poster-paper, in Proceedings of Cluster 2003, December 2003.

Run-Time Prediction of Parallel Applications on Shared Environments

Byoung-Dai Lee
blee@cs.umn.edu
Dept. of Computer Science and Engineering
University of Minnesota, Twin Cities, MN

Jennifer M. Schopf
jms@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL

Abstract

Application run-time information is a fundamental component in application and job scheduling. However, accurate predictions of run times are difficult to achieve for parallel applications running in shared environments where resource capacities can change dynamically over time. In this paper, we propose a run-time prediction technique for parallel applications that uses regression methods and filtering techniques to derive the application execution time without using standard performance models. The experimental results show that our use of regression models delivers tolerable prediction accuracy and that we can improve the accuracy dramatically by using appropriate filters.

1. Introduction

Application run-time information is needed in most application and job-scheduling approaches for parallel and distributed systems, as well as in resource selection in Grid computing environments ([5][9][11][12]). However, accurate predictions of application run times are difficult to achieve, especially for parallel applications running in shared environments, where resource capacities (e.g., CPU load, bandwidth, latency) can change dynamically over time and poor predictions can dramatically affect the performance of the scheduler. In order to achieve accurate predictions of application run-times, many conventional approaches used in scheduling research assume that accurate performance models of the applications are available (which can be costly or even impossible to achieve) or that the applications execute only on space-shared resources where no two processes can run simultaneously.

In this paper, we propose a run-time prediction technique for parallel applications running in shared environments. Our approach derives application execution times without using performance models. Instead, it discovers the relationship between variables that affect the run times of the application (e.g., the input to the application, resource capacities) and the actual run times from the past application run history.

Our approach uses regression methods and a filtering technique: regression methods are applied only to *subsets* of past history (evaluated by using filters) in order to discover the relationship. Our work assumes very limited performance information (similar to Kapadia et al. [1] and Lee and Weissman [3]) and yet functions in a dynamic environment (as seen in Parashar and Hariri [6], Schopf and Berman [8], and Taylor et al.[10]). We evaluate the performance using two parallel applications: an N-body simulation code and a heat distribution code. The experimental results show that the use of regression methods delivers tolerable prediction accuracy and that we can improve the accuracy dramatically by using appropriate filters.

2. Run-Time Prediction Using Regression Methods

Predicting the execution times based on past application run history without using accurate performance models raises several significant issues. To make the prediction problem tractable, we consider the following constraints (detailed further in [2]).

- The set of application input parameters that can affect the application run-time is known. It is unknown, however, to what extent the input parameters affect the run time; we assume only that this set should be tracked.
- We do not consider parallel applications with run times that are nondeterministic or that depend on the distributions of the input data.
- The parallel applications are not instrumented, so the applications need not be modified.
- Every application executes on the same resource set. In other words, we do not predict how long an application that ran on a system X will now take on a system Y.

Figure 1 shows the steps we use to predict the runtime of a parallel application run. The set of information needed for a given run we call a *query point*. This includes the input to the application run, the number of processors to use for the run, and the current status of the processors and links connecting the processors.

When each application run finishes, the following information is recorded: (1) the input to the application

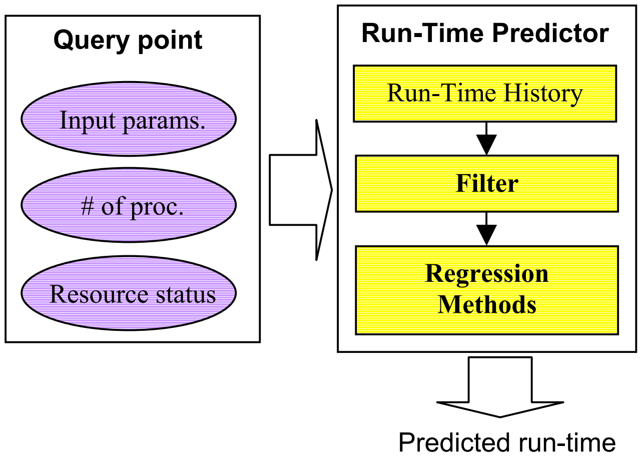

Figure 1. Sequence of steps to predict the run time of a parallel application run. The information for the application run is given by the query point.

run, (2) the number of processors used, (3) average CPU load, (4) average bandwidth, (5) average latency, and (6) the actual run time. Note that in order to compute (3), (4), and (5), only the processors and communication links that will be used for the run are considered and that they are computed before the application run starts.

When the query point is fed into the run-time predictor, it first extracts a subset of past run-time histories that are relevant to the query point as defined by a set of filters (see Section 2.2). Using this selected dataset, the predictor applies regression methods to discover the relationship and then predicts the run time given the query point (see Section 2.1).

2.1. Regression Methods

Regression methods [7] are mathematical tools that are often used to predict the behavior of one variable (e.g., the actual run time), the *dependent variable*, from multiple *independent variables* (e.g., the input, the number of processors to use, and the resource status). In our work we use linear regression approaches that are less computationally expensive than other approaches. We do not, however, assume that the performance models of the applications are linear.

Linear regression methods can be applied in different ways depending on which independent variables are used. Direct prediction (DP) treats as independent variables every variable that affects the run time; inindirect prediction (IP) the run time is predicted by a two-step process. Unlike DP, the linear regression model is applied to predict base time (time to compute the unit size of the problem on a processor). The predicted base time and query point then are used to generate the final run-time prediction. Both of these approaches are explained in additional detail in [2].

DP, the more traditional approach, generates accurate predictions when the regression models can capture the relationship between dependent variable and independent variables. On the other hand, IP introduces new independent variables that are a function of existing independent variables. Our experiments, described in Section 3, show that this method is superior to DP in most instances.

2.2. Filtering Technique

Any linear regression technique attempts to fit a straight line to the available data to minimize the sum of squared deviations of the predicted values from the actual observations. Therefore, if the observations do not show strong linearity, applying the linear regression model will not generate accurate predictions. To support parallel applications whose performance models are not linear, we use a filtering technique to extract subsets of observations that are close to the query point and show strong linearity.

The closeness of each data point to the query point is defined by the distance function of the filter. Since the range and distribution of variables used for run-time predictions are unknown, the distance function should normalize distances with respect to the query point. In addition, the extent to which individual variables influence the run time is also different. Therefore, the importance of each variable with respect to the run time also should be incorporated into the distance function. Formally, the distance between data point ($d_1, d_2, \ldots, d_n$) and query point ($q_1, q_2, \ldots, q_n$) is defined as:

$$Distance = \sum_{i=1}^{n} LocalDistance(d_i, q_i) * w_i$$

$$LocalDistance(d_i, q_i) = \left| \frac{d_i - q_i}{\max\limits_{1 \le k \le t} D_i^k - \min\limits_{1 \le k \le t} D_i^k} \right|$$

D_i^t : *observed value of* d_i *at time* t

w_i : *weight*

For each dimension, a local distance is computed that denotes how distant the value of a variable is from the corresponding variable in the query point, and is normalized to 1. The denominator used in the local distance function represents the range of the variable based on the observed data.

We used four filters in our experiments. Table 1 shows the names and variables used to calculate the distance metric for each filter. We use the number of processors in all of our filters; the rest of the data is first sorted by this variable, and then others are considered as part of the filtering function. Hence, only the past run time history data that has the same value as the one in the query point is used to compute distances.

Table 1. Filters and their variables

Filter Name	Variables Used
NP	Number of processors used
NP_R	Number of processors used, Resource capacities
NP_PARM	Number of processors used, Input parameters
NP_R_PARM	Number of processors used, Resource capacities, Input parameters

3. Empirical Analysis

We conducted experiments using two parallel applications implemented in MPI: an N-body simulation code and a heat distribution code. We also evaluated two types of background load: homogeneous and heterogeneous. Because of space constraints, we detail here only heterogeneous load and the N-body application; full experimental results are given in [2,3].

The N-body problem is concerned with determining the effects of forces between bodies (for example, astronomical bodies that are attracted to each other through gravitational forces). The N-body problem also appears in other areas, including molecular dynamics and fluid dynamics [13]. We implemented an $O(N^2)$ version of an N-body simulation code using the master-slave paradigm. For each time step, the master sends the entire set of bodies to each slave and also assigns a portion of the bodies to each slave. The slaves compute the new positions and velocities for their assigned bodies and then return the new data to the master. We randomly select problem sizes from 2,000 to 13,000 bodies and a number of processors from one to twenty.

We deployed the applications on the data grid nodes at Argonne National Laboratory. This testbed consists of 20 dual 845 MHz Intel Pentium III machines with 512 MB memory interconnected with 100 Mbit Ethernet. Resource status such as CPU load, bandwidth, and latency are measured every 5 minutes by the Network Weather Service (NWS) [14].

The status of each participating processor can affect the performance of the parallel applications. For this reason, we compared the performance of our technique using a heterogeneous background workload, in which each machine shows different background workload patterns. The competing workloads on the resources were generated by other users on the system in the normal course of use. They were not simulated or generated from traces. Hence, every run experienced a slightly different load.

We measured the prediction accuracy using the normalized percentage error, defined by

$$\%Error = \frac{\sum \left| RunTime_{measured} - RunTime_{predicted} \right|}{Size * AVG(RunTime)} * 100$$

$Size$: $total\ number\ of\ predictions$

$AVG(RunTime)$: $average\ of\ measured\ run-time$

3.1. Results

Figure 2 shows the results for the N-body application with a heterogeneous background load. Additional experiments are given in [2,3]. Filtering improved the prediction accuracy significantly in our experiments under both homogenous and heterogeneous loads. The run-time predictors that take into account both CPU load and latency always produce better prediction accuracy. The best two were the predictor using NP_PARM filter with 20.2% error in DP and the predictor using NP_PARM filter with 22.9% error in IP, respectively.

When IP is applied, the NP_PARM filter (which uses only the number of processors and input parameters) generated noticeably improved performance. In IP, however, the improvement achieved by the other filters was either negligible or worse than the prediction accuracy of the run-time predictor without filters. This result can be explained by the following equation modeling IP.

$$\begin{aligned} RunTime &= \boldsymbol{S * T} \\ &= \{\boldsymbol{S} * \boldsymbol{\alpha_0}\} + \{\boldsymbol{S * rs * \alpha_1}\} \end{aligned}$$

When IP is applied, the NP_PARM filter (which uses only the number of processors and input parameters) generated noticeably improved performance. In IP, however, the improvement achieved by the other filters was either negligible or worse than the prediction accuracy of the run-time predictor without filters. This result can be explained by the following equation modeling IP.

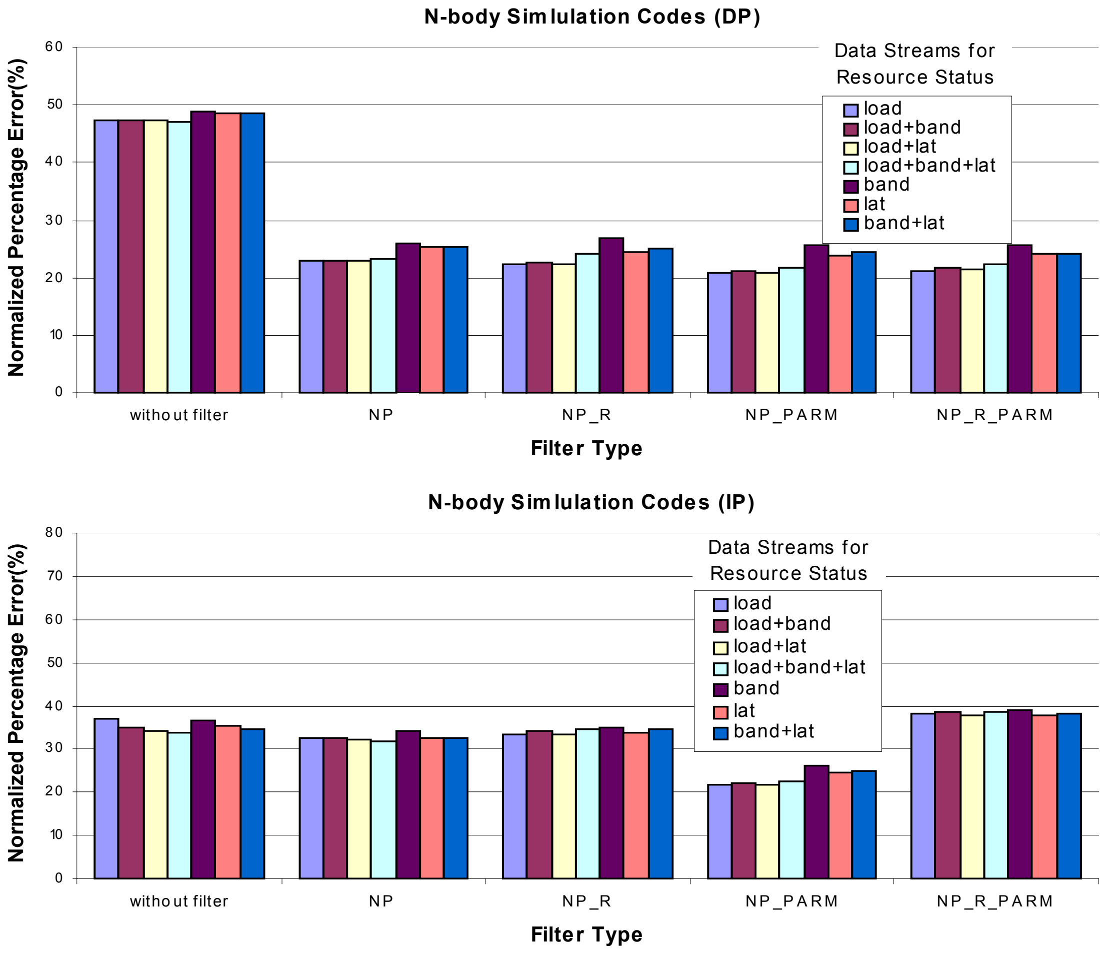

Figure 2. Normalized percentage error of various run-time predictors under heterogeneous background workloads.

$$\begin{aligned} RunTime &= \boldsymbol{S * T} \\ &= \{\boldsymbol{S * \alpha_0}\} + \{\boldsymbol{S * rs * \alpha_1}\} \end{aligned}$$

The first term ($S*\alpha_0$) explains the *main effects,* and the second term ($S*rs*\alpha_1$) defines the *interaction effects* between S and rs. Typically, the main effects have a larger impact on the, as reflected by the magnitude of the coefficients of each term. Therefore, filtering over the main effects can collect closer data points to the query point. In the N-body simulation example, *S* represents the quantity {*the number of bodies/the number of processors*}, and filters that include *S* generate more accurate predictions. The NP_R_PARM filter, which includes the number of bodies and the number of processors, as well as resource status, performs worse than the other predictors even though it uses the main effect for filtering. We believe this is because the filter only allows the consideration of data points very close to the query point, and therefore cannot capture the relationships correctly.

4. Conclusions and Future Work

In this paper, we propose a run-time prediction technique that is based on linear regression methods and filtering techniques in order to discover the relationship between variables that affect the run times of applications running on shared clusters and the actual run times. The experimental results show that without accurate performance models, our prediction techniques can generate satisfactory prediction accuracies for parallel applications running in shared environments.

Acknowledgments

We thank Charles Bacon for testbed support. This work was supported in part by the Mathematical Information and Computational Sciences Division Subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38.

References

[1] Nirav H. Kapadia, Jose A. B. Fortes, and Carla E. Brodley, "Predictive Application-Performance Modeling in a Computational Grid Environment", *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, 1999.

[2] Byoung-Dai Lee and Jennifer M. Schopf, "Run-Time Prediction of Parallel Applications on Shared Environments (long-version)", Argonne National Laboratory Technical Report #ANL/MCS-P1088-0903, September 2003.

[3] Byoung-Dai Lee and Jon B. Weissman, "Adaptive Resource Scheduling for Network Services", *Proceedings of the 3rd International Workshop on Grid Computing*, 2002.

[4] Byoung-Dai Lee, Run-Time Prediction Logs, http://www.cs.umn.edu/~blee.

[5] Muthucumaru Maheswaran and Howard Jay Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems", *Proceedings of the 7th IEEE Heterogeneous Computing Workshop*, 1998.

[6] M. Parashar and S. Hariri, "Interpretive Performance Prediction for Parallel Application Development", *Journal of Parallel and Distributed Computing*, vol. 60, no.1, pp.17-47, 2000.

[7] John A. Rice, "Mathematical Statistics and Data Analysis", Duxbury, 1995.

[8] Jennifer M. Schopf and Francine Berman, "Performance Prediction in Production Environments", *Proceedings of IPPS/SPDP*, 1998.

[9] A. Takefusa, H. Casanova, S. Matsouka and F. Berman, "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid", *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, 2001.

[10] Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin li, and Zhiling Lan, "Prophesy: Automating the Modeling Process", *Proceedings of 3rd International Workshop on Active Middleware Services* 2001.

[11] Sathish S. Vadhiyar and Jack J. Dongarra, "A Metascheduler for the Grid", *Proceedings of the 3rd International Workshop on Grid Computing*, 2002.

[12] Jon B. Weissman, "Predicting the Cost and Benefit for Adapting Data Parallel Applications on Clusters", *Journal of Parallel and Distributed Computing*, vol. 62: p 1248-1271, 2002.

[13] B. Wilkinson and M. Allen, "Parallel Programming", Prentice Hall, 1999.

[14] Rich Wolski, Neil T. Spring and Jim Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Journal of Future Generation Computing Systems*, 1998.