Predicting Running Time of Grid Tasks based on CPU Load Predictions

Yuanyuan Zhang[#1], Wei Sun[#2], Yasushi Inoguchi[*3]

[#]*Graduate School of Information Science, Japan Advanced Institute of Science and Technology*
1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan
[1]yuanyuan@jaist.ac.jp
[2]sun-wei@jaist.ac.jp

[*]*Center for Information Science, Japan Advanced Institute of Science and Technology*
1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan
[3]inoguchi@jaist.ac.jp

***Abstract*— The ability to accurately predict task running time is of great importance for interactive applications and scheduling algorithms which need to determine how to use time-shared resources in a dynamic grid environment. In this paper we present and evaluate a new method to predict the running time of tasks in a grid. The prediction of task running time is based on a novel CPU load prediction method and is calculated from predictions of CPU load. We conducted evaluations using more than 10,000 randomized testcases run on load traces sampled from 39 heterogeneous machines. Our experimental results demonstrate that both our CPU load prediction method and task running time prediction strategy outperform significantly the widely used AR(16) load prediction model and the task running-time prediction method based on this model.**

I. Introduction

Grid computing [1], the internet-based infrastructure that aggregates geographically distributed and heterogeneous resources to solve large-scale problems, is becoming increasingly popular because it provides us the ability to dynamically link resources together as an ensemble, to support the execution of large-scale, resource-intensive, and distributed applications. Task scheduling of the applications is an important component for achieving high application performance in a grid computing environment. In grid there are often some interactive applications in which we need to schedule short tasks with deadlines on one of the available grid resources. If we could predict the running time of each task on each of the resources, we can easily select the appropriate resources. Therefore, task running time prediction can be useful for guiding scheduling strategies to achieve high application performance and efficient resource use. In this paper we propose a new strategy to predict running time for compute-bound tasks.

Grid is a non-dedicated environment. Such non-dedicated feature results in dynamic behavior. Every grid resource needs to execute its own tasks as well as tasks from grid applications. From the viewpoint of a grid application, all the tasks from both the grid resource and other grid applications are "load" on the resource. The grid application can obtain such load information from grid information service, and decide how and where to execute their tasks based on such information [2]-[5]. Since the execution time of a CPU-bound task on a host is tightly related with the CPU load it experiences while it is run on the host, we can make use of the CPU load prediction technique to predict the task running time. If we could predict the load on a host during the execution of a task, we could predict easily the execution time of the task on the host. However, in a grid environment, since applications compete for resources with unknown workloads from other users, resource contention causes host load and availability to vary over time, and makes the load prediction problem more difficult. In this paper first we introduce a new multi-steps-ahead CPU load prediction strategy we propose, and then we make task running time predictions by mathematic way based on such CPU load predictions. For the CPU load prediction, first we propose a new one-step-ahead load prediction method, then we predict CPU load several steps ahead by repeatedly using the one-step-ahead prediction strategy. Our one-step-ahead CPU load prediction strategy is a kind of tendency-based method. It predicts the load one step ahead based on the change tendency several steps backward, and uses a 2nd or 3rd order polynomial fitting method to determine the prediction value. We also conduct the prediction based on a search of previous similar "patterns".

Our experimental results on a commonly used host load measurement dataset show that our proposed task running time prediction strategy consistently outperforms the task running time prediction method which is based on AR(16) load prediction model.

The rest of the paper is structured as follows. Section 2 introduces the related work. Section 3 describes our CPU load prediction strategy in detail. Section 4 introduces the way to estimate the task running time using the CPU load predictions. Section 5 describes the results of experiments in which our prediction method was applied to realistic load measurements and compared with previous work. Finally in Section 6 we conclude the paper.

II. Related work

Both Smith et al. [6] and Gibbons [7] predict the running times of parallel applications based upon the running times of "similar" applications that have been executed in the past. Gibbons makes predictions by examining categories derived

from some templates until a category that can provide a valid prediction is found. This prediction is then used as the running time prediction. The novelty of [6] compared with [7] is that [6] applies search techniques such as greedy and genetic algorithm search techniques to determine the application characteristics that yield the best definition of similarity, to partition jobs into categories within which jobs are judged to be similar.

Downey [8] predicts the running times of parallel applications by categorizing all applications in the workload, and then he models the cumulative distribution functions of the running times in each category, and finally uses these functions to predict application running times.

Stochastic values represent a range of likely behavior and can be used as model parameters. Schopf and Berman [9] use stochastic values to parameterize performance models of applications. They use environment and application characteristics as parameters, such as bandwidth, available CPU, message size, operation counts, etc, to model the application performance and provide a prediction of such performance. Model parameters are represented by a distribution of possible values over a range, or in other words stochastic values. The running time prediction is then calculated by using the stochastic values of the model parameters and is represented as a distribution of running times. The deficiency of their technique is that accurate performance models of the applications are required.

Lee and Schopf [10] predict the running times of applications by using regression methods to establish the relationship between the actual running times of the past applications and the variables which affect the application running times. Only some subsets of past application history are used and filtering technique is used to select such subsets.

The difference between our work and the above papers is that they predict the running times of applications with long duration, while the applications we consider are those interactive ones, and what we predict here are the running times of the short compute-bound tasks in the applications, other than the running times of the applications themselves.

Dinda [11] predicts the task running times also based on CPU load predictions. His work and ours use the same method to calculate task running time from load predictions. However, the CPU load prediction strategy used in his work is the AR(16) linear time series model [12], while we use a different load prediction strategy we proposed. Moreover, in his work the predicted task running time is expressed as a confidence interval given some confidence specified by the application, while here we predict the task running time as a point value, other than a confidence interval.

III. CPU LOAD PREDICTION

This section describes our CPU load prediction strategy. Our multi-steps-ahead load prediction is based on the one-step-ahead load prediction strategy that we proposed [13]. In this section first we describe the one-step-ahead load prediction method, and then introduce the multi-steps-ahead load prediction strategy. Based on our study on the statistical properties of host load traces, our one-step-ahead prediction strategy predicts the next load value based on polynomial fitting method and "similar" patterns.

We use the following notations in the description of the prediction strategy:

V_T: the measured load at the T_{th} measurement.

P_{T+1}: the predicted load for the $(T+1)_{th}$ measurement.

N: number of historical data points used for the decision of P_{T+1}.

A. Polynomial Fitting

In scientific experiments, it is often necessary to disclose the relationship between the independent variable x and the dependent variable y from a set of experimentally observed data (x_i, y_i), i = 0, 1, ···. Such a relationship usually can be approximately expressed by the function: y = f(x). One method to produce function f(x) is a Least Squares Polynomial Fitting, which can be expressed as follows:

Given a discrete sampling of N data points D_1, D_2, ···, D_N which have coordinates

$$D_i = (x_i, y_i) \qquad i = 1,2,\cdots,N, \tag{1}$$

it is assumed that the value of y can be correlated to the value of x via an approximate function f with the expression:

$$f(x) = A_n x_n + A_{n-1} x_{n-1} + \cdots + A_1 x + A_0, \tag{2}$$

which corresponds to a nth order polynomial expansion. The expansion coefficients A_i are determined by least-squares fitting the data points to this expression. The resulting continuous function may then be used to estimate the value of y over the entire region of x where the approximation has been applied.

B. One-step-ahead Prediction Strategy

1) Prediction based on Polynomial Fitting: Our one-step-ahead load prediction strategy is a kind of tendency-based method because we predict based on the increases or decreases of several previous measurements. The method we use to predict the increment or decrement for next step is based on multi-order polynomial fitting method. To determine the order of the polynomial function, i.e., the number of data points to be used for the polynomial fitting, we have studied the regulation that the CPU load traces vary and run a set of experiments. From our observations we found that 2nd or 3rd order polynomial fitting achieves much better (and the best) fitting effect than the 1st order (linear) fitting does when a load trace varies smoothly and monotonously; therefore in our prediction we try both 2nd and 3rd order polynomial fitting and choose the one that is closer to the current data as the predicted value for the next load measurement. The detail is that when the last 3 (V_{T-2}, V_{T-1}, V_T) or 4 (V_{T-3}, V_{T-2}, V_{T-1}, V_T) load measurements increase or decrease monotonously, we use these measurements to produce the 2nd or 3rd order polynomial fitting function and then use the function value at the next time point as P_{T+1}.

2) *Prediction based on Similar Patterns:* The polynomial fitting prediction method achieves satisfying performance when the load traces vary smoothly and monotonously,

however, our observations show that such fitting does not work well for predicting a "turning point", the time when a time series changes its "direction", (that is, the point at which a time series begins to decrease after a number of successive increases, or starts to increase after some successive decreases), or when a "turning point" is used as one data point to fit the polynomial function.

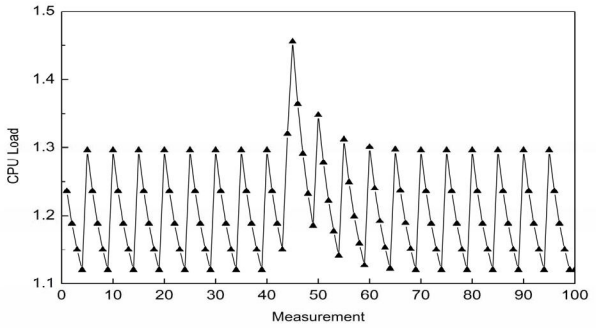

Fig. 1 There are many similar "patterns" in the load traces

To deal with this problem, we predict the load at a possible turning point based on the information of previous similar "patterns". Our observation on the load traces shows that there are many successive and similar "patterns" occurring in the time series. As shown in Fig. 1, such a pattern occurs repeatedly: the time series decreases successively for several times and then one turning point happens, after this turning point the series decreases successively again, and after several times another turning point is reached. We call the successive increases or decreases between two neighbouring turning points a "pattern", and two patterns with same number of data points are thought to be "similar". Because of this observation, we can predict the value of a possible turning point based on the information of previous similar patterns: if the point to be predicted comes after several (at least 3) successive decreases (the first point of these successive decreases is a turning point), while same number of successive decreases can be seen before this series of decreasing points, then we assume that the current point to be predicted is quite possibly a turning point and these two successively decreasing series are similar "patterns", so we use the value of last turning point as the prediction value for the current point, P_{T+1}. If we can't find such similar patterns we will still use the polynomial fitting method to predict this point.

For the prediction of a point following a turning point or two steps after a turning point, a polynomial fitting based prediction will also result in a large error because its prediction involves the use of the turning point, so we predict such points also based on information of similar patterns: we use the increment for the point after the previous turning point (or accordingly two steps after last turning point) as the increment for the current point.

To predict a point after several (at least 3) successive increases, if we can find a successively increasing pattern immediately before this series of increasing points, we use the information of this pattern to judge if current point is a turning point or not and then predict: if the number of points in this pattern has arrived at the same number as that of the last pattern, then we think that next point will be a turning point and use the measurement of last turning point as P_{T+1}; if the number here is less than that of last pattern, then we use polynomial fitting to predict next value; if we can't find a successively increasing series we will predict using a "conservative" strategy: we use V_T as P_{T+1}.

For the other cases we also choose a conservative prediction strategy that set the increment (decrement) between V_T and P_{T+1} as 0.

Formally, our proposed prediction strategy can be expressed as follows:

TABLE I
ONE-STEP-AHEAD PREDICTION STRATEGY

Algorithm Predict (V_1, V_2, ···, V_T)
Input: CPU Load Trace History (V_1, V_2, ···, V_T)
Output: P_{T+1}, Prediction Value of V_{T+1}

Begin
If ($V_T < V_{T-1} < V_{T-2} < V_{T-3}$) {
If ($V_{T-3} < V_{T-4} < V_{T-5}$, V_{T-5} is a turning point, and last five points before V_{T-5} also decrease successively)
$P_{T+1} = V_{T-5}$
Else Predict P_{T+1} using polynomial fitting method
}
If (V_T is a turning point)
Search for last turning point before V_T. Use the increment of that measurement as the increment to V_T for predicting P_{T+1}
If (V_{T-1} is a turning point) {
If (last pattern changes direction after 5 points)
Use the increment of the point two steps following last turning point as the increment to V_T
Else $P_{T+1} = V_T$
}
If ($V_T > V_{T-1} > V_{T-2} > V_{T-3}$) {
If ($V_{T-3} > V_{T-4}$) $P_{T+1} = V_T$
Else $P_{T+1} = V_T - (V_T - V_{T-1})$
}
For all the other cases $P_{T+1} = V_T$
End

3) *Multi-steps-ahead Prediction Strategy:* The process of multi-steps-ahead CPU load prediction based on the above one-step-ahead prediction strategy is as follows: we predict the multi-steps-ahead CPU load repeatedly using the one-step-ahead prediction, that is, when we predict the ith-step ahead load P_{T+i} ($i > 0$), we predict P_{T+1}, P_{T+2}, …, P_{T+i-1} one by one using the one-step-ahead prediction strategy, then we use the historical load trace and all the predicted load values before P_{T+i} to predict P_{T+i}, using the one-step-ahead prediction strategy.

IV. PREDICTION OF TASK RUNNING TIME

A. Continuous-time Task Running Time Estimation

The problem of task running time prediction is that, given the CPU load prediction history on a machine, and the nominal time t_{nom}, the execution time of a task on an unloaded machine, we want to predict the expected execution time of the task on the loaded host, t_{exp}. The algorithm we use to predict the running time of grid tasks is the one proposed by Dinda [11].

Indeed, t_{exp} is the time when the available time on a host arrives at its nominal time, t_{nom}, that is, given the available time function at(t) (t > 0), we have:

$$at(t_{\exp}) = t_{nom}. \qquad (3)$$

The load we discuss here is the number of processes that are running or ready to run, therefore the available CPU time until time t, at(t) (t > 0), can be expressed as:

$$at(t) = \frac{t}{1 + al(t)} \qquad t > 0. \qquad (4)$$

Here al(t) is the average load function until future time t, which is the average value of the load signal V(t),

$$al(t) = \frac{1}{t}\int_0^t V(\tau)\,d\tau \qquad t > 0. \qquad (5)$$

V(0) is the current load. The available time decreases as the increase of the average load, accordingly the execution time of the task increases.

B. Discrete-time Task Running Time Estimation

If we can predict the future CPU load V(t), we can easily calculate the expected task running time using the above equations, however, since the load history information we obtain is not a continuous-time signal, but a discrete approximation of the continuous-time signal, the future load value we can predict is also discrete signal, therefore we must discretize the above equations to make use of the discrete historical load measurements. Suppose the sample interval is Δ, the available time until the ith sample (iΔth second) in the future can be expressed as:

$$at_i = \frac{i\Delta}{1 + al_i} \qquad i > 0, \qquad (6)$$

where al_i is the average CPU load until the ith sample, which can be calculated using the following equation:

$$al_i = \frac{1}{i}\sum_{j=1}^{i} V_{T+j} \qquad i > 0, \qquad (7)$$

where V_{T+j} is the load value at the ith sample into the future. at(t), the continuous-time signal which represents the available time until t then can be estimated using linear interpolation:

$$at(t) = at_{\lfloor t/\Delta \rfloor} + \frac{t - \lfloor t/\Delta \rfloor}{\Delta}(at_{\lceil t/\Delta \rceil} - at_{\lfloor t/\Delta \rfloor}). \qquad (8)$$

C. Predict Task Running Time using CPU Load Prediction

To use the CPU load prediction strategy we introduce in Section 3 to predict the task running time, we substitute the predicted load signal P_{T+j} for V_{T+j} so that we obtain the predicted value for al_i using (7), and then we calculate the predicted discrete-time available time using (6) and its corresponding continuous-time approximation using (8).

V. Experimental Evaluations

A. Experimental Environment

We ran a series of experiments using our running time prediction strategy in order to evaluate its effectiveness under load traces with a variety of statistical properties. In total we used 39 load traces which are the load time series sets on a Unix system and were collected by Dinda [14]. The load on a Unix system at any time is the number of processes that are running or ready to run. The kernel samples the number at some period and exponentially averages some previous samples to produce a load average. To satisfy the Nyquist criterion, Dinda chose a 1 HZ sample rate and exponentially averaged with a time constant of five seconds. This captures all of the dynamics made available to us by the operating system.

The 39 load traces were sampled from heterogeneous machines which cover a broad range including interactive machines, batch machines, compute servers, a testbed, and desktop workstations. The mean and standard deviation (SD) of the 39 traces are shown in Fig. 2. Here the captions of the horizontal axis represent the names of the traces. From the figure we can see that the statistical properties of these traces are significantly different. Some traces are with high mean and high SD; some are with high mean while low SD compared with the mean; while some are with low mean but high SD compared with the mean.

B. Evaluation results for multi-steps-ahead CPU load prediction

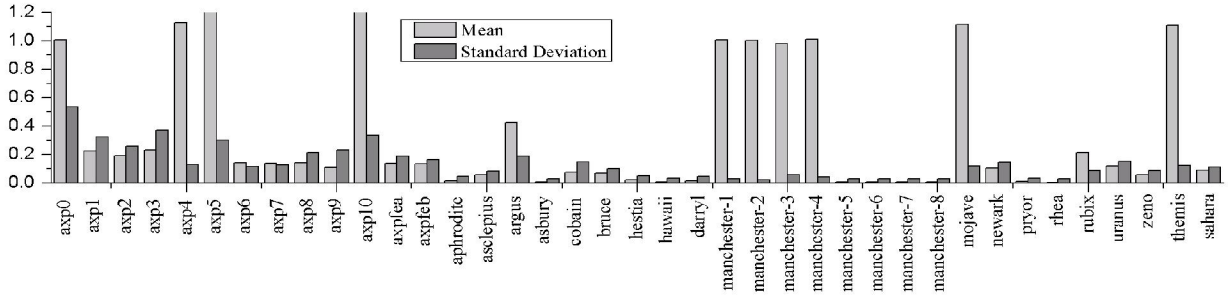

Fig. 2: Mean and standard deviation of the 39 load traces

Before we evaluate our task running time prediction strategy, we firstly ran some experiments using the 39 load traces to validate the effectiveness of our multi-steps-ahead CPU load prediction method because it is the basis for the task running time prediction. If we can predict the CPU load accurately, we are sure that we can also provide satisfying task running time prediction because only some mathematic calculation is needed to obtain the task running time prediction from CPU load prediction.

The evaluation results for one to five steps ahead CPU load predictions using our proposed strategy and AR(16) model are shown respectively in Fig. 3 to Fig. 7. We only show results for one to five steps because of the limitation of space, and also, because what we consider are the short tasks. The vertical axis in each of the figures represents the mean of prediction errors for a given load trace. The prediction error for a measurement is the ratio between the absolute value of prediction error (the difference between a predicted value and the measurement) and the measurement. The mean of prediction errors is the average error ratio for the prediction error ratios of all the data in a time series.

In all of the figures from Fig. 3 to Fig. 7, our proposed CPU load prediction strategy predicts much more accurately than the AR(16) model. We don't show the exact number of reduction ratio of prediction error because the effect is clearly shown in the figures. For each load prediction strategy and each load trace, the average prediction error increases gradually from one step to five steps. This is because that although as Dinda has studied [15], the CPU load reveals the characteristic of long-rang dependence, as time lasts, the dependence between load measurements becomes weaker, therefore it's more difficult to predict the load values in the farther future.

For each step or in each figure, the average prediction errors on the 39 traces are much different using any of the two load prediction systems. For example, in Fig. 3, the average prediction errors for some traces such as *asclepius* and *bruce* are higher than 16%, while for some other traces such as *manchester-1* and *manchester-2* the average prediction errors are even lower than 1%. Compared with Fig. 2, we can see that the absolute values of mean and SD of the load traces corresponding to high or low prediction errors can be large or small, but a more detailed observation tells us some rule: the load traces with high prediction errors are those whose relative SD, the ratio between SD and Mean, is high, while the load traces with small relative SD result in low prediction errors. For example, both the mean and SD of trace *asclepius* are not so high compared with other traces, but its SD is even larger than its mean. This tells us that no matter the load on a host is heavy or light, it is more difficult to predict a host with high dynamicity.

C. Evaluation Results for Task Running Time Prediction

To evaluate the effectiveness of our task running time prediction strategy given a particular trace, we compared it with Dinda's method which is based on AR(16) load prediction model. For a given load trace, the following process was executed repeatedly for 300 times:

- Choose a random value between 100ms and 10 seconds from a uniform distribution as the nominal time of a task, t_{nom}.
- Select a CPU load prediction system from our method and AR(16) model.
- Calculate the expected task running time t_{exp} using load predictions derived from the above selected load prediction system.
- Calculate the actual task running time t_{act} on the host with the load measurements in the trace.
- Calculate the prediction error for the selected load prediction system.

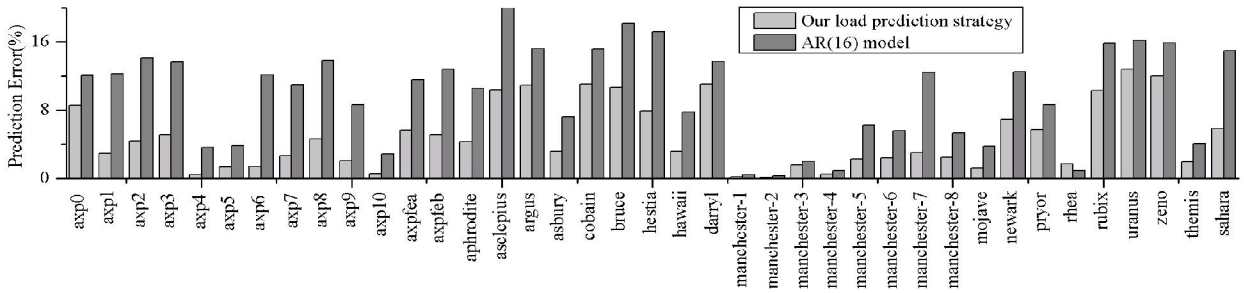

Fig. 3: One-step-ahead load prediction errors on the 39 traces

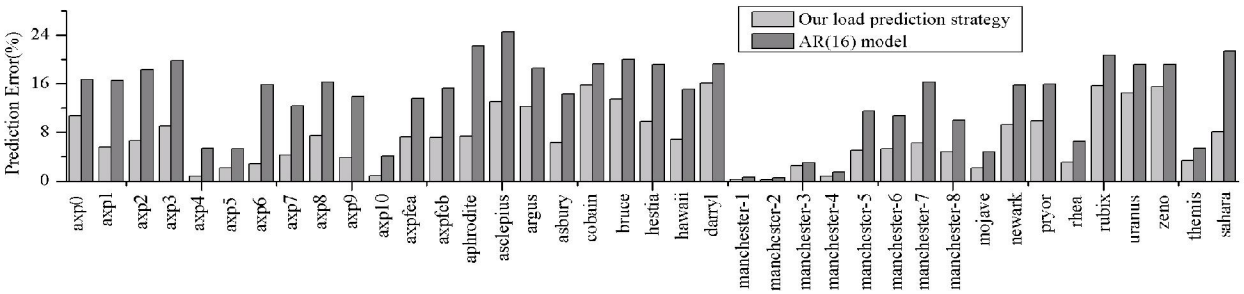

Fig. 4: Two-steps-ahead load prediction errors on the 39 traces

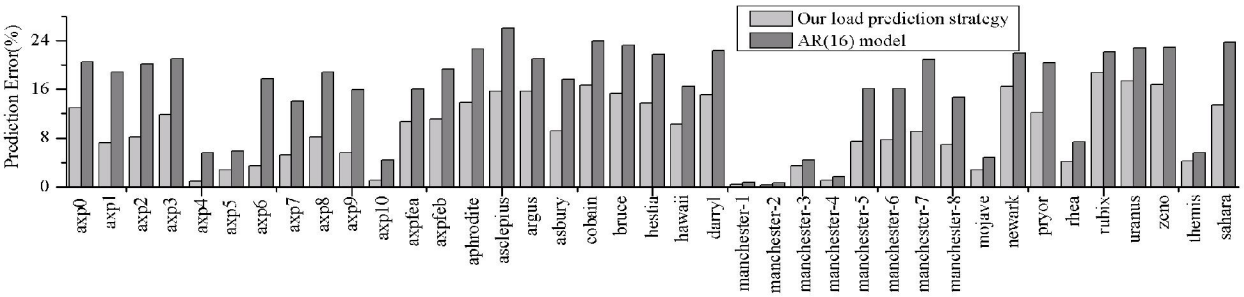

Fig. 5: Three-steps-ahead load prediction errors on the 39 traces

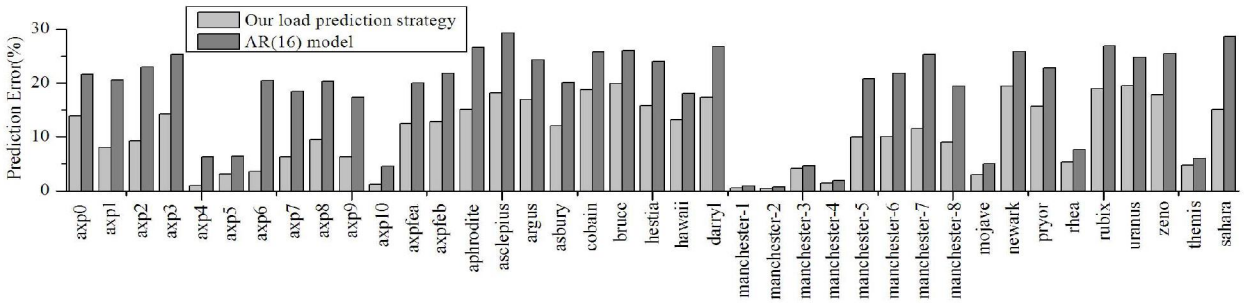

Fig. 6: Four-steps-ahead load prediction errors on the 39 traces

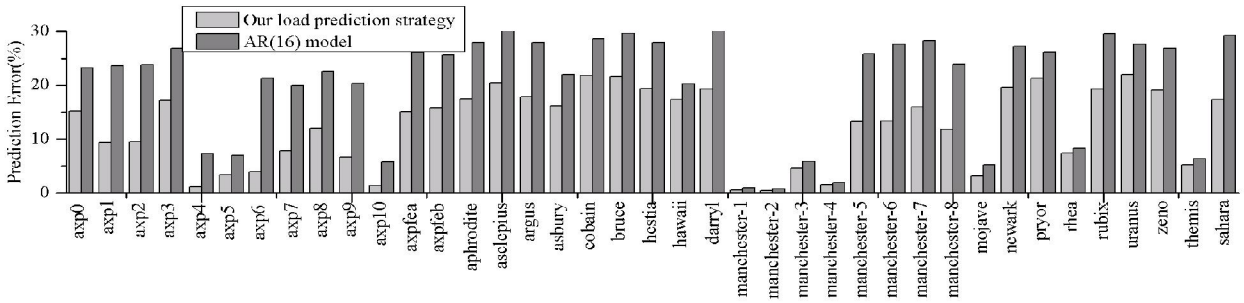

Fig. 7: Five-steps-ahead load prediction errors on the 39 trace

After all of the 300 testcases were finished, we calculated the average prediction error among these 300 testcases for a load prediction system with a given load trace. Here the prediction error for a nominal time t_{nom} is the ratio between the absolute value of prediction error (the difference between t_{exp} and t_{act}) and t_{act}.

The evaluation result, using our prediction strategy to predict the task running time given the 39 load traces, is shown in Fig. 8. We can see that our running time prediction strategy based on our load prediction method always outperforms the prediction based on AR(16) load prediction model. Moreover, for each of the two running time prediction methods, among the 39 traces, the traces with low CPU load prediction errors always correspond to low task running time prediction error, or vice versa. This is easy to understand because both of the two running time prediction strategies get their predictions from the calculation from load predictions. What encourages us very much is that compared with the load prediction, the prediction error of task running time prediction is much lower for all of the 39 traces. The prediction error of task running time is quite low even for traces with quite high CPU load prediction error. This reveals the effectiveness of the idea to use CPU load prediction for task running time prediction.

VI. Conclusions

Predictions of task running time can be used to improve resource selection and scheduling in a dynamic grid environment. In this paper we propose and evaluate a new task running time prediction strategy which conducts the prediction based on the calculation from multi-steps-ahead CPU load predictions. The results of experiments that we conducted demonstrate that this new prediction strategy outperforms the task running time prediction method using AR(16) load prediction model. Moreover, our experimental results also show that for any load trace, the prediction error of task running time prediction is much lower than that of CPU load prediction. This clearly proves the effectiveness of task running time strategies based on CPU load predictions.

Acknowledgement

This research is conducted as a program for the "21st Century COE Program" by Ministry of Education, Culture, Sports, Science and Technology, Japan.

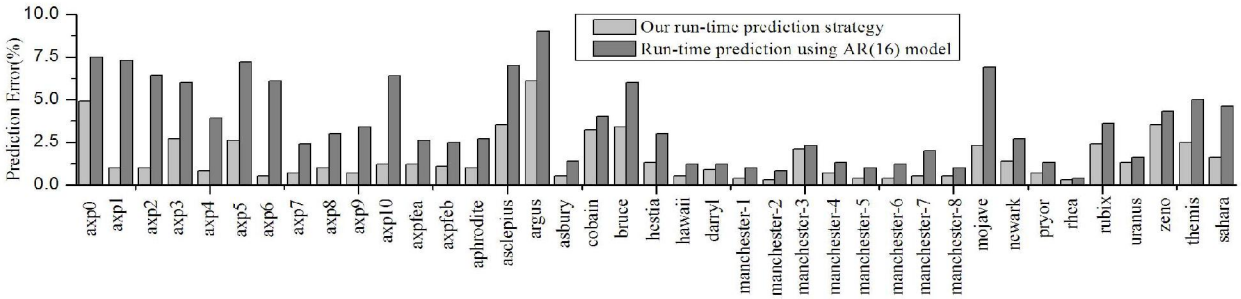

Fig. 8: Running-time prediction errors on the 39 traces

REFERENCES

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Fransisco, CA, 1999.

[2] L. Yang, J. M. Schopf, and I. Foster, "Conservative scheduling: using predicted variance to improve decisions in dynamic environment," in *Supercomputing'03*, pp. 1-16, 2003.

[3] P. A. Dinda, "A prediction-based real-time scheduling advisor," in *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS 2002)*, pp. 35-42, 2002.

[4] D. Lu, H. Sheng, and P. Dinda, "Size-based scheduling policies with inaccurate scheduling information," *12th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, pp. 31-38, 2004.

[5] S. Jang, X. Wu, and V. Taylor, "Using performance prediction to allocate grid resources," Technical report, GriPhyN 2004-25, pp. 1-11, 2004.

[6] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times with Historical Information," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1007-1016, Sept. 2004.

[7] R. Gibbons, "A Historical Application Profiler for Use by Parallel Scheduler," *Lecture Notes on Comput. Science*, vol. 1297, Springer, Berlin, pp. 58-75, 1997.

[8] A. Downey, "Predicting queue times on space-sharing parallel computers," *International Parallel Processing Symp.*, pp. 209-218, Apr. 1997.

[9] J. Schopf and F. Berman, "Using Stochastic Information to Predict Application Behavior on Contended Resources," *Int'l J. Foundations of Computer Science*, vol. 12, no. 3, pp. 341-363, 2001.

[10] B. D. Lee and J. M. Schopf, "Run-Time Prediction of Parallel Application on Shared Environment," *Cluster 2003*, pp. 1-19, Sept. 2003.

[11] P. A. Dinda, "Online Prediction of the Running Time of Tasks," *Journal of Cluster Computing*, vol. 5, no. 3, pp. 225-236, July 2002.

[12] P. A. Dinda and D. R. O'Hallaron, "Host Load Prediction using Linear Models," *Journal of Cluster Computing*, vol. 3, no. 4, pp. 265-280, 2000.

[13] Y. Zhang, W. Sun, and Y. Inoguchi, "CPU Load Predictions on the Computational Grid," *IEEE, Proc. in 6th International Conference on Cluster Computing and the Grid (CCGrid06)*, pp. 321-326, May 2006.

[14] http://www.cs.cmu.edu/~pdinda/LoadTraces/.

[15] P. A. Dinda and D. R. O'Hallaron, "The statistical properties of host load," *Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR 98)*, pp. 1-23, 1998.