Run-Time Prediction of Parallel Applications on Shared Environment

Byoung-Dai Lee[1] and Jennifer M. Schopf[2]

blee@cs.umn.edu

jms@mcs.anl.gov

[1]Department of Computer Science and Engineering, University of Minnesota, Twin Cities, MN

[2]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL

Abstract

Application run-time information is a fundamental component in application and job scheduling, as well as in resource selection. 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. Many conventional approaches used in scheduling research activities assume that accurate performance models of the applications are available or that the applications execute only on space-shared resources. We believe, however, that it is not always possible to obtain such performance models of complex applications, as they require a deep understanding of the application and a significant number of experiments to validate. 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.

Keywords: performance prediction, filtering, regression, scheduling, Grid computing

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 [1, 2, 10, 17, 19, 20]. 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 and increase application run-times [3].

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 or that the applications execute only on space-shared resources where no two processes can run simultaneously. For accurate predictions of run times of parallel applications, such performance models must include parameters for both system-specific and application-run-specific information such as CPU load, bandwidth, the number of processors to use, message size, and operation counts, etc. For simple applications such as matrix multiplication codes, developing accurate performance models of the applications is relatively easy. In general, however, obtaining such performance models of complex applications is not always possible, as they require a deep understanding of the application and a significant number of experiments to validate the models developed.

In this paper, we propose a run-time prediction technique for parallel applications running in shared environments. The novel aspect of our technique is that it derives the application execution times without using performance models of the applications. 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. The proposed technique is based on regression methods and a filtering technique. The key idea of our approach is

that regression methods are applied only to *subsets* of past history in order to discover the relationship. A filtering technique is used to select such subsets. We evaluated the performance of our technique using two parallel applications: an N-body simulation code and a heat distribution code. These applications are implementations of two representative paradigms of high-performance distributed computing: master-slave and regular SPMD. 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.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes our prediction technique in detail and Section 4 presents the experimental results and analysis. Section 5 briefly discusses future work.

2. Related Work

Significant research has been conducted in the area of performance prediction. Much of this research, however, is tightly coupled with the applications in the sense that a thorough analysis of the codes is required. For example, much previous work assumed that operation counts and other low level details were available to predict application behavior [4,11,12,16].

Kapadia et al. [7] use three instance-based learning algorithms to predict resource usage for each application run in the PUNCH network-computing environment. Their approach is similar to ours in that only subsets of past performance information is used to learn the relationship. However, they do not consider parallel applications. In addition, by exploiting temporal and spatial locality of application runs, they implicitly assume that system-specific and application-run-specific information does not change significantly.

Lee and Weissman [8] use local linear regression and clustering techniques to predict the run times of parallel applications using columnwise predictions that reflect the performance change depending on input parameters given the same resource set and rowwise predictions that reflect performance change depending on the resource set given the same input parameters to the application. This work is limited to dedicated execution environments where no two processes run simultaneously on a single processor.

Parashar and Hariri [12] propose an interpretive performance prediction technique that uses a characterization of the high-performance computing system and the application. In order to characterize the application, however, very detailed application information is required.

Schopf and Berman [15] use stochastic values of parameters in performance models to predict behavior over a range of likely system states. They address the problem of using performance models parameterized by single values of resource status on shared environments. However, their technique can be applied only when accurate performance models of the applications are available.

Taylor et al. [18] develop an infrastructure, called Prophesy, that automates the performance analysis and modeling processes. The application codes can be instrumented at different levels of granularity and the resultant performance data is automatically stored in the performance database. The performance models are developed based upon performance data from the performance database, model templates from the template database, and system characteristics from the system database. These models are used to predict the performance of the application under different system configurations. Prophesy also uses curve fitting as one of its automatic modeling technique

Our work assumes very limited performance information (similar to Kapadia et al. and Lee and Weissman) and yet functions in a dynamic environment (as seen in Parashar and Hariri, Schopf and Berman, and Taylor et al.).

3. Run-Time Prediction Using Regression Methods

To predict the execution times based on past application run history without using accurate performance models raises several significant issues. For example, one must evaluate which variables affect the run time of the application and to what extent. Furthermore, since the ranges and distributions of values of individual variables are unknown, appropriate scaling factors must be determined in order to compare values of different variables. To make the prediction problem tractable, we make the following assumptions:

- The set of application input parameters that can affect the application run-time is known. In addition, some of the values have been converted into a more appropriate form; for example, if the application has the input parameter of a file name and the run time depends directly on the size of the file, the run-time predictors will be fed with the file size, instead of the file name. To what extent the input parameters affect the run time is unknown; we assume only that this set should be tracked. In our experience, users of the applications understand the meanings of individual input parameters and know which input parameters affect the run time of the application, so this assumption does not significantly affect the usability of our approach.

- We do not consider parallel applications with run times that are nondeterministic or that depend on the distributions of the input data. For example, in iterative Jacobi computation, the number of iterations depends on the distribution of eigenvalues of the input matrix, which is difficult to compute in advance. To our knowledge, however, no existing performance prediction technique considers these types of application.

- The parallel applications are not instrumented. Therefore, the only information we can collect is the start times and end times of the application runs. By making this assumption, we believe
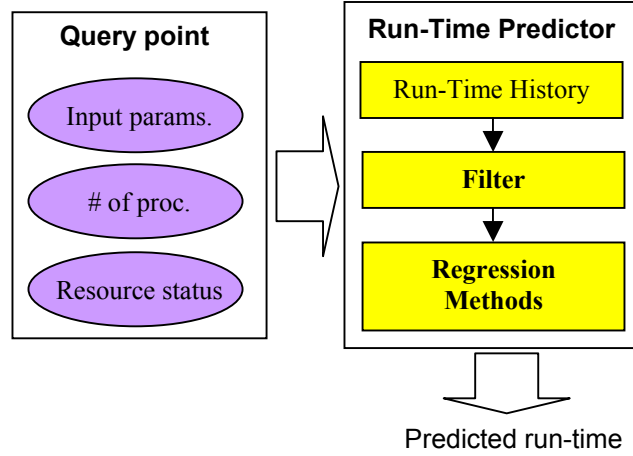
Figure 1. Sequence of steps to predict the run time of a parallel application run. The information for the application run is given by the query point.

that our technique can be applied to many parallel applications without modifying the applications.

- 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 run time 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) input to the application run, (2) number of processors used, (3) average CPU load, (4) average bandwidth, (5) average latency, and (6) 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; these 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 3.2). 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 3.1).

3.1. Regression Methods

Regression methods 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). The principle underlying regression methods is to minimize the sum of squared deviations of the predicted values from the actual observations [13]. In this work, we use the linear regression method because the computation cost is less than that of nonlinear regression methods such as quadratic and cubic methods. Note that using a linear regression method in our prediction technique does not mean that we assume that the performance models of the applications are linear. Instead, only a set of past run history sufficiently close to the query point, which is selected by filters, will be approximated by a linear relationship.

Linear regression methods can be applied in different ways depending on which independent variables are used.

- **Direct Prediction)**

Direct prediction (DP) treats as independent variables every variable that affects the run time. Therefore, the following equation holds.

$$RunTime \sim f(ip, np, rs)$$
$$= \beta_0 + \beta_1 * ip + \beta_2 * np + \beta_3 * rs$$

ip : the input parameters to the parallel application run
np : the number of processors to use for the run
rs : the current resource status
β_i : linear regression coefficients

In this model, the run time of the parallel application run is a linear function of the input to the run, the number of processors, and the current resource status. Note that if there are n input parameters, the term β_1*ip is expanded into n terms. For example, if two input parameters, x, y, are used for the application run, β_1*ip is replaced with $\beta_{11}*x+\beta_{12}*y$, where β_{11} and β_{12} are linear regression coefficients. In addition, depending on which data stream is used for resource status, β_3*rs term also can be expanded into multiple terms. For example, if both CPU load and bandwidth are considered, then $\beta_{31}*cpu\ load +\beta_{32}*bandwidth$ will be used for the linear equation, where β_{31} and β_{32} are linear regression coefficients.

- **Indirect Prediction**

In indirect 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.

$$RunTime = \mathbf{T * S}$$
$$T \sim f(rs)$$
$$= \boldsymbol{\alpha_0} + \boldsymbol{\alpha_1} * \mathbf{rs}$$
$T :base\,time$
$S : problem\,size\,per\,processor$
$rs :the\,current\,resource\,status$
$\alpha_i :linear\,regression\,coefficients$

This method is made available because we assume that the individual input parameters to the applications are already known. Therefore, the problem size that each participating processor should compute can be easily computed.
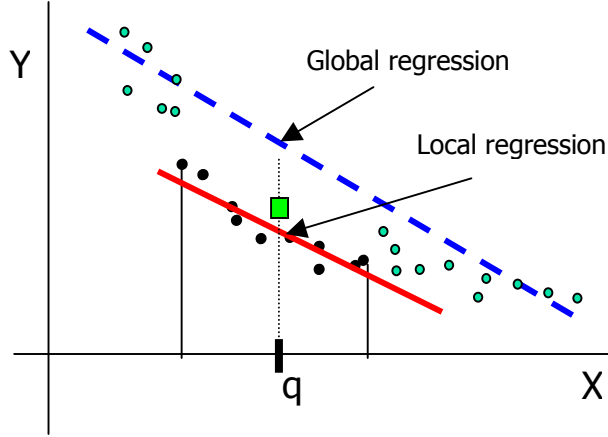
Figure 2. The actual value of Y given X is q is represented by rectangle. The dashed line is generated by a global linear regression method and the solid line by a local linear regression model.

DP is the more traditional way of applying regression methods. It will generate accurate predictions when the regression models used 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 4, show that this method is superior to DP as long as the values of the new variables are correct.

3.2. Filtering Technique

Any linear regression technique attempts to fit a straight line to the available data in order 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. Figure 2 shows the effects of using filters. Since the global regression model considers all of the observed data in order to predict that the actual value of Y, given X, is q, it generates a straight line in such a way that it minimizes the error for data points not only close to q

but also far from q. On the other hand, in a local regression model, since only data points close to q are considered in order to generate a straight line, the prediction accuracy is improved.

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 $(d_1, d_2, ..., d_n)$ and query point $(q_1, q_2, ..., q_n)$ is defined as follows.

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

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

D_i^t : observed value of d_i at time t
w_i : weight

For each dimension, local distance is computed, which denotes how far the value of a variable is distant from the corresponding variable in the query point and is normalized to 1. The denominator used in local distance function represents the range of the variable based on observed data so far. The weight associated with each dimension represents the importance of the dimension with respect to the run times. For example, for applications that require small amounts of data transfer among constituent processes, fluctuation of network status will affect the run times less than the CPU load will.

Many possible filters exist, and each is differentiated by which variables are used to define the distance function. We used four filters for the experiments. Table 1 shows the filter names and variables that are used for each filter.

Table 1. Filters and their constituent 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

We use the number of processors in all of our filters; hence, the rest of the data is first sorted by this variable, and then others are considered as part of the filtering function. Thus, only the past run time history data that has the same value as the one in the query point is considered to compute distances.

4. Empirical Analysis

We conducted experiments using two parallel applications implemented in MPI: an N-body simulation code and a heat distribution code. These applications are implementations of two representative paradigms of high-performance distributed computing: master-slave and regular SPMD.

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 [21]. 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.

The heat distribution code simulates the temperature changes of the surface over time, given that temperatures along each of edges of the surface are known. Temperature distribution is simulated by covering the area with an evenly spaced grid of points. In our parallel implementation, the grid is
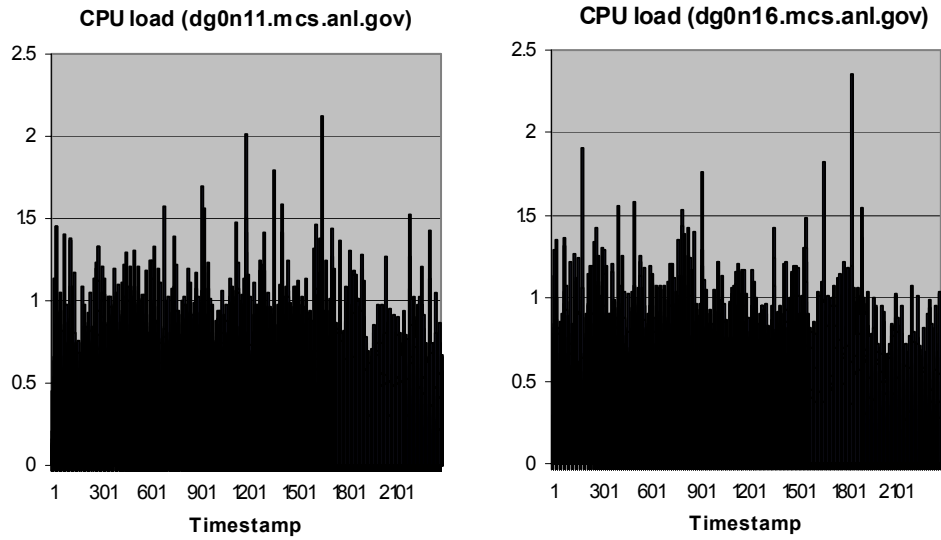
Figure 3. Homogeneous background workloads.
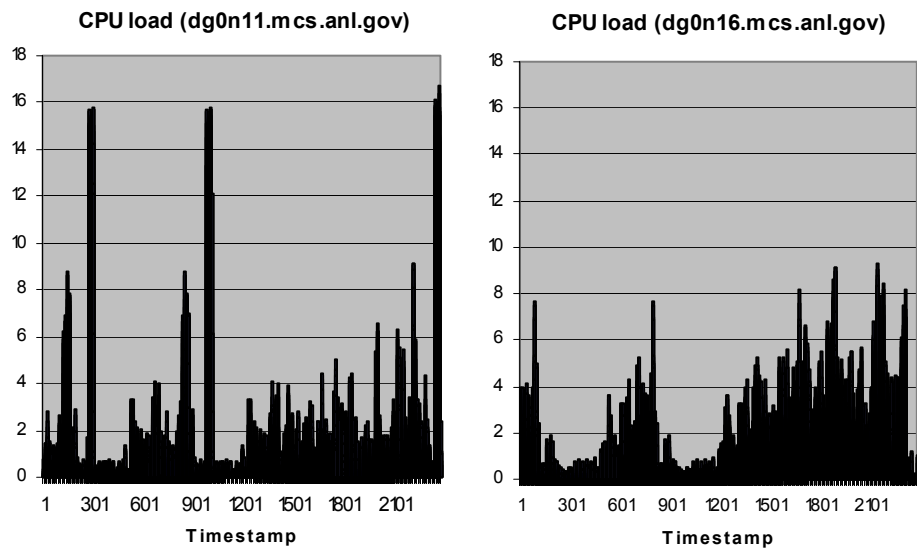
Figure 4. Heterogeneous background workloads.

decomposed and then distributed by rows to the each process. At each time step, participating processes must exchange border data with neighboring processes because a grid point's current temperature depends on its previous time step value and the values of the neighboring grid points.

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) [22].
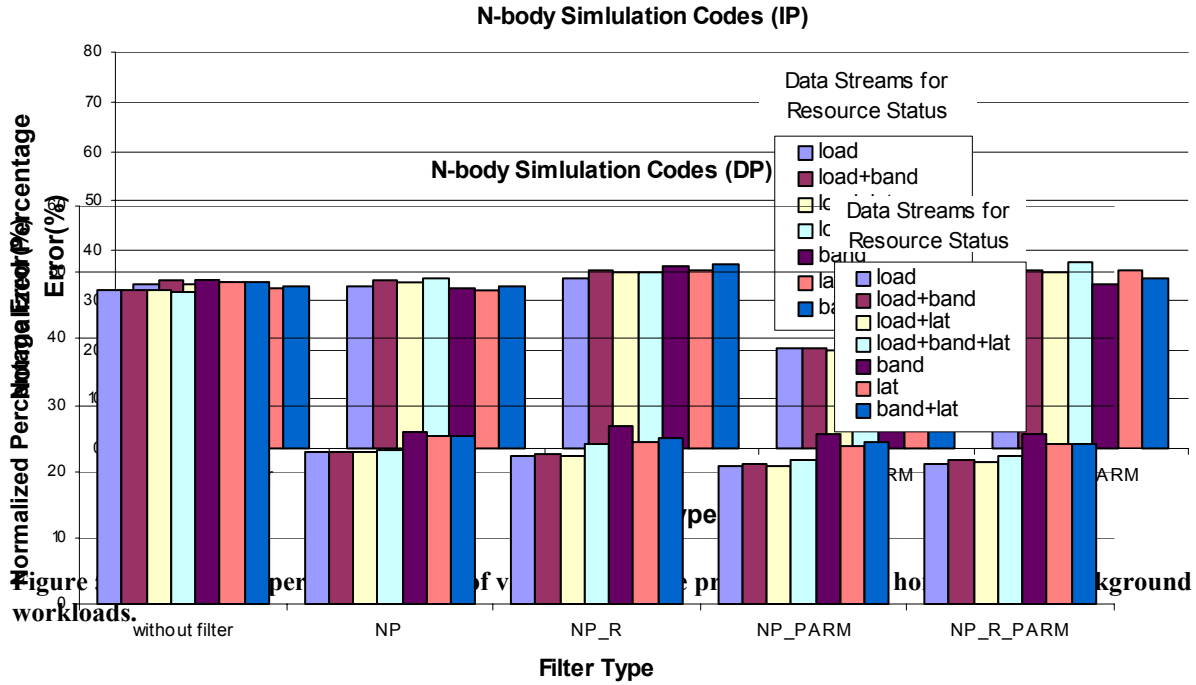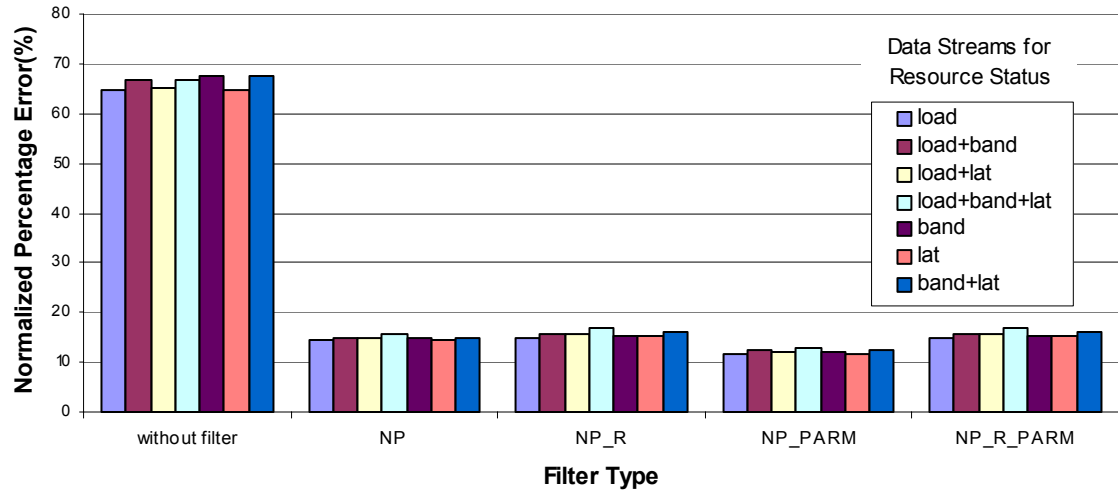
The status of each participating processor can affect the performance of the parallel applications. For this reason, we compared the performance of our technique under two different background workloads: homogeneous and heterogeneous. In homogeneous background workloads every processor in the testbed shows similar patterns of background workloads during the experiments (Figure 3), whereas in heterogeneous background workloads each machine shows different background workload patterns (Figure 4). 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.

Table 2 shows the configuration for the experiments. For each application run, the values are randomly selected.

Table 2. Configuration for the experiments

	N-body Simulation Codes	**Heat Distribution Codes**
Problem size	2,000 – 13,000 bodies	100x100 – 800x800 grids
Number of processors	1-20	1-20

We measured the prediction accuracy using the normalized percentage error, defined as follows.

N-body Simlulation Codes (DP)

N-body Simlulation Codes (IP)

Figure ... **per** ... **of v** ... **e pr** ... **ho** ... **ckground workloads.**

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

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

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

We present the results from the N-body simulation code ... or the heat distribution code shows similar trends. The full results are available in [9]. We conducted several experiments for each filtering technique while varying weights for distance functions, since we did not know how much

Figure 6. **Normalized percentage error of various run-time predictors under heterogeneous background workloads.**

each variable would affect the run times. For each filter, we applied several different sets of weights selected so that the sum of the weights was 1.0. These weights were selected based on preprocessing of a related set of experiments and were fixed throughout. The presented graphs are only subsets of the experimental results.

4.1. Homogeneous Workloads

Figure 5 presents the normalized percentage errors of several run-time predictors under homogeneous background workloads. In the graph, run-time predictors are grouped by the filter used. And, under each filter, run-time predictors are differentiated by what data streams for resource status are used. For example, one predictor used only the load status of the processors, whereas another used both load status and latency.

Without filters, IP outperforms DP. The reason is that DP predicts the run time using linear relationship with all variables, which is not the case with parallel applications. For both IP and DP, the prediction accuracy can be improved by using appropriate filters. However, we did not observe significant difference in the performance with different data streams for resource status. The reason is that the resource status of each machine is homogeneous and the system itself is lightly loaded.

4.2. Heterogeneous Workloads

Figure 6 shows results under heterogeneous workloads. As with homogeneous workloads, filtering improved the prediction accuracy. In addition, depending on the information used for resource status, the prediction accuracy can also be improved. 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.

Under both homogeneous and heterogeneous background workloads, when IP is applied, the NP_PARM filter (which uses only the number of processors and input parameters) generated noticeably improved performance. However, in IP, 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.

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

The first term ($S*\alpha_0$) is called *main effects,* and the second term ($S*rs*\alpha_1$) is called *interaction effects between S and rs*. Typically, the main effects have a larger impact on the response than do the interaction effects, 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 N-body simulation example, *S* can be represented by {*the number of bodies/the number of processors*}. Therefore, only filters that include *S*, which include both the number of processors and the number of bodies, can generate more accurate predictions. An interesting result is that the filter that includes the number of bodies and the number of processors, as well as resource status, performed worse than the other predictors even though it uses the main effect for filtering. This filter collects data points too much concentrated around the query point and therefore cannot capture the relationship correctly.

In both homogeneous and heterogeneous background workloads, run-time predictors with appropriate filters improved the prediction accuracy dramatically compared with those without filters. The reason is that only parts of history that are close to the query point and show strong linearity are

considered to predict the run time. In addition, using appropriate data streams for resource status helped improve the accuracy.

5. Conclusions and Future Work

In this paper, we proposed a run-time prediction technique based on linear regression methods and filtering techniques in order to discover the relationship between variables that affect the run times of parallel applications 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.

Since this work is at early stage, many issues need to be addressed. In particular, in the current technique, the variables and weights used in filters are manually selected. In general, however, we do not know which variables are more important than others. To address the problem, we are investigating principal component analysis [6] as a means of identifying a set of variables that affect the application run-times most. In addition, we are examining nonlinear regression models to derive the relationship of past history selected by filters. We also plan to apply our technique to the two LHC physics codes, ATLAS [1] and CMS [5].

References

[1]   ATLAS – A Toroidal LHC Apparatus, http://atlas.web.cern.ch/Atlas/Welcome.html.

[2]   Rajkumar Buyya, Manzur Murshed and David Abramson, "A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids," Technical Report, CSSE-2002/109, Monash University, Melbourne, Australia.

[3]   Su-Hui Chiang, Andrea Arpaci-Dusseau, and Mary K. Vernon, "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance," in *Proceedings of 8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.

[4]   Mark J. Clement and Michael J. Quinn, "Analytical Performance Prediction on Multicomputers," *Proceedings of Supercomputing*, 1993.

[5]   CMS – Compact Muon Solenoid, http://cmsdoc.cern.ch/cms/outreach/html/index.html.

[6]   I. T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, 1986.

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

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

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

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

[11] V. W. Mak and S. F. Lundstrom, "Predicting the Performance of Parallel Computations," *IEEE Transactions on Parallel and Distributed Systems*, pp. 106–113, July 1990.

[12] 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.

[13] John A. Rice, *Mathematical Statistics and Data Analysis,* Duxbury, 1995.

[14] Rafael H. Saavedra and Alan J. Smith, "Analysis of Benchmark Characteristics and Benchmark Performance Prediction", Technical Report CSD-93-767, University of California, Berkeley, 1992.

[15] Jennifer M. Schopf and Francine Berman, "Performance Prediction in Production Environments," *Proceedings of IPPS/SPDP*, pp. 647–653, 1998.

[16] Jens Simon and Jens-Michael Wierum, "Accurate Performance Predictions for Massively Parallel Systems and its Applications," in *Proceedings of Euro-Par '96*, Vol. 12, pp. 675–688, 1996.

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

[18] Valerie Taylor, X. Wu, X. Li, J. Geisler, Z.Lan, M. Hereld, I. Judson, and R. Stevens, "Prophesy: Automating the Modeling Process," in *Proceedings of 3rd International Workshop on Active Middleware Services 2001*, San Francisco, 2002.

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

[20] Jon B. Weissman, "Predicting the Cost and Benefit for Adapting Data Parallel Applications on Clusters," *Journal of Parallel and Distributed Computing*, Vol. 68, no. 8, pp. 1248–1271, 2002.

[21] B. Wilkinson and M. Allen, *Parallel Programming,* Prentice Hall, 1999.

[22] 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*, Vol. 15, no. 5–6, pp. 757–768, 1999.