Performance Modeling based on Multidimensional Surface Learning for Performance Predictions of Parallel Applications in Non-Dedicated Environments *

Jay Yagnik
Google Inc.
1600 Amphitheatre Pky.
Mountain View,CA 94043
USA.
jayyagnik@gmail.com

H.A. Sanjay and Sathish Vadhiyar
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore - 560012
India
{sanjay@rishi.serc, vss@serc}.iisc.ernet.in

Abstract

Modeling the performance behavior of parallel applications to predict the execution times of the applications for larger problem sizes and number of processors has been an active area of research for several years. The existing curve fitting strategies for performance modeling utilize data from experiments that are conducted under uniform loading conditions. Hence the accuracy of these models degrade when the load conditions on the machines and network change. In this paper, we analyze a curve fitting model that attempts to predict execution times for any load conditions that may exist on the systems during application execution. Based on the experiments conducted with the model for a parallel eigen value problem, we propose a multi-dimensional curve-fitting model based on rational polynomials for performance predictions of parallel applications in non-dedicated environments. We used the rational polynomial based model to predict execution times for 2 other parallel applications on systems with large load dynamics. In all the cases, the model gave good predictions of execution times with average percentage prediction errors of less than 20%.

1 Introduction

Performance predictions of parallel applications have been mainly used for scheduling decisions, identification of bottlenecks in applications and systems and fine tuning of algorithms to provide scalability for larger problem sizes and larger number of processors. Over the years, many performance modeling efforts [9, 12] have been undertaken to predict the performance of parallel applications on various systems. The performance modeling strategies are of different types including trace-event simulations [2, 12], parametrized analytical models [9, 13] and curve-fitting models [13].

Most of the existing curve-fitting modeling strategies assume uniform loading conditions on the systems or dedicated environments when the experiments for modeling are conducted and use the models to predict execution times for large problem sizes and/or larger number of processors for the same loading conditions [13]. This assumption is unrealistic in non-dedicated environments. Although major scientific parallel applications are primarily executed on dedicated space-shared systems, non-dedicated environments form important testbeds for testing, developing and fine-tuning of parallel applications and also for executing less time-critical applications. On these systems, it is essential to predict execution times of applications for various kinds of scheduling decisions.

In this paper, we explore a curve-fitting strategy based on rational polynomials for predicting execution times of applications in non-dedicated systems where the external CPU and network loads on the systems during predictions can be different from the loads when experiments for modeling were conducted. Our work considers predicting execution times of a parallel application for different problem sizes for a fixed number of processors, i.e., given execution times of the application for few different problem sizes for a certain number of processors, our methodologies predict the execution times of the application for certain other problem sizes for the same number of processors in non-dedicated environments. Unlike the previous models that require detailed knowledge of the applications, our model requires only approximate complexities of the applications.

*This work is supported by Indian Institute of Science's 10^{th} Plan Grant SERC Part(2A) Special Grant (45/SERC)

Also, our model does not require detailed instrumentation and profiling of applications but only require observation of total execution times. The modeling techniques are intended for simple parallel application kernels which are invoked in complex parallel applications. These parallel kernels have single phases of computation and communication and are integral to many scientific applications. In most cases, the developers of the parallel kernels will be able to provide coarse computation and communication complexities. These complexities are used in our models to predict the execution times of the kernels. Although the coarse complexities are available, automatically refining the model by careful parametrization of the coarse complexity terms with adequate application and system parameters in order to provide decent predictions on noisy non-dedicated systems is a non-trivial task.

By conducting experiments for predicting execution times of a parallel eigen value problem on 4 processors with small load dynamics[1], we found that the model based on rational polynomials provides good accuracies and that the resulting performance prediction trends for different problem sizes closely match with the trends shown by actual values. We then evaluated the rational polynomial based model on 8-processor and 32-processor systems with large load dynamics by predicting execution times for 3 parallel applications: eigen value problem, Fast Fourier Transforms (FFT), and conjugate gradient (CG). Approximate complexities of the applications and approximations of the load dynamics were used to parametrize the model for the applications. In all cases, the model gave less than 20% average percentage prediction errors. Models with prediction errors of less than 20% are considered to be reasonable in the literature especially when used to predict execution times in non-dedicated environments with high load dynamics.

In the next section, detailed description of the the curve-fitting model based on rational polynomials is given. The model is evaluated in terms of prediction accuracies by conducting experiments with a parallel eigen value application on a 4-processor system with small load dynamics. Based on the results, we claim that models based on rational polynomials can give close to accurate predictions. In Section 3, we propose a generic model based on rational polynomials for predicting execution times of generic parallel applications executing in non-dedicated environments with potential large load dynamics. We evaluate the generic rational polynomial based model in Section 4 by showing prediction accuracies for three parallel applications on 8-processor and 32-processor systems with large load dynamics. In Section 5, relevant work is presented. Section 6 gives conclusions and Section 7 details future work needed for automatic building of performance models.

[1] We consider a system as having small load dynamics when the load conditions change from one application execution to another but remain the same throughout the duration of an application execution. We consider a system as having large load dynamics when the load conditions can change even during an application execution.

2 Models and Their Evaluations

In this section we describe the general methodology for modeling and evaluation of the model. We then describe the rational polynomial based model and evaluate its accuracy.

2.1 Methodology

We evaluate the models with the help of ScaLAPACK [4] parallel eigen value problem. The ScaLAPACK kernel that was used for the experiments is PDSYEV, the kernel for parallel eigen value solver for a double-precision symmetric matrix. The matrix was distributed in 2D block-cyclic fashion across processors.

The experiments for modeling were conducted on a Intel Pentium IV based Linux cluster consisting of 8 nodes with small load dynamics. All the 8 nodes are connected to each other by 100 Mbps Ethernet link using a 8-port 100 Mbps Ethernet switch. Each of the nodes has a single 2.8 GHz CPU, a 512 MB RAM, a 80 GB hard disk and running Fedora Core 2.0, Linux 2.6.5-1.358 operating system. For the experiments reported in this section, only 4 processors were used. Before conducting an experiment corresponding to the application execution with a problem size, synthetic CPU and network loads were applied to the processors and links between processors, respectively. The loads were maintained at constant amounts throughout the duration of the application execution but were varied by random amounts for different application executions.

In order to construct models and predict the execution times for larger problem sizes for a given number of processors, P, experiments were conducted by executing the application on P processors for smaller problem sizes. In our work, we restrict our predictions to only those problem size ranges where the entire data needed by the application will fit in the local memories of the processors and hence will not incur disk swapping costs.

At the start of conducting an experiment with a problem size, the transient CPU and network characteristics were obtained. Network Weather Service (NWS) [16], a tool for measuring and forecasting system parameters, was used for obtaining available CPUs of the processors used for application execution and available bandwidths on the links between the processors. Available CPU is a fraction of the CPU that can be used for the application and inversely proportional to the amount of CPU load. Available bandwidth of a link to an application is usually lesser than the link capacity and is also inversely proportional to the network load on the link. *min_avail_cpu* and *min_avail_band* are then calculated as the minimum of available CPUs of all processors

involved in the application execution and minimum of available bandwidths of all the links between the processors, respectively. Then the experiment is conducted by executing the application with the problem size and the duration of application execution is observed.

The problem sizes, *min_avail_cpu* and *min_avail_band* values, and the execution times, corresponding to different experiments with different small problem sizes, were used as inputs for training our models. The trained models were then used for predicting execution times for a large problem size for given values of *min_avail_cpu* and *min_avail_band*. For the ScaLAPACK eigen value problem, experiments were conducted for matrix sizes 100-8000 in step sizes of 100 and the measurements were used for training the models. The resultant models were used to predict execution times for matrix sizes 9000-12000 in step sizes of 100.

The models were evaluated and compared based on average percentage prediction errors. For a given model, the average percentage prediction error is given by:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{|(actual_i - predicted_i)|}{|actual_i|}$$

where *N* is the number of experiments, $actual_i$ is the measured execution time and $predicted_i$ is the predicted execution time by the model for experiment *i*. Since in some cases, the average percentage prediction errors can be misleading, we also evaluated the models by comparing the trends shown by the prediction and actual values.

In this section, the *min_avail_cpu* and *min_avail_band* values were measured only once for an experiment at the beginning of the application execution. Although calculation of the values only once for an experiment does not capture all kinds of load dynamics, it is adequate for the experiments shown in this section where constant load is maintained throughout an application execution. Although our models have provisions for taking into account maximum available latencies of the links, latency values were not considered in our models for the results in this paper due to the negligible impact of latencies on the execution times in the cluster environment we use for our experiments.

The following subsection describe the model that was used:

2.2 A Rational Polynomial Model

The model we investigated was a rational polynomial based model taking into account the computation and communication complexities of the problem. ScaLAPACK parallel eigen value problem has cubic computation complexity and quadratic communication complexity in terms of problem sizes. If *n* is the problem or matrix size, the rational

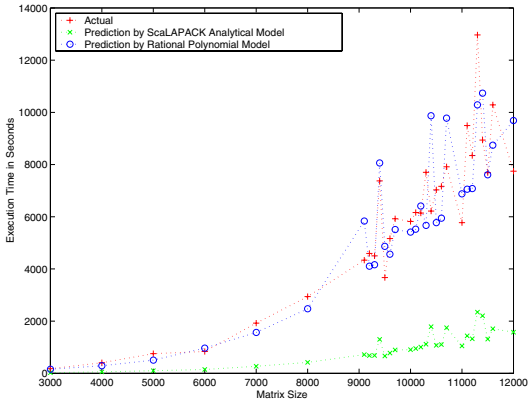

Figure 1. Predictions with Rational Polynomial based Model

polynomial based model can be formulated as:

$$P_n = a_1 n^3 + b_1 n^2 + c_1 n + d_1 \quad (1)$$

$$P_{cpu} = \frac{a_2 n^3 + b_2 n^2 + c_2 n + d_2}{min\_avail\_cpu} \quad (2)$$

$$P_{bw} = \frac{a_3 n^2 + b_3 n + c_3}{min\_avail\_band} \quad (3)$$

$$t_{predicted} = P_n + P_{cpu} + P_{bw} \quad (4)$$

Equation 1 calculates the time needed for problem initializations and synchronizations that happens once at the beginning of application execution. This time depends on the problem size and is not impacted significantly by the external load. Equation 2 calculates the total predicted time for computation and equation 3 calculates the total predicted time for communication. The total predicted time for the application is given by equation 4. The coefficients in the equations are determined by polynomial fit using data points in the training samples. The problem of finding the coefficients is a linear regression problem and is equivalent to solving a system of linear equations. The determined coefficients along with the system and application parameters are used in the equations to predict the execution time for a given problem size. Figure 1 shows the predictions of execution times with the rational polynomial based model. The curve corresponding to predicted times follows very closely with the curve corresponding to actual execution times. The average percentage prediction error is 18% with variance of 0.01 and indicates the reasonableness of the model. Thus our rational polynomial based model gives good predictions for non-dedicated environments.

Figure 1 also shows a third curve corresponding to the parametrized analytical model for the ScaLAPACK eigen value problem. The parametrized analytical model was

formulated by the ScaLAPACK authors. The formula expresses total execution time in terms of matrix size, number of processors, latency, bandwidth, and speed of the processors. We scaled down the theoretical speed of the processors by the available CPU obtained from NWS to obtain effective speeds [2] and also used the minimum available bandwidth and maximum available latency in the formula. As can be observed, our model performs much better than the analytical model for the ScaLAPACK eigen value problem. This is because the ScaLAPACK analytical model does not take into account contentions due to external loads while our curve-fitting model embeds the average contention behavior in the coefficients used in its equation.

3 Proposed Model

We propose a general model involving rational polynomials for predicting execution times of generic parallel applications executing in non-dedicated environments with potential large load dynamics. On systems with large load dynamics, the amount of external loads on the processors involved in application execution can change drastically during the course of application execution. Hence using *min_avail_cpu* and *min_avail_band* values, that are measured once before the application execution, for training the models and predicting execution times will lead to large prediction inaccuracies. Instead, during training of the models, we measure available CPUs and available bandwidths on all processors and links involved in application execution at periodic intervals of time from the beginning to the end of the application execution. We then calculate for each processor and link, *avg_avail_cpu* and *avg_avail_band*, respectively. These are the averages of the periodic available CPUs and available bandwidths collected for the processor and the link, respectively, during the application execution. Finally, we calculate *min_avg_avail_cpu* and *min_avg_avail_band* values by finding the minimum of *avg_avail_cpu* and *avg_avail_band* values, respectively, on all processors and links. The *min_avg_avail_cpu* and *min_avg_avail_band* values are used along with the problem sizes and execution times for training the models.

The trained models are used to predict the execution time of the application when executed with a particular problem size and for particular values of *min_avg_avail_cpu* and *min_avg_avail_band* of the system. But unlike *min_avail_cpu* and *min_avail_band* values that were used for predicting execution times on systems with small load dynamics, *min_avg_avail_cpu* and *min_avg_avail_band* values cannot be measured before the application execution to predict the execution time. This is because the *min_avail_cpu* and *min_avail_band* values represent the system loads **that exist** at the beginning of application execution for which the execution time is predicted, while the *min_avg_avail_cpu* and *min_avg_avail_band* values represent the system loads **that will exist** during the period of the application execution.

Hence, we forecast the *min_avg_avail_cpu* and *min_avg_avail_band* values based on the history of previous measured values and use the forecasted values in our models to predict the execution time of an application. The history is continuously updated with new measured values. Thus, we forecast the load dynamics of the system based on the observed load dynamics. We use the forecasting tools from NWS to forecast *min_avg_avail_cpu* and *min_avg_avail_band*. NWS uses "mixture of experts" strategy [15] to forecast a value, where different forecasting algorithms are used to postcast the previous measured values. The forecasting algorithm with the minimum postcast error is then used to forecast the next value. Thus, by using NWS tools and updating history of measured values with recent values for forecasting *min_avg_avail_cpu* and *min_avg_avail_band*, our models that use these forecasted values for predicting execution times are able to "self correct" based on the prediction errors for previous values.

The resultant model based on rational polynomials for predicting execution time, $t_{predicted}$, is given by:

$$t_{predicted} = O(n^{comp}) + \frac{O(n^{comp})}{min\_avg\_avail\_cpu} + \frac{O(n^{comm})}{min\_avg\_avail\_band} \quad (5)$$

where $O(n^{comp})$ is the computational complexity and $O(n^{comm})$ is the communication complexity of the application. The *min_avg_avail_cpu* and *min_avg_avail_band* used in Equation 5 are measured values during training the model and forecasted values for prediction. Equation 5 is a generalization of Equations 1-4 used for predicting execution time of ScaLAPACK eigen value problem.

The advantage of the proposed model is that the application developer only needs to specify the approximate computational and communication complexities of the parallel application. The exact parameters are learned by regression with few trial runs. The experiments in the next section use this proposed model for predicting execution times.

4 Experiments and Results

Experiments were conducted validating the proposed model in Section 3 in terms of predictions of execution times for 3 applications: ScaLAPACK parallel eigen value problem, parallel Conjugate Gradient (CG) application from NAS Parallel Benchmarks (NPB) suite [3], and

[2] Available CPUs are between 0 and 1. When there is no contention, available CPU is 1 and the effective speed will be equal to the theoretical speed

parallel FFT application from FFTW [7] package. All the three applications have significant computation and communication complexities.

The experiments in this Section were conducted both on the 8-processor Intel Pentium IV system described in Section 2 and on a 32-processor IBM P720 system arranged in 8 nodes. Each node of the 32-processor system is a 4-way IBM Power 5 SMP with hard disk capacity of 146 GB and running Suse Linux 9.0 sp 1 operating system. Each processor in a node has 1.65 GHz CPU speed and 1 GB RAM. All the 8 nodes are connected to each other by Gigabit Ethernet links through Gigabit Nortel switch.

Large load dynamics was ensured on both the systems by continuously running synthetic CPU and network loading programs on the processors in the background. For loading the CPUs of a system at a given point of time, a set of processors was randomly chosen out of the available processors in the system and synthetic loading programs were run on the processors in the set. The amount of loading on each processor was randomly varied such that the available CPU value of the processor is varied between 6.5%-72% of the total CPU. Small available CPU percentages imply large loading of the processor. The duration of the load is also randomly varied between 3-8 minutes. This process of random selection of processors, introducing random amounts of loads on the processors, and maintaining the loads for random durations of time, is repeated continuously on the system. For network loading, we used a loading program to introduce synthetic network loads on the links of the system and to reduce the available bandwidths of the links. The amount and duration of the load can be specified to the loading program. The loading program takes as input a source and destination host. It then continuously sends packets of fixed sizes from the source to the destination thereby reducing the end-to-end bandwidth from the source to the destination host. At a given point of time, a random number (between 1-8) of source-destination pairs is chosen out of all possible source-destination pairs in the system. Random amounts of network loads are introduced on the links between the source-destination pairs by running the synthetic network program so as to vary the available bandwidths of the links between the hosts from 2% to 80% of the total bandwidth capacities of the links. The network loads are maintained for random durations between 3-8 minutes. Similar to CPU loading, the network loading process is repeated continuously.

During the course of an application execution, available CPUs of the nodes and available bandwidths of the inter-node links are collected every 2 minutes. Due to the restrictions of NWS tool on the 4-way SMP system, we were able to collect the available CPU values only on a per-node basis and not on a per-processor basis in the 32-processor system. This introduces some measurement inaccuracies and hence

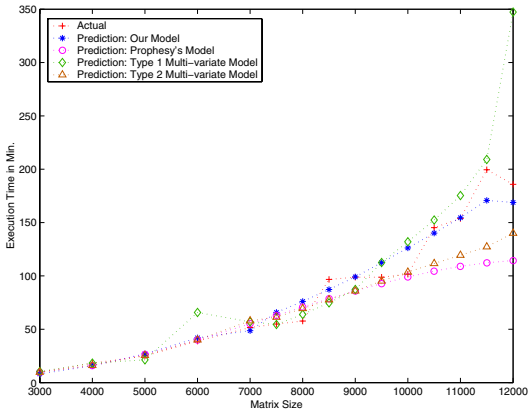

Figure 2. Predictions for Eigen Value Problem on 8 Intel Processors

leads to modeling errors as will be shown in the subsequent subsections. The following subsections report the results for the three applications.

4.1 ScaLAPACK Eigen Value

As mentioned in the previous section, cubic computation complexity and quadratic communication complexity in terms of problem size were used in the rational polynomial based models for ScaLAPACK parallel eigen value problem. On the Intel system, problem sizes from 3000 to 7000 were used for training the model and problem sizes from 7500 to 12000 were used for predictions. On the IBM system, problem sizes from 5000 to 12000 were used for training and problem sizes from 12500 to 20000 were used for predictions. Figures 2 and 3 show the predictions by the model on 8 Intel Pentium IV processors and 32 IBM Power 5 processors respectively. The average percentage prediction errors were 11.86% on the Intel system and 15% on the IBM system.

Figure 2 also shows 3 other results corresponding to 3 different models. Prophesy [13, 14] and many other previous curve-fitting approaches follow a simple one-dimensional polynomial model of degree n for curve fitting when the complexities of the application are polynomials of degree n. For the ScaLAPACK eigen value problem, the model that will be used by Prophesy is $ax^3 + bx^2 + cx + d$. This model, as Prophesy recognizes, does not separate system and application parameters and encapsulates all kinds of system dynamics in the coefficients. As can be seen in the result, even though the average percentage prediction error is 16.42%, the execution times predicted by the model do not match with the actual results as closely as our rational polynomial-based model. The other two results correspond to two simple generic multi-variate models in-

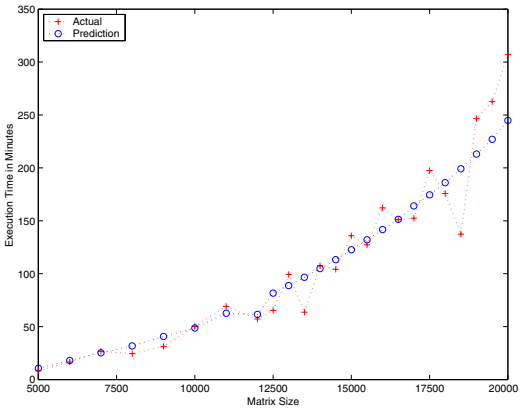

Figure 3. Predictions for Eigen Value Problem on 32 IBM Processors

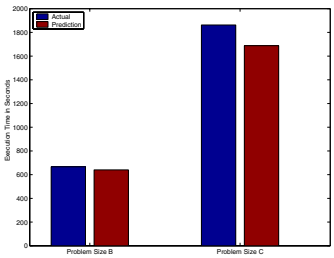

Figure 4. Predictions for Conjugate Gradient Problem on 8 Intel Processors

volving 3 variables of problem size, *min_avg_avail_cpu* and *min_avg_avail_band*. For these two models, we used the following generic 3-dimensional equation for the models.

$$\frac{an^3 + bn^2 + cn + d}{g(min\_avg\_avail\_cpu, min\_avg\_avail\_band)} \quad (6)$$

where g is a function of 2 variables. For simplicity, we used the following two functions, involving variables of degree 1, found in popular curve-fitting packages [5]:

$$g = 1 + (e \times min\_avg\_avail\_cpu) + (f \times min\_avg\_avail\_band) \quad (7)$$

$$g = 1 + e \times min\_avg\_avail\_cpu \times min\_avg\_avail\_band \quad (8)$$

e and f are the coefficients of the variables used in the calculation of g. The substitution of functions 7 and 8 in Equation 6 result in 2 models referred to as *type 1 multi-variate* and *type 2 multi-variate*. The presence of 1 in the functions help avoid homogeneity in Equation 6. As seen in Figure 2, the execution times predicted by the generic multi-dimensional models don't match with the actual results as closely as our model. The average percentage prediction errors of *Type 1 Multi-variate* and *Type 2 Multi-variate* models are 20.15% and 13.84% respectively. By careful seggregation of initiation, computation and complexities and associating *min_avg_avail_cpu* and *min_avg_avail_band* system related terms with computation and communication complexities, respectively, our model is able to give better predictions than the multi-dimensional models where the system related terms are encapsulated in generic functions without specific associations. In future, we plan to compare our model with more multi-variate models.

Comparing the predictions on the Intel and the IBM systems, we find that the model gave better predictions on the Intel system than on the IBM system. This is because the periodic available CPU values used for calculating *min_avg_avail_cpu* were collected on a per-processor basis on the Intel system whereas they were collected on a per-node basis on the 32-processor IBM system. Thus, the available CPU of a node in the IBM system represents an approximation for 4 processors and hence lead to inaccuracies in the *min_avg_avail_cpu* values. Since the model is parametrized by the *min_avg_avail_cpu*, the model results in more prediction errors on the IBM system than on the Intel system.

4.2 NPB CG

For the NPB's parallel CG application, the complexities that were used for the model are linear in terms of problem size for both communication and computation. The CG application has different classes corresponding to different problem sizes. We used classes S, W, A for training the model and used classes B and C for validating the predictions of execution times. The bar chart in Figure 4 shows predicted and actual execution times for classes B and C for 8 Intel processors. Similar to the parallel eigen value problem, we observe good predictions for the CG application with the rational polynomial based model. The average percentage prediction error was 7%.

The bar chart shows that as the problem size was increased, the model gave large prediction error. This is because the problem sizes used in training, namely classes S, W and A, were able to complete under less than 2 minutes even in the presence of external loads. Hence, only one set of available CPU and available bandwidth values were collected for each processor at the beginning of application execution. In this case, the *avg_min_avail_cpu* and *avg_min_avail_band* values degenerate to *min_avail_cpu* and *min_avail_band*, respectively, collected at the beginning of application execution. Thus the model trained with these measurement values gave large prediction inaccuracies for large problem sizes where the loads on the system vary drastically during the application execution.

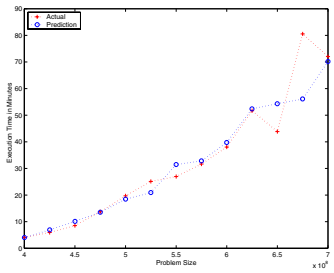

Figure 5. Predictions for FFT Problem on 32 IBM Processors

4.3 FFTW's Parallel FFT

Computation complexity of O(NlogN) and communication complexity of O(N) were used for the model for parallel FFT application. On the IBM system, problem sizes from 400 million to 550 million were used for training the model and problem sizes from 575 million to 700 million were used for predictions. Figure 5 shows predicted and actual execution times for 32 processors in the IBM system.

The average percentage prediction error for FFT application was 11%. Although the prediction error is low, we find that the predictions differ from actual values by large amounts for 650 million and 675 million problem sizes. This is due to the errors in forecasting of *min_avg_avail_cpu* and *min_avg_avail_band* values by NWS. Although the forecasting algorithms used in NWS give good predictions, accurate forecasting is difficult to achieve for drastic changes in measured values as is evident for 650 million and 675 million problem sizes shown in Figure 5. Also, during such drastic load changes, our method of averaging used in the calculation of *min_avg_avail_cpu* and *min_avg_avail_band* values does not adequately capture the drastic load dynamics.

5 Related Work

Some modeling strategies including PACE [8] and POEM [1] depend on the availability of the source codes for applications for dynamic instrumentation and task graph construction and analysis. Our modeling strategy works with the executable binaries for the applications. Most of the curve-fitting models [13] assume uniform loading conditions when experiments for modeling are conducted and when predicting execution times. Prophesy's [13,14] curve-fitting model utilizes both experimental results and the complexities of the applications to predict execution times of applications for larger problem sizes. The curve-fitting models based on linear and non-linear regression techniques have the disadvantages mentioned earlier. Prophesy also uses parametrized analytical models to address the Parametrized analytical models have several drawbacks when compared to our approach. The biggest drawback is that it assumes that the model developer has detailed knowledge about the components of the application while our model only needs computation and communication complexities. Secondly, benchmark experiments have to be conducted for various application specific subcomponents including broadcasts, floating-point additions, multiplications etc. The number of such benchmarks increase as more applications are added since different applications have different subcomponents. Our model uses only generic benchmarks relating to CPU loads, and latencies and bandwidths of network links for all applications. These benchmarks are integral to many parallel, distributed and Grid systems.

The only efforts regarding predicting execution times on non-dedicated systems that we are aware of are the efforts by Dinda [6] and Schopf et. al. [9–11]. The work by Dinda analyzes the impact of system loads on predicted execution times. Their experimental results, where they analyze the confidence intervals of their predictions, are reported only for sequential application traces. The work that is most closely related to our work is the one by Schopf and Berman [9–11]. Their structural prediction modeling approach builds a model for an application in terms of the models for the components of the applications. The motivation and the scope of their work is similar to ours in that they try to predict execution times in dynamic non-dedicated environments where the load can vary at different times. They utilize arithmetic operations based on stochastic values to obtain stochastic prediction execution times. Unlike our work, their work requires profiling tools for determining the communication and computation requirements of the application. Their component models are also based on detailed parametrized analytical models requiring user intervention for conveying the complexities of the application in terms of system and application characteristics and hence have the same drawbacks as Prophesy's parametrized analytical models. The biggest advantage of our model is that it gives predicted execution times in non-dedicated environments as point values rather than stochastic values. For large load dynamics, stochastic techniques give large ranges of predicted execution times and hence the usefulness of such large ranges are unclear.

6 Conclusions

Modeling parallel applications using curve fitting models based on experiments to predict execution times has been found to be an attractive approach. In this paper, we evaluated a rational polynomial based strategy for modeling. We find that the models based on rational polynomials satisfy the desirable properties of modeling, namely, ac-

curacy, simplicity and less time consuming than the other models. We evaluated the rational polynomial based models based on experimental results for 3 applications on small and large parallel systems with both small and large load dynamics. We used the approximate complexities of the applications and novel forecasting approach to predict the system load dynamics for parameterizing the model for the applications. Overall, the rational polynomial based models gave less than 20% average percentage prediction errors in most cases.

7 Future Work

Techniques will be developed to predict execution times for different types of load dynamics in the environment and for varying number of processors. In our current approach, we expect that approximate complexities of the application are provided to our models by the user. We plan to develop systematic methods to determine the approximate complexities of any application automatically. Specifically, we plan to include a preprocessing step to our modeling where lightweight curve-fitting of the execution times collected during the training phase will be employed. The execution times will be curve-fitted with a list of popular polynomial, logarithmic and exponential functions. The set of functions that closely fit the execution times data will then be used for approximate complexities in our model. The approximate curve-fitting will be first employed on data collected for single processor executions to determine computation complexities. Having fixed the computational complexities, the model data collected on sample multi-processor executions will be curve-fitted with the popular functions to determine approximate communication complexities.

We plan to augment our techniques for predicting execution times for complex multi-phase and multi-component applications where the computation and communication complexities can drastically change between different phases of application execution. Techniques will be developed to determine the number and nature of different phases using change in execution patterns. We also plan to extend our model to include I/O related parameters for predicting the behavior of I/O intensive scientific applications dealing with large datasets.

References

[1] V. Adve and M. Vernon. Parallel Program Performance Prediction using Deterministic Task Graph Analysis. *ACM Transactions on Computer Systems*, 22(1):94–136, 2004.

[2] R. Badia, J. Labarta, J. Gimenez, and F. Escale. DIMEMAS: Predicting MPI Applications Behavior in Grid Environments. In *Workshop on Grid Applications and Programming Tools (GGF8)*, Seattle York, U.S.A., June 2003.

[3] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, Nasa Ames Research Center, December 1995.

[4] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[5] DataFit. `http://www.curvefitting.com/datafit.htm`.

[6] P. Dinda. Online Prediction of the Running Time of Tasks. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, pages 383–394, San Francisco, U.S.A., August 2001.

[7] M. Frigo and S. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on Program Generation, Optimization, and Platform Adaptation.

[8] G. Nudd, D. Kerbysin, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems. *The International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[9] J. Schopf. Structural Prediction Models for High-Performance Distributed Applications. In *Proceedings of the Cluster Computing Conference (CCC '97)*, Atlanta, U.S.A., March 1997.

[10] J. Schopf and F. Berman. Performance Prediction in Production Environments. In *Proceedings of the 12th. International Parallel Processing Symposium*, page 647, Orlando, U.S.A., March 1998.

[11] J. Schopf and F. Berman. Using Stochastic Information to Predict Application Behavior on Contended Resources. *International Journal on Foundation in Computer Science*, 12(3):341–364, June 2001.

[12] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A Framework for Performance Modeling and Prediction. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–17, Baltimore, U.S.A., November 2002.

[13] V. Taylor, X. Wu, J. Geisler, X. Li, Z. Lan, M. Hereld, I. Judson, and R. Stevens. Prophesy: Automating the Modeling Process. In *Proceedings of the Third Annual International Workshop on Active Middleware Services*, pages 3–11, Tokyo, Japan, August 2001.

[14] V. Taylor, X. Wu, and R. Stevens. Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):13–18, March 2003.

[15] R. Wolski. Dynamically Forecasting Network Performance using the Network Weather Service. *Journal of Cluster Computing*, 1(1):119–132, 1998.

[16] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999.