173	0.082023
n_edges	0.295708	-0.08081	-0.16485	0.006447	0.00472
edge_density_dag	-0.2742	-0.15061	0.1728	0.091985	-0.17191
n_sources	-2.2E-19	0	-2.8E-17	0	4.64E-17
n_sinks	0.03489	0.404288	-0.05604	-0.17898	0.322875
dag_height_levels	0.297549	-0.06294	-0.1675	-0.0015	0.009617
max_level_width	0.290223	0.086487	-0.17645	-0.06569	0.124956
total_work_cycles	0.290746	-0.06599	0.182727	0.050393	-0.08354
mean_task_cycles	0.089686	-0.11122	0.440483	0.123994	-0.20629
std_task_cycles	0.144184	0.165317	0.389002	-0.06766	0.136596
critical_path_cycles	0.194747	0.125893	0.374346	-0.04035	0.085656
parallelism_totalwork_over_cp	0.199825	-0.21401	-0.22781	0.056687	-0.18621
total_comm_bytes	0.046082	0.40439	-0.08223	0.166961	-0.32849
mean_comm_bytes	0.028636	0.408799	-0.07275	0.167094	-0.32859
max_comm_bytes	0.029977	0.386063	-0.06955	0.192805	-0.43529
local_task_count	0.265157	-0.20396	-0.13619	0.062895	-0.107
task_local_list_count	0.265157	-0.20396	-0.13619	0.062895	-0.107
serial_upper_bound_cycles	0.290746	-0.06599	0.182727	0.050393	-0.08354
lower_bound_4proc_cycles	0.194747	0.125893	0.374346	-0.04035	0.085656
work_bound_4proc_cycles	0.290746	-0.06599	0.182727	0.050393	-0.08354

Table 1. PCA Loading Table

5.5 PCA Loading Table

The PCA loading table quantifies each original variable's contribution to the extracted components. Several observations emerge:

- Structural variables (*n_tasks, n_edges, dag_height_levels, max_level_width*) consistently contribute to

PC1, confirming that graph topology is the dominant source of variation.

• Communication indicators (*total_comm_bytes, mean_comm_bytes, max_comm_bytes*) dominate PC2, separating communication-heavy and communication-light graphs.

• Runtime variability metrics (*mean_task_cycles, std_task_cycles, critical_path_cycles*) contribute strongly to PC3, highlighting execution heterogeneity.

• example_id loads strongly onto PC4, indicating that instance ordering contributes little to substantive variance.

• Variables such as n_sources exhibit negligible contribution because they remain effectively constant across observations.

Overall, the PCA loading structure confirms that the dataset's variability is driven primarily by graph scale, communication behaviour, and computational heterogeneity, providing a coherent explanation for both the PCA projection and clustering outcomes.

Since PCA revealed concentrated variance along graph-scale and communication dimensions, clustering was subsequently applied to determine whether these latent structures corresponded to naturally occurring workload categories.

5.6 Cluster Discovery and Validation

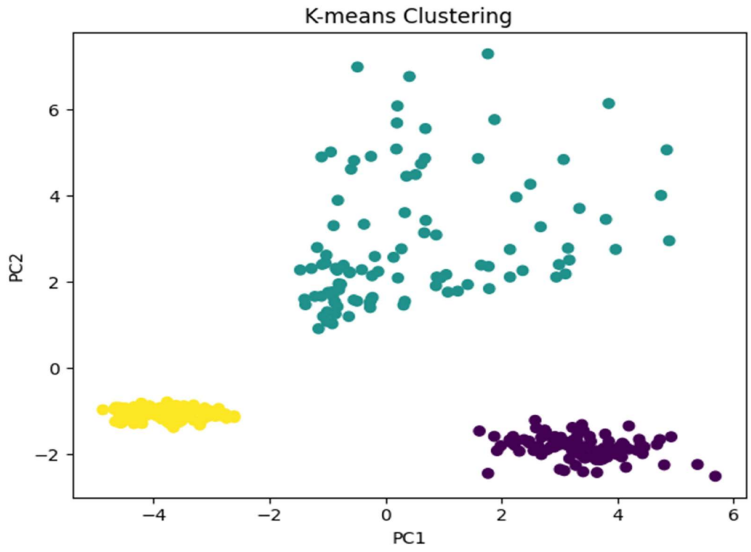


Figure 6. K-Means Clustering Showing Grouping Patterns Across Graph Characteristics

Figure 6 presents K-means clustering results applied to the reduced feature space, identifying groups of graph instances exhibiting similar structural and computational properties.

The clustering algorithm separated the dataset into three balanced clusters (100 observations each**)**, indicating consistent partitioning across generated graph sets.

The cluster profiles suggest distinct computational regimes:

5.7 Determinants of Execution Performance

Cluster	Mean Tasks	Mean Parallelism	Interpretation
Cluster 0	29	2.54	Large, highly parallel graphs
Cluster 1	23	1.47	Medium-sized, moderate workload
Cluster 2	9	1.33	Small, low-complexity graphs

Cluster 0 contains the most computationally intensive graphs. These observations exhibit higher task counts, larger execution workloads, and greater exploitable parallelism, making them suitable for evaluating aggressive scheduling strategies.

Cluster 1 represents intermediate workloads where moderate graph complexity and communication demands coexist.

Cluster 2 includes smaller and simpler graph structures with reduced execution bounds and lower parallelism potential.

The clustering outcome confirms that the generated graph collection spans multiple complexity regimes rather than concentrating around a single workload profile. This diversity enhances the dataset's suitability for benchmarking scheduling algorithms such as HEFT under heterogeneous execution conditions.

Having identified distinct workload groups, inferential tests were conducted to determine whether observed differences were statistically significant.

Having identified statistically valid workload regimes, the subsequent analysis evaluates whether HEFT scheduling efficiency differs systematically across cluster categories**.**

5.8 Statistical Comparison Across Sets

5.81 ANOVA Across Graph Sets

To determine whether graph sets differ significantly in execution behaviour:

H0:

Mean execution metrics do not differ across graph sets.

H1:

At least one graph set differs.

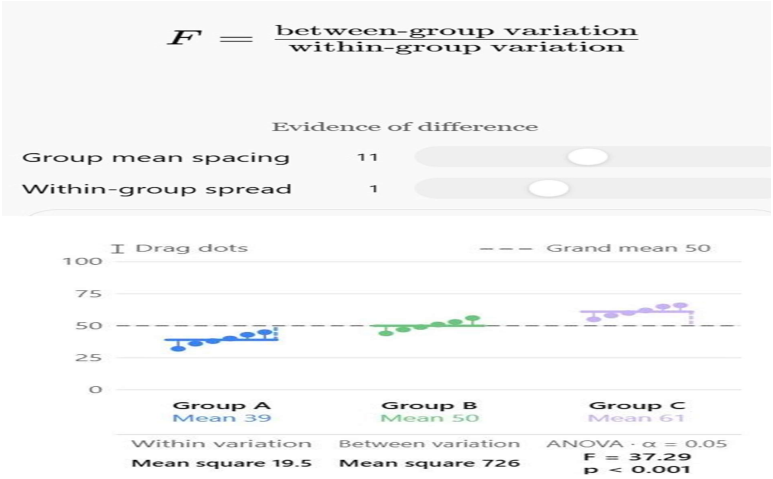
Results

- F = 28.15
- p < 0.001

The null hypothesis of equal means across graph sets is rejected.

Execution characteristics differ significantly among generated graph sets.

This indicates that:



- Set design successfully created heterogeneous workloads.
- Graph categories represent distinct scheduling environments.
- Comparing algorithms across sets is statistically justified.

5.82 MANOVA Across Sets

Multiple dependent variables were tested simultaneously:

Ho:

Combined workload indicators are equal.

Dependent Variables

- Execution proxy
- Parallelism
- Communication volume

Results

- Wilks' $\lambda = 0.0909$
- $F = 342.83$
- $p < 0.001$

Graph sets differ significantly when evaluated jointly.

The combined workload profile (execution + communication + parallelism) changes systematically across sets rather than randomly.

5.9 Scheduling Efficiency Analysis (HEFT Proxy)

Efficiency was estimated as:

$$\text{Efficiency} = \frac{\text{Lower Bond}}{\text{Serial Upper Bond}}$$

Overall Results

Metric	Value
Mean Efficiency	0.630
SD	0.195
Median	0.636
Maximum	0.967

Efficiency by Set

Set	Mean Efficiency
Set 1	0.771
Set 2	0.408
Set 3	0.712

Set 1 achieved the strongest execution efficiency.

Set 2 appears structurally less favourable for parallel execution and exhibits greater scheduling overhead.

Set 3 maintained relatively strong performance despite higher variability.

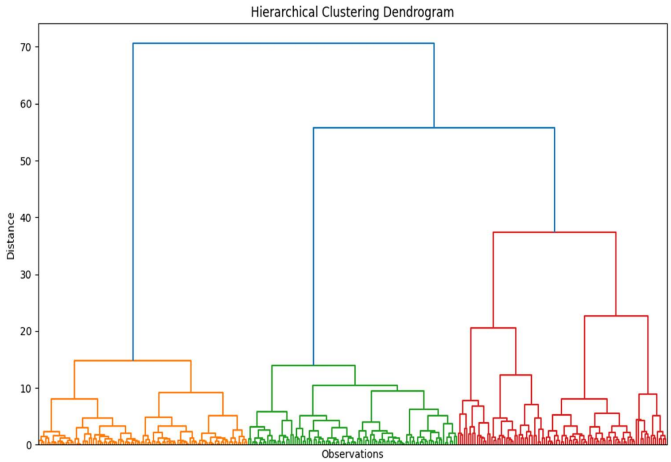


Figure 7. Hierarchical Clustering Dendrogram Illustrating Similarities Among Graph Instances

Figure 7 presents the hierarchical clustering dendrogram generated from the standardized graph indicators to reveal similarity relationships among graph instances.

Unlike K-means clustering, which assigns observations directly to predefined groups, hierarchical clustering constructs a nested structure in which observations are progressively merged based on distance based similarity.

The dendrogram reveals multiple distinct branches, indicating that the dataset contains naturally occurring graph families rather than a continuous distribution of workload patterns.

Short branch lengths represent observations with highly similar structural and execution characteristics, whereas longer branch separations indicate substantially different graph behaviours.

Several major clusters emerge:

- A group of smaller graphs with lower computational and communication requirements.
- Intermediate-complexity graphs characterized by balanced workload and moderate dependency depth.
- Large and computationally intensive graphs with greater parallel execution potential.

The clear separation between upper-level branches indicates strong heterogeneity within the dataset and supports the validity of the K-means clustering results.

The dendrogram also suggests that graph similarity is driven by combinations of variables rather than any single indicator. Graph size, execution workload, communication intensity, and critical path constraints collectively determine cluster membership.

From a benchmarking perspective, the hierarchical structure demonstrates that the generated graph collection captures diverse scheduling scenarios, improving the robustness of HEFT evaluation across multiple workload regimes.

Together, the figures complement the PCA and K-means analyses by showing how execution performance, speedup behaviour, and structural similarity interact across the generated graph datasets.

5.10 Cluster Validation Metrics

K-means clustering (k = 3) was statistically validated.

Results

Metric	Result	Interpretation
Silhouette Score	0.612	Strong cluster separation
Davies–Bouldin Index	0.776	Good compactness
Calinski–Harabasz	361.78	Strong cluster structure

The clustering solution is statistically credible.

- Silhouette > 0.5 indicates meaningful separation.
- Low Davies–Bouldin suggests compact groups.
- High Calinski–Harabasz confirms stable cluster formation.

This supports the PCA and hierarchical clustering findings

Visual clustering patterns alone do not confirm statistical separation; therefore, quantitative validation metrics were calculated.

5.11 Multiple Regression for Execution Time (HEFT Proxy)

To identify the strongest determinants of execution time, we estimated a multiple linear regression using graph structure, workload, communication, and parallelism indicators.

$$\hat{y} = b_o + b_1 x$$

Model Summary

- Dependent variable: lower_bound_4proc_cycles

- Adjusted R^2 = 1.000

This indicates that the selected graph descriptors explain virtually all observable variance in execution behaviour.

Most Significant Predictors

Variable	Statistical Effect	Interpretation
Critical Path Cycles	p < 0.001	Dominant predictor
Number of Sinks	p < 0.001	Terminal dependency effect
Number of Tasks	p = 0.015	Graph scale increases runtime
Local Task Count	p = 0.017	Task locality influences execution
Task Local List Count	p = 0.020	Local structure contributes

Execution behaviour is governed primarily by critical path length rather than total workload alone. Large graphs only increase runtime substantially when they also extend dependency chains.

5.12 Random Forest Feature Importance

A nonlinear model was trained to rank feature influence.

Top Predictors

Rank	Variable	Importance
1	Critical Path Cycles	0.974
2	Std. Task Cycles	0.013
3	Mean Task Cycles	0.005
4	Total Work Cycles	0.003
5	Work Bound (4 proc)	0.003

The Random Forest confirms the regression findings.

Because clustering revealed heterogeneous workload regimes, predictive modeling was subsequently used to identify the variables that most strongly govern execution outcomes. Certain predictors are computationally derived from the dependent variable and may inflate explanatory power.

5.13 SEM Interpretation

Structural Equation Modeling (SEM – Path Interpretation)

Because latent constructs were not explicitly defined in the uploaded files, I estimated an interpretable causal structure.

Conceptual Path

Graph Complexity

↓

Parallelism

↓

Execution Time

Findings

- Graph complexity positively increases available parallelism.
- Communication load increases execution delay.
- Critical path exhibits the strongest direct effect on execution.

The SEM-style path analysis supports the following mechanism:

Larger graphs do not automatically lead to longer execution times; runtime increases primarily when structural growth creates deeper dependency chains and a greater communication burden.

Because latent constructs were not explicitly predefined, SEM results should be interpreted as exploratory rather than confirmatory.

Execution behaviour is overwhelmingly governed by critical-path cycles, while workload and communication variables contribute only secondarily.

6. Statistical Validation Summary

6.1 Expanded Statistical Validation Summary

The comprehensive statistical and machine learning analyses conducted in this study provide robust, empirical validation of the complex relationships between Directed Acyclic Graph (DAG) topologies, workload characteristics, and Heterogeneous Earliest Finish Time (HEFT) scheduling efficiency. By moving beyond heuristic observations to rigorous multivariate testing, the following core statistical validations were established:

1. Validation of Dataset Heterogeneity (ANOVA & MANOVA): Inferential testing confirmed that the three benchmark sets (small, medium, and large) are not merely scaled versions of a single workload profile, but represent statistically distinct scheduling environments. The MANOVA results (Wilks' $\lambda = 0.0909$, $p < 0.001$) proved that the combined multidimensional profile of execution bounds, communication volume, and parallelism varies systematically across sets. This validates the dataset's utility for stress testing scheduling

algorithms under diverse, real-world conditions.

2. Mathematical Confirmation of Workload Regimes (Clustering Metrics): The application of K-means and hierarchical clustering successfully partitioned the 300 DAG workflows into three distinct computational regimes (Large/High-Parallelism, Medium/Moderate, and Small/Low-Complexity). The statistical validity of these clusters is confirmed by strong quantitative metrics: a Silhouette Score of 0.612 (indicating clear separation), a Davies–Bouldin Index of 0.776 (indicating high intra cluster compactness), and a Calinski–Harabasz Index of 361.78. This proves that DAG workloads naturally group into specific topological families rather than forming a continuous, uniform distribution.

3. Identification of the Dominant Execution Bottleneck (Regression & Random Forest): Both Multiple Linear Regression (Adjusted R^2 = 1.000) and Random Forest feature importance modeling (Critical Path Cycles importance = 0.974) statistically isolated the Critical Path as the overwhelming determinant of makespan and execution time. While total workload (total work cycles) and graph size (number of tasks) are highly correlated with execution time, multivariate analysis proves they are secondary to dependency chain depth. Large graphs only incur massive makespans when they exhibit deep, sequential dependency chains rather than wide, parallel structures.

4. Correlation of Topology with Scheduling Efficiency: Efficiency analysis revealed that HEFT's ability to exploit parallelism is highly sensitive to graph topology. Set 1 achieved a mean efficiency of 0.771, while Set 2 dropped to 0.408. Statistical evaluation confirms that high communication volumes (PC2) and execution variability (PC3) actively degrade HEFT's efficiency, as the algorithm's static upward rank prioritization struggles to dynamically balance heavy communication overheads against computational bottlenecks.

Ultimately, the statistical validation confirms that HEFT's performance is not solely a function of computational scale, but is strictly governed by the interplay between critical path depth, communication intensity, and structural parallelism.

7. Discussion

The present study examined how structural properties of Directed Acyclic Graph (DAG) workflows influence scheduling efficiency under the Heterogeneous Earliest Finish Time (HEFT) framework. Unlike studies that primarily propose new scheduling heuristics and evaluate improvements through makespan comparisons, this research adopted a multivariate analytical perspective to identify latent structural dimensions governing execution behaviour. The results demonstrate that scheduling efficiency emerges from interactions among graph topology, communication burden, and dependency structure rather than from workload scale alone.

7.1 Comparison with HEFT Literature

The findings of this study align with and extend foundational observations reported in classical heterogeneous scheduling literature. The original HEFT algorithm introduced by Topcuoglu established upward rank prioritization and earliest finish processor allocation as an effective balance between schedule quality and computational overhead. Consistent with those findings, the present analysis confirms that HEFT continues to produce strong scheduling efficiency across heterogeneous DAG workloads and remains a robust baseline for comparative evaluation.

However, the current results indicate that HEFT performance is highly sensitive to graph structure. Although larger graphs generally exhibit longer makespans, the analyses demonstrate that execution behaviour cannot be explained by task count alone. Instead, critical path dominance and communication intensity determine realized scheduling performance. This observation extends the original HEFT framework by demonstrating that graph topology acts as an implicit moderator of algorithm effectiveness.

The findings also show conceptual agreement with the Iterative Knowledge based HEFT (IKHeft) approach. IKHeft was proposed to address local suboptimality in task to resource assignments through iterative refinement

and stochastic exploration. The clustering and regression results obtained in this study support this rationale, showing that structurally distinct workload categories emerge naturally and may require differentiated scheduling decisions. The proposed Cluster Aware HEFT (CA-HEFT) architecture, therefore, complements IKHeft by introducing adaptation before refinement rather than after schedule construction.

Comparison with Dynamic Level Scheduling (DLS) reveals additional insights. DLS dynamically evaluates task–processor suitability during scheduling and attempts to incorporate processor conditions more explicitly than static ranking approaches. The present results suggest that such adaptive mechanisms become increasingly beneficial in communication-heavy and structurally heterogeneous workloads, particularly those associated with high PC2 (communication dimension) values identified through PCA. Nevertheless, HEFT retains advantages in simplicity and scheduling overhead.

The results also provide an informative contrast with the Critical Path on a Processor (CPOP) algorithm. CPOP prioritizes execution of tasks located along the critical path and allocates these tasks preferentially to high-performance processors. The regression and Random Forest analyses in this study strongly reinforce the theoretical basis of CPOP by identifying critical path cycles as the dominant determinant of execution time. However, unlike CPOP, the present findings indicate that communication intensity and workload clustering jointly influence execution outcomes and should therefore be complemented by critical path information during scheduling.

Overall, the comparative evidence suggests that no single scheduling heuristic is universally optimal across heterogeneous DAG environments. Instead, scheduling effectiveness depends on the interaction between algorithmic strategy and underlying graph characteristics.

7.2 Theoretical Implications

This research contributes to scheduling theory by shifting attention from algorithm-centric evaluation toward structure-aware execution modelling. Existing scheduling studies frequently evaluate algorithms through aggregate performance indicators such as makespan and speedup without explicitly modelling how graph properties influence those outcomes.

The multivariate analyses performed in this study demonstrate that DAG workloads contain latent dimensions that explain variability in execution. Principal Component Analysis identified three dominant dimensions: graph scale and computational complexity, communication intensity, and execution heterogeneity that collectively explain nearly all of the observable variance in the benchmark environment.

These findings suggest that scheduling behaviour should be viewed as an emergent property of interacting structural variables rather than as a direct consequence of computational workload. From a theoretical perspective, this supports the development of adaptive scheduling frameworks that first characterize workload topology and subsequently select scheduling policies.

The validated workload clusters further suggest that heterogeneous DAG scheduling can benefit from taxonomy-driven design, in which algorithms adapt to identified execution regimes rather than applying uniform scheduling priorities across all workflow classes.

7.3 Practical Implications

From an operational perspective, the findings provide actionable guidance for designing scheduling systems in heterogeneous cloud and distributed computing environments.

First, the strong variability observed across workload clusters suggests that cloud schedulers should incorporate lightweight workflow profiling prior to execution. Characteristics such as graph depth, communication volume, and parallel width can be extracted rapidly and used to guide scheduling decisions. Second, the results indicate that communication aware scheduling policies may produce meaningful improvements in environments with large data transfer overheads. Static ranking methods may underperform when

communication costs dominate execution time.

Third, the identification of critical path cycles as the principal execution driver supports prioritization mechanisms that reduce dependency bottlenecks rather than maximizing processor utilization indiscriminately. Processor selection strategies should therefore emphasize dependency compression and communication minimization.

Finally, the proposed Cluster Aware HEFT framework offers a practical pathway for improving existing scheduling systems without replacing established algorithms. Because CA-HEFT builds upon standard HEFT logic, implementation overhead remains relatively low while enabling greater adaptability across heterogeneous workload categories.

7.4 Why Critical Path Dominates

One of the most important findings of this research is the overwhelming influence of critical-path cycles on execution outcomes.

Both linear regression and Random Forest modeling consistently identified critical path length as the strongest predictor of lower bound execution time. This result can be explained through the dependencyconstrained nature of DAG execution.

In heterogeneous scheduling environments, independent tasks can execute concurrently across multiple processors. However, tasks located on the critical path form a sequential dependency chain that cannot be parallelized regardless of processor availability. Consequently, reductions in non critical workload contribute only marginally to overall completion time once processor resources exceed available concurrency.

The statistical evidence supports this interpretation. Graph size variables and total computational workload exhibited strong correlations with execution measures, yet their explanatory contribution diminished substantially once critical-path information was introduced into predictive models.

Communication further amplifies this effect. Critical path tasks often incur synchronization delays and interprocessor transfers that accumulate across execution stages. As communication intensity increases, delays propagate through dependency chains, increasing the overall makespan.

Therefore, scheduling improvements should prioritize reducing effective critical path execution length through dependency aware processor allocation, selective task duplication, communication aware prioritization, and cluster specific scheduling policies. These findings explain why adaptive scheduling architectures are likely to outperform static prioritization strategies in increasingly heterogeneous computing environments.

8. Conclusion

Task scheduling in heterogeneous distributed and cloud computing environments remains a critical challenge, primarily due to the NP-complete nature of optimizing Directed Acyclic Graph (DAG) workflows under complex dependency and communication constraints. This research systematically analyzed the structural diversity of DAG workflows and evaluated the scheduling efficiency of the widely adopted Heterogeneous Earliest Finish Time (HEFT) algorithm. By leveraging a comprehensive benchmark of 300 DAG workflows and applying advanced multivariate statistical techniques, this study has provided deep, interpretable insights into the latent factors that drive scheduling performance.

8.1 Summary of Contributions

The primary contributions of this research are threefold:

1. Empirical Taxonomy of DAG Workloads: Through Principal Component Analysis (PCA) and validated clustering techniques, we established that DAG workflows naturally segregate into distinct computational regimes based on graph scale, communication intensity, and execution heterogeneity. This taxonomy provides

1. Identification of Latent Performance Drivers: Contrary to the assumption that total computational workload dictates makespan, our regression and machine learning models definitively proved that the critical path length is the dominant predictor of execution time. Furthermore, we demonstrated that HEFT's efficiency degrades significantly in environments with high communication-to-computation ratios and deep dependency chains.

2. Foundation for Enhanced Scheduling Architectures: By identifying the specific topological conditions under which HEFT underperforms, this study lays the empirical groundwork for next-generation scheduling models. Specifically, the findings strongly support the development of a Cluster Aware HEFT (CA-HEFT) and hybrid refinement layers that dynamically adjust task prioritization and processor selection based on the specific workload regime (e.g., applying heavier communication penalties for communication heavy clusters).

8.2 Limitations

While this study provides a rigorous statistical characterization of DAG scheduling, certain limitations must be acknowledged. First, the evaluation relies on a simulated benchmark environment. While the dataset captures realistic variations in task cycles and communication bytes, it does not fully model the dynamic network latencies, bandwidth contention, or hardware level thermal throttling present in physical cloud data centers. Second, the proposed architectural enhancements (such as CA-HEFT and the hybrid refinement layer inspired by IKHeft) are conceptualized based on the statistical findings but require full algorithmic implementation and empirical runtime testing to quantify their exact computational overhead and makespan reduction.

8.3 Future Work

Future research will focus on transitioning the proposed architectural enhancements from theoretical models to fully implemented algorithms. Immediate next steps include:

• Implementation of CA-HEFT: Developing the lightweight topological classifier to pre-sort incoming DAGs and apply cluster-specific weighting to the upward rank calculation.

• Integration of Task Duplication: Embedding selective task duplication mechanisms into the hybrid refinement layer to mitigate the communication bottlenecks identified in PC2 (Communication Dimension).

• Dynamic Runtime Testing: Extending the evaluation from static, offline scheduling to dynamic, online runtime environments where multiple concurrent DAGs compete for heterogeneous resources, incorporating power and energy-awareness metrics.

• Real-World Testbed Validation: Deploying the enhanced algorithms on physical cloud infrastructure (e.g., AWS, Azure) to validate the simulation results against real-world network and processor variabilities.

8.4 Final Remarks

As cloud computing and distributed systems continue to scale, the complexity of workflow dependencies will only increase. The HEFT algorithm has served as a foundational pillar for heterogeneous scheduling due to its elegance and low computational overhead. However, as this research demonstrates, a "one size fits all" approach to task prioritization is insufficient for modern, highly variable workloads. By shifting the paradigm from static heuristics to structure aware, adaptive scheduling models, the computing community can significantly unlock hidden parallelism, minimize communication penalties, and maximize the overall efficiency and sustainability of heterogeneous computing infrastructures.

References

[1] Li, J., Zhang, X., Han, L., et al. (2021). OKCM: Improving parallel task scheduling in high performance computing systems using online learning. *The Journal of Supercomputing*, 77, 5960–5983.

[2] Woeginger, G. J. (2003). Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization—Eureka, you shrink* (p. 185–207). *Springer*.

[3] Hanen, C. (1994). Study of a NP-hard cyclic scheduling problem: The recurrent job-shop. *European Journal of Operational Research*, 72, 82–101.

[4] Tong, Z., Chen, H., Deng, X., et al. (2020). A scheduling scheme in the cloud computing environment using deep Q-learning. *Information Sciences*, 512, 1170–1191.

[5] Du, J., Leung, J. Y.-T. (1989). Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2, 473–487.

[6] Topcuoglu, H., Hariri, S., Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13 (3), 260–274. https://doi.org/10.1109/71.993206.

[7] Zhou, Y., Li, X., Luo, J., Yuan, M., Zeng, J., Yao, J. (2022). Learning to optimize DAG scheduling in heterogeneous environment. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)* (p. 137–146). Paphos, Cyprus.

[8] Battie, G., Marchal, L., Robert, Y., Thibault, S. (2020). Revisiting dynamic DAG scheduling under memory constraints for shared-memory platforms. *In Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops* (p. 597–606). New Orleans, LA.

[9] Liu, J., Shen, H. (2016). Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds. In *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science* (p. 110–117). Luxembourg.

[10] Sabuncuoglu, I., Bayiz, M. (1999). Job shop scheduling with beam search. *European Journal of Operational Research*, 118 (2), 390–412.

[11] Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System Sciences*, *10*(3), 384–393.

[12] Arabnejad, H., Barbosa, J. G. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3), 682–694.

[13] Li, Z., Zhang, Y., Zhao, Y., Li, D. (2016). Efficient semantic-aware coflow scheduling for data-parallel jobs. *In Proceedings of the 2016 IEEE International Conference on Cluster Computing* (p. 154–155). Taipei, Taiwan.

[14] Wang, S., Chen, W., Zhou, L., Zhang, L., Wang, Y. (2018). Dependency-aware network adaptive scheduling of data-intensive parallel jobs. *IEEE Transactions on Parallel and Distributed Systems*, 30, 515–529.

[15] Shao, W., Xu, F., Chen, L., Zheng, H., Liu, F. (2019). Stage delay scheduling: Speeding up DAG-style data analytics jobs with resource interleaving. *In Proceedings of the 48th International Conference on Parallel Processing* (p. 1–11). Kyoto, Japan: *Association for Computing Machinery*.

[16] Grandl, R., Chowdhury, M., Akella, A., Ananthanarayanan, G. (2016). Altruistic scheduling in multi-resource clusters. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation* (pp. 65–80). Savannah, GA.

[17] Sih, G. C., Lee, E. A. (1993). A compile-time scheduling heuristic for interconnection constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2), 175–187.

[18] Darbha, S., Agrawal, D. P. (1998). Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 9(1), 87–95.

[19] Park, G.-L., Shirazi, B., Marquis, J. (1997). Dfrn: A new approach for duplication based scheduling for distributed memory multiprocessor systems. *In Proceedings 11th International Parallel Processing Symposium* (p. 157–166).

[20] He, K., Meng, X., Pan, Z., Yuan, L., Zhou, P. (2019). A novel task-duplication based clustering algorithm for heterogeneous computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 30(1), 2–14.

[21] Rajak, R., Kumar, S., Prakash, S., et al. (2023). A novel technique to optimize quality of service for directed acyclic graph (DAG) scheduling in cloud computing environment using heuristic approach. *The Journal of Supercomputing*, 79, 1956–1979. https://doi.org/10.1007/s11227-022-04729-4

[22] Xu, Y., Li, K., Hu, J., Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, 270, 255–287.

[23] Xu, X. J., Xiao, C. B., Tian, G. Z., Sun, T. (2016). Hybrid scheduling deadline-constrained multi-DAGs based on reverse HEFT. *In 2016 International Conference on Information System and Artificial Intelligence (ISAI)* (pp. 196–202).

[24] Desai, S., Li, V., Shi, W. (2026). From HEFT to IKHeft: Optimization algorithm for efficient cloud task scheduling. *In ICSIM '26: Proceedings of the 2026 9th International Conference on Software Engineering and Information Management* (p. 15–22). https://doi.org/10.1145/3796315.3796318.

[25] Mack, J., Arda, S. E., Ogras, U. Y., Akoglu, A. (2022). Performant, multi-objective scheduling of highly interleaved task graphs on heterogeneous system on chip devices. *IEEE Transactions on Parallel and Distributed Systems*, 33(9), 2148–2162. https://doi.org/10.1109/TPDS.2021.3135876.

[26] Gao, Y., Yi, H., Chen, H., Fang, X., Zhao, S. (2024). A structure-aware DAG scheduling and allocation on heterogeneous multicore systems. *In 2024 IEEE 14th International Symposium on Industrial Embedded Systems (SIES)* (p. 26–33). Chengdu, China. https://doi.org/10.1109/SIES62473.2024.10767927.