Chapter 5

Dependency Prediction of Long-Term Resources

High-performance computing (HPC) has revolutionized scientific computation and simulation infrastructure. However, the challenge of uneven load distribution among processors often leads to reduced computational efficiency and increased resource allocation demands for simulations. This necessitates accurate long-term resource utilization prediction to optimize HPC performance. This chapter[1] introduces an innovative ensemble technique incorporating the Feature-based Capability Prediction Algorithm (FBCA) and the Accuracy and Relative Runtime Error Prediction Algorithm (ARRE). Additionally, a three-level architectural framework comprising the simulation environment, resource prediction, and resource queue is proposed and evaluated using Phold and SoS simulations. The framework addresses the computational and simulation requirements effectively. The FBCA algorithm reduces feature redundancy, while the ARRE algorithm enhances the ensemble technique's precision. Comparative analysis against established methods like the Regressive Approach, Linear Regression, and Random Forest reveals that our approach achieves superior accuracy improvements ranging from 8% to 18%.

[1]The work is published by Navin Mani Upadhyay, Ravi Shankar Singh, and Shri Prakash Dwivedi, in IEEE Access, titled "Ensemble Techniques for Long-Term Resource Utilization Prediction in HPC Environments, Volume 11, Pages 141871-141888, in 2023. [SCI]

5.1 Introduction

In a cluster computing environment, the dynamic nature of workloads and resource utilization patterns pose a substantial challenge in anticipating long-term dependencies among resources. The lack of a comprehensive and accurate predictive framework hampers the efficient allocation and management of resources in clusters, leading to suboptimal performance and resource wastage. So, the problem is scheduling the workflow and predicting the sequence of execution of tasks so that the required long-term resources can be easily identified.

Multi-core clusters face challenges in dynamic workload variability, characterized by unpredictable changes in task execution, arrival rates, and prioritization. Inter-core interference complicates resource dependency predictions, as activities on one core impact the performance of others. Long-term resource utilization predictions must consider temporal dynamics, capturing how dependencies evolve over extended periods. Heterogeneous workloads and scalability concerns arise as multi-core clusters expand, impacting the effectiveness of long-term dependency predictions. Adaptability to various cluster configurations, accommodating diverse hardware architectures and workload compositions, is again challenging for accurate predictions.

The challenge involves scheduling workflows and forecasting task execution sequences to effectively manage long-term resource utilization. Our goal is to optimize the load balancing as defined by the below objective function.

Balancing the load is inherently conflicting, thus we adopt the weighted sum approach pioneered by Hoang et al. [103] and Chen et al. [104]. These objectives are constrained to sum up to unity as expressed in the conditional equation Eq. (5.1) below:

$$\sum_{i=1}^{k} w_i = \begin{cases} 1 & \text{if } k \leq 3 \\ 0 & \text{Otherwise} \end{cases} \qquad \text{Eq. (5.1)}$$

Since, our setup consists of 4 CPUs so the maximum distribution of weight (w_i) can not be greater than 3. Further, suppose $\boldsymbol{L}_{ik}$ will be the load on processor $\boldsymbol{p}$, such that either the task is executed over a processor (p_k), and represented by the below conditional equation Eq.

(5.2):

$$L_{ik} = \begin{cases} 1 & \text{if task } t_i \text{ is executed by processor } p_k \\ 0 & \text{Otherwise} \end{cases} \quad \text{Eq. (5.2)}$$

By using $\boldsymbol{w_i}$ and $\boldsymbol{L_{ik}}$, we get the following equations Eq. (5.3) and Eq. (5.4) to find the load and weight:

$$\sum_{j=1}^{n} L_{jk} = \begin{cases} 1 & \{\forall\ j \mid 1 \leq j \leq n\} \\ 0 & \text{Otherwise} \end{cases} \quad \text{Eq. (5.3)}$$

$$\sum_{j=1}^{3} w_j = 1, \{\forall\ j \mid 0 < j \leq 3\} \quad \text{Eq. (5.4)}$$

$\boldsymbol{L_{ik}}$ is the load, and $\boldsymbol{w_i}$ is the weight of the loaded resources on processor $\boldsymbol{p_k}$. The assigned tasks will also be passed through our proposed framework to determine the respective execution time with its consumed voltage and frequency. So our job is to predict $\boldsymbol{w_i}$ concerning $\boldsymbol{p_k}$.

5.2 Proposed Algorithm

In this section, we provide an overview of the ensemble algorithm followed by a comprehensive discussion on the various phases incorporated within our novel approach. Additionally, we outline our evaluation methodology along with the metrics utilized to assess the efficacy of our proposed algorithm. This structured approach ensures a clear understanding of the ensemble algorithm's implementation and its operational phases. Furthermore, it establishes a framework for evaluating its performance based on predefined metrics, fostering a systematic analysis of its effectiveness.

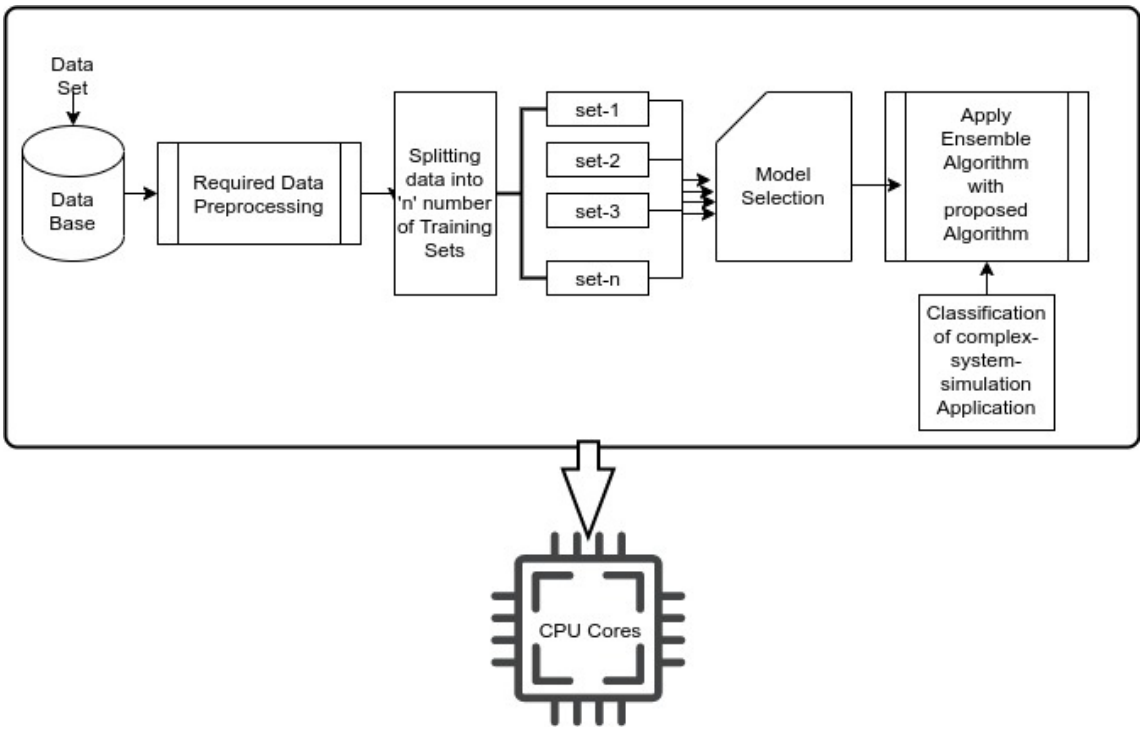

Figure 5.1: Flow diagram of dataset processing and passing through the CPU cores.

5.2.1 Overview of Ensemble Algorithm

In Figure 5.1, we present the flow diagram outlining our proposed framework. The framework employs an ensemble learning approach where multiple independent models are trained simultaneously. Specifically, we introduce the Feature-Based Compatibility Prediction Algorithm (FBCA) for feature selection across various training datasets. This algorithm aims to enhance the quality and efficacy of feature selection.

Additionally, to further optimize performance, we introduce the Accuracy and Relative Runtime Error Prediction Algorithm (ARRE). This algorithm is utilized to predict accuracy and relative runtime errors, thereby aiding in model selection and improvement strategies.

As illustrated in Figure 5.1, the integration of FBCA and ARRE enables long-term resource prediction within the framework. The subsequent sections elaborate on each component depicted in the flow diagram.

- **Dataset Selection and Preparation**: To accurately predict long-term resource needs,

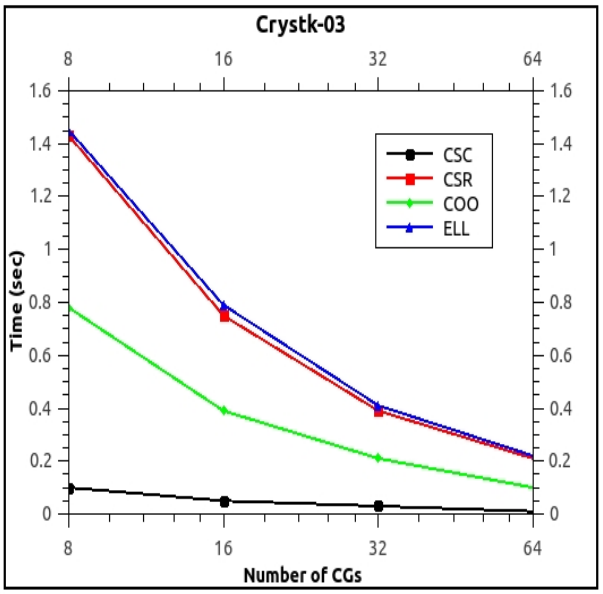

Figure 5.2: Feature distribution based on number of SPECs (CGs) and time domain (loading time)

Algorithm 4 Feature-based Capability Prediction Algorithm (FBCA)

Inputs: $Feature\_Set : F, class\_Set : D, No.\_of\_features : n, Selected\_features : x$
Output: $Selected\_Set\_of\_features : F_s$
Initialization : $t = 1$
  for $i = 1$ *to* n **do**
    Compute weight (F_i, D)
  end
Select feature $F_m, max\_weight(F_i, D)$
$F_s = F_s\ U\ F_m$ $\quad$ $//F_s = F_{s1}\ U\ F_{s2}\ U\ F_{s3}\ U\ \ldots\ U\ F_{sn}$ **for** $x = 1$ *to* m **do**
  if $(x > t)$
  end
    remove F_m from F

end if for *i=1 to Len(F)* **do**
  Compute $F(x_i)$
  end

Select the feature set $F_x | max\_weight(F_x, D)$
$F_s = F_s\ U\ F_x$
$t = t + 1$ **return** F_s

a substantial dataset is essential. Therefore, we generated simulation-based datasets using an application prototype, chosen for its ability to capture relevant features [12]. These datasets were tailored by varying parameters such as maximum and minimum main memory, cache memory sizes, and processing frequencies. Subsequently, a standard FP benchmark [105] was employed to validate our generated datasets.

- **Data Preprocessing**: Data preprocessing is crucial for enhancing dataset quality. Our approach involved two phases: data cleaning and transformation. In the data cleaning phase, outliers were identified and handled to ensure data integrity. Outlier instances were visualized using Figure 5.2, where outliers were defined as instances lying outside specified bounds relative to the baseline. The transformation phase addressed outlier replacement; instances showing significant deviation from neighboring data points were replaced with their mean values to maintain dataset coherence and prepare it for subsequent algorithmic processing. Following these adjustments, the dataset underwent normalization using the Z-score technique.

  The Z-score normalization method was chosen for its ability to transform data into a standard normal distribution, facilitating consistent comparison and analysis across different experiments and benchmarks like the Standard Performance Evaluation Corporation (SPEC) Dataset. This technique centers the data around a mean of zero and scales it according to the standard deviation, thereby reducing skewness and kurtosis in the distribution. Skewness measures asymmetry, while kurtosis quantifies peak or flatness characteristics. By standardizing the data, outliers are effectively neutralized, resulting in a more symmetrical distribution with reduced skewness and kurtosis values.

  In comparison, alternative normalization techniques like min-max scaling and decimal scaling adjust data within specific ranges or by decimal shifts, respectively. However, these methods may be influenced by outliers and could potentially lose information during scaling. Conversely, the Z-score technique's robustness against outliers ensures that the original data's information is preserved, making it particularly suitable for the diverse characteristics and performance metrics encountered in SPEC datasets.

  Thus, for our selected SPEC Dataset, the Z-score normalization emerges as the preferred method due to its robustness, ability to handle data variations effectively, and established reliability in statistical and machine learning practices.

- **Feature selection**: Feature selection is one of the best phases in machine learning techniques because using this method can improve accuracy and time complexity prediction. Both constraints (time and accuracy) are vital in problem-solving and algorithms. Therefore, feature selection also becomes a crucial task to select the best set of features. The main advantage of feature selection is that it reduces the redundancies of chosen components and improves the performance of simulation application algorithms. Webar et al. [106] have found that most of the feature selection algorithms are based on two different features: the first one is maximizing the relevance instance value, and the second one is minimizing the redundancies. In this paper, we have proposed an FBCA algorithm shown in Algorithm 4 to create an impact over existing feature selection.

Suppose M_{info} is the mutual information shared between two individuals $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$, then the measurement of dependency between $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$ will be written as $\boldsymbol{M_{info}(I_1, I_2)}$. The entropy $\boldsymbol{H(I_1)}$ is defined as the dependency disorder or randomness or uncertainty of the mutual information. The uncertainty measurement will be done on 36 instances, represented as $\boldsymbol{I[36]}$. The relation between $\boldsymbol{M_{info}(I_1, I_2)}$ and $\boldsymbol{H(I_1)}$ is represented by the below-mentioned Eq. (5.5):

$$
\begin{aligned}
M_{info} &= \sum_{I_1 \in I} \sum_{I_2 \in I} D(I_1, I_2) log(\frac{D(I_1, I_2)}{D(I_1) D(I_2)}) \\
H(I_1) &= \sum_{I_1 \in I} D(I_1) log(I_1) \\
H(I_2) &= \sum_{I_2 \in I} D(I_2) log(I_2)
\end{aligned}
\qquad \text{Eq. (5.5)}
$$

$\boldsymbol{D(I_1)}$ and $\boldsymbol{D(I_2)}$ are defined as the normal distribution of $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$. $\boldsymbol{D(I_1, I_2)}$ is defined as joint distribution of $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$.

We have selected normal and joint distribution for the above Eq. (5.5) because the mutual information will be the intermediate part of the information from both individuals. So, the normalized mutual information will be written as Eq. (5.6).

$$Norm(M_{info}(I_1, I_2)) = 2 \times \frac{M_{info}(I_1, I_2)}{H(I_1) + H(I_2)} \quad \text{Eq. (5.6)}$$

Since the information depends upon both individuals $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$. So, we have multiplied our normalized mutual information by 2. Also, it is notable that the distribution is joint with respect to $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$. So, we have used their entropy in an additive manner.

To calculate the feature capabilities of the Contribution of individual $\boldsymbol{Is}$ we have assumed that the Contribution of $\boldsymbol{I_1}$ candidate feature is $\boldsymbol{I}_{contri} | (\boldsymbol{I_1}, \boldsymbol{I_2}) \in \boldsymbol{C}$, where $\boldsymbol{C}$ is specific feature class for $\boldsymbol{I}_{contri}$. So, the feature capabilities $(\boldsymbol{f}_{cap})$ is defined as Eq. (5.7):

$$f_{cap}(I_{contri}, C) = Norm(M_{info}, C) + Const \times (1 - (\frac{H(I_{contri})}{log\ k})) \quad \text{Eq. (5.7)}$$

where $\boldsymbol{Const}$ is any constant $0 \leq Const \leq k$ defined as the weight between the distribution of mutual information $\boldsymbol{D}(\boldsymbol{I}_{contri}, \boldsymbol{C})$, and $\boldsymbol{k}$ is the total number of contributed candidate features $(\boldsymbol{I}_{contri})$ of $(\boldsymbol{I_1}, \boldsymbol{I_2})$ categories.

Based on the above assumptions, the redundancies between selected feature $(\boldsymbol{I}_{select})$ and contributed feature $(\boldsymbol{I}_{conti})$ can be defined as Eq. (5.8):

$$M_{info}(I_{contri}, I_{select}) = \begin{cases} \text{Overall} & \forall (\ I_{contri} \leq \text{redundancy} \leq I_{select}) \\ \text{Ignored} & (if\ (I_{contri} - I_{select}) = 0) \end{cases} \quad \text{Eq. (5.8)}$$

Therefore, we have used standard Deviation (SD) as $\boldsymbol{Const.}$, defined in $\boldsymbol{f}_{cap}(\boldsymbol{I}_{contri}, \boldsymbol{I}_{select})$ formula. The standard Deviation (SD) is a measure of the amount of variation or dispersion of a set of values. A low standard deviation (SD) indicates the expected value since the weight tends to be close to the mean, whereas a high SD indicates that the values are spread out over a broader range. A detailed definition of Standard Deviation (SD) is shown below in Eq. (5.9):

Let $\boldsymbol{\mu}$ be the expected value (the average) of distribution $\boldsymbol{D(I_1, I_2)}$ with density of selected features $\boldsymbol{I_{select}}$. Then,

$$\mu \equiv \frac{1}{\mid I_{select} \mid} \sum_{I_{select}(I_1,I_2) \in I_{contri}} Norm(M_{info}(I_{contri}, I_{select})) \quad \text{Eq. (5.9)}$$

The standard deviation (SD) denoted as $\boldsymbol{\sigma}$ of $\boldsymbol{I_{select}}$ is defined as, $\sigma \equiv \sqrt{\frac{1}{|I_{select}|} \sum_{I_{select}(I_1,I_2) \in I_{contri}} ((Norm(M_{info}(I_{contri}, I_{select}))) - \mu)^2}$ This is because the standard deviation is the square Root of the variance of $\boldsymbol{I_{contri}}$ & $\boldsymbol{I_{select}}$ concerning $\boldsymbol{I_1}$ & $\boldsymbol{I_2}$. Based on the above formula, if the standard deviation $\boldsymbol{\sigma}$ is low, then the redundancy between $\boldsymbol{I_{contri}}$ and $\boldsymbol{I_{select}}$ is closer to mean μ. To evaluate its contribution we have expressed $\boldsymbol{E(I_{contri})}$ in the following Eq. (5.10):

$$E(I_{contri}) = f_{cap}(I_{contri,C}) - const. \times (1-\sigma) \sum_{I_{select}(I_1,I_2) \in I_{contri}} Norm(M_{info}(I_{contri}, I_{select})) \quad \text{Eq. (5.10)}$$

The $(1-\sigma) \sum_{I_{select}(I_1,I_2) \in I_{contri}} Norm(M_{info}(I_{contri}, I_{select}))$ is defined as the representation of both overall and individual feature selection between $\boldsymbol{I_{contri}}$ and $\boldsymbol{I_{select}}$.

The proposed Algorithm 4 is divided into five major steps. The steps are described as follows:

1. Compute the feature capability of all features available in feature-set $(\boldsymbol{i.e.\ Feature_{Set} : F})$.
2. Select the feature with the highest feature capability and make the time-span $(\boldsymbol{t = 1})$.
3. Based on feature selection criteria, rank them into their corresponding computed scores and evaluate them.

4. Merge the feature having the highest capability score to the feature set and increase the time span $(i.e.\ t = t+1)$.

5. Repeat the steps from 3 to 5 until the selected feature has the highest score $(i.e.\ x > t)$.

5.2.2 Performance Evaluation of Proposed Algorithms

The performance of any model depends on the following characteristics: Accuracy, RMSE, F1-Score and Relative Error. So, to evaluate the performance measurement of our proposed algorithm, we have also used the above-said metrics, which are defined as follows:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n}[I_{predict}[i] - I_{select}[i]] \quad \text{Eq. (5.11)}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[I_{select} - I_{predict}]^2} \quad \text{Eq. (5.12)}$$

$$Pi = \frac{TP_i}{TP_i + FP_i} \quad \text{Eq. (5.13)}$$

$$Ri = \frac{TP_i}{TP_i + FN_i} \quad \text{Eq. (5.14)}$$

$$F1 - Score = \frac{1}{C}\sum_{i=1}^{n}\frac{2PiRi}{Pi + Ri} \quad \text{Eq. (5.15)}$$

$$RelativeError = \lambda * Accuracy + (1-\lambda) * (1 - \frac{\sum_{i=1}^{n}[I_{select} - I_{predict}]^2}{\sum_{i=1}^{n}[I_{select} - \bar{I}_{predict}]^2}) \quad \text{Eq. (5.16)}$$

Where, $\boldsymbol{TP_i, FP_i, FN_i}$ are true positive, false positive, and false negative scores for the $\boldsymbol{I_{predict}}$ and $\boldsymbol{I_{select}}$ class. Hence, the selection of several CPU cores is referred to $\boldsymbol{I_{select}}$ and $\boldsymbol{I_{predict}}$. $\boldsymbol{n}$ is the total number of samples, and $\boldsymbol{C}$ is the total number of classes. $\boldsymbol{\lambda}$ is the

weight selected according to the user's performance requirement and accuracy. The Relative Error is only the indicator for comprehensive computation of accuracy and deviation of $\boldsymbol{TP_i}$ and $\boldsymbol{I}_{predict}$.

- **Training and Model Selection**: Supervised learning was chosen for the classification-based problem addressed in Chapter 3, where the mapping from input to output levels is continuous for predicting the required number of CPU cores. To establish a baseline model for CPU requirement prediction and application simulation, we experimented with several algorithms including K-nearest neighbor (KNN), Decision Tree (DT), and Support Vector Machine (SVM) using different parameters. Among these, SVM, KNN, and DT proved to be the most accurate, as illustrated in Table 5.1.

  SVM was particularly favored due to its capability in handling non-linear relationships between features and target variables, which is crucial in performance evaluation scenarios where such relationships are common, such as in the SPEC dataset. KNN was found effective in managing noisy data, a characteristic often present in SPEC due to variations in hardware configurations and operating systems, influencing parameters like Min-Memory, Max-Memory, Min-Clock, Min-Cache, and Min-Frequency.

  Similarly, Decision Trees are well-suited for datasets with a mix of continuous and categorical variables, a typical scenario in performance evaluation where benchmarks exhibit diverse features and may contain missing values due to measurement errors in datasets.

  A sequential and parallel execution of selected Dataset (SPEC) has shown in Figure 5.3, Figure 5.4, Figure 5.5, and Figure-5.6.

  We partitioned the dataset into two distinct sets: a training dataset and a test dataset, optimizing their utility in simulation applications. Employing a methodology akin to traditional machine learning systems, our simulation mode underwent training and testing processes.

  In our approach to model selection, we adopted ensemble techniques. These methods are favored due to the inherent limitations of individual machine learning models in addressing the computational demands of complex systems. By combining multiple base models, ensemble methods leverage diverse perspectives to achieve enhanced

Table 5.1: Comparison of different normalization algorithms, based on accuracy, precision, recall and F1-score for SPEC dataset.

Classifier	Accuracy	Precision	Recall	F1-Score
Naive Bayes	0.82	0.84	0.86	0.85
Gradient Boosting	0.85	0.84	0.83	0.82
SVM	0.92	0.92	0.92	0.91
KNN	0.92	0.91	0.94	0.92
DT	0.88	0.87	0.89	0.88

results. Notably, they demonstrate superior generalization capabilities compared to singular machine learning models.

For the basic understanding, ensemble models are defined as an integration of different base models. In such base models, the diversity will increase whenever the accuracy increases. Based on its accuracy and diversity, its performance will also increase. So, for our Simulation, we have to increase the disturbance of the sample data set to train our base model. This will increase the existing difference between the base and selected models. It will also improve its generalization performance.

Notably, none of any base models has that much of a positive impact on ensemble techniques based on performance. So, we have proposed Algorithm 5, which will be able to choose the correct Set of base models as per the user's requirement. Since it can choose the required model per the user's needs, it can also reduce the execution time by discarding the timestamp selection of weak models. The proposed Algorithm 5 is shown below:

Algorithm Explanation: The Proposed Accuracy and Relative Runtime Error (ARRE) Algorithm 5 is processed into four steps. The corresponding steps are described as follows:

1. After taking a set of base models as input, the algorithm computes each base model's Accuracy and Relative Error.
2. The obtained accuracy and relative error values will be arranged into ascending order (the algorithm will perform a sorting operation).
3. Select all the based models having the same or higher accuracy and relative error.

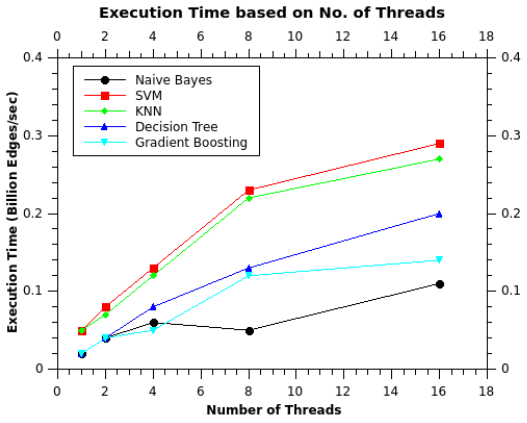

Figure 5.3: Sequential execution of uniformly distributed datasets over selected algorithms (SVM, KNN, DT, GBT, and Naive Bayes)

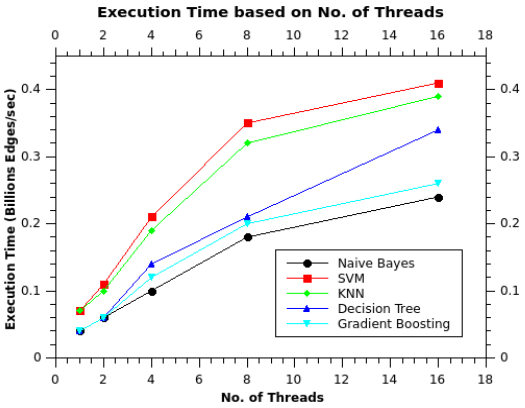

Figure 5.4: Sequential execution of randomly distributed datasets over selected algorithms (SVM, KNN, DT, GBT, and Naive Bayes)

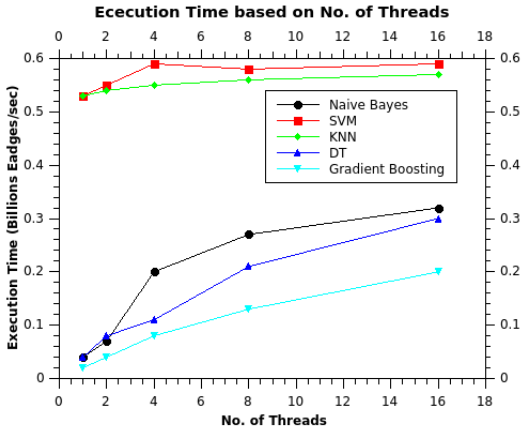

Figure 5.5: Parallel execution of uniformly distributed datasets over selected algorithms (SVM, KNN, DT, GBT, and Naive Bayes)

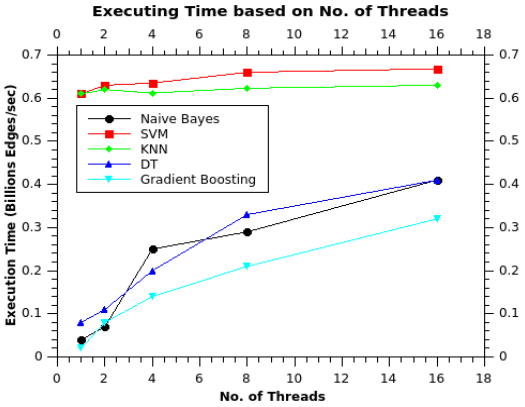

Figure 5.6: Parallel execution of randomly distributed datasets over selected algorithms (SVM, KNN, DT, GBT, and Naive Bayes)

4. Apply the ensemble technique to select the best combined base models.

5. Finally, select the best base model set based on their maximum accuracy and minimum relative error.

Once the predicted base models have been obtained, the ensemble technique is applied to them. The following formula Eq. (5.17) is used to apply the ensemble technique:

Algorithm 5 Accuracy and Relative Runtime Error (ARRE)

Inputs: *Base Model Sets* $BM_1, BM_2, BM_3, ..., BM_n, Dataset(D_s)$
Output: *Predicated Base model sets* $PBM_1, PBM_2, ..., PBM_k$
Initialization : $t = 1$
for *i=1 to n* **do**
Compute weight(F_i, D)
end

Select feature $F_m | max\_weight(F_i, D)$
$F_s = F_s \ U \ F_m$
// $F_s = F_{s1} \ U \ F_{s2} \ U \ F_{s3} \ U \ ... \ U \ F_{sn}$ **for** *x=1 to m* **do**
if $(x > t)$
remove F_m from F
end if
end
for *i=1 to Len(F)* **do**
Compute $F(x_i)$
end

Select the feature set $F_x | max\_weight(F_x, D)$
$F_s = F_s \ U \ F_x$
$t = t + 1$
return F_s

$$\begin{aligned} &Ensemble\_I_{predict}\{PBM_1 \ U \ PBM_2 \ U \ ... \ U \ PBM_k\} \\ &= I_{select}(\Sigma_{j=1}^{t} PBM_j\{PBM_1 \ U \ PBM_2 \ U \ ... \ U \ PBM_t\}) \end{aligned} \quad \text{Eq. (5.17)}$$

Where $\boldsymbol{I}_{predict}\{PBM_1 \ U \ PBM_2 \ U \ ... \ U \ PBM_k\}$ is the information predication among predictive base models sets, is the information class described earlier.

5.3 Experimental Setup and Simulation

We used Phold [68] and SOS (Social Opinion Systems) [68] in our experiments. Both of these applications are CPU-intensive and supported by resource prediction algorithms. To find better performance, we have done different studies on the configuration of these applications. The detailed configuration parameters are shown in the Table 5.2.

Table 5.2: Benchmark with the number of instances used for simulation

Used Benchmark	No. of Instances used	Remarks
Phold	(50,400)	No. of objects participated in Simulation
	(10,100)	Objects/Events generated from Simulation
	(1000)	Total running time of Simulation/Experiments
	Preserved	Time Distribution based on resources
	(0.1,1)	Look Ahead Carry Buffer
Social Opinion System	[100, 1000]	No. of Profiles
	(5,20)	No. of Cities Selected
	(10)	No. of Infrastructure
	(5%,20%)	Probability of IP Networks
	(1000)	Total running time of Simulation/Experiments
	Preserved	Time Distribution based on resources
	(0.1,1)	Look Ahead Carry Buffer

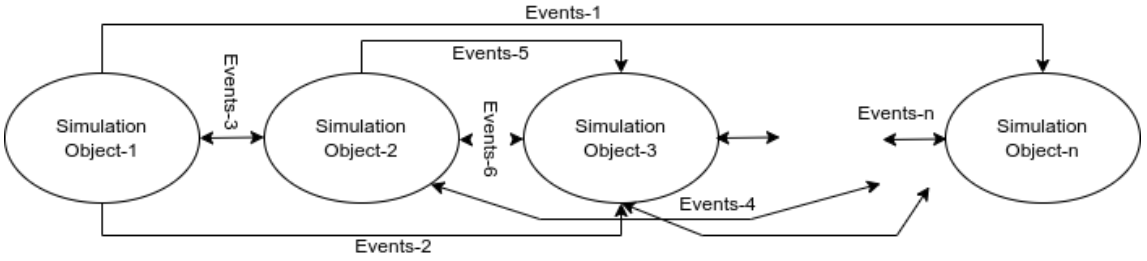

Figure 5.7: Phold structure with n simulation objects

5.3.1 Phold

In the Parallel and Discrete Simulation Environment (PEDS), the Phold is the most powerful and effective performance evaluation benchmark. Many Linear programming simulation object models (LPs) are connected with Phold. All events pass through the event simulator object state. These transferred events are classified into three categories (local, autonomous, and simulation object/event). A detailed structure of Phold is shown in the Figure 5.7. In Figure 5.7, the Simulation of the Phold Benchmark is shown. A single simulation object contains the following Set of components and is processed in different sets of steps.

- Initialization: At the start of the Simulation, n simulation objects are created. Each object has its initial state, attributes, and defined behaviour.

- Event Generation: Each simulation object can generate events at a specific time based on their pre-defined conditions. These events has shown as event-1, event-2, ... , event-n in the Figure 5.7.
- Event Queue: All generated events are placed into an event queue in FCFS order. Simulation objects further process all the events.
- Event Execution: Event execution and an execution clock keep track of the current executed event and Simulation.

5.3.2 Social Opinion Systems (SOS)

SOS exhibits a unique and intricate nature within the realm of Parallel and Discrete Simulation Environments (PDSE), particularly evident in its ability to capture complex interdependencies, as detailed in Table 5.2. The table provides insights into various aspects of the simulation:

- **Number of Simulation Objects (50,400):** This figure underscores the substantial scale of the simulation, involving 50,400 objects, each representing a distinct entity within the simulated environment. This scale highlights the simulation's robustness and complexity.
- **Objects/Events Generated (10,100):** The simulation generated 10,100 objects or events, indicating its dynamic nature and the myriad interactions and events simulated, enriching the complexity of the environment.
- **Total Simulation Running Time (Phold) (1000):** The simulation ran for 1000 units, offering a temporal perspective on the duration over which events unfolded and processes operated within the simulation environment.
- **Preserved Time Distribution Based on Resources (0.1, 1):** This aspect highlights the simulation's consideration of varying resource availability, crucial for scenarios where resource dynamics impact simulation outcomes.
- **Look Ahead Carry Buffer (100, 1000):** The inclusion of a look-ahead carry buffer ranging from 100 to 1000 enhances decision-making within the simulation, allowing for anticipation of future events or conditions.

- **Number of Profiles (5, 20):** The simulation incorporated between 5 to 20 profiles, representing diverse configurations or characteristics within simulated entities, thereby enriching simulation variability.
- **Number of Cities Selected (10):** Selection of ten cities within the simulation context suggests spatial relevance, potentially influencing urban-centric scenarios such as resource distribution or policy-making.
- **Infrastructure Social Opinion System Integration:** Integration of an infrastructure social opinion system highlights the simulation's depth by simulating how infrastructure influences societal opinions and dynamics.
- **Probability of IP Networks (5%, 20%):** Variation in IP network probability (5% to 20%) demonstrates the simulation's flexibility in modeling network connectivity dynamics and their impact on simulated environments.
- **Total running time of Simulation/Experiments Preserved Time Distribution based on resources (1000):** Similar to the earlier mention, this remark reiterates the total running time of the Simulation or experiments, emphasizing the importance of the 1000-unit duration. The preserved time distribution based on resources further underscores the consideration of resource dynamics throughout the Simulation.
- **Look Ahead Carry Buffer (0.1,1):** This repetition emphasizes the utilization of a look-ahead carry buffer with a range of 0.1 to 1, highlighting its significance in influencing the Simulation's decision-making processes.

These elements collectively portray the complexity and comprehensive scope of the simulation, encompassing entity scale, temporal aspects, resource dynamics, and specific features such as profiles, cities, infrastructure, and social opinion systems.

Moreover, the simulation's complexity deepens in scenarios where infrastructure is absent or compromised, affecting societal dynamics through disruptions in critical services like electricity and essential resources. Such infrastructural dependencies significantly influence policy-making processes based on city-specific contexts and infrastructure conditions. Figure 5.8 illustrates a foundational structure of the SOS.

In Figure 5.8, the infrastructure encompasses both physical and digital frameworks crucial to societal operations, including communication networks and storage facilities. Media

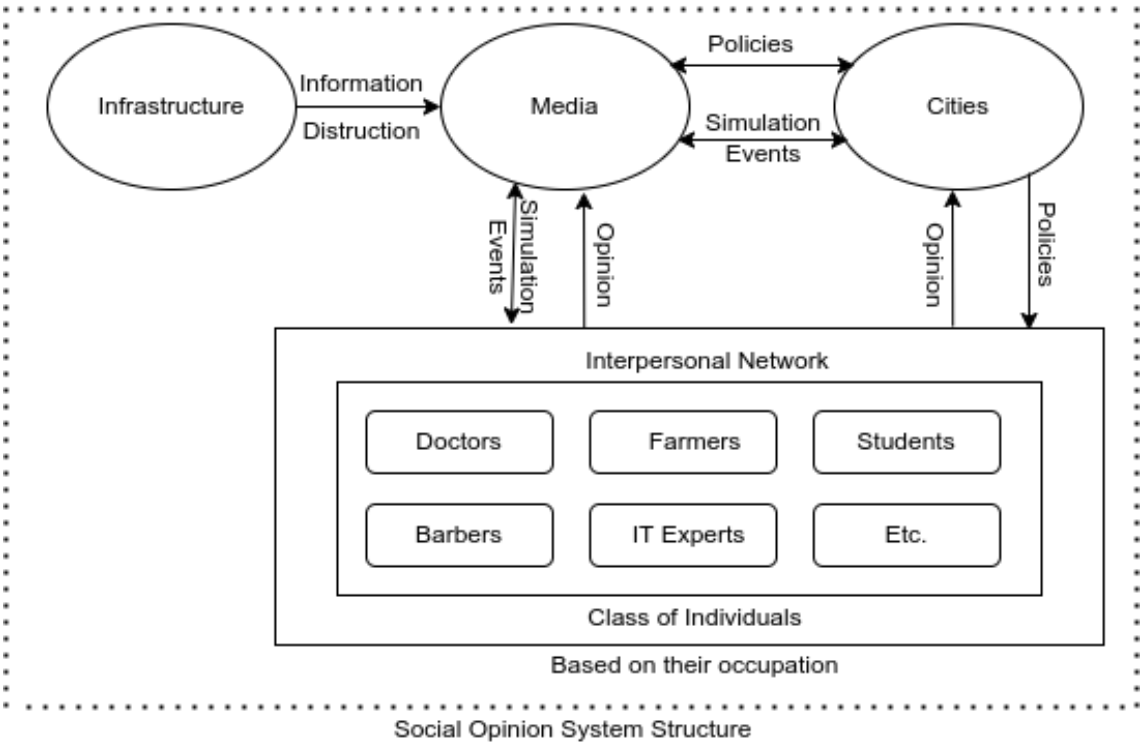

Figure 5.8: Social opinion system structure

channels play a pivotal role in disseminating information, influencing public opinion and feedback, thereby impacting policy decisions made by urban authorities in response to infrastructural changes and societal needs.

5.3.3 Data Set Description

All the datasets are standard and taken from the official website of Phold-IBM[2]. We have adjusted the parameters from the datasets to generate the simulation application. The adjustment of datasets are already described in Algorithm 4 and Algorithm 5 are used for selected benchmark simulation shown in Table 5.2. The adjustments are followed by a 10-step procedure, which includes Defining the event generation, configuring events and event queues for First Come, First Serve (FCFS) scheduling, event processing, setting of simulation time and termination criteria, collecting the results and analyzing the data recursively followed by parameter tuning.

[2]ibm.com/docs/en/zos/2.3.0?topic=initialization-page-data-set-sizes

Then, we deployed them to our virtual HPC environment. The Phold and Social Opinion System (SOS) contains 10050 and 5010 samples, respectively. Table 5.2 contains a detailed description of these used dataset features. The selected features are based on their minimum execution time for all the selected CPU cores.

5.4 Results and Discussions

5.4.1 Experimental Setup

We employed a system consisting of eight nodes, each equipped with a 64-bit Operating System. These nodes were chosen from a range of multi-core components previously detailed and documented in the literature [105]. The specific configuration of these nodes is presented in Table 5.3, which lists the detailed specifications used in this experimental setup. This selection ensured consistency and comparability across our evaluations, maintaining the integrity of our findings with respect to the hardware environment employed.

Table 5.3: A detailed configuration of selected multi-cores for virtual cluster

Nehalem Architecture	**Multi-core**	**GPU**	**SC10-EP**	**SC10-EX**
Core Archi.	Intel Nehalem	Nvidia Fermi	Intel Core	Nvidia Core
Model No.	Xeon X5550	Nvidia C2050	Xeon E5345	GeForce GTX275
Core Freq.	2.67 GHz	1.15 GHz	2.33 GHZ	1.40 GHz
No. Socket	2	1	2	1
No. Cores	4	14*2(a)	4	30
HW-threads	2	≈32	1	≈32
SIMD/SIMT	- (not used)	32	-	32
Total Cache	16 MB	2MB	8 MB	-
Main Memory	24 GB	3GB	32 GB	896MB
Memory Band	100 GB/s	128 GB/s	10.4 GB/s	127 GB/s

For this experiment, we utilized a Docker environment configured with a single core and a minimum of 2GB of memory. Details of the node configuration are provided in Table 5.3. Long-term resource predictions are illustrated in Figure 5.9. The experimental workflow involves preprocessing datasets, followed by partitioning into training and testing sets based on specific requirements. The selection of the optimal dataset led to the application of an ensemble algorithm within a simulation framework to achieve the most pertinent

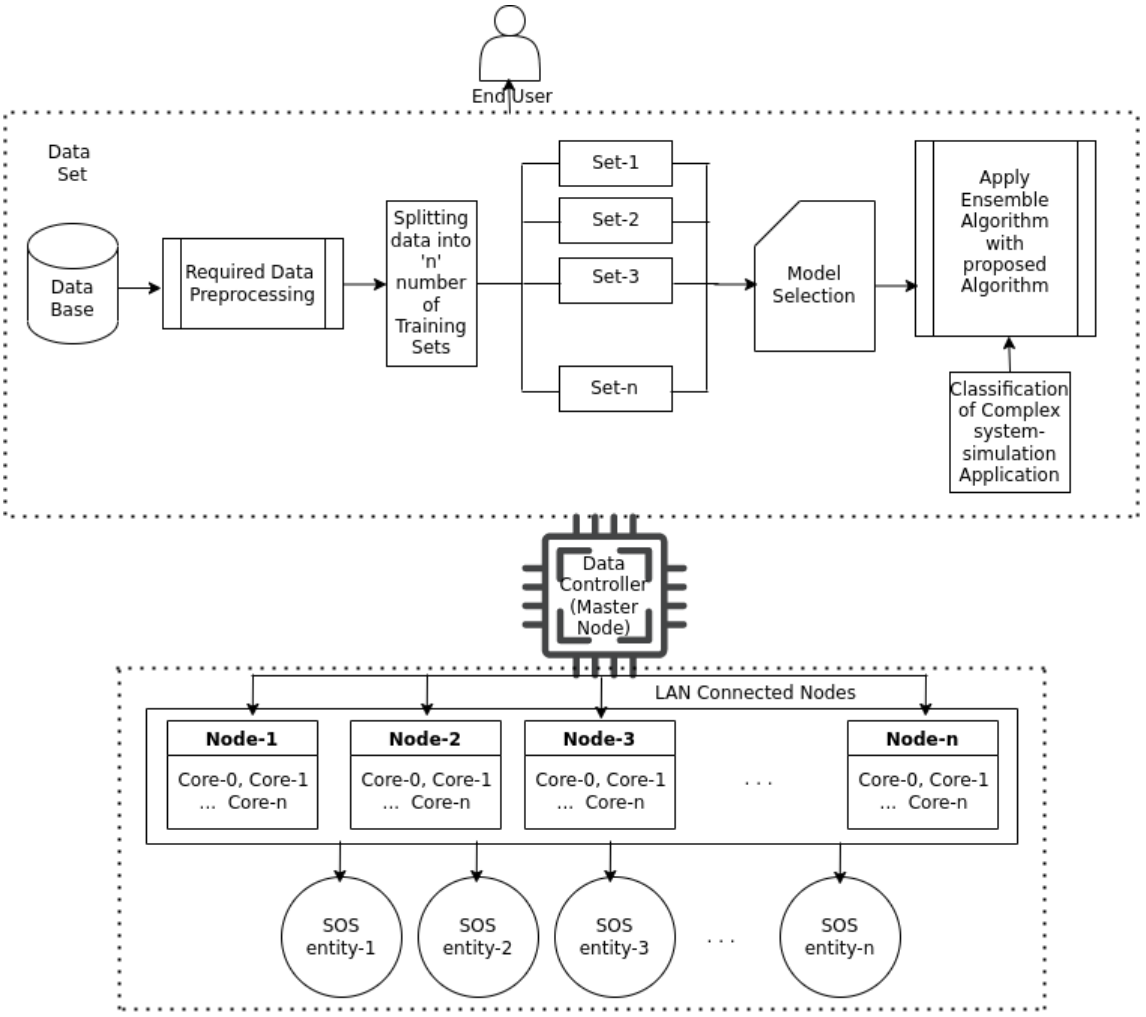

Figure 5.9: Long-term running resource prediction structure.

results.

In this architecture, the Linear Programming models (LPs) play a crucial role in distributing computational load across all cores. Each LP partition maintains identical CPU configurations encompassing memory, frequency, and core allocations. The Objective functions detailed in Algorithm 4 and Algorithm 5 guide the partitioning process, along with Decision Variables and constraints on resource allocation and utilization. Additionally, the models incorporate techniques for model selection and forecasting, ensuring efficient resource management. Central to this setup is the master node, which serves as the control hub for aggregating resource information. Subsequently, this information enables the controller node to apply predictive algorithms aimed at optimizing computing resources for enhanced system performance.

5.4.2 Experimental Results and Analysis

We have performed our analysis on three different aspects.

1. Based on parameter selection
2. Based on result validation
3. Based on Results comparison

- **Parametric Experiments:** The algorithm's performance is influenced by several parameters, as detailed in the problem formulation section, including feature selection dimensions and the ensemble of base models. To evaluate the algorithm's performance under different parameter settings, 10-fold cross-validation was employed. The results are presented in Figure 5.10 and Figure 5.11.

  In our study, we opted for k=10 in the 10-fold cross-validation, a widely recommended practice in machine learning. The rationale for choosing k=10 is as follows:

  - **Common Practice:** Using k=10 in k-fold cross-validation is well-established in the machine learning community for robust performance evaluation.
  - **Bias-Variance Trade-off:** Smaller values of k (e.g., k=5) tend to introduce higher variance in error rate estimates due to increased bias, whereas larger values (e.g., k=9 or 10) strike a balance between bias and variance. We chose k=10 to achieve stable estimates of error rates.
  - **Empirical Evidence:** Empirical studies consistently support using k=10 as it provides reliable estimates of test error rates without significant bias or variance issues.
  - **Computational Efficiency:** Apart from its statistical merits, k=10 is computationally efficient, particularly suitable for datasets like SPEC, allowing us to perform cross-validation effectively.

  In Figure 5.10, the evaluation metrics show that as the number of selected feature dimensions increases, accuracy, F-1 score, and RMSE initially improve and then stabilize within a specific range. However, beyond this range, their performance

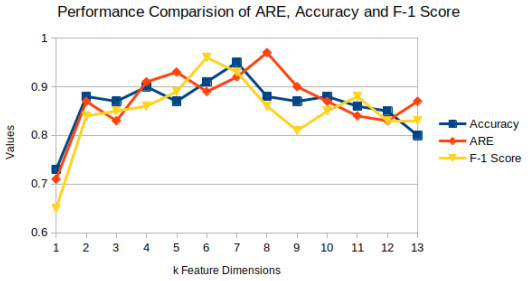

Figure 5.10: Performance comparison of intelligent ensemble algorithms with k-feature dimensions (ARE, Accuracy, F-1 Score).

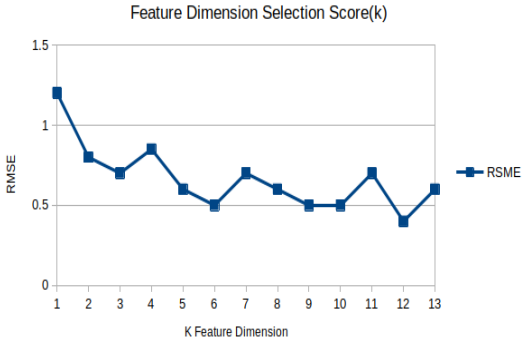

Figure 5.11: Feature dimension selection core based on RSME.

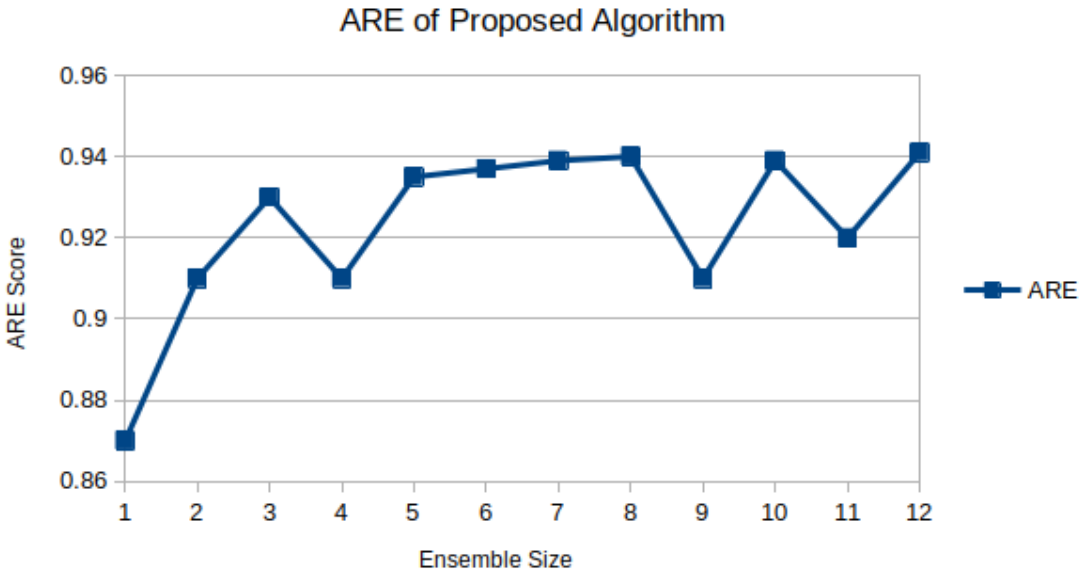

Figure 5.12: Absolute root mean error of proposed algorithm

tends to decrease. Notably, lower feature dimensionality poses challenges due to unstable prediction accuracy over time. Redundancy in features can lead to a decline in performance beyond a certain point, as evident in Figure 5.10.

Figure 5.11 illustrates that RMSE decreases with increasing feature dimensions, stabilizes within a range, and then begins to rise again. Optimal performance in terms of RMSE is observed within the range of 3 to 13 feature dimensions.

Figure 5.12 depicts the effect of ensemble size on the proposed algorithm's

performance using the ARRE method for base model selection. The figure indicates that as the ensemble size increases, so does the ARE, albeit with diminishing returns after reaching a certain threshold. This saturation effect underscores the importance of selecting appropriate base models for our intelligent ensemble algorithm.

By leveraging these insights into parameter tuning and model selection, our approach enhances the robustness and effectiveness of ensemble algorithms in predictive modeling tasks.

- **Experimental Result Verification:** We performed experiments in the simulated virtual environment and validated our proposed algorithm. We have also compared our algorithm with existing KNN and Traditional Bagging algorithms. One notable thing is that the KNN uses a grid-based searching technique to find its best optimal parameters. In contrast, the traditional Bagging algorithm uses a bagging approach to club its base models. Table 5.4 contains a detailed comparison report on both KNN and traditional bagging techniques.

Table 5.4: Performance comparison of the proposed algorithm with other selected algorithms from different benchmark

Bench Mark	Model	Accuracy (%)	RSME	ARE(%)	F1-Score	Time
Phold	SVM	87.05	0.94	72.12	89.06	10.11
	DT	83.44	1.79	90.42	75.68	10.03
	KNN	89.63	1.23	90.37	92.83	08.93
	Bagging Ensemble	91.82	1.38	69.72	83.23	16.77
	proposed algorithm	**94.11**	**0.06**	93.54	**94.68**	11.31
SOS	SVM	84.71	1.02	86.31	90.32	10.34
	DT	87.69	0.93	89.64	82.86	12.08
	KNN	83.78	0.93	90.43	93.73	10.42
	Bagging Ensemble	89.28	1.02	79.32	86.84	59.23
	proposed algorithm	84.08	**0.78**	96.11	90.52	10.43

From the above Table 5.4, it is clear that our proposed model has a higher accuracy and a lower Root mean square error (RMSE). As we know, the RSME results from the deviation between predicted and actual values, so based on RSME, it is also clear that our proposed model is perfect in terms of variation. From the F1-score, the difference between performed activity between bagging and the proposed algorithm, the proposed algorithm shows a higher value $\approx 10\%$. Based on the result in terms of time and F1-score, we can say that the Bagging algorithm performs lower than

our proposed algorithm. The proposed algorithm is independent of the accuracy or true and false prediction. The compared method shows less training time, but our proposed algorithm's accuracy is 4% faster.

The proposed Algorithm performs better because it chooses its base models wisely. The selection is based on Phold and SOS simulation services, which reduce the influence of poorly executed models.

- **Performance Comparison:** The Table 5.5 shows a specific comparison between the proposed method and other primary resource prediction methods like (the regressive ensemble approach for predicting resource (REAP) [107], Bayesian Approach (BN) [107], cost-aware auto-scaling approach (LR) [107], the advanced model for efficient workload prediction (RF) [108], and fuzzy neural network (FNN) [108].

Table 5.5: Performance comparison between major resource prediction methods and proposed method

Bench Mark	Model	Accuracy (%)	RSME	ARE(%)	F1-Score	Time
Phold	REAP	87.27	0.84	82.12	89.06	10.11
	BN	83.51	1.19	91.42	85.68	10.03
	LR	89.63	1.32	90.37	82.83	08.93
	RF	91.82	0.38	79.72	87.13	16.77
	FNN	91.82	0.38	90.72	81.29	16.77
	proposed algorithm	94.11	0.53	93.54	94.68	11.31
SOS	REAP	90.71	1.12	86.31	90.32	10.34
	BN	91.69	0.23	89.64	82.86	12.08
	LR	83.78	0.93	90.43	93.73	10.42
	RF	87.28	0.72	79.32	86.84	59.23
	FNN	89.28	0.46	79.32	86.84	59.23
	proposed algorithm	84.08	0.78	96.11	90.52	10.43

We have used Root mean square error and absolute root error at $(\lambda = 0.5)\%$ for all our experiments. We have also used the F1-score to evaluate the comparative prediction with the indices parameter. During our experiments, we observed that the comparative experiments' parameters need to be discretized. A clear difference can be seen in Table 5.5.

This Table also shows that our proposed algorithm is significantly better than other traditional models. Other methods like REAP can achieve good accuracy, but it is not ideal with other performance evaluation indices. If we look at BN and LR, they take

less time but have a lower accuracy. The FNN requires more training time than the RMSE and the proposed method. So, overall, we can say that our proposed method can achieve high accuracy, low training time and better user requirements to improve the QoS.

5.5 Summary

In this study, we introduce two novel algorithms for long-range resource prediction problems within an intelligent ensemble learning framework: the Feature-based Capability Prediction Algorithm (FBCA) (Algorithm 4) and the Accuracy and Relative Runtime Error (ARRE) (Algorithm 5). The FBCA algorithm addresses noise reduction in datasets, while the ARRE algorithm enhances model adaptability beyond traditional approaches.

The experimental setup involved eight nodes comprising various CPU components, including Nehalem CPU cores, Nvidia GPU cores, Intel Core (SC10-EP), and Nvidia core (SC10-EX) configurations. Our algorithms demonstrated significant performance gains in resource prediction and load distribution optimization across extensive experiments. Parametric studies, incorporating a 10-fold cross-validation at $k = 10$, revealed optimal performance under varying conditions. Stability in accuracy metrics, F1-score, and absolute root mean square error (RMSE) across different feature dimensions further underscored the algorithms' robustness.

Comparative analyses against traditional models (SVM, DT, KNN, Bagging Ensemble) and primary resource prediction methods (REAP, BN, LR, RF, FNN) consistently highlighted the superiority of our proposed algorithms. For instance, compared to SVM and DT, the algorithms achieved an accuracy of 94.11%, an RMSE of 0.06, and an F1-score of 94.68%. In contrast to RF, our algorithms attained an accuracy of 91.82%, an RMSE of 0.38, and an F1-score of 87.13%.

Furthermore, we introduced a novel SoS (System of Systems) and Phold-based framework for long-term resource prediction in scientific computing, demonstrating improved accuracy ranging from 8% to 18%. Validation efforts using inter-process communication and L1-cache memory-based miss ratio techniques ensured the suitability of simulation entities for modern CPUs and SPEC benchmarks.