# Predict task running time in grid environments based on CPU load predictions

Yuanyuan Zhang [a] Wei Sun [a,*] Yasushi Inoguchi [b]

[a] *School of Information Science, JAIST, Nomi, Ishikawa, 923-1292 Japan*
[b] *Center of Information Science, JAIST, Nomi, Ishikawa, 923-1292 Japan*

**Abstract**

A good running time prediction of tasks is very helpful and important for job scheduling and resource management of Grid. In this paper we present a running time predicting method for Grid tasks based on our previous work, which is a novel CPU load prediction method. In order to eliminate the interference of other factors, such as the memory accessing, network performance, and fluctuation of CPU processing capacity and so on, we produce a simulation to test and evaluate our prediction method. In this simulation we use more than 10,000 randomized test cases run on load traces sampled from 39 different machines. The simulation results are excellent and demonstrate that our running time prediction of Grid tasks outperforms significantly a widely existing prediction method.

*Key words:* Grid, task, running time prediction, polynomial fitting, pattern

## 1. Introduction

Grid computing [6] is a hotspot in recent research field which derives its name from the analogy with the electricity grid. A basic and commonly used definition of grid is the internet-based infrastructure that aggregates geographically distributed and heterogeneous resources as an ensemble to solve large-scale problems.

The information of running time prediction for tasks in a grid application is very helpful and important for both job scheduling and resource management of Grid because most of such work involves predicting the performance of the tasks on the prospective resources. Such prediction information can be derived in two ways: application-oriented and resource-oriented. For the application-oriented approaches, they predict the running-time of grid tasks directly using information about the application, such as the running-time of previous similar tasks; while for the resource-oriented approaches, they predict the future performance of a resource such as CPU load and availability at first, using the available information about the resource, and then such predictions of resource performance are used to predict the running time of a task given the information of the task's resource requirement.

In this paper what we discuss are the interactive applications in grid. Each of the short tasks with deadlines in one of such applications is needed to be scheduled on one of the available grid resources. If we could predict the running time of the task on each of the resources, we can easily select the appropriate resource. In this paper we introduce a strategy to predict the running time for computation-intensive tasks. Our approach is resource-oriented: we predict the expected task running time based on the predic-

* Corresponding author.
*Email address:* sun-wei@jaist.ac.jp (Wei Sun).

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



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

tions of the future performance of the resources. In detail, such performance parameter is the CPU load on the resources. The precondition of our task execution time prediction is the task's resource requirement, which is expressed by the nominal execution time of the task on an otherwise vacant machine.

Grid computing is a dynamic environment because users who have competing goals with each other share resources, resources are upgraded, fail, join or leave grid system dynamically, and so on. Also, such dynamicity comes from the fact that every grid resource needs to execute its own tasks as well as tasks from grid applications. Since for application scheduling, everything in the system is evaluated by its influence on the application, from the viewpoint of a grid application, all the tasks from both a 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 [3], [10], [16]. Since the execution time of a computation-intensive 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 information about CPU load 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, 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 CPU load prediction strategy we proposed, and then we predict task running time by a mathematic way based on such CPU load predictions. For the CPU load prediction, first we introduce a new one-step-ahead load prediction method, and then we predict multi-steps-ahead CPU load 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 one-step-ahead load based on the increase or decrease tendency of the historical load data, and uses a 2nd or 3rd order polynomial fitting method to produce the prediction value. To make up for the deficiency of polynomial fitting, we also make prediction based on a search of previous similar "patterns".

To implement the running time prediction of tasks is very difficult to realize in a real environment and to compare with previous work, so we evaluate the performance of our strategy in simulation environment and compare it with an existing method to eliminate the interference of other factors, such as the memory accessing, network performance, and fluctuation of CPU processing capacity and so on.

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

The paper structure is as follows. In Section 2, we discuss related work. In Section 3, we describe our CPU load prediction strategy in detail. In Section 4, we introduce the approach to estimate the task running time making use of the information of the CPU load predictions. Section 5 describes the evaluation results in which our prediction method was applied to realistic load dataset and compared with previous work. Finally in Section 6 we conclude the paper.

## 2. Related work

As we have introduced before, the approaches to predict task execution time can be divided into two categories: resource-oriented and application-oriented. For the existing running time prediction strategies, [5], [7], [9], [11], [12] belong to application-oriented approaches, while [1] is resource-oriented.

Both Smith et al. [12] 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 time. 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 [12] is that [12] applies search techniques like 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 [5] predicts the running times of parallel applications by categorizing all applications in the workload, and it then models the cumulative distribution functions of the running times in each category, and finally uses these functions to predict application running times.

Schopf and Berman [11] use stochastic values to parameterize performance models of applications. They use environment and application characteristics as parameters, such as bandwidth, available



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

CPU, message size, operation counts, etc, to model the application performance and provide a prediction of such performance. Model parameters are represented by stochastic values. They then calculate running time prediction by using the stochastic values of the model parameters and the prediction 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 [9] predict the running time of applications by using regression methods to establish the relationship between the actual running times of the past application run history and the variables which affect the application running times. Only some subsets of past history are used and filtering technique is used to select such subsets.

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

Dinda [1] predicts the task running times also based on CPU load predictions, that is, his work is also resource-oriented. Our work and his use the same method to calculate the task running time from load predictions. However, the CPU load prediction strategy used in his work is the AR(16) linear time series model [4], 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.

## 3. CPU load prediction

This section describes our CPU load prediction strategy. Our approach is to predict the one-step-ahead load at first, and then predict multi-steps-ahead load based on such information. In this section first we describe the one-step-ahead prediction method we have proposed [14], 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 load value one step into the future based on polynomial fitting method and "similar" patterns.

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

$V_T$: the load value at the Tth measurement.

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

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

### 3.1. *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, $\cdots$. 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 $D_1$, $D_2$, $\cdots$, $D_N$ with 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 an nth order polynomial expansion. The expansion coefficients Ai 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 x region where the approximation has been applied.

Polynomial models are a great tool for most engineering and manufacturing applications, hence polynomial fitting is sometimes used to estimate and predict the future data and their direction [8]. In most modern data analysis tools for example Matlab and Origin, the polynomial fitting is a basic function. In the modern computer science polynomial fitting has become a common approach for signal processing and data analysis [15].

### 3.2. *One-step-ahead prediction strategy*

#### 3.2.1. *Prediction based on polynomial fitting.*

The existing CPU load prediction methods mainly use the time series analysis models to predict



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

CPU load [4]. Using polynomial model to predict CPU load is a new approach in load prediction. Although it hasn't been demonstrated obviously, a method in [16] in fact sometimes uses the first-order polynomial model to predict CPU load for grid environment. However, the multi-order polynomial model hasn't been used to predict CPU load. In this research we are going to use the multi-order polynomial fitting to predict the CPU load.

Our one-step-ahead load prediction strategy is a kind of tendency-based method because we predict based on the increase or decrease tendency 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, we have studied the regulation that the CPU load traces vary. 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 the 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. In detail, 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 a 2nd or 3rd order polynomial fitting function and then use the function value at the next time point as $P_{T+1}$.

3.2.2. *Prediction based on similar patterns.*

The polynomial fitting prediction method achieves satisfying prediction effect when load traces vary smoothly and monotonously, however, 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. We also found that usually the load value at a turning point changes acutely, and therefore tendency-based strategy with polynomial fitting method will produce large prediction error for such cases.

To deal with this problem, we predict the load value at a possible turning point based on the information of previous similar "patterns". Our observation of the load traces shows that there are many

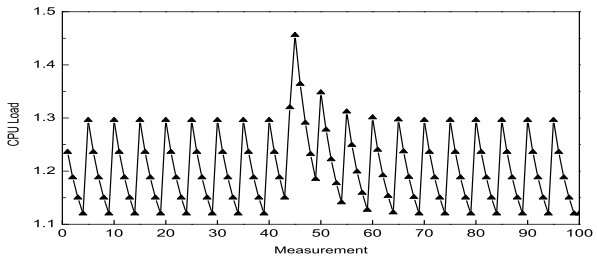


Fig. 1. Similar "patterns" in the load traces.

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 happens. We call the successive increases or decreases between two neighboring 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 previous similar pattern: 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. If we can't find such similar pattern 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 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 this 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 will 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 of the last pattern, then we think that next point will be a turning point and use the measurement of last turning point as



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

PT+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 sets the increment (decrement) between $V_T$ and $P_{T+1}$ at 0.

### 3.3. *Multi-steps-ahead prediction strategy*

The prediction of multi-steps-ahead CPU load based on the above one-step-ahead load prediction strategy is as follows: we predict the multi-steps-ahead CPU load by using the one-step-ahead prediction repeatedly, that is, when we predict the ith-step-ahead load $P_{T+i}$ (i > 0), we predict $P_{T+1}$, $P_{T+2}$, $\cdots$, $P_{T+i-1}$ 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.

## 4. Task running time prediction

### 4.1. *Continuous-time task running time prediction*

Task running time prediction is the problem that given the nominal running time $t_{nom}$, the execution time of a task on an unloaded machine, and the CPU load prediction history on a host, 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 [1].

In fact, the expected execution time of a task on a host is the time when the available CPU time on the host arrives at the nominal task execution time, that is, given the available time function at(t) (t > 0), we have:

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

The background of our research is a time shared environment. Because of this, the load we discuss here is the number of processes that are running or ready to run, therefore the available time until time t, at(t) (t > 0), can be expressed as:

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

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

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

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.

### 4.2. *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, therefore we need discrete version of the above equations to make use of the discrete historical load measurements. Suppose the sample interval is $\Delta$, the available time until the ith sample (the $i\Delta$th second) in the future can be expressed as:

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

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

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

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

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

### 4.3. *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 ali using (7), and then we calculate the predicted discrete-time available time using (6) and its corresponding continuous-time approximation using (8).



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

## 5. Simulation evaluations

### 5.1. *Simulation setup*

As we have introduced before, we evaluate the performance of our task running time prediction strategy in simulation environment and compare it with an existing method to eliminate the interference of other factors. The simulation is run on a workstation of Sun Blade 1500. We make a simulated predictor which embeds the above load prediction strategy and the algorithm to calculate task running time using the load predictions. The load traces we use are with a variety of statistical properties. In total we use 39 load traces which are derived from a Unix system and were collected by Dinda [17]. 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. 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.

Our simulation process is shown in Fig. 3. The inputs of the simulation are the data of load trace and the nominal execution time of a task, which is chosen randomly between 100 ms and 10 seconds from a uniform distribution. Given the inputs, the simulated predictor calculates the number of steps needed for CPU load prediction, and then it uses a load prediction strategy to determine these load prediction values. Such CPU load predictions are then used by the task running time algorithm to calculate the expected task execution time, which is finally outputted to a trace recording the task running time predictions.

### 5.2. *Evaluation results for multi-steps-ahead CPU load prediction*

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. 4 to Fig. 8. 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. 4 to Fig. 8, 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 [14], 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 16the 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.



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

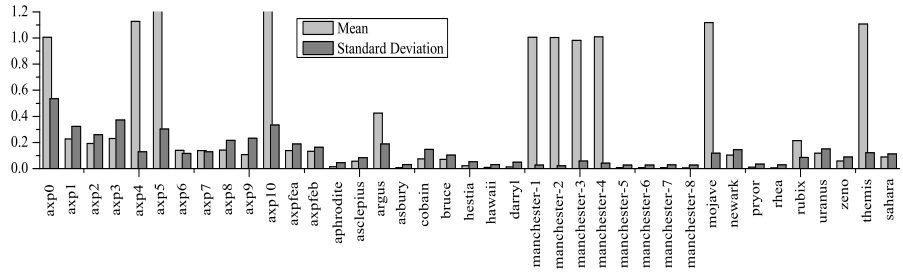


Fig. 2. Mean and standard deviation of the 39 load traces.

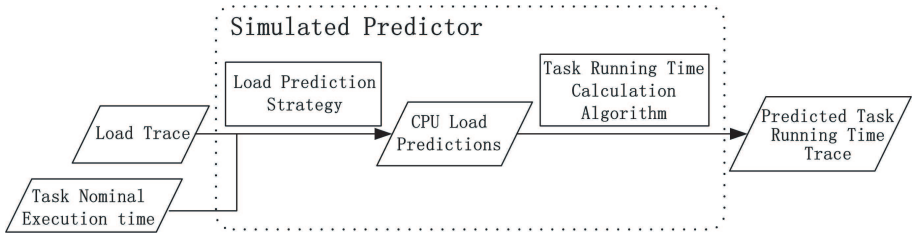


Fig. 3. Simulation process of the task running time prediction.

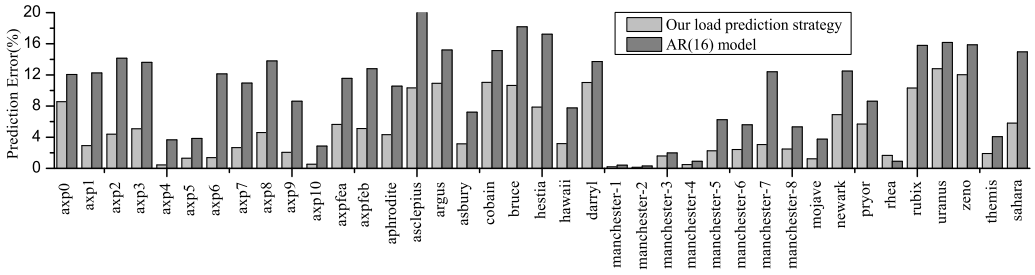


Fig. 4. One-step-ahead load prediction errors on the 39 traces.

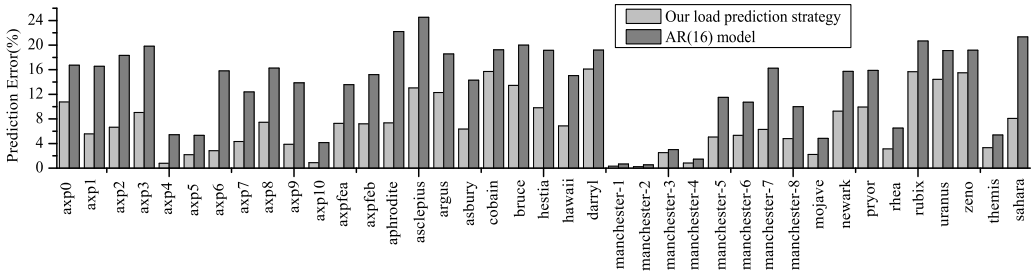


Fig. 5. Two-steps-ahead load prediction errors on the 39 traces.



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

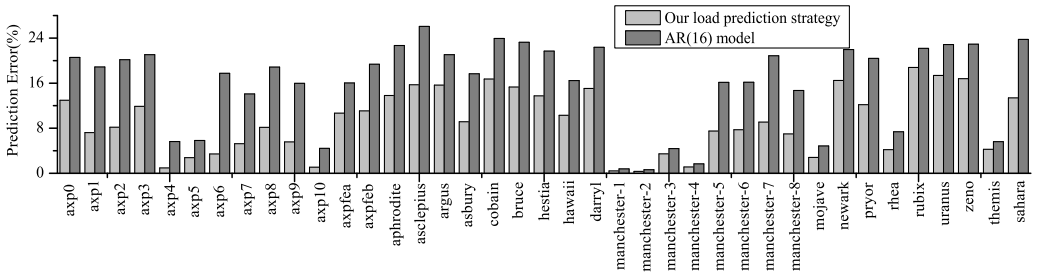


Fig. 6. Three-steps-ahead load prediction errors on the 39 traces.

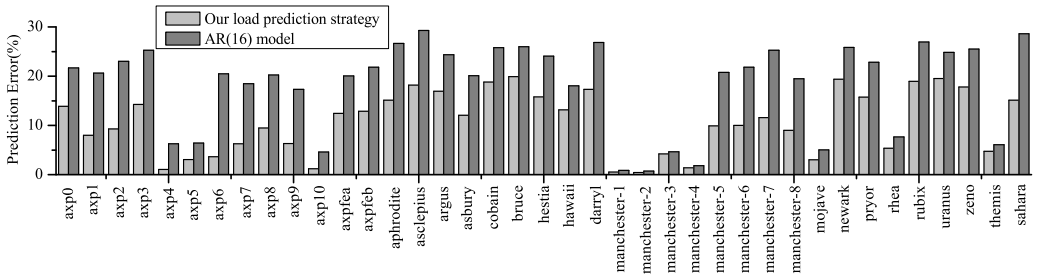


Fig. 7. Four-steps-ahead load prediction errors on the 39 traces.

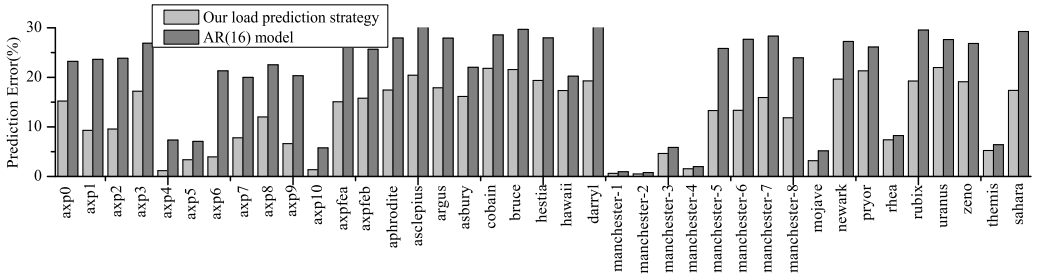


Fig. 8. Five-steps-ahead load prediction errors on the 39 trace.

### 5.3. *Evaluation results for task running time prediction*

We compare our task running time prediction strategy with Dinda's method which is based on AR(16) load prediction model. For a given load trace, we choose a task nominal execution time using the above method and then use one of our CPU load prediction strategy and Dinda's AR(16) model in the simulated prediction to predict the running time of the task. Then actual task running time can be calculated given the load trace. Finally the prediction error is calculated, which is the value derived from the absolute prediction error divided by the actual task running time, while absolute prediction error is the difference between the actual time and predicted time. Such process is executed repeatedly for 300 times for one combination of load trace, load prediction strategy. After all of the 300 testcases are finished, we calculate the average prediction error among these 300 testcases for a load prediction system with a given load trace.

#### 5.3.1. *Nominal time independent evaluation of quality of expected running times.*

This part evaluates how well the expected running time provided by the two running time prediction algorithms predicts the actual running time, encoun-



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

tered when running the task. The simulated predictor is a composite whose errors are due to inaccurate host load predictions as well as modeling errors in transforming from the host load predictions and task CPU demand to task running time. We are interested in how this composite performs as a whole.

In total we run 11,400 randomized testcases (300 testcases for each trace) for each prediction strategy and studied the results. The testcases are randomized with respect to their nominal running time (0.1 to 10 seconds). We evaluate the quality of prediction using average prediction error.

We evaluate the prediction quality of different traces, load prediction methods (ours and AR(16)), and nominal running time. The main conclusion is that our running time predictor does indeed produce high quality predictions for task running times. The nature of the effect depends on how heavily loaded the host is.

The evaluation result is shown in Fig. 9. We can see that our task running time prediction strategy based on our load prediction method always outperforms the prediction based on AR(16) load prediction model.

5.3.2. *Effect of nominal time on quality of predicted running times.*

The performance of the running time prediction depends on the nominal time of the task, $t_{nom}$. To illustrate this dependence, we use the same methodology as described in the previous section, but we classify the results based on $t_{nom}$. Recall that the range of $t_{nom}$ is from 0.1 to 10 seconds. In the previous part, for each testcase we produce a nominal time randomly from this entire range. In this part, we divide the range of $t_{nom}$ into three classes: 0.1 to 3 seconds ("small tasks"), 3 to 6 seconds ("medium tasks"), and 6 to 10 seconds ("large tasks"). Then, for each of these classes we conduct 300 testcases for each of the 39 load traces.

Figure 10 to 12 illustrate how the average prediction error value measured for each of the load traces depends on the nominal time. Here we only show the result of our prediction strategy because the running time prediction based on AR(16) model behaves similarly.

As can be seen from the figures, prediction error values increase gradually as tasks grow in size. For small tasks (Figure 10), for all of the traces generally the average prediction error is lower that when the task size is uniformly selected between 0.1 and 10, while for large tasks (Figure 11), usually for each of the traces the average prediction error is higher than that when the task size is between 0.1 and 10. The reason for such phenomenon is that as the task size grows, we need to predict CPU load longer in the future. As the increase of average prediction error for the multi-step-ahead load prediction, the prediction error of task running time also increases.

## 6. Conclusion

In this paper we propose a task running time prediction method, which is based on the calculation from multi-steps-ahead CPU load predictions, and a simulation by which our prediction method is tested and evaluated. The simulation uses the real load traces sampled from 39 different machines and be designed to purely show the advantage of our prediction method over the other one. The simulation results prove that our method based on CPU load prediction is effective, our prediction error is lower than the other one, and the prediction error of task running time prediction is much lower than that of CPU load prediction.

## References


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

[2] 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.

[3] P.A. Dinda, "A prediction-based real-time scheduling advisor," Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS 2002), pp.1-8, 2002.

[4] P.A. Dinda, D.R. O'Hallaron, "Host load prediction using linear models," Cluster Computing, vol. 3, no. 4, pp. 265-280, 2000.

[5] A. Downey, "Predicting Queue Times on Space-sharing Parallel Computers," International Parallel Processing Symposium, pp. 209-218, Apr. 1997.

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

[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] http://www.itl.nist.gov/div898/handbook/

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

[10] D. Lu, H. Sheng, and P. Dinda, "Size-based scheduling policies with inaccurate scheduling information,"




This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

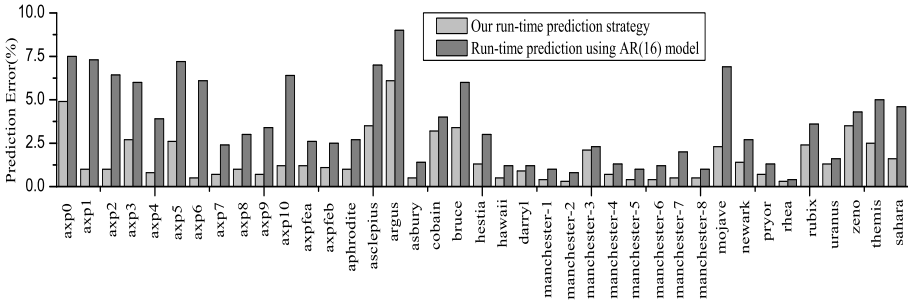


Fig. 9. Running-time prediction errors on the 39 traces.

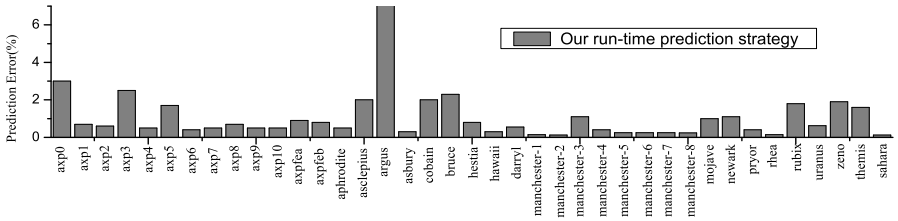


Fig. 10. Effect of nomial time on prediction error, 39 traces. 3 to 6 second tasks.

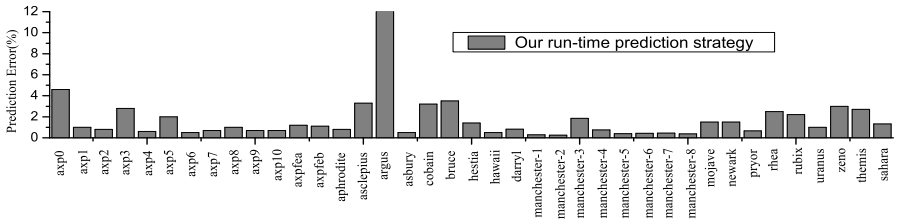


Fig. 11. Effect of nomial time on prediction error, 39 traces. 3 to 6 second tasks.

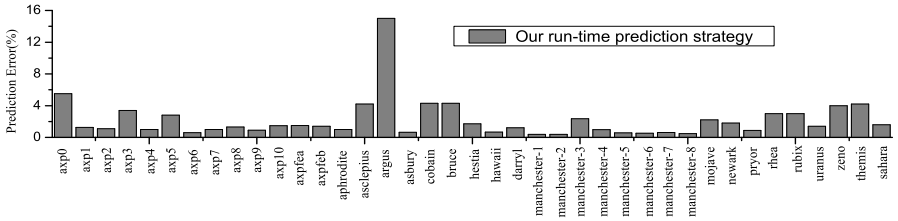


Fig. 12. Effect of nomial time on prediction error, 39 traces. 3 to 6 second tasks.

12th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems(MASCOTS'04), pp. 31-38, 2004.

[11] 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.

[12] 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, Sep. 2004.

[13] L. Yang, J.M. Schopf, and I. Foster, "Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environment," Proc. of the ACM/IEEE SC2003 Conf. on High Performance Networking and Computing, pp.1-16, 2003.

[14] 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, 2006.

[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, "Introduction to Algorithms", Second Edition, The MIT



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.

Press, 2001.
[16] L. Yang, I. Foster, and J. Schopf, "Homeostatic and Tendency-based CPU Load Prediction," Int'l Parallel and Distributed Processing Symp. (IPDPS 2003), pp. 42-50, 2003.
[17] http://www.cs.cmu.edu/ pdinda/LoadTraces/



This is the author's version of an article published in Future Generation Computer Systems Volume 24, Issue 6, June 2008, Pages 489-497, http://dx.doi.org/10.1016/j.future.2007.07.003.