Prediction of Completion Time of an Application using Decision Tree Regression Model

Master of Computer Application
In the Faculty of Engineering & Technology
Jadavpur University

By

SOUMYAJIT HAIT

Exam Roll No: MCA2360040
Registration No: 154250 of 2020-21

Under the Guidance of
Prof. Nandini Mukhopadhyay
Department of Computer Science & Engineering

Department of Computer Science & Engineering
Jadavpur University
Kolkata - 700 032
2023

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Recommendation

This is to certify that the dissertation entitled **“Prediction of Completion Time of an Application using Decision Tree Regression Model”** has been carried out by Soumyajit Hait (University Registration No.: 154250 of 2020- 2021, Examination Roll No.: MCA2360040) under my guidance and supervision and be accepted in partial fulfilment of the requirement for the Degree of Master of Computer Application. The research results presented in this work have not been included in any other paper submitted for the award of any degree in any other University or Institute.

....

Prof. Nandini Mukhopadhyay (Supervisor)
Professor
Department of Computer Science and Engineering
Jadavpur University, Kolkata-32

Countersigned

....

Prof. Nandini Mukhopadhyay
Head, Department of Computer Science and Engineering,
Jadavpur University, Kolkata-32.

....

Prof. Ardhendu Ghosal
Dean, Faculty of Engineering and Technology,
Jadavpur University, Kolkata-32.

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Approval

The foregoing project report entitled "Prediction of Completion Time of an Application using Decision Tree Regression Model" is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood by this approval the undersigned do not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn therein but approve the report only for the purpose for which it has been submitted.

....

Signature of Examiner 1

Date:

....

Signature of Examiner 2

Date:

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this work entitled "Prediction of Completion Time of an Application using Decision Tree Regression Model" contains an original work by the undersigned candidate as part of my Degree of Master of Computer Application.
All information has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Soumyajit Hait

Registration No: 154250 of 2020- 2021

Exam Roll No.: MCA2360040

………………………………………………
Signature with Date

ACKNOWLEDGEMENTS

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my postgraduate experience has been one that I will cherish forever.

Foremost, I would like to express my profound gratitude and sincere thanks to my adviser, **Dr. Nandini Mukhopadhyay**, **Professor, Department of Computer Science & Engineering**, **Jadavpur University**, for her valuable suggestions, guidance, constant encouragement and intent supervision at every stage of my work. I have been amazingly fortunate to have her as my project Supervisor who gave me the freedom to explore on my own, and at the same time guided me to recover when my steps faltered. It has been a great learning process for me.

Moreover, Prof. **Nandini Mukhopadhyay** has been always there to listen and provide valuable advice. I am deeply grateful to her for the long discussions that helped me sort out the technical details of my work. I am thankful to her for her insightful comments and constructive criticisms at different stages of my research which were thought-provoking and helped me focus on my ideas. I am also thankful to her for encouraging the use of correct grammar and consistent notation in my writings and for carefully reading and commenting on countless revisions of this manuscript.

I am grateful to him for allowing me to use the computing facilities of “DST” laboratory, Department of Computer Science and Engineering, Jadavpur University.

I am also thankful to the system staff who maintained all the machines in my lab so efficiently that I never had to worry about viruses, losing files, creating backups or installing software.

Many friends have helped me stay sane through these difficult years. Their support and care helped me overcome setbacks and stay focused on my postgraduate study. I greatly value their friendship and I deeply appreciate their belief in me.

Most importantly, none of this would have been possible without the love and patience of my family. My family, to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express my heart-felt gratitude to my family.

Date:

Place: Kolkata

SOUMYAJIT HAIT
M.C.A. (C.S.E)
Roll No: **MCA2360040**

CONTENT

Abstract:

Efficiently estimating the completion time of applications is crucial for project planning, resource allocation, and meeting stakeholders' expectations. In this study, we propose a predictive model using the Decision Tree Regression (DTR) algorithm to accurately forecast the completion time of software applications during the development lifecycle. The dataset used for training and testing the model comprises historical records of completed projects, containing various application characteristics and their corresponding actual completion times.

First, we preprocess the dataset to handle missing values, normalize numerical features, and encode categorical variables appropriately. We then employ the Decision Tree Regression algorithm to build a predictive model by recursively partitioning the data based on feature thresholds, resulting in a tree-like structure. The DTR model offers interpretability, which allows us to gain insights into the important factors influencing the completion time.

To evaluate the model's performance, we adopt various metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R2) score. Additionally, we compare the DTR model's performance with other regression models, such as Linear Regression and Random Forest Regression, to showcase its effectiveness in handling application completion time prediction.

The experimental results demonstrate that the proposed Decision Tree Regression model outperforms other traditional regression methods in accurately forecasting the completion time of applications. Moreover, the model's interpretability aids project managers in understanding the underlying relationships between application features and completion time, facilitating better decision-making and resource management.

In conclusion, our study presents a robust approach to predict the completion time of software applications using the Decision Tree Regression model. The proposed methodology can significantly benefit software development projects, providing stakeholders with reliable estimates for better planning and execution, ultimately leading to improved project success rates.

CHAPTER 1

1. INTRODUCTION

The introduction of high-performance computing systems, as well as the increased demand for processing large-scale scientific and data-intensive applications, has prompted academics to develop efficient task scheduling techniques for parallel computing platforms. DAG-based task models have developed as a standard representation for a wide range of applications, from scientific simulations to data processing workflows.

The HEFT algorithm is used to optimize the completion time of a DAG-based task plan. The completion time, which indicates the time it takes to execute all tasks in the DAG, is an important performance metric that influences the overall efficiency of the system. Reducing completion time can result in faster outcomes, higher throughput, and better resource use.

In this work, we investigate how well the HEFT algorithm schedules tasks on different types of computational resources. In contemporary parallel computing settings, heterogeneous resources are frequent and have a range of processing and communication rates. The goal of the algorithm is to reduce the total execution time of the DAG by locating a task-to-resource mapping that strikes a balance between communication and computation expenses.

With mean task weights, a communication-to-computation ratio, and resource characteristics all taken into account, we offer an updated DAG construction and scheduling algorithm. We also assess the algorithm's performance by calculating its average calculation cost, average communication cost, and completion time, and we compare it to other scheduling strategies.

The findings of this study help to clarify how effective job scheduling algorithms for parallel computing systems work. We intend to provide useful insights for optimizing the execution of large-scale applications in distributed and parallel contexts by investigating the performance of the HEFT algorithm.

The overall goal of this work is to increase the effectiveness and performance of task scheduling in parallel computing systems, hence boosting the capabilities of high-performance computing applications such as data processing and scientific simulations. By reducing the completion time, we hope to develop task scheduling technology and advance the field of parallel and distributed computing as a whole.

1.1 BACKGROUND

Directed acyclic graphs (DAGs) have become an indispensable tool for modelling complex systems in a variety of fields, such as parallel computing, bioinformatics, and project management. A DAG is made up of nodes that represent tasks or events and directed edges that show the dependencies between them. The graph is acyclic because there are no cycles, which enables effective task scheduling and parallel processing.

The jobs within a DAG frequently have different processing timeframes, dependencies, and resource needs in real-world applications. Consequently, it is difficult to anticipate the completion time of such DAGs with any degree of accuracy. The estimation of the completion time is essential for meeting deadlines, allocating resources efficiently, and enhancing system performance as a whole.

Analytical techniques, simulation-based methods, and heuristic techniques are common methodologies for predicting the completion times of DAGs. But these approaches frequently have drawbacks, including a lack of generality, a high computational complexity, and a reliance on parameterized data from a particular area.

1.2 MOTIVATION

Large computations are carried out by supercomputers and high-performance computing facilities like clusters. Although these infrastructures are designed specifically for computers, they are challenging to set up, maintain, and use. Scientists now have a completely new way of exploiting the computing infrastructure thanks to Grid and Cloud computing, which contrasts with such a specialized model of computation. When not in use, resources can be dynamically provided on a pay-per-use basis and released. Consequently, As a result, computationally intensive scientific applications are appropriate for a distributed environment, where users don't need to own specialized resources and where high-capacity and high-availability resources can Users only pay for the time that they use the resources, which are reserved or accessed on demand. But accurate resource requirement forecasting is required in order to maximize the rates paid by the customers. As a result, it is necessary to develop a job modelling scheme for predicting the resource needs of jobs, particularly with regard to scientific applications.

It can be shown that scientific applications typically involve extensive computations that call for computing resources with high performance and high availability. To complete the computation with a satisfactory level of efficiency, effective algorithms and appropriate data structures may need to be used. The majority of scientific applications rely on the deft interplay of many approximations and ask for in-depth scientific expertise. In many cases, the identical code segments are used in scientific applications to carry out specific numerical operations that are widespread in various applications. In many scientific applications, there may therefore be a lot of comparable parts. These applications are typically created by scientists, who have a thorough understanding of their field but might not be familiar with current methods and skills in software engineering. Therefore, although the structure and syntax of various code pieces created by scientists may appear to be different, they may actually execute the same function. Scientific applications also have another crucial quality. They are created and developed once, then utilized again with the same or other datasets.

A software-clone based job modelling technique is suggested with the aforementioned traits in mind. In accordance with this concept, static and dynamic data (including performance data) connected to jobs that have already been run inside a distributed environment (Grid or Cloud) are maintained in an Execution History. A freshly submitted job's clone level is determined in relation to other jobs that were previously executed in the environment. The resource needs of the new job are then estimated using performance information from clone jobs that have been saved in the Execution History.

1.3 <u>OBJECTIVE</u>

Research Goals: The main goals of this thesis are as follows:

a) Use an application that has already been scheduled and is using the same set of resources to estimate the time it will take for an application that can be described in terms of a DAG to complete.

b) To research various machine learning techniques, including supervised and unsupervised approaches for building prediction models.

c) To use case studies and experiments to show how well-suited and useful the generated models are in practical situation.

By attaining these goals, we hope to develop DAG completion time prediction methods and encourage the use of machine learning techniques in this field.

1.4 <u>OUTLINE OF THE PROJECT</u>

The rest of the project is organized as follows:

<u>Chapter2</u>

Explanation of resource management in the context of the project. A literature survey summarizing the existing knowledge and research related to resource management in this domain.

<u>Chapter3</u>

Description of the methodology proposed for the project, which will be used for predicting the completion time of an application.

<u>Chapter4</u>

The process of creating a dataset for the project. Extraction of relevant features from the dataset, which will be used in the decision tree regression model.

Chapter 5

Explanation of the working methodology, i.e., how the decision tree regression model is implemented to predict the completion time of an application.

Chapter 6

An overview of the entire work conducted in the project. Discussion about the limitations faced during the project.

CHAPTER 2

2.1 RESOURCE MANAGEMENT

In the context of our project on predicting DAG completion time, resource management played a crucial role in maximizing the efficiency and effectiveness of DAG executions. Efficiently allocating and optimizing resources is vital for meeting performance objectives and reducing overall computation time. We adopted a multi-faceted approach to resource management to ensure seamless workflow execution and optimal resource utilization.

Resource Allocation and Task Scheduling:

Resource allocation and task scheduling are fundamental aspects of efficient DAG execution. To enhance the utilization of available resources, we implemented intelligent task scheduling algorithms. Our system dynamically assigns tasks to appropriate resources based on task characteristics, resource availability, and dependency relationships among tasks. By considering data locality and resource constraints, we achieved task parallelism and minimized resource contention, leading to faster DAG completion times.

Load Balancing Strategies:

Load balancing is crucial in distributed computing environments, especially when executing large-scale DAGs. Unevenly distributed tasks can result in resource bottlenecks, leading to delayed or stalled DAG execution. To address this challenge, we designed and integrated load balancing strategies that evenly distribute tasks across computational resources. Our load balancing techniques dynamically adapt to workload variations, ensuring equitable resource utilization and optimized execution times.

Cloud-Based Resource Scaling:

Leveraging the elastic nature of cloud computing, we implemented resource scaling mechanisms to handle varying DAG workloads. Our system automatically scales resources up or down based on workload demands, ensuring that DAGs with different complexities are executed efficiently. Cloud-based resource scaling also enabled cost optimization, as resources are provisioned only when needed, reducing operational expenses.

Performance Monitoring and Resource Optimization:

To continuously improve resource management, we established a comprehensive performance monitoring system. Real-time monitoring of resource usage, task completion rates, and network latencies enabled us to identify performance bottlenecks and resource inefficiencies. We proactively addressed issues such as resource contention, overloaded nodes, and data transfer delays through adaptive resource optimization strategies, resulting in reduced completion times and improved overall system performance.

Energy-Aware Resource Management:

Considering the environmental impact and cost of energy consumption in cloud computing environments, we also incorporated energy-aware resource management techniques. Our system dynamically optimized resource usage based on energy consumption profiles, minimizing the carbon footprint while maintaining high performance. By striking a balance between computational efficiency and energy conservation, we contributed to sustainable and eco-friendly computing practices.

The resource management strategies employed in our project not only facilitated accurate DAG completion time predictions but also contributed to scalable, energy-efficient, and cost-effective workflow executions in cloud computing environments. By effectively managing resources, our system significantly enhanced the overall performance and user experience in the context of DAG-based applications.

2.2 LITERATURE SURVEY

A large application that is composed of multiple tasks or jobs can be represented as a directed acyclic graph (DAG), where each task of the application is represented by a node of the DAG, and dependencies among them are represented by directed edges. Each of the tasks has a non-zero execution time. There is a data transfer among dependent tasks, and there is a non-zero data transfer time between dependent tasks if they are executed on different resources. In an environment like the cloud, cloud service users (CSUs) place their resource demands with the cloud service providers, and virtual resources are reserved as per the demand of the CSUs. These virtual resources are available on a reservation basis, and different reservation costs can be incurred as per the reservation policy of the cloud service provider. Over-prediction of these virtual resources will lead to non-utilization of resources, while under-utilization will lead to higher costs. As a result, accurate job demand prediction is critical, both in terms of price and timing. However, the problem of demand prediction is well known for its difficulty, as it belongs to the NP-Complete class. . A feedback-guided job modelling scheme is discussed in, where the history of previously executed jobs is stored in a history. When a new job is created, its similarity to one or more previous jobs is determined. Based on these similarities, the resource requirement of a new job is predicted. However, in all of the previous work, only dependent tasks were considered. DAG scheduling algorithms are classified as heuristics-based or learning-based, with heuristics-based algorithms further subdivided into list scheduling, cluster-based scheduling, and task duplication-based scheduling algorithms. A task duplication-based learning algorithm is discussed in. The scheduling length or make span prediction of a newly arrived DAG is not discussed in any of the previous work.

CHAPTER 3

PROPOSED METHODOLOGY

Problem Statement

A parallel/distributed application is decomposed into multiple tasks with data

dependencies among them. In our model an application is represented by a directed acyclic graph (DAG) that consists of a tuple G = (V, E, P, W, data, rate), where V is the set of v nodes/tasks, E is the set of e edges between the nodes, and P is the set of processors available in the system. (In this paper task and node terms are used interchangeably used.) Each edge (i, j) ∈ E represents the task-dependency constraint such that task n_i , should complete its execution before task r_{ij} can be started.

W is a v x p computation cost matrix, where v is the number of tasks and p is the number of processors in the system. Each W_{ij} gives the estimated execution time to complete task n_i on processor p_j. The average execution costs of tasks are used in the task priority equations. The average execution cost of a node n,- is defined as $\overline{w_i} = \sum_j^p w_{i,j} /p$. Data is a v x v matrix for data transfer size (in bytes) between the tasks. The data transfer rates (in bytes/second) between processors are stored in a p x p matrix, rate.

The communication cost of the edge (i, j), which is for data transfer from task n_i (scheduled on p_m) to task n. j (scheduled on p_n), is defined by $c_{i,j}$ = data(n_i,r_j)/rate(p_m,p_n). When both n,- and rij are scheduled on the same processor, p_m = p_n, then C_{jj} becomes zero, since the intra-processor communication cost is negligible compared with the inter processor communication cost. The average communication cost of an edge is defined by $\overline{c_{i,j}} = data(n_i, n_j)/\overline{rate}$, where $\overline{rate}$ is the average transfer rate between the processors in the domain.

Proposed Algorithm

In this section we present our scheduling algorithms the Heterogeneous Earliest

Finish Time (HEFT) Algorithm.

HEFT Algorithm

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \left(\overline{c_{i,j}} + rank_u(n_j) \right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

The Heterogeneous-Earliest-Finish-Time (HEFT) algorithm is a scheduling algorithm designed to efficiently schedule tasks in a directed acyclic graph (DAG) onto a bounded number of heterogeneous processing elements (PEs). The goal is to minimize the total execution time of the application by assigning tasks to processors in an optimal manner. The algorithm prioritizes tasks based on their upward rank and uses the earliest finish time (EFT) to determine the best processor for each task.

Here's a step-by-step explanation of the HEFT algorithm:

1. Upward Rank ($rank_u(n_i)$) Calculation (Equation 1):

   - For each task n_i in the DAG, calculate its upward rank ($rank_u(n_i)$). The upward rank of a task n_i is recursively defined as the sum of the estimated computation time (w_i) for task n_i and the maximum value obtained by considering all its immediate successor tasks r_{ij}.

   - The computation time (w_i) for task n_i is estimated as the average execution cost (W_{ij}) of task n_i on all available processors (p_j) divided by the number of processors (P) in the system. This average execution cost represents the expected execution time for task n_i on any processor.

2. **Task Priority Order**

   - Sort the tasks in descending order based on their upward rank ($rank_u$). The tasks with higher upward rank values will be scheduled first, as they are more critical to the overall execution time.

3. **Processor Selection (Earliest Finish Time - EFT)**

   - The algorithm then proceeds to assign each task to an appropriate processor. For non-insertion-based scheduling algorithms like HEFT, the earliest available time of a processor p_j, denoted as T-Available[j], is calculated as follows:

     - T-Available[j] is initially set to 0 for all processors.

- When scheduling a task n_i on processor p_j, its earliest finish time (EFT) is determined by considering the completion time of the last assigned node on p_j (T-Available[j]) and the communication cost (C_{ij}) between the last assigned task and task n_i on processor p_j.

- EFT (n_i, p_j) = W_{ij} + max(T-Available[j], EFT (n_m, p_k) + $C_{m, i}$)

- Here, W_{ij} is the estimated computation time for task n_i on processor p_j, and max(...) is used to select the maximum value between the current earliest finish time and the time needed for communication (if any) from the last assigned task (n_m) on processor p_k to task n_i on processor p_j.

4. **Task Scheduling**

- Assign task n_i to the processor that yields the minimum EFT (n_i,p_j).
- Update T-Available[j] to be EFT (n_i, p_j) for the selected processor p_j.

5. **Repeat Steps 3-4**

- Continue the scheduling process until all tasks are assigned to processors.

The HEFT algorithm uses the upward rank to prioritize tasks and then employs the EFT calculation to find the best processor for each task, taking into account the communication costs between tasks scheduled on different processors. By considering both computation and communication costs, the HEFT algorithm aims to minimize the overall execution time of the application in a heterogeneous computing environment.

CHAPTER 4

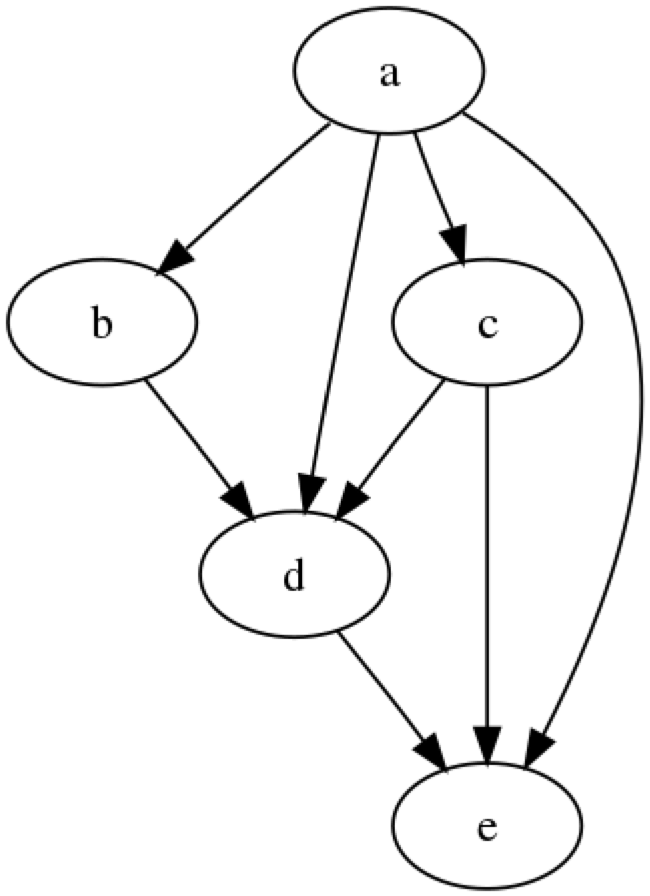
4. CONSTRUCTION OF DATASET

4.1 DAG CREATION:

Definition:

A graph is formed by vertices and by edges connecting pairs of vertices, where the vertices can be any kind of object that is connected in pairs by edges. In the case of a directed graph, each edge has an orientation, from one vertex to another vertex. A path in a directed graph is a sequence of edges having the property that the ending vertex of each edge in the sequence is the same as the starting vertex of the next edge in the sequence; a path forms a cycle if the starting vertex of its first edge equals the ending vertex of its last edge. A directed acyclic graph is a directed graph that has no cycles.

A vertex *v* of a directed graph is said to be reachable from another vertex *u* when there exists a path that starts at *u* and ends at *v*. As a special case, every vertex is considered to be reachable from itself (by a path with zero edges). If a vertex can reach itself via a nontrivial path (a path with one or more edges), then that path is a cycle, so another way to define directed acyclic graphs is that they are the graphs in which no vertex can reach itself via a nontrivial path



The Directed Acyclic Graph (DAG) is generated to represent the tasks and their dependencies. The generated DAG is a matrix representation, where each element of the matrix indicates whether there is a directed edge between two tasks.

Generating DAG:

Here we use a fixed height and node_per_level to construct the DAG. It creates a DAG of n levels with m nodes per level (except the last level, which may have fewer nodes). The nodes are numbered from 0 to `n_task - 1`.

The DAG generation logic ensures that each node in the DAG is connected to some other nodes in the subsequent levels, forming a directed acyclic graph. The connectivity is determined by the variables `height`, `node_per_level`, and the number of tasks `n_task`.

For example,

if `height = 14`, and `node_per_level = 7`, the code creates a DAG with 14 levels, and each level (except the last) has 7 nodes. Each node in level i is connected to some nodes in level i+1 to form dependencies.

The matrix representation `matrix` is used to store the DAG. The value of `matrix[i][j]` is 1 if there is a directed edge from node i to node j, and 0 otherwise. This matrix is printed as `matrix_new` after generation.

```
[[0 1 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
        ...
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 0]]
```

<u>Task Weights</u>

The code generates random weights for each task within a range determined by `Beta`. The weights represent the computation cost of each task. These weights are stored in the `weight_task` list.

<u>Final DAG with Weights</u>

The code multiplies the weights with the CCR to determine the communication cost between tasks. The result is stored in the `matrix_new` matrix, representing the final DAG with task weights and communication costs.

The DAG generation process is based on a predefined height and the number of nodes per level. This may not be a flexible or realistic way to represent real-world task dependencies. The generated DAG structure can vary depending on these input parameters. In practical scenarios, DAGs may have more complex structures and dependencies that need to be modeled more explicitly.

```
[[  0 126 126 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
          ...
 [  0   0   0 ...   0   0 117]
 [  0   0   0 ...   0   0 118]
 [  0   0   0 ...   0   0   0]]
```

4.2 <u>FEATURE EXTRACTION</u>

The process of extracting features from a Directed Acyclic Graph (DAG) entails numerically representing the graph's topology and node properties for use in machine learning applications. There are no cycles (directed paths that begin and terminate at the same node) and edges in DAGs have a definite direction.

There are two types of Feature Extraction—

1. Before Execution feature extraction.
2. After Execution feature extraction.

<u>BEFORE EXECUTION FEATURE</u>:

1. Average of average Computation Cost of the Task (AACOMPCT)
2. 25_percentile of AACOMPCT
3. 50_percentile of AACOMPCT
4. 75_percentile of AACOMPCT
5. Standard Deviation of AACOMPCT
6. Maximum of AACOMPCT
7. Minimum of AACOMPCT
8. 25_Percentile of rank up
9. 50_Percentile of rank up
10. 75_Percentile of rank up
11. Standard Deviation of rank up
12. Average of rank up
13. Maximum of rank up
14. Minimum of rank up
15. 25_Percentile of rank down
16. 50_Percentile of rank down

17. 75_Percentile of rank down
18. Standard Deviation of rank down
19. Average of rank down
20. Maximum of rank down
21. Minimum of Rank down
22. Average of average Communication cost of task (AACOMMCT)
23. Maximum of AACOMMCT
24. Minimum of AACOMMCT
25. 25_Percentile of AACOMMCT
26. 50_Percentile of AACOMMCT
27. 75_Percentile of AACOMMCT

<u>AFTER EXECUTION FEATURE:</u>

The Completion Time.

Here to predict the completion time we use the HEFT Algorithm.

CHAPTER 5

5.WORKING METHODOLOGIES

Machine Learning is a branch of Artificial Intelligence that aims to provide computers with the ability to learn how to perform specific tasks without being explicitly programmed by a human. This technique is based on the design of models that learn from data and make decisions or predictions when new data are available. Deep Learning can be seen as an evolution of machine learning that uses a structure of multiple layers called Artificial Neural Network (ANN). Deep learning algorithms require less involvement of humans because features are automatically extracted. However, an important difference with respect to other machine learning techniques is that deep learning requires massive data to work properly.

Although deep learning are recent concepts, the first computer learning program was written by Arthur Samuel in 1952 and the first neural network was proposed by Frank Rossenblatt in 1957. Since the 1990s, the development in both ML and deep learning has been significant, mainly due to the increment of computation power and the availability of large amounts of data.

There exist many machine learning approaches that can be applied to solve different problems. In this section, we will discuss only those algorithms that have been used for predicting pollutant measures and rain prediction. We can distinguish between the ones based on regression analysis and the ones using neural networks. Moreover, in the first category we will distinguish between the use of classical regression algorithms and machine learning algorithms.

- **Decision Tree:** The aim of this algorithm is to design a model for predicting a quantitative variable from a set of independent variables. The algorithm is based on a recursive partitioning. Trees are composed of decision nodes and leaves. regression usually is built by considering the standard deviation reduction to determine how to split a node in two or more branches. The root node is the first decision node that is divided on the basis of the most relevant independent variable. Nodes are split again by considering the variable with the less sum of squared estimate of errors (SSE) as the decision node. The dataset is divided based on the values of the selected variable. The process finishes when a previously established termination criterion is satisfied. The last nodes are known as leave nodes and provide the dependent variable prediction. This value corresponds to the mean of the values associated to leaves. In **Figure 1** Balogun and Tella (2022) [16] shows a graphical representation of the general structure of a standard Decision Tree.

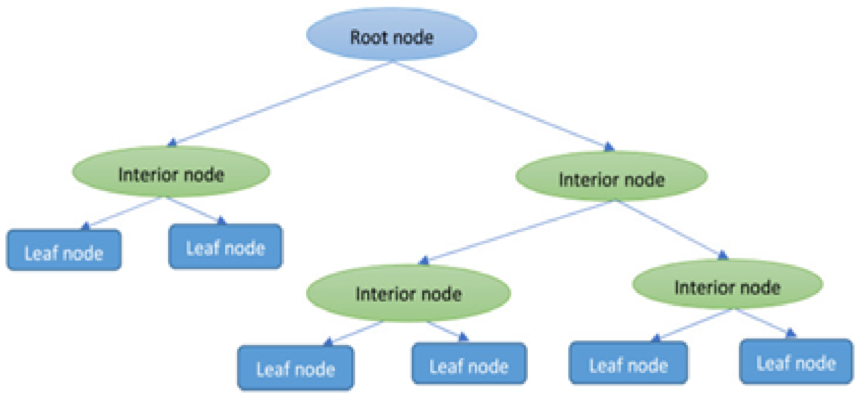


Figure 1 Decision tree structure

RESULTS AND DISCUSSION

In this section, we take a closer look at the datasets we used in our study. We provide detailed information about where the data came from, how we collected it, and the steps we took to prepare it for analysis. We then present the results sharing what we found and how well different models and algorithms performed.

EVALUATION METRICS

In this study, Mean Squared Error and r2 score metrics have been used. These validation metrics are used to determine whether regression models are accurate or misleading.

- **Mean Squared Error (MSE):** Mean squared error (MSE) is a popular evaluation metric in regression analysis, used to quantify the average squared difference between the predicted values and the true values. It is calculated by taking the sum of the squared residuals (the differences between the predicted and true values) and dividing it by the total number of observations. The formula for MSE is:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

where n is the number of observations, y represents the true values, and $\hat{y}$ denotes the predicted values. The MSE provides a comprehensive measure of the model's accuracy, with larger errors being emphasized due to the squaring operation. Minimizing the MSE during model training helps to identify the best-fitting model that minimizes the overall squared differences, indicating a closer alignment between the predicted and actual values.

- **R^2 score:** Coefficient of determination also called as R2 score is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.
  Mathematical Formula:
  $$R^2 = 1 - SS_{res} / SS_{tot},$$
  Where, SS_{res} is the sum of squares of the residual errors.
  SS_{tot} is the total sum of the errors.

DETAIL EXPERIMENTAL FINDINGS

In this section, we will discuss the results we get after evaluating all models which we mention previously.

In this dataset, we evaluate the decision tree model. After evaluate the models, we calculate the errors which are shown in **Table 1.**

Table 1Errors using Decision Tree

Parameter	Value
MSE	2.578703333333334
R^2 SCORE	0.9757867264585409

and we plot the actual values and the predicted values which is shown in **Figure 2**.

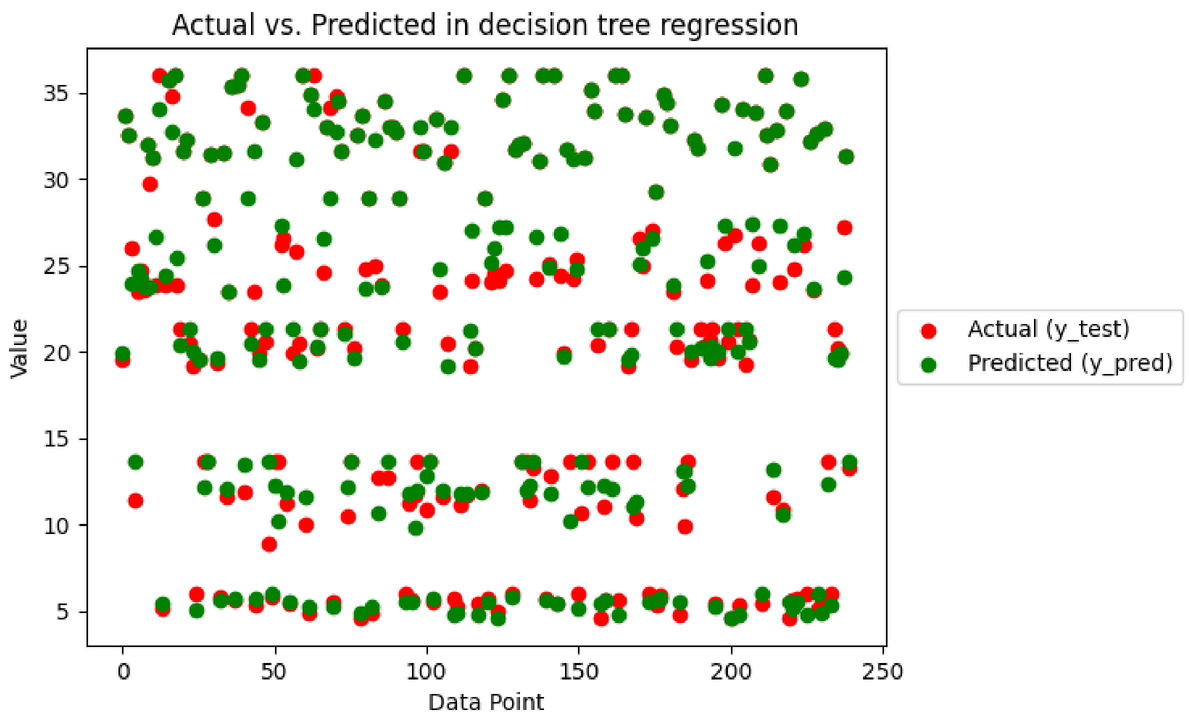


Figure 2 Actual vs Predicted values using decision tree

CHAPTER 6

CONCLUSION

6.1 OVERVIEW OF LOOK

In this thesis, the goal was to reduce the Directed Acyclic Graphs (DAGs) completion time by task scheduling tasks on parallel computing systems using the HEFT algorithm. The study examined the performance of the HEFT algorithm in modern parallel computing environments with heterogeneous resources on various kinds of computational resources.

The development of an updated DAG construction and scheduling method that considers mean job weights, the communication-to-computation ratio, and resource characteristics is one of the study's major conclusions. Average calculation costs, average communication costs, and completion times were used to evaluate the algorithm's performance in comparison to other scheduling techniques.

The study showed that getting quicker DAG completion times requires effective resource management and work scheduling. The system might accomplish task parallelism and decrease resource contention, leading to increased efficiency, by intelligently distributing jobs to suitable resources and utilizing load balancing algorithms.

The paper also suggested a method for predicting the resource needs of jobs based on software clones, particularly for scientific applications in distributed contexts like Grid and Cloud computing. To estimate the resource requirements of new jobs, this method used historical execution data, which improved resource use and reduced costs.

Machine learning approaches were investigated to further improve the efficacy of work scheduling and resource management. Different machine learning methods were tested for this purpose as features were taken from the DAGs to forecast the execution time of new jobs. Decision Tree was one of the machine learning methods employed, and it showed promising results with a high R2 score and a reasonably low Mean Squared Error.

6.2 <u>LIMITATIONS</u>

There are several restrictions to take into account, despite the fact that the research produced substantial findings and insightful information. The study concentrated on particular application kinds and DAG topologies, and the efficiency of the suggested solutions may change depending on the circumstances. Fixed height and node_per_level parameters, which may not always precisely reflect task interdependence in the actual world, were used to generate the DAG.

6.3 <u>FUTURE WORK</u>

To further increase prediction accuracy, future research may examine the use of machine learning techniques besides Decision Trees, such as Neural Networks and Deep Learning. Furthermore, a more thorough comprehension of the potential and constraints of the suggested algorithms and methodologies would result from testing them in a wider variety of parallel computing contexts and actual applications.

It would also be advantageous to look into how scheduling and resource management are affected by heterogeneous resources, taking into account various hardware and communication traits. Exploring adaptive and dynamic scheduling algorithms that can adapt to shifting system conditions may be a promising area of research as parallel computing technology progress.

References:

[1] Valouxis, Christos & Gkogkos, Christos & Alefragis, Panayiotis & Goulas, George & Voros, Nikolaos & Housos, Efthymios. (2013). DAG Scheduling using Integer Programming in heterogeneous parallel execution environments.

[2] Topcuoglu, Haluk & Hariri, Salim & Wu, Min-You. (1999). Task scheduling algorithms for heterogeneous processors. 3 - 14. 10.1109/HCW.1999.765092.

[3] Chen, Huangke & Zhu, Xiaomin & Liu, Guipeng & Pedrycz, Witold. (2018). Uncertainty-Aware Online Scheduling for Real-Time Workflows in Cloud Service Environment. IEEE Transactions on Services Computing. PP. 1-1. 10.1109/TSC.2018.2866421.

[4] Khatua, Sunirmal & Sur, Preetam & Das, Rajib & Mukherjee, Nandini. (2014). Heuristic-based Optimal Resource Provisioning in Application-centric Cloud.

[5] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman, 1979

[6] Sarkar, Madhulina & Mondal, Triparna & Roy, Sarbani & Mukherjee, Nandini. (2013). Resource requirement prediction using clone detection technique. Future Generation Computer Systems. 29. 936–952. 10.1016/j.future.2012.09.010.

[7] Luo, Jinhong & Li, Xijun & Yuan, Mingxuan & Yao, Jianguo & Zeng, Jia. (2021). Learning to Optimize DAG Scheduling in Heterogeneous E