TSUBAME 共同利用　令和 4 年度　学術利用　成果報告書

利用課題名　MPI アプリケーションにおけるプロファイルおよびトレース予測手法の評価
英文：Evaluation of a Method for Predicting Profiles and Traces of MPI Applications

利用課題責任者　三輪忍
First name Surname　Shinobu Miwa

所属　電気通信大学
Affiliation　The University of Electro-Communications
URL　http://www.hpc.is.uec.ac.jp/miwa_lab/index.html

邦文抄録（300 字程度）

本研究では，小規模実行によって得た性能メトリクスを用いて大規模実行時の並列アプリケーションの性能メトリクスを予測するツールである Extra-P を関数コール回数予測に応用した場合の評価を行う．我々の実験結果によると，Extra-P は 5 つの並列アプリケーションに対して関数コール回数を 2.23%以下の誤差で予測できた．また，我々は，関数コール回数予測の応用として，関数コール予測の結果をも言いて関数の実行時間を予測する手法を提案する．我々の実験結果によると，提案手法はさまざまな関数の総実行時間の予測精度を平均 1.41%改善した．

英文抄録（100 words 程度）

In this study, we provide the first evaluation of Extra-P, which is a tool for predicting the performance metrics of a parallel application executed at a large scale using the performance metrics of the application executed at multiple small scales, in a use case of function call count prediction. Our experimental results demonstrated that Extra-P predicted the function call counts for five parallel applications within an error of 2.23%. Additionally, we propose a technique to predict the total execution time of a function using the call count predicted as a use case of function call count prediction. Our experimental results demonstrated that the proposed technique improved the prediction accuracy of the total execution time of various functions by 1.41% on average.

Keywords: Extra-P, parallel applications, function call count, evaluation

1. Introduction

The analysis of the dynamic behavior of parallel applications is widely performed by supercomputing users for various purposes (e.g., performance tuning). Because an application's behavior depends on the scale of execution (i.e., process count and input data), the collection of performance metrics (e.g., function call, and memory and cache access counts) usually requires the execution of a targeted application combined with a profiling tool (e.g., TAU) at a targeted scale. However, this approach for collecting the performance metrics of large-scale applications consumes a large amount of computing time and resources. A more cost-effective approach to collecting performance metrics is required for large-scale applications.

In this context, various methods for modeling the performance of parallel applications have been developed. These methods enable us to collect the various performance metrics (e.g., execution time) of a given application executed at a large scale with reasonable computing time and resources, by creating their scalability models empirically or analytically. Specifically, Extra-P, which is the most powerful tool for the performance analysis of parallel applications, automatically generates a model to provide the extrapolated performance from a subset of metric data of a given application executed at small scales. In previous work, the authors reported that Extra-P is very useful for estimating some performance metrics (e.g., execution time, number of floating-point instructions, and

communication message size) of a given application executed at a large scale; however, but it is still unknown whether Extra-P can be used for scalability prediction for the other metrics. In particular, the function call count is an important metric for understanding the application behavior and is widely used for the performance analysis of parallel applications; however, to the best of our knowledge, researchers have not reported the effectiveness of Extra-P in function call count prediction.

In this paper, we show the first evaluation of Extra-P for function call count prediction. Therefore, as a use case for function call count prediction, we propose a method that predicts the total execution time of function using the number of predicted function calls. Our experimental results demonstrated that Extra-P predicted the number of function calls with sufficiently high accuracy for various functions and our proposed method predicted the total execution time of various functions with higher accuracy than the conventional method.

We summarize the main contributions of this study as follows:

- We demonstrate that Extra-P is also useful for creating models of function call counts.
- We demonstrate that combining function call count prediction is effective in predicting the total execution time of functions.
- We demonstrate that Extra-P greatly reduces the cost of collecting the total execution time of functions.

2. Background

2.1. Profiling

Profiling parallel applications is widely used for performance analysis and tuning in the field of high-performance computing. Profiling involves the collection of the performance metrics of a targeted application. Such metrics include the number of function calls and the total execution time of functions. To collect performance metrics, an instrumentation code is inserted into the targeted application using a compiler or dynamic library, and the application is then executed on a targeted system. Profiling tools, such as TAU and Score-P, are available for the insertion of the instrumentation code. We note that the performance metrics collected through profiling are measured values. We believe that the costs (computing time and resources) of performance-metric collection through profiling are high because this approach requires a targeted application to be executed on a targeted system. Many techniques, such as selective instrumentation, have been developed to reduce the costs of performance-metric collection; however, performance is limited by the execution of the application plus the required collection overhead.

2.2.Extra-P

Extra-P is a performance analysis tool for parallel applications that can be used for the quick analysis of parallel applications. Extra-P extrapolates a performance metric of a targeted application executed at a large scale from those of the application executed at multiple small scales. To accomplish this, Extra-P generates a scalability model for the performance metric using regression. The independent variables of the model are the process count and problem size, and the dependent variable is the performance metric to be predicted.

The general form of the model used in Extra-P is as follows:

$$f(x_1,\cdots,x_m)=\sum_{k=1}^{n} c_k \prod_{l=1}^{m} x_l^{i_{kl}} (\log x_l)^{j_{kl}} \qquad (1)$$

where m is the number of independent variables, x_l is an independent variable, and c_k is a

regression parameter. n is the number of product terms, each of which has a different combination of $\prod_{l=1}^{m} x_l^{i_{kl}} (\log x_l)^{j_{kl}}$. The exponents i_{kl} and j_{kl} are called the hypotheses and characterize the form of the equation. Extra-P users can specify the sets of i_{kl} and j_{kl} (referred to as I and J, respectively) used for the regression from the command line interface.

Using Extra-P, we can collect various performance metrics of a targeted application without executing the application at a targeted scale; however, the performance metrics reported by Extra-P are predicted values. Additionally, the effectiveness of Extra-P has been proven for a few performance metrics (e.g., total execution time of a function and communication message size). It is still unknown whether Extra-P can also be used for function call count prediction.

3. Function Call Count Prediction using Extra-P

3.1. Function Call Count and Performance Tuning

The function call count is helpful for tuning the performance of an application. Because calling a function has a performance overhead, a reduction in the function call count may improve overall application performance. For example, inlining a function can remove the calling overhead from an application, but it increases the application code size. The function call count is fundamental information in performance tuning; therefore, many profiling tools support functionality to report the call counts of the functions executed.

The function call count is a dynamic attribute of an application. It is usually affected by various factors of running applications (e.g., inputs, process count, and results of branch instructions) and cannot be computed using static code analysis. Thus, to obtain the exact number of function calls, we require a targeted application to be executed at a targeted scale.

Table 1. System configuration of TSUBAME3.0

Total peak performance	12.15PFlops（double）
# of nodes	540
Total memory capacity	138,240GB
Network topology	Full-bisection fat tree

Table 2. Node configuration of TSUBAME3.0

CPU	Processor name # of physical (logical) cores Frequency	Intel Xeon E5-2680 V4 14(28) 2.40GHz
Memory	Capacity Bandwidth	256GB 153.6GB/s
GPU	Processor name	NVIDIA Tesla P100

3.2. Experimental Methodologies

In this subsection, we evaluate the accuracy of Extra-P in function call count prediction. Specifically, we first collect sample data (i.e., call counts per function) by executing a targeted application at various small scales and then input them into Extra-P. Next, Extra-P automatically generates an optimal model to predict the scalability of the call count for each function. Finally, we predict the function call count at a large scale with using the model generated and compare the result with the measured call count.

To enable the generation of multi-variable models, we use multi-parameter for the modeler option in Extra-P. Moreover, we use $I = \{-1, 0, 1, 2, 3\}$ for the exponents of the polynomial and use $J = \{0, 1\}$ for the exponents of the logarithm in Equation (1). We also set force_combination_exponents and allow_negative_exponents to True.

We conducted our experiment on the supercomputer TSUBAME3.0. It consists of 540 compute nodes, each of which has two CPUs (Intel Xeon E5-2680 V4). The system and node configurations of TSUBAME3.0 are shown in Tables 1 and 2, respectively. We used TAU as a profiling tool when collecting the actual number

Table 3. Benchmark programs and scales used for model generation. Abbreviations in italics in the Scales column represent the as following: *pc*=process count, *ps*=problem size, *gs*=grid size, *it*=iterations, *kc*=key count, and *mv*=max value.

Name	Scales
EP	pc : (2, 4, 8, 16, 32, 64), ps : (24, 25, 28, 30, 32)
FT	pc : (2, 4, 8, 16, 32, 64), gs : (32, 54, 128, 256, 512), it : (5, 10, 15, 20, 25)
IS	pc : (2, 4, 8, 16), kc : (18, 20, 22, 24), mv : (9, 11, 13, 15)
MG	pc : (4, 8, 16, 32, 64), ps : (4, 8, 16, 32, 64), it : (5, 10, 15, 20, 25)
LULESH	pc : (8, 27, 64, 125, 216, 343), it : (8, 16, 32, 64, 128), ps : (16, 24, 32, 48)

Table 4. Scales used for prediction

Name	Process counts			Input data		
	Small	Medium	Large	Small	Medium	Large
EP	128	256	512	$ps = 36$	$ps = 40$	$ps = 44$
FT	128	256	512	$gs = 128, it = 25$	$gs = 256, it = 25$	$gs = 512, it = 25$
IS	32	64	128	$kc = 28, mv = 20$	$kc = 29, mv = 20$	$kc = 30, mv = 20$
MG	128	256	512	$ps = 16, it = 25$	$ps = 32, it = 25$	$ps = 64, it = 25$
LULESH	512	729	1,000	$ps = 32, it = 48$	$ps = 64, it = 48$	$ps = 128, it = 48$

of function calls.

We tested five HPC applications shown on TSUBAME3.0. Four out of the five applications (i.e., EP, FT, IS, and MG), which we selected from the NAS Parallel Benchmark suite, are simple and stress particular properties. The remaining application, LULESH, is more complex and we selected it from the CORAL-2 Benchmark suite.

The scales used for model generation are summarized in Table 3. We executed an application for every combination of the scale parameters shown in the table and then entered all the resulting function call counts into Extra-P to generate the models. For example, we executed EP at 30 scales (6 types of process counts × 5 types of problem sizes) in total.

Using the models generated by Extra-P, we predicted the function call counts of each application for the three types of process counts and input data shown in Table 4. We note that these targeted scales are larger than the data collection scales shown in Table 3. In the next section, we focus on the functions executed at all the scales listed in Tables 3 and 4. Our experimental results exclude a few functions executed only at a specific scale.

Table 5. Fitting errors in function call count prediction

Name	MAPE[%]	Min MAPE per func.[%]	Max MAPE per func.[%]
EP	324.79	0.00	13775.78
FT	1.58	0.00	283.65
IS	23.66	0.00	2024.92
MG	10.92	0.00	903.92
LULESH	2.64	0.00	94.32

We used the following two metrics to evaluate prediction accuracy.

MAPE: The accuracy of a model is often evaluated using the mean absolute percent error (MAPE), which is the average of the absolute error rate between the model's output values (F_t) and the observed values (A_t). It can be expressed by the following formula:

$$MAPE[\%] = \frac{100}{N}\sum_{t=1}^{N}\frac{|A_t - F_t|}{A_t} \quad (2)$$

where N is the number of data points.

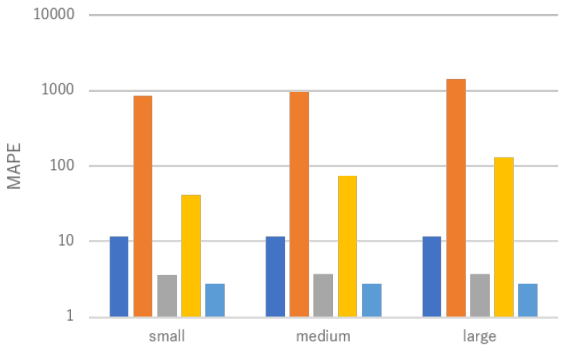
Weighted MAPE: In the MAPE, the errors in each data point contribute to the overall error equally; however, the more precise prediction of a large function call count is helpful for performance tuning. Therefore, we introduce the weighted MAPE as the other metric for prediction accuracy. The weighted MAPE is a weighted average of the absolute error rate by the number of function calls and can be expressed by the following formula:

$$WeightedMAPE[\%] = \frac{100}{c_{sum}}\sum_{t=1}^{N} c_t\frac{|A_t - F_t|}{A_t} \quad (3)$$

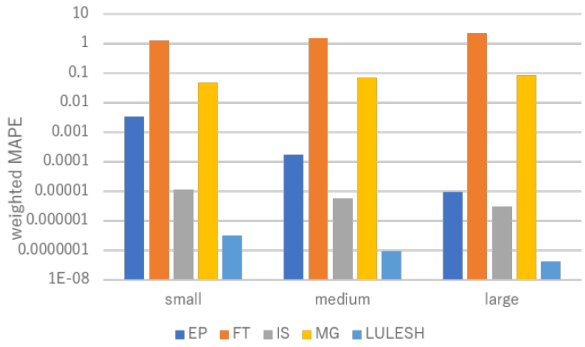
where c_t and c_{sum} represent the number of function calls measured for the function t and the total number of function calls measured during the execution of a targeted application at a given scale, respectively.

3.3. Experimental Results

Table 5 summarizes the fitting errors of the function-call-count models generated by Extra-P.

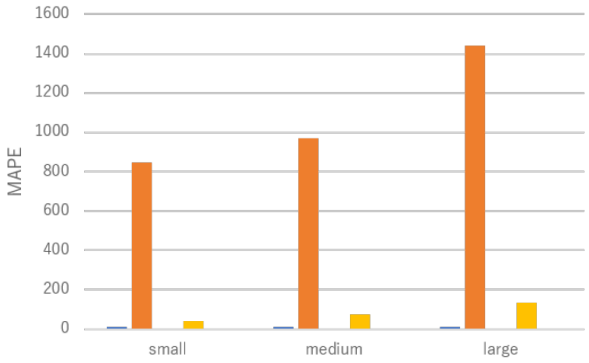


(a) MAPE

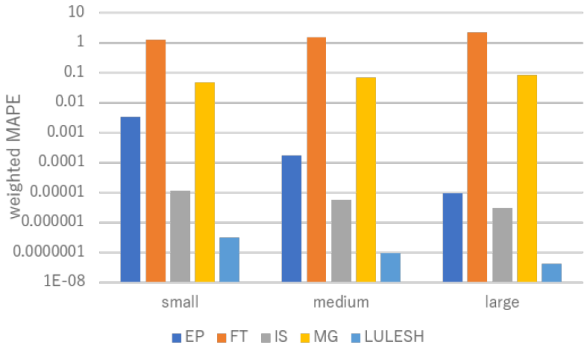


(b) Weighted MAPE

Figure 1. Accuracy of function call count prediction for various process counts using large input data



(b) MAPE



(b) Weighted MAPE

Figure 2. Accuracy of function call count prediction for various input data using large process counts

As shown in the table, Extra-P produced models well -fitted to the collected data, in many cases. In particular, the function-call-count models generated by Extra-P had an error of 1.58% on average for FT. A few applications, such as EP and IS, had large MAPEs because they included some functions that had irregular patterns in the function call count. However, the numbers of calls for such functions were relatively small. Extra-P provided very accurate models for many functions that were important to overall performance, even in such applications.

Figure 1 shows the prediction accuracy of the generated models for various process counts. The x-axis represents the process count, and the y-axis represents MAPE or weighted MAPE. We note that the y-axis uses the logarithmic scale. The five bars on each x-label represent the five applications. We used large input data for this experiment.

Figure 1 (a) shows that two out of the five applications (i.e., IS, and LULESH) had small errors (within 10%). As shown in Table 5, Extra-P produced function-call-count models well -fitted to these applications so that the generated models had high prediction accuracy at the targeted scales.

By contrast, FT and MG had large MAPEs because of the inaccuracy of the call-count models for some functions. However, this inaccuracy may not become a problem in many use cases because such functions have relatively small call counts and are therefore less important than the other functions. Figure 1 (b) exemplifies this. The weighted MAPE was up to 2.23%.

Figure 2 shows the prediction accuracy of the

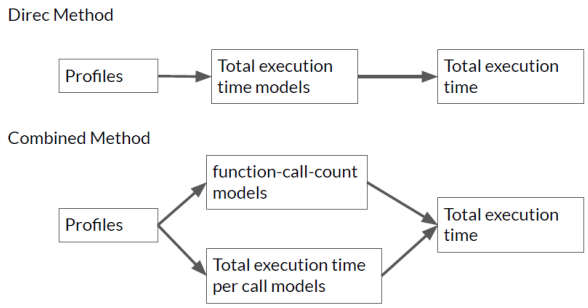


Figure 3. Overview of prediction of total execution time of functions

generated models for various input data. We used large process counts for this experiment. Similar to Figure 1, the weighted MAPEs of the models generated were very small for all combinations of applications and input data, whereas the MAPEs were large for a few applications. Thus, we conclude that Extra-P produced models sufficient for practical use in function call count prediction.

4. Combining Function Call Count Prediction and Total Execution Time of Functions Prediction

As described in the previous section, Extra-P produced accurate function-call-count models, in many cases. In this section, as a use case for predicting function call counts, we combine function call count prediction with the scalability prediction of the total execution time of a function.

4.1. Predicting the Total Execution Time of Functions

The prediction of the total execution time of a function is often performed to identify a scalability bug in a targeted application. As shown in previous work, it is the main use case of Extra-P.

The total execution time of function (T_F) can be expressed as follows.

$$T_F = \sum_{i=1}^{N_F} t_F^i \quad (4)$$

where N_F is the function call count and t_F^i is the i-th total execution time of function F.

Generally, Extra-P predicts the total execution time of function directly. Specifically, we first collect the total execution time of each function (T_F) while executing a targeted application at various small scales. Next, we enter the collected data into Extra-P and then create a total execution time of function model for each function. Finally, we predict the total execution time of function executed at a targeted scale using the generated model (the direct method in Figure 3).

Hereafter, we call this the direct approach because we need to distinguish between this approach and our approach proposed in the next section.

4.2. Combined Approach

Theoretically, the total execution time of function can be expressed as follows.

$$T_F = N_F \times \bar{t_F} \quad (5)$$

where $\bar{t_F}$ is the average execution time of function per call.

Because the execution scales (i.e., process counts and input data) have different influences on the call count (N_F) and total execution time of function per call ($\bar{t_F}$), the direct modeling of T_F is difficult for some functions. For such functions, prediction accuracy can be improved by developing individual models for N_F and $\bar{t_F}$, and then combining the predicted results using the models (the combined method in Figure 3). We call this the combined approach and use Extra-P to model both N_F and $\bar{t_F}$.

4.3. Experimental Methodologies

Many profiling tools report two types of total execution time of function. One is inclusive time, which includes the time taken by all callees. The other is exclusive time, which represents the time taken by the targeted function itself. In this study, we performed the prediction for both inclusive and exclusive time.

Table 6. Fitting errors in the total execution time of function prediction. In "*x* (*y*-*z*)," *x* is the MAPE, and *y* and *z* are the minimum and maximum MAPE per function, respectively

Name	Exclusive		Inclusive	
	Direct [%]	Combined [%]	Direct [%]	Combined [%]
EP	539.18 (0.14-15599.76)	516.85 (0.078-14251.43)	439.78 (0.31-15599.76)	417.70 (0.08-14251.43)
FT	20711.60 (0.02-4125476.68)	409.52 (0.0-51945.34)	415.20 (0.0-84141.52)	472.66 (0.0-51527.31)
IS	402.96 (0.01-48590.21)	421.29 (0.03-66931.58)	364.00 (0.02-49760.63)	408.55 (0.07-66931.58)
MG	104.06 (0.0-11042.49)	46.45 (0.01-1015.20)	120.58 (0.01-26427.56)	138.32 (0.01-42872.94)
LULESH	152.02 (0.0-15216.29)	106.57 (0.0-13155.79)	145.28 (0.0-15816.07)	146.16 (0.0-13155.79)

We used the same experimental system described in Section 3.2. We also used the benchmark programs and execution scales listed in Tables 3 and 4.

We evaluated the cost of data collection, in addition to fitting errors and prediction accuracy. We define the cost of collecting data (C) for an application (*app*) executed on M cores as follows:

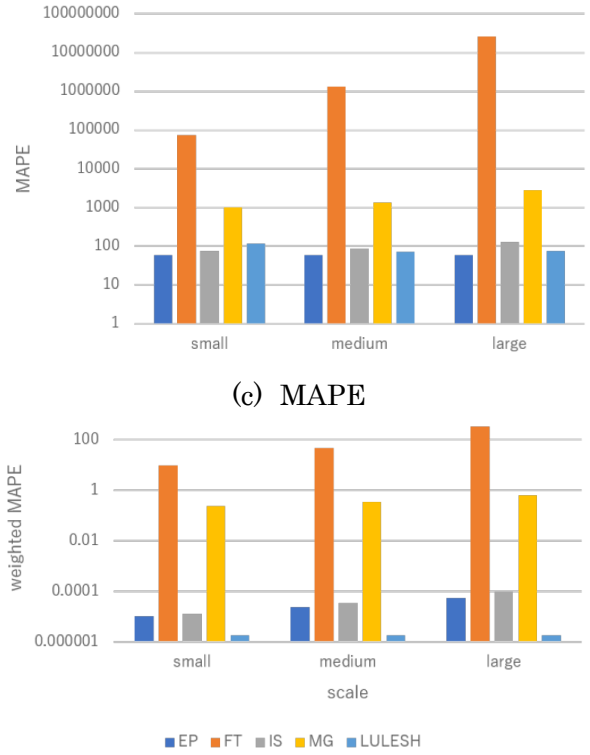
$$C = M \times T_{app}$$

Because many supercomputing services require users to buy points, which are consumed based on the product of the computing time and number of computing resources.

Both direct and combined approaches require model generation and prediction processes in addition to data collection. These two processes also consume computing time and resources; however, they can be executed very quickly, even on a single compute node. Because the costs of model generation and prediction are relatively small compared with the cost of data collection, we ignore them in this study.

4.4. Experimental Results

The fitting errors for the total execution time of function models are shown in Table 6. Each entry represents the MAPE of all models generated by an approach for an application, and the range represents the minimum and maximum values in the case of computing the MAPE for each function. The table shows that the combined approach produced models with higher accuracy than the direct approach when predicting the exclusive time of functions. In particular, compared with the direct approach, the combined approach reduced the fitting error by 20,302% for FT. This is because, with respect to the exclusive time, Extra-P can create more precise models for N_F and $\bar{t_F}$ than T_F.



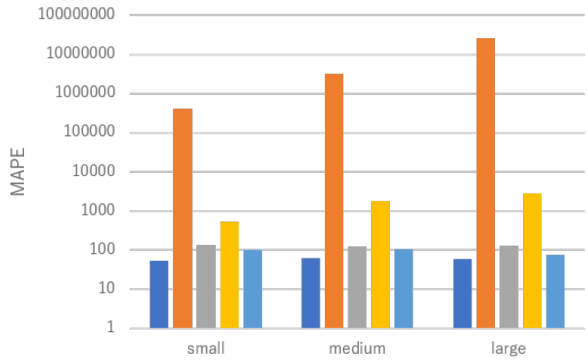
(c) MAPE

(b) Weighted MAPE

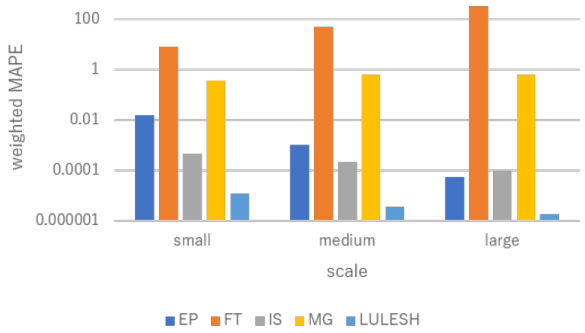
Figure 4. Accuracy of prediction of total execution time of functions for various process counts using large input data

Figure 4 shows the accuracy in predicting the total execution time of function. The x-axis represents the process count, and the y-axis represents the MAPE or weighted MAPE. We note that the y-axis uses logarithmic scale. We used large input data.

As shown in Figure 4 (a), Extra-P on average



(d) MAPE



(b) Weighted MAPE

Figure 5. Accuracy of prediction of total execution time of functions for various input data using large process counts

produced inaccurate models in the total execution time of function prediction. The smallest MAPE was 58.95+%, even when we used the combined approach. In particular, FT had a MAPE of 10,000,000+% for the large process count. Because FT executed many all-to-all communication functions and they were called from several locations within the code, it was difficult for Extra-P to precisely extrapolate the performance of such functions.

By contrast, we observed different properties of the models produced by Extra-P in view of the weighted MAPE. Figure 4 (b) shows that the models achieved very high accuracy (within 1% of the weighted MAPE) for four out of five applications. Therefore, we consider that the models achieved sufficient accuracy in practice.

Figure 5 shows the prediction accuracy of the generated models for various input data. We used large process counts. Similar to Figure 4, the weighted MAPEs were small, whereas the MAPEs were large. The models generated had a weighted MAPE of 1.41%, on average. Thus, we conclude that Extra-P produced models sufficient for practical use in total execution time of function prediction.

Figure 6 shows the cost of collecting the total execution time of functions. The x-axes represent the number of execution scales to be collected, while the y-axes represent costs. The y-axis uses the logarithmic scale. The blue lines represent the case of predicting the total execution time of function and the red lines represent the case of collecting the total execution time of function based on actual runs (i.e., using profilers). We note that the data collection time for the direct approach was the same as that for the combined approach.

The figure shows that the cost of predictively collecting the total execution time of functions was completely constant across the number of predicted scales. This is because we ignored Extra-P's execution cost, which was relatively small compared with the cost of data collection, as described in Section 4.3. By contrast, the cost of collecting the total execution time of function based on actual runs increased gradually as the number of predicted scales increased. In particular, in the case of collecting the total execution time of function at 27 scales, the predictive approach reduced the cost of data collection by 99% for EP compared with the approach of actually-running the application. Thus, the predictive approach was effective, particularly for collecting the total execution time of function at multiple scales.

5. Conclusions

In this study, we evaluated Extra-P in function

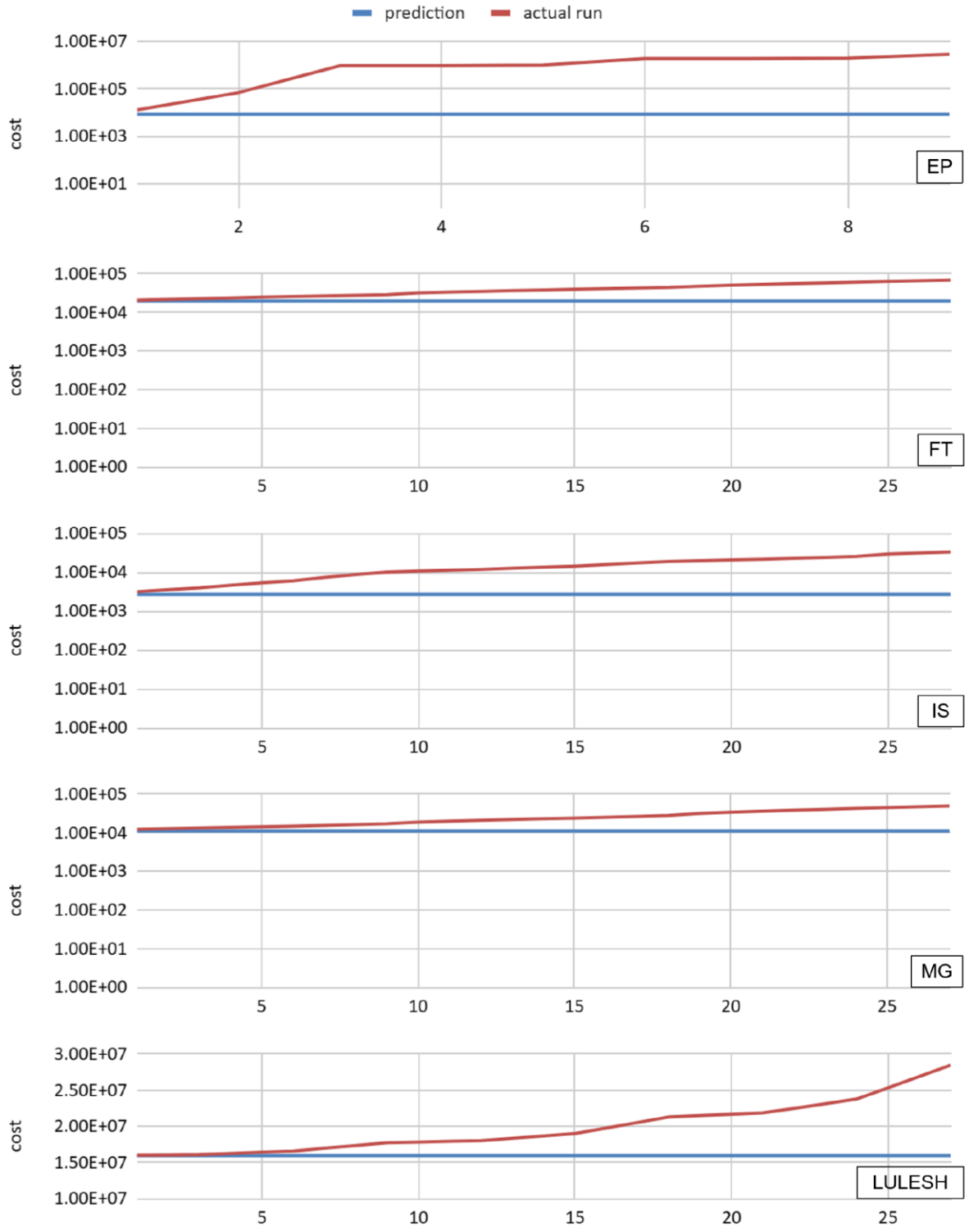


Figure 6. Cost of prediction of total execution time of functions

call count prediction. Our experimental results showed that Extra-P produced highly accurate scalability models for various functions in the case of function call count prediction. Additionally, we showed that the predicted function call counts were helpful for improving the prediction accuracy of the total execution time of functions. The predicted function call counts may be useful for the prediction of performance metrics, with the exception of the total execution time of functions.

In future work, we will extend the use case of function call count prediction to the prediction of other metrics.