# Machine Learning Approach to Generate Pareto Front for List-scheduling Algorithms

Nam Khanh Pham
ECE Department, NUS
Data Storage Institute (DSI),
A*STAR, Singapore
phamnamkhanh@u.nus.edu

Akash Kumar
Center for Advancing
Electronics Dresden (cfaed)
TU Dresden, Germany
akash.kumar@tu-dresden.de

Khin Mi Mi Aung
Data Storage Institute (DSI),
A*STAR, Singapore
Mi_Mi_AUNG@dsi.a-star.edu.sg

## ABSTRACT

List Scheduling is one of the most widely used techniques for scheduling due to its simplicity and efficiency. In traditional list-based schedulers, a cost/priority function is used to compute the priority of tasks/jobs and put them in an ordered list. The cost function has been becoming more and more complex to cover increasing number of constraints in the system design. However, most of the existing list-based schedulers implement a static priority function that usually provides only one schedule for each task graph input. Therefore, they may not be able to satisfy the desire of system designers, who want to examine the trade-off between a number of design requirements (performance, power, energy, reliability ...). To address this problem, we propose a framework to utilize the Genetic Algorithm (GA) for exploring the design space and obtaining Pareto-optimal design points. Furthermore, multiple regression techniques are used to build predictive models for the Pareto fronts to limit the execution time of GA. The models are built using training task graph datasets and applied on incoming task graphs. The Pareto fronts for incoming task graphs are produced in time 2 orders of magnitude faster than the traditional GA, with only 4% degradation in the quality.

## CCS Concepts

•**Computer systems organization** → **Embedded software;**
•**Hardware** → **Operations scheduling;**

## Keywords

Design space exploration; List-scheduling; Machine Learning

## 1. INTRODUCTION

Since its first introduction, list-based scheduling has been extensively used in different domains from operational research to electronic system design and cloud computing [14, 21]. In the field of electronic system design, the spirit of list-based approach is widely adopted with the usage of cost/priority function to guide various resource management processes: mapping, scheduling, placement and routing. For scheduling problems, all tasks in a task graph are sorted into a list by a priority function; then they are allocated on the computational platform by their order in the list. While the very first schedulers were mostly performance-oriented and focused on reducing the schedule length; in recent list-based approaches, the cost function evolves over time to cope with different requirements of the design process: new terms added to reflect the pay-off in hardware resource, energy consumption, or reliability [15,18]. However, the main limitation of existing list-based schedulers is their ability to produce only one scheduling result for each application; therefore, the trade-off between different design objectives cannot be examined.

To address this problem, multi-objective approaches such as Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) were proposed [9]. In these approaches, the components in the cost functions are parameterized; hence, with the same task graph, different parameter sets give different scheduling results. Each combination of different choices for these parameters provides a single option in terms of design objectives (performance, energy...) and forms a specific design point in the design space. Thereafter, these algorithms can help the designers to efficiently traverse the design space and generate a set of points that are superior in one of the objective dimensions. These points form the Pareto front, which is the Holy Grail for system designers since it not only provides the insight into the trade-off between different objectives but also allows them to choose the most efficient design for different purposes. However, the process of traversing the design space in GA and PSO is usually very time-consuming due to the exponential increase in the number of design points to the dimension of the space, which are the number of coefficients in the priority functions. The problem is worse when the scheduling algorithm is complicated and the evaluation time for each design point is long.

Recently, learning based techniques are emerging in the resource management domain of electronic system design. Several works have used Machine Learning (ML) techniques to solve time consuming issues such as placement and routing [11, 16]. For scheduling problem, the ML approaches are well-studied for performance modeling and resource allocation in cloud computing [10]. However, to the best of our knowledge, there is no reported ML-based attempts that try to solve the time consuming problem of generating Pareto front for list schedulers during GA optimization.

**Our contributions:** Towards the same purpose, we propose a multi-level Machine Learning framework that utilizes Spline Regression and Linear Regression to build predictive models for Pareto



*SCOPES '16, May 23-25, 2016, Sankt Goar, Germany*

DOI: http://dx.doi.org/10.1145/2906363.2906380

fronts from a training set of task graphs (TG) and applies these predictive models to accurately estimate the Pareto fronts of new incoming tasks in a fraction of time when compared to GA approach. Following are our main contributions in this work:

- Developing a comprehensive multistage framework for integrating GA and ML techniques to optimize existing list-based schedulers: from generating data to building predictive models and predicting Pareto fronts for new TGs;
- Building a systematic representation of Pareto front curves with Spline regression models;
- Applying the Linear Regression techniques to model the dependency between Spline model of Pareto front and TG's features;
- Applying Density-base Clustering Algorithm to generate near-Pareto-optimal design points.

**Paper Organization:** Section 2 presents state-of-the-art related to GA and ML techniques in scheduling domain. Section 3 provides the overview of our multistage framework. The details of Pareto front Generating phase are presented in Section 4. Section 5 covers the main Regression techniques used in Model Building Phase. Section 6 describes the implementation of Predicting Phase. In Section 7, experimental results are reported and Section 8 provides the conclusion.

## 2. RELATED WORK

Genetic algorithm approaches have been used intensively for optimizing scheduling algorithms, especially in cloud computing systems [9]. However, when it comes to multiprocessor systems (MPS) with tight timing requirements, the applications of GA are quite limited because of its time-consuming behavior. Sutar et al. proposed memetic algorithm that combines GA with simulated annealing to solve the scheduling problem of precedence constrains tasks [23]. Towards using GA-based scheduling algorithm with primary-backup scheme to improve the fault-tolerance of real-time MPS, Zarinzad et al. and Samal et al. proposed their frameworks in [25] and [20] respectively. Obviously, none of above-mentioned studies incorporates ML techniques to solve the time-consuming problem of GA methods in scheduling domain. That unique point makes our work stand out from previous studies, which also try to apply GA approaches for solving scheduling problems.

Recently, ML techniques have emerged as a promising and efficient solution for resource management problems. A comprehensive survey on existing learning-based approaches for the same problems on cloud computing systems has been conducted by Hormozi et al. [10]. More specific overview on the direction of energy minimization is presented by Berral et al. [2]. As summarized from these works, the main application of ML techniques in scheduling problem is performance modeling and Quality of Service (QoS) modeling. For performance modeling purpose, the historical data on execution trace of previous applications are used to build predictive models to forecast the performance of new coming applications [12]. In the other hand, the models for QoS are usually built based on the dependency with available resource (CPU, memory, bandwidth ...) and application requirements [3]. These models are then used to assist the scheduler at runtime to efficiently allocate the resources. Second approach to apply ML techniques in resource management is to classify applications and make decision

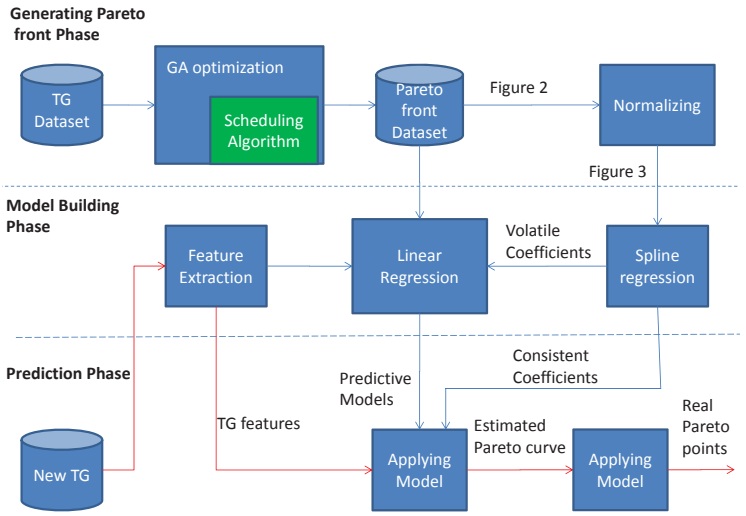



Figure 1: Proposed framework

based-on the classification results [7]. Using unsupervised learning techniques such as Reinforcement Learning to build autonomic self-management schedulers is another trend not only in cloud computing [17] but also in digital system design [5].

Amongst above-mentioned body of works, our framework is most related to the first application of ML in resource management domain since we also use regression techniques to build predictive models. However, the differences in purpose and the interaction between scheduling algorithms and ML techniques make our framework unique and novel. While the existing works try to assist the schedulers by predicting the performance or QoS of new applications, our framework tries to model the behavior of schedulers during GA optimization process and build predictive model for the result of that procedure (i.e. Pareto front).

## 3. OVERALL FRAMEWORK

In this section, we provide an overview of the working flow and the general functionality of the components in our framework. Basically, we explain how the Genetic Algorithm (GA) and Machine Learning (ML) techniques are utilized to optimize the list-based schedulers. As can be seen from Fig.1, our framework has 3 main phases, 2 of them execute at training time: Generating Pareto front and Model Building Phase, while the other Prediction Phase runs at execution time.

### 3.1 Phase 1: Generating the training database

In the first phase, the original Scheduling Algorithm is wrapped by the *GA optimization* process, which takes a bunch of previously generated task graphs (TG) as the input, iterates through their design spaces and generates the optimal Pareto front for each TG. The generated Pareto fronts are stored in a database to feed to the Model Building Phase after being processed by the *Normalizing* block. The implementation details of this phase are discussed in Section IV.

### 3.2 Phase 2: Building the predictive models

The Model Building phase contains main contributions and most of the novelties of our work. The procedure in this phase starts from *Spline Regression* block, which takes the normalized Pareto front curves from Phase 1 as input and builds Spline Regression models that fit the Pareto curve with acceptably small error. There-

after, it filters out the most Volatile Coefficients of generated Spline models and sends them to *Linear Regression* block, which is the second most important ML block. This module receives historical data, which are Volatile Coefficients from *Spline Regression* block and range of Pareto front from Phase 1, as well as the Features of respective TGs in the TG dataset. From these inputs, it builds Linear Regression models that characterize the dependences of Spline Coefficients and Pareto range on the TG features. The Predictive models output from this module are sent to Phase 3 for use at execution time. The last component of Model Building Phase is *Feature Extraction* block, which computes the most important metrics of TG and creates new concise and systematic representation for TG. In the training phase, this block processes the TG from historical Dataset and sends the features to *Linear Regression* block; while in Prediction Phase, it computes features for new TGs and feed them to the *Applying Model* block. The components of Model Building Phase are further presented in Section V.

### 3.3 Phase 3: Prediction at execution time

The last phase in our framework utilizes the results from previous stages to generate the Pareto front for a new TG at execution time. The first building block in this phase is *Applying Model* component, which takes the Linear Regression models and features of the new TG to build the estimated Pareto curve. The *Trace back* block produces the real design points on Pareto front from previously estimated curves.

### 3.4 Advantages

Our framework is developed with a fashion of modular approach so that the designers can freely customize by plugging in new schedulers, new multi-objective optimization approaches or new ML techniques. At the same time, the framework is also uniformly practical in the sense that the designer can quickly apply for a new scheduling algorithm just with the built-in components. The only part that might need to be customized is the *Feature Extraction* block, which needs to be adapted for the application models (i.e. Task Graph, SDF, or Kahn Processing Network (KPN)...). For the ease of representation and reducing the level of abstraction, the detailed implementation of the components in the follow Sections will be presented with examples, which show how to apply our framework to a specific algorithm named energy-conscious scheduling (ECS) [15].

## 4. PHASE 1:GENERATING THE TRAINING DATABASE

During the design process, system designers need to explore the design space to find the solutions that satisfy the trade-offs between often conflicting criteria such as: performance (throughput, latency), hardware usage, energy consumption and reliability. The commonly-used tools to facilitate this exploration process are multi-objective optimization algorithms such as: Evolutionary Algorithm and Particle Swarm Optimization (PSO). The result of these optimizations is the Pareto front on the objective space that contains non-dominated design points, which have no other design points that better than themselves in all dimensions of the objective space. In our example with ECS, the objectives under consideration are Schedule Length and Energy consumption.

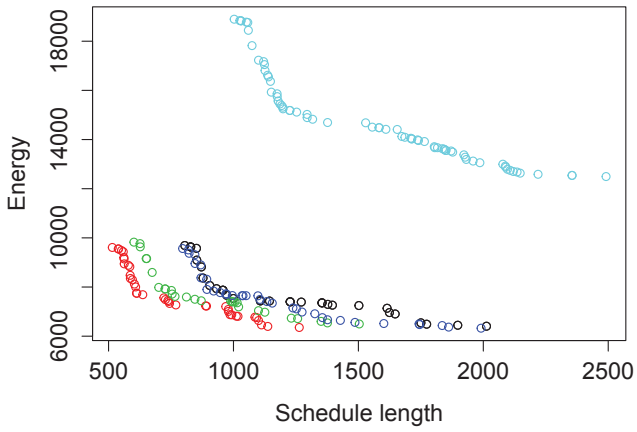



Figure 2: Original Pareto front of 5 task graphs

### 4.1 Generating the Pareto fronts with Genetic Algorithm

To apply the GA for scheduler ECS, we first need to parameterize the cost function from [15]. The original cost function given by Eqn.1 provides only one scheduling result for each TG, while the parameterized version in Eqn.2 offers various scheduling solutions with different set of parameters $(\alpha, \beta, \gamma, \delta, \eta)$.

$$RS(n_i, p_j, v_{j,k}, p', v') = \frac{E(n_i, p_j, v_{j,k}) - E(n_i, p', v')}{E(n_i, p_j, v_{j,k})} + \frac{EFT(n_i, p_j, v_{j,k}) - EFT(n_i, p', v')}{E(n_i, p_j, v_{j,k}) - min(EFT(n_i, p_j, v_{j,k}), EFT(n_i, p', v'))} \quad (1)$$

$$RS(n_i, p_j, v_{j,k}, p', v') = \frac{\alpha * E(n_i, p_j, v_{j,k}) - \beta * E(n_i, p', v')}{E(n_i, p_j, v_{j,k})} + \frac{\gamma * EFT(n_i, p_j, v_{j,k}) - \delta * EFT(n_i, p', v')}{E(n_i, p_j, v_{j,k}) - \eta * min(EFT(n_i, p_j, v_{j,k}), EFT(n_i, p', v'))} \quad (2)$$

Where $E(n_i, p_j, v_{j,k})$ and $E(n_i, p', v')$ are the energy consumption of task $n_i$ on processor $p_j$ with operating voltage $v_{j,k}$ and that of task $n_i$ on $p'$ with $v'$, respectively, and similarly the earliest finish times of the two task-processor allocations are denoted as $EFT(n_i, p_j, v_{j,k})$ and $EFT(n_i, p', v')$. The relative superiority $RS(n_i, p_j, v_{j,k}, p', v')$ is the objective function that balances both performance considerations. More details on ECS can be found in [15]. The parameters $(\alpha, \beta, \gamma, \delta, \eta)$ are chosen to capture all the important factors that might affect the result of objective function.

Thereafter, the GA is used to explore the space of these parameter set to find the Pareto front in the objectives space. In general, the GA encodes the parameters in the form of chromosome and uses the objectives as criteria to heuristically search for better parameters by iterating from generation to generation. The good parameter sets are transferred through generations by inheritance while the new potential parameter sets are explored through mutation. There are quite a number of different implementations of GA but we use the NSGA II algorithm because of its proven efficiency and popularity [22]. The choice of GA depends on the designers' taste and by no means limits the generalization capability of our framework.

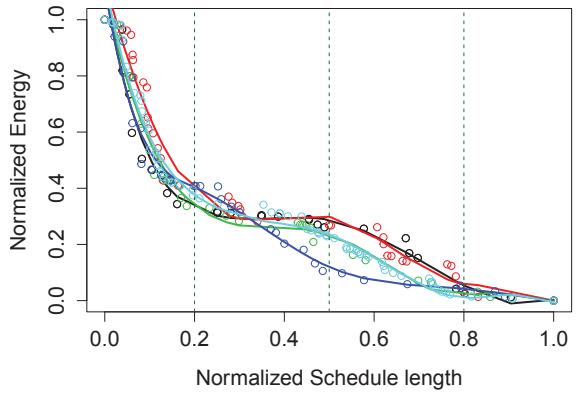



Figure 3: Normalized Pareto fronts and their Spline Models

## 4.2 Normalizing the Pareto fronts to uniform curves

Fig.2 shows an example of Pareto fronts of 5 TGs, which are the outcome from GA block. As can be observed, the general shapes of the Pareto curves are somehow similar while the range and the scale of these curves have major differences. To overcome this problem and make the Pareto fronts easier to interpret and more uniformly across the TG dataset, we normalize the curves so that all the Pareto fronts fit in the range of $[0, 1]$ for all dimensions of objectives (Schedule length and Energy). The formulas used in the normalization process are given in Eqn.3 - Eqn.4. The Pareto fronts after normalizing step are presented in Fig.3. As can be seen, the common pattern of Pareto curves becomes more apparent when they are nicely fitted in the range of $[0, 1]$.

$$T_i = (T_i - T_{min})/(T_{max} - T_{min}) \quad (3)$$
$$E_i = (E_i - E_{min})/(E_{max} - E_{min}) \quad (4)$$

Where $T_i$ and $E_i$ denote the Schedule length and Energy consumption of the $i$-th point on the Pareto front. $T_{max}$, $T_{min}$ and $E_{max}$, $E_{min}$ represent the range of Pareto curve in two objective dimensions.

# 5. PHASE 2: BUILDING THE PREDICTIVE MODELS

As discussed earlier, the Model Building phase contains two main blocks that integrate Spline Regression and Linear Regression techniques into our framework.

## 5.1 Build Spline Regression for Pareto curves

After observing the similar pattern in normalized Pareto curves, we try to quantify the similarity by transforming the curves into a more systematic representation, which is a function describing the relationship between Energy and Schedule Length of the points on Pareto curve. Based on the continuity and the curvy shape of the Pareto front, a number of different regression models have been tested to find appropriate function such as: piece-wise polynomial regression, smoothing spline, local regression [8]. Amongst them Cubic Spline Regression is nicely fitted into our framework due to the balanced trade-off between accuracy and computational complexity [8].

In general, Spline Regression partitions the whole range of predictor (Schedule Length) into $K$ distinct intervals. Then, in each interval, it tries to fit a polynomial function to the data. For the Cubic SplineâĂŹs case, 3-degree polynomials are used. Unlike normal Piecewise Polynomial Regression, a set of constraints on

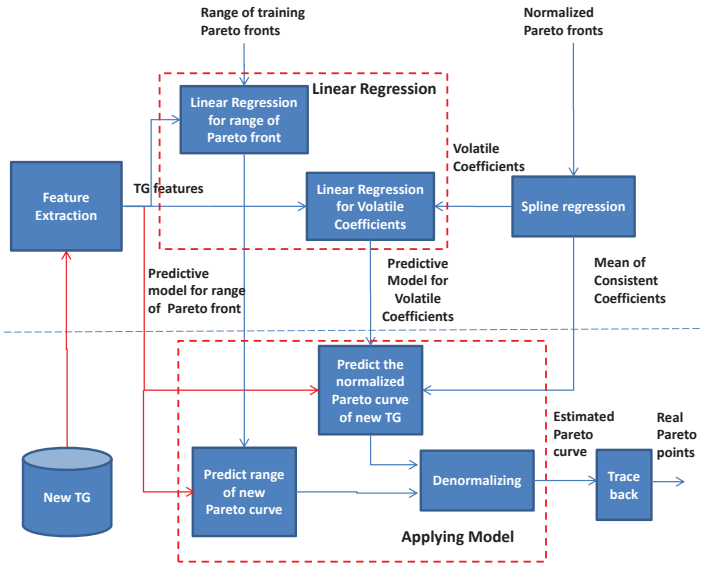



Figure 4: Details of Model Building and Prediction Phases

continuity are applied to ensure smooth transformation between intervals. The division points are called *knots* and the choices of their number and values are very important factors in Spline Regression. $K = 3$ has been found empirically to provide the best curve fitting vs. computation trade-off. The general formulation of Cubic Spline model is given in Eqn.5:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i$$
$$b_1(x_i) = x_i$$
$$b_2(x_i) = x_i^2$$
$$b_3(x_i) = x_i^3$$
$$b_{k+3}(x_i) = (x_i - \xi_k)_+^3, \quad k = 1, \ldots, K$$
$$where$$
$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3, & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Where $y_i$ is the response and $x_i$ is the predictor. In our case $y_i$ and $x_i$ is the Energy and Schedule Length of the $i$-th point in Pareto front; $\beta_0 - \beta_{K+3}$ are the coefficients of the models. They are different from TG to TG and each coefficient set characterize the Pareto curve of a specific TG. $b_1 - b_{K+3}$ are the basic functions of the models. $\xi_k$ are the knots; the basic functions relative to the knots $b_4 - b_{K+3}$ imply the constraints that the curve will be continuous up to 2-orders of derivatives at each knot; hence, ensure the smoothness of the curve. From the Eqn.5, we need $(K + 4)$ Coefficients to define a unique Cubic Spline or Pareto curve of a specific TG.

Fig.4 presents more details on the functionality of blocks and process in Phase 2 and Phase 3. After generated in Spline Regression block, the Spline Coefficients are classified into *Volatile* and *Consistent Coefficients*. The *Consistent Coefficients* have a small variance compared with their average ($\leq$ 10%) and they do not vary much across different TGs. So, we can use their mean for the new coming Task graphs. Therefore, they are transferred directly to *Predict Normalized Pareto* block. In contrast, the coefficients with large variance, i.e. more than 10% of their mean, are defined as *Volatile Coefficients*. These coefficients change values from TGs to

TGs and are dependent on the TG features. Therefore, we need the Linear Regression model in Subsection V-B to characterize this dependency and they are sent to *Linear Regression* block. The threshold of 10% is derived empirically and might be tuned for different Regression techniques. Generally, for majority of the regression techniques, the threshold of 10% gives a good trade-off between computational effort and accuracy of final results. For example, Fig.5 shows Boxplot graph of $3 - knots$ Cubic SplineâĂŹs coefficients from dataset of 40 TGs. It is obvious that only 5 out of 7 coefficients vary across the TGs; hence, they are potential candidates of Volatile Coefficients.

## 5.2 Build Linear Regression for Spline's Volatile Coefficients and Pareto's range

From the above discussion, the real Pareto front of a TG can be rebuilt based on 3 types of parameters: the range of Pareto curve, the Volatile Coefficients and the Consistent Coefficients of the Spline Model. Amongst them, only the Consistent Coefficients are unchanged across the TGs while the others are dependent on the features of TG. Therefore, we need to build predictive models to capture the dependencies of the range of Pareto curve and the Volatile Coefficients on the TG's features. That also describes the role of **Linear Regression** (LR) block in our framework. As can be seen from Fig.4, there are 2 sub-modules in this block: *Linear Regression for Pareto's range* and *Linear Regression for Volatile Coefficients*. The former sub-module takes input from training Pareto curves and associate TG features to generate the LR models for predicting the min, max of Pareto curves. The later sub-module uses training Volatile Coefficients generated from Spline Regression block to build the model for predicting normalized Pareto curve in Prediction Phase. The general formulation of LR model is given in Eqn.6:

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon_i \qquad (6)$$

Where $Y_i$ is the outcome. In our framework it will be the $min$, $max$ of Schedule Length and Energy of points on Pareto curve or the Volatile Coefficients of Spline Model. $\beta_i$ is the coefficients of LR model. In our example, there will be 9 models and coefficient sets in total, 5 for predicting the Volatile Coefficients and 4 for estimating the range of Pareto front. $X_i$ refers to the features extracted from TGs such as: number of tasks, number of edges, maximum bottom level, maximum top level, mean of task size, variance of task size, mean of edge length, variance of edge length. These features are selected from popular metrics of a TG [14] and its statistics. The reason behind the choices of TG's features and LR is once again the trade-off between accuracy and computational complexity. In fact, the simple LR model can well describe the dependency between the TG's features and the outcomes since it can explain more than 95% of the variation in the dataset ($R^2 >= 0.95$). As mentioned above, the features selection depends on the application model used by original scheduling algorithm. For other application models such as SDF or KPN, the designers may need different *Feature Extraction* blocks.

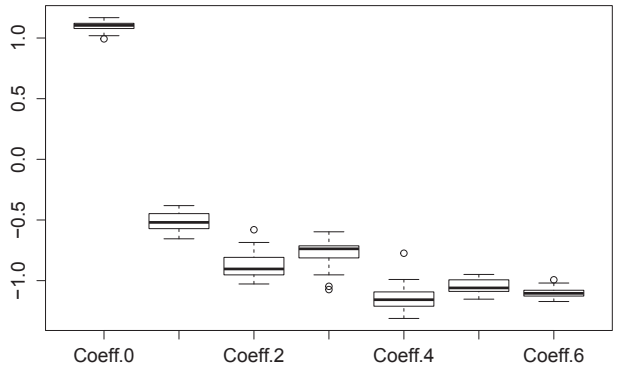



Figure 5: Boxplot of Spline's coefficients

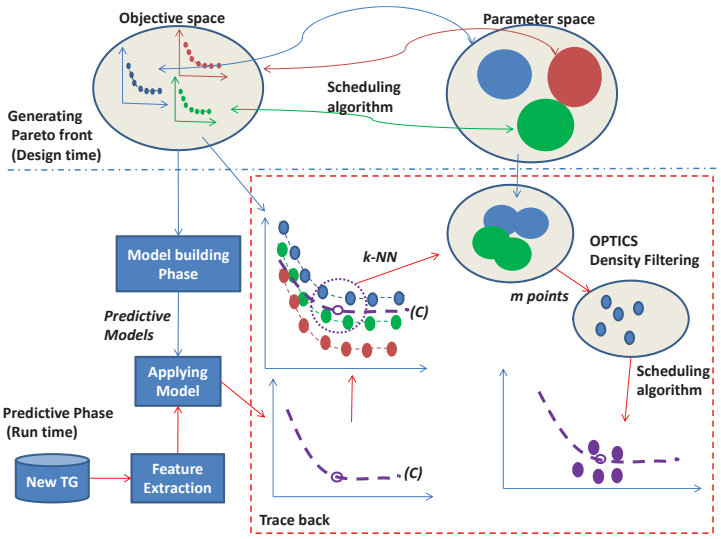



Figure 6: Details of Traceback module

# 6. PHASE 3: APPLYING THE ML MODELS FOR PREDICTION AT RUNTIME

Fig.6 presents the execution procedure in the third phase of our framework. When a new TG comes, its features are extracted and sent to **Applying Model** block to generate an estimated Pareto curve, which is denoted as curve *(C)*. The **Trace back** module is introduced to obtain real Pareto points on the curve. The detailed steps of **Trace back** are presented in the dotted rectangle. First, the *targeted point* (unfilled-point) and *estimated curve* (curve *(C)*) are put on the same normalized objective space with the Pareto fronts of training TGs. Then, k-nearest neighbors (k-NN) of the *targeted point* are extracted. Subsequently, the parameter space of these k-NN points are fed from historical data and merged together to form a *potential parameter space*. Thereafter, a clustering algorithm called *Ordering points to identify the clustering structure* (OPTICS) [1] is applied to *potential parameter space* to filter out *m potential parameter sets*, which have the largest local density factor. Finally, the scheduling algorithm is called for these *m potential parameter sets* to generate the desired points on objective space that are closest to the targeted point. The rationale behind the k-NN and OPTICS steps is to extract the most potential parameter sets from the historical parameters space.

There are 2 obvious use cases, where our framework can be applied efficiently.

• **Generating the most efficient design points that satisfy predefined constraints on objectives:** this use case is similar to the procedure described above where the *targeted point* is defined by the objective's constraints. Our approach provide a huge advantage over the Multi-objective Algorithms (MOAs) in term of execution time since we just need to evaluate several points on the

objective space while the traditional MOAs need to generate the whole Pareto fronts before obtaining the design points satisfied the constraints. The difference is especially significant when the bottleneck is usually attributed to the scheduling procedure which is called to generate a design point on the objective space.

• **Generating the whole Pareto front of a new TG:** in this scenario, the designer can divide the estimated Pareto curve in *n- intervals* and run **Trace back** procedure for each point of these intervals. The results are combined to form the Pareto front for the new TG. The traditional alternatives for this use case are existing Multi-objective Algorithms and GA is one of the most prominent candidate.

Although our framework is presented with an example of 2 dimensional objective space, applying our framework to multi dimensional objective space is straight forward. The only major change is in the Spline model for Pareto fronts (Subsection 5.1). The steps to modify our framework for multi-dimensional objective space are described below:

• Choose one of the objectives as the response ($y_i$);

• Consider the remaining objectives as the predictors ($x_i$ , $z_i$ , $u_i$ ...) ;

• Build the Spline model for the response based on the polynomial of predictors. Eqn.5 will now include the components of $(z_i, u_i \ldots)$ similar to the ones for $(x_i)$.

• Apply the rest of the framework as described above.

The consequence of multi-dimensional design space is that the number of Volatile Coefficients might be increased and the execution time of the whole framework might be longer. However, the same implication exists for GA methods as well. Therefore, we expect the speedup of our framework to remain the same or even improve further.

# 7. EXPERIMENTAL RESULTS

A number of experiments are conducted to evaluate the performance and efficiency of our framework. Because of the limited space, in this section we report only the result of experiments conducted for the second use case, where the designer wants to generate the whole Pareto front. This also allows comparison of our framework directly with the **GA method**. For the first use case, the run time performance can be approximated from the run time reported here, whereas the quality of the outcome needs more thorough examination. We have applied our framework to a list-based scheduler named ECS [15]. The scheduler is coded in Java, the GA optimization is implemented with the NSGA II algorithm from NGPM package [22] in Matlab 2013 and run with a configuration of 50 population size and 100 generations. Finally, the ML techniques are developed with R 3.2 and Splines package [19]. All experiments are performed on an Intel Core i7 2.26GHz CPU with 8 GB RAM. The platform under scheduling has 4 heterogeneous processors that can operate in different levels of Supply Voltage as in the original platform model [15]. The Energy and Schedule Length are obtained with the energy model and execution model used in the original Scheduler ECS [15]. The criteria for comparison in our experiments are quality of generated Pareto fronts and the execution time of all methods.

## 7.1 GA method over original scheduler

**In the first experiment**, we examine the efficiency of GA methods and the accuracy of our *Pareto front estimation*, which is the result of our ML framework prior to applying the **Trace back** module. This experiment is performed with 3 synthesized groups of

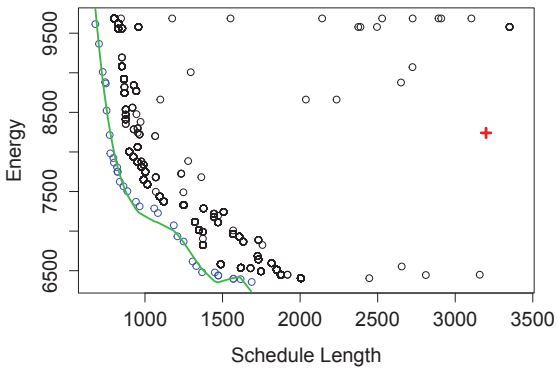



Figure 7: Result for single TG from fat group

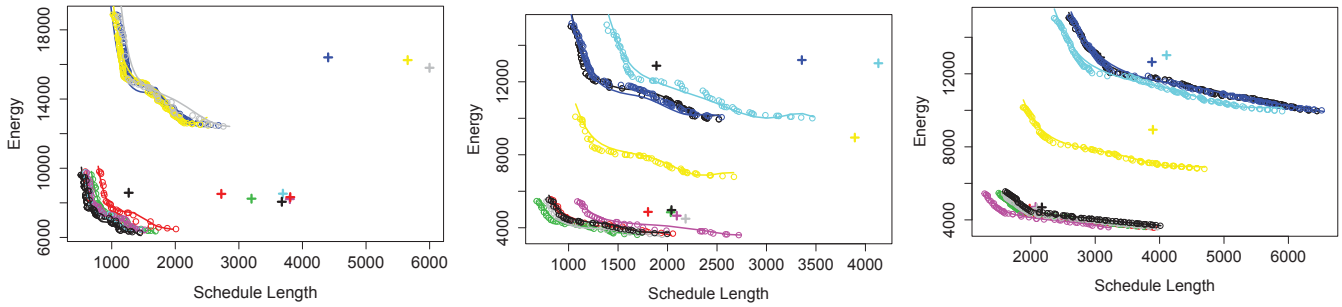
TGs, each with 50 TGs, which are generated from TGFF tool [6] with different levels of parallelism: *fat*, *medium*, *slim*. Out of 50 TGs in each group, 40 TGs are used as training set in **Phase 1** and **Phase 2** of our framework. The predictive models are built with 10-fold Leave One Out Cross Validation process [8] to assure the generalization capability of the models. The other 10 TGs are used as new TGs to test the accuracy of the ML techniques. All the results shown in this Section are from the test set.

Fig.7 shows the design space for 1 TG in the *fat* type. The red plus sign presents the scheduling result from original ECS, while the *real* Pareto front from GA method is marked with blue circle points and the *estimated* Pareto curve by ML techniques is presented as continuous green line. It is obvious that the GA algorithm provides far better results in all objective dimensions when compared with original scheduler. It is easy to understand since the GA has to pay a huge trade-off in running time to achieve such a superior result as can be seen later in the runtime analysis part. The more interesting observation is that the Pareto curve estimated by our 2 levels ML techniques is very close to the real Pareto front generated by GA. This result proves that both the Spline Regression and Linear Regression have done a good job in modeling the dependency between TG's features and Pareto front curve.

Fig.8 combines all the results from 10 TGs in the test set into one plot. As can be seen from these figures, the superiority of GA over original schedulers and the accuracy of our estimated Pareto curve hold true for all the TGs in the test set across 3 different TG types. Another interesting phenomenon is that the shape of Pareto fronts becomes more homogeneous when moving from *fat* to *slim* group; the results of GAs also become less dominating over the results of original scheduler. This can be explained by the fact that the TGs with higher parallelism will have more different ways to be placed on the computational platform; hence, their design spaces are bigger and more heterogeneous. The chance that original scheduler produces a result in suboptimal region of the design space is also higher.

## 7.2 Our framework over GA method

While previous experiment has shown that the *estimated Pareto curves* are very close to the GA Pareto front, they are just intermediate result and need to be processed by **Trace back module** to generate real design points on objective space. In this experiment, we examine the ultimate result of our framework which are generated after **Trace back module**. These results are compared directly with the Pareto fronts from GA method. Fig.9a and Fig.9b show the result for 1 TG in the fat type and medium type. The red




(a) Fat group (b) Medium group (c) Slim group

Figure 8: Combined result of all TGs in test set

points are the Pareto front generated by GA, the green line is the estimated Pareto curve from the beginning of the **Trace back** procedure. All the points generated from **Trace back** step are marked with blue color, where the plus signs and square signs represent the result when $m = 1$ and $m = 10$ respectively. As can be seen, the Pareto fronts generated by our framework are very close to the ones from GA method. The figure also shows how the quality of our Pareto fronts improve with the increase in $m$.

Since the most time consuming process in both GA and our framework is executing the scheduler to get the design point on objective space, we designed this experiments around two hyper-parameters: *n-interval* and *m-potential candidates*, which directly decide on the number of evaluation points in our framework. After varying the value of $m$ and $n$, we quantify the quality of the Pareto fronts using popular metric in the MOA domain: R2-indicator (R2I) [4], where the reference set is the origin of the objective space. The quality degradation of the results generated by our framework when compared with GA's Pareto fronts are measure relatively by the Quality Trade-off (QT) in percentage of the GA's R2-indicator. The measurements are averaged over all the 10 TGs of the test set and reported a long with the execution time in Table 1. As can be observed from both the Table and the Figure, the quality of our framework is approached to the one generated by the GA method when increasing the number of evaluations (by increasing $m$ or $n$) and the pay-off for that improvement is the nearly linear increase in execution time. However, to achieve the comparable quality to the result from GA we need only a fraction of time. With $m = 1$ and $n = 20$, we can achieve 2 orders of speed-up over the GA with less than 1% deficiency in the quality of the Pareto front for all types of task graph. Such an achievement is due to the fact that all the heavy computation is moved to the training phase and take advantage of the ML models built upon the historical data. As discussed in subsection 6, the runtime overhead of ML method can be broken down to 4 main components: Feature Extraction, applying Linear Regression model, applying Cubic Spline model and Denormalizing. While the Feature Extraction part has more or less the same complexity as the original scheduler, the other components are very simple computations: vector multiplications for applying Linear Regression and Denormalizing blocks, 3-degree polynomial computation for Applying Cubic Spline.

## 7.3 Real life applications

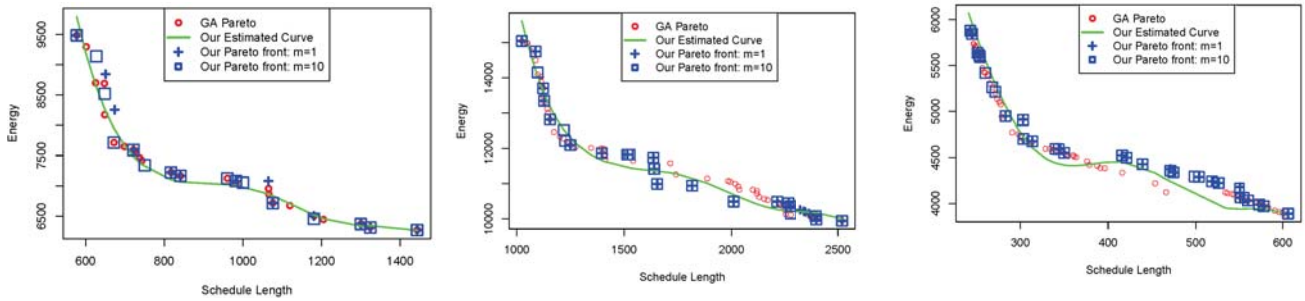
In this experiment, we used the predictive model built from *fat* group to apply for task graphs of realistic applications: *MP3 decoder* [13], *robot control*, *sparse matrix solver* and *fpppp* from the benchmark [24]. The choice of the model from *fat* training set is explained by the fact that *fat* type has the largest variance in parallelism from the 3 groups; hence, model built on these TGs is the most flexible and generalizable. Fig.9c presents the qualitative results from the *sparse* task graph. The figure again demonstrates the capability of our ML methods to accurately generate the Pareto fronts of new TGs just based on their features (without any prior information about TG). The quantitative result is shown in the second half of Table 1 with the same metrics as in the 1st experiment. The same 2 orders of magnitude speed-up can be achieved for the simplest configuration (m=1, n=20) with small degradation in the Quality (the trade-off is still less than 4%).

Table 1: Execution time and quality comparison

| TGs | | Our approach | | | | | GA |
|---|---|---|---|---|---|---|---|
| | m | 1 | 5 | 10 | 5 | 5 | 50 |
| | n | 20 | 20 | 20 | 40 | 80 | 100 |
| Fat | R2I | 0.42011 | 0.41979 | 0.41963 | 0.41953 | 0.41951 | 0.41936 |
| | QT (%) | 0.17 | 0.10 | 0.06 | 0.04 | 0.03 | 0 |
| | Time | 23.989 | 68.181 | 124.138 | 132.425 | 260.932 | 2653.873 |
| Medium | R2I | 0.41786 | 0.41715 | 0.41671 | 0.41659 | 0.41650 | 0.41644 |
| | QT (%) | 0.34 | 0.16 | 0.06 | 0.03 | 0.01 | 0 |
| | Time | 21.221 | 58.902 | 105.499 | 112.353 | 222.093 | 2564.354 |
| Slim | R2I | 0.42024 | 0.41859 | 0.41829 | 0.41804 | 0.41786 | 0.41775 |
| | QT (%) | 0.59 | 0.20 | 0.12 | 0.07 | 0.02 | 0 |
| | Time | 19.104 | 57.349 | 105.587 | 114.533 | 220.870 | 2542.166 |
| MP3 | R2I | 0.42091 | 0.42021 | 0.41996 | 0.41961 | 0.41951 | 0.41848 |
| | QT (%) | 0.58 | 0.41 | 0.35 | 0.27 | 0.24 | 0 |
| | Time | 16.701 | 31.125 | 48.002 | 58.179 | 115.961 | 2347.545 |
| robot | R2I | 0.43159 | 0.42075 | 0.41885 | 0.41925 | 0.41885 | 0.41640 |
| | QT (%) | 3.64 | 1.04 | 0.59 | 0.68 | 0.58 | 0 |
| | Time | 21.201 | 52.194 | 91.767 | 101.441 | 197.809 | 2553.820 |
| sparse | R2I | 0.42282 | 0.41880 | 0.41846 | 0.41871 | 0.41857 | 0.41694 |
| | QT (%) | 1.41 | 0.44 | 0.36 | 0.42 | 0.39 | 0 |
| | Time | 20.300 | 52.395 | 92.754 | 101.203 | 200.654 | 2832.736 |
| fpppp | R2I | 0.42503 | 0.42065 | 0.41638 | 0.42050 | 0.41610 | 0.41434 |
| | QT (%) | 2.57 | 1.52 | 0.49 | 1.48 | 0.42 | 0 |
| | Time | 30.912 | 89.415 | 154.891 | 214.854 | 424.732 | 6632.461 |

# 8. CONCLUSION

This work presents a generic framework that utilizes Genetic Algorithm and Machine Learning techniques to predict the Pareto front of multi-objective list-based schedulers. While the GA optimization provides Pareto front that are far better than original schedulers with huge trade-off in execution time, our multilevel ML techniques with Spline Regression and Linear Regression can accurately generate the Pareto curve with a much lower execution time and less than 4% deficiency in quality of the Pareto front. This result offers the potential of using the Pareto front information for resource management at runtime, which is was hitherto infeasible due to the time consuming limitation of multi-objective approaches




(a) Fat group (b) Medium group (c) Sparse Matrix Solver

Figure 9: Pareto fronts generated from GA and our framework

such as GA or PSO. The modular design of our framework allows multiple directions for extension: new multi-objective optimization algorithm (PSO) and new ML models can be easily integrated into our framework. Moreover, the framework can be applied to other resource management problems that utilize priority/cost functions such as: mapping, placement and routing.

# 9. REFERENCES


[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[2] J. Berral et al. Toward Energy-Aware Scheduling Using Machine Learning. *Energy Efficient Distributed Computing Systems*, pages 215–244, 2012.

[3] J. L. Berral et al. Power-Aware Multi-data Center Management Using Machine Learning. *ICPP*, pages 858–867, 2013.

[4] D. Brockhoff, T. Wagner, and H. Trautmann. On the properties of the r2 indicator. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 465–472. ACM, 2012.

[5] A. K. Das et al. Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems. In *Design Automation Conference*, pages 1–6. IEEE, 2014.

[6] R. P. Dick et al. Tgff: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101. IEEE Computer Society, 1998.

[7] H. Eom et al. MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training. *International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 51–60, 2015.

[8] J. Friedman et al. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[9] M. Guzek et al. A survey of evolutionary computation for resource management of processing in cloud computing [review article]. *Computational Intelligence Magazine, IEEE*, 10(2):53–67, 2015.

[10] E. Hormozi et al. Using of machine learning into cloud environment (a survey): Managing and scheduling of resources in cloud systems. *Proceedings - 3PGCIC*, pages 363–368, 2012.

[11] K. Jeong et al. Accurate machine-learning-based on-chip router modeling. *Embedded Systems Letters, IEEE*, 2(3):62–66, 2010.

[12] N. H. Kapadia et al. Predictive application-performance modeling in a computational grid environment. In *International Symposium on High Performance Distributed Computing*, pages 47–54. IEEE, 1999.

[13] P. Kumar et al. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *ASP-DAC*, pages 123–128. IEEE, 2011.

[14] Y.-K. Kwok et al. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[15] Y. C. Lee et al. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *CCGRID*, pages 92–99. IEEE, 2009.

[16] R. Manimegalai et al. Placement and routing for 3D-FPGAs using reinforcement learning and support vector machines. *18th International Conference on VLSI Design*, 2005.

[17] J. Perez et al. Utility-based reinforcement learning for reactive grids. In *International Conference on Autonomic Computing*, pages 205–206. IEEE, 2008.

[18] N. K. Pham et al. A multi-stage leakage aware resource management technique for reconfigurable architectures. In *Proceedings of GLSVLSI*, pages 63–68. ACM, 2014.

[19] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[20] A. K. Samal et al. Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. *Swarm and Evolutionary Computation*, 14:92–105, Feb. 2014.

[21] J. Schutten. List scheduling revisited. *Operations Research Letters*, 18(4):167–170, 1996.

[22] L. Song. Ngpm–a nsga-ii program in matlab. 2011.

[23] S. Sutar et al. Task scheduling for multiprocessor systems using memetic algorithms. In *4th International Working Conference Performance Modeling and Evaluation of Heterogeneous Networks*, 2006.

[24] T. Tobita et al. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[25] G. Zarinzad et al. A novel intelligent algorithm for fault-tolerant task scheduling in real-time multiprocessor systems. In *Convergence and Hybrid Information Technology*, volume 2, pages 816–821. IEEE, 2008.