Design and Development of the Prophesy Performance Database for Distributed Scientific Applications *

Xingfu Wu [†]***, Valerie E. Taylor*** [†]***, Jonathan Geisler*** [†]***, Xin Li*** [†]***, Zhiling Lan*** [†]***, Rick Stevens*** [‡]***, Mark Hereld*** [‡] ***and Ivan R. Judson*** [‡]

1 Introduction

Efficient execution of a scientific computing application requires insights into how system features impact the performance of the application. A distributed system consists of heterogeneous components, such as networks, processors, run-time systems, operating systems, etc. This heterogeneity complicates the task of gaining insights into the performance of the application. The Prophesy project [21] is an infrastructure that aids in gaining this needed insight based upon one's experience and that of others. The core component of the Prophesy system is a relational database that allows for the recording of performance data, system features and application details to analyze and improve the performance of scientific applications. The Prophesy infrastructure can be used to develop models based upon significant performance data, identify the most efficient implementation of a given function based upon the given system configuration, explore the various trends im-

*This research was supported in part by the National Science Foundation under NSF grant EIA-9974960 and a grant from NASA Ames.

[†]Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208.

[‡]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

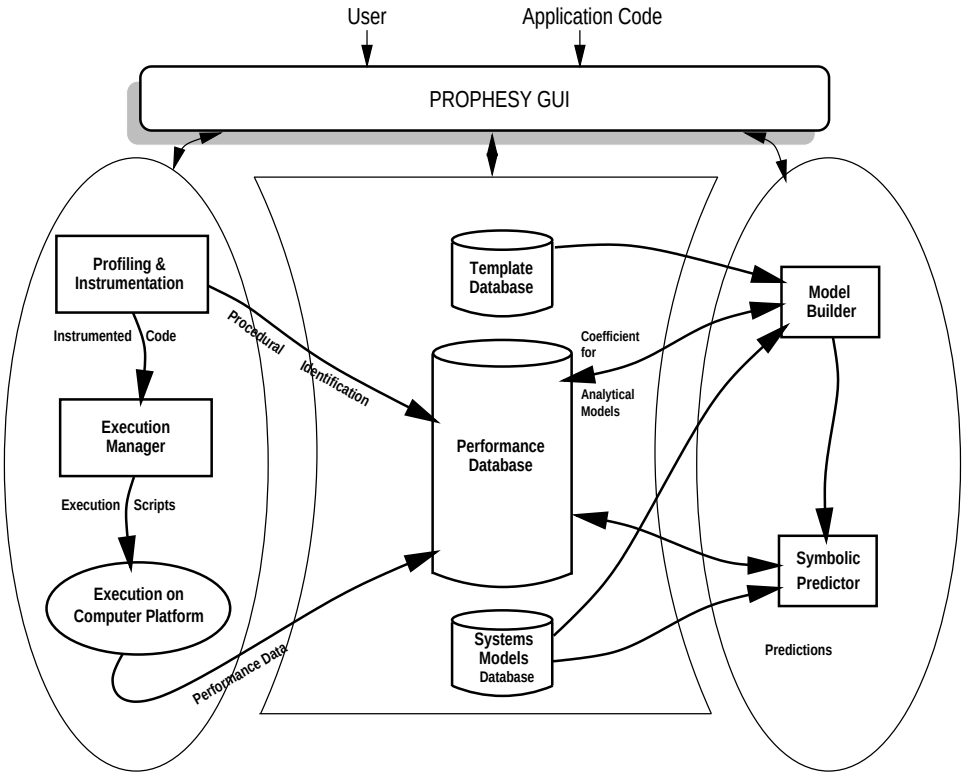

Figure 1. *Prophesy framework*

plicated by the significant data, and predict the performance on a different system. This paper focuses on the design and development of the Prophesy database and illustrates the necessity for the various entities.

The Prophesy framework consists of three major components: data collection (left section), data analysis (right section), and the three central databases, as illustrated in Fig. 1. The data collection component focuses on the automatic instrumentation of codes at the level of basic blocks, procedures, or functions. The default mode consists of instrumenting the entire code at the level of basic loops and procedures. A user can specify that the code be instrumented at a finer granularity than that of loops or identify the particular events to be instrumented. The resultant performance data is automatically placed in the performance database and is used by the data analysis component to produce an analytical performance model with coefficients, at the granularity specified by the user. The models are developed based upon performance data from the performance database, model templates from the template database, and system characteristics from the systems database. The interface uses web technology to allow users to access the Prophesy system from anywhere.

An application goes through three stages (instrumentation of the application,

performance data collection of many runs, and model development using optimization techniques) to generate an analytical performance model. The Prophesy system allows for the development of linear as well as nonlinear models. These models, when combined with data from the system database, can be used by the prediction engine to predict the performance on a different compute platform. The use of databases with the Prophesy system allows users to explore the performance models developed for different kernels, applications and systems. The data in the databases are organized in a hierarchical manner, allowing for the development of analytical models of different granularities. The Prophesy system is an infrastructure designed to explore the plausibility and credibility of various techniques in performance evaluation (such as scalability, efficiency, speedup, performance coupling between application kernels, etc.) and allow users to use various metrics collectively to bring performance analysis environments to the most advanced level. In this paper, we describe the Prophesy Database (PD), and discuss the design and development of all three databases: performance, system and template (used to identify the appropriate optimization technique for generating the models).

The remainder of this paper is organized as follows. Section 2 describes the design and development of the PD in detail. section 3 depicts an example that demonstrates the use of the PD. Section 4 compares the PD with related work. Section 5 summarizes the paper.

2 Prophesy Database

Recall that the Prophesy Database must accommodate queries that lead to the development of performance models, allow for prediction of performance on other systems, and allow for one to obtain insight into methods to improve the performance of the application on a given distributed system. Hence, the database must facilitate the following query types:

- Identify the best implementation of a given function for a given system configuration (identified by the run-time system, operating system, processor organization, etc.)
- Use the raw performance data to generate analytical (nonlinear or linear) models of a given function or application; the analytical model can be used to extrapolate the performance under different system scenarios and can be used to assist programmers in optimizing the strategy or algorithms in their programs.
- Use the performance data to analyze application-system trends, such as scalability, I/O requirements, communication requirements, etc.
- Use the performance data to analyze user specific metrics such as coupling between functions [5].

In this section, we present the details of the implementation of the database that accommodates the aforementioned query types and data uses. We first provide details about the structure of applications.

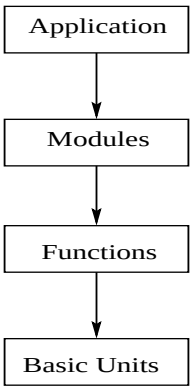

Figure 2. *Hierarchical structure of applications*

2.1 Hierarchical Structure of Applications

It is assumed that scientific computing applications have the following hierarchical structure shown in Fig. 2. The description of each structure is described in detail below. Notice that the assumption of applications is general. As an example, we shall use NAS Parallel CG benchmarks [1] to account for the hierarchical structure of the applications as follows.

- Application: refers to the complete large-scale scientific computing application, which may have different versions corresponding to the development of additional application functionalities over time. For example, for the CG benchmark, it has the different versions such as version 1.* for PVM and 2.* for MPI.
- Modules: refer to the various files that comprise the application; it is assumed that the application designer uses some modularity in the application design. For example, the parallel CG program consists of three Fortran files: cg.f, randi8.f and print_results.f. For some other applications, it is possible that different modules are implemented in different programming languages such as C, C++, Fortran77, Fortran90, HPF, etc.
- Functions: refer to the different function routines that may be contained in a given module or file. For example, the module cg.f includes functions such as makea(), conj_grad(), and so on. Users will be asked to associate a "pure function" name with their given function where appropriate. For example, a user may identify their function "genfft" as the pure function FFT or the function conj_grad as a Conjugate Gradient function. Pure functions are used to facilitate the query types that seek the best implementation of a given function. By associating a pure function name with a given function, we eliminate the need to try to identify a function based upon a function name, which may be very obscure.
- Basic Units: refer to a code segment that is of smaller granularity than a

function but higher granularity than a basic block. Currently, basic unit is defined as the code between two high-level language control constructs. For example, a set of nested 'do loops' may be considered one basic unit.

2.2 Database Organization

The PD has a hierarchical organization, consistent with the hierarchical structure of the applications. The Entity-Relationship (ER) diagram for the PD is given in Fig. 3; this diagram was developed using the Dezign database design tool [6]. The PD includes all three databases given in Fig. 1: performance database, system models database and template database. The entities in Fig. 3 are organized into four areas: application information, executable information, run information and performance statistics. Descriptions of these four areas are given below.

- Application Information: includes two entities: Application and Owner. The Application entity gives the application name, version number, a short description and the application owner name. It is assumed that an application goes through various versions as one adds different functionalities over time. Data is placed into this entity when a new application is being developed. The addition of a new version of a given application is reflected in in the Executable Information, for which it is assumed that a new executable would be generated for this new version. The Owner entity includes the owner name, password and the owner's email address. The password is used so that information relating to the application can only be modified or updated by the owner.
- Executable Information: includes all of the entities related to generating an executable of an application. These entities include details the modules and functions that comprise the executable; the compilers and compiler flags used for each module to generate the executable; the libraries used with the application; the control flow at the level of functions; and the model template which identifies the optimization method used to generate a model for the given function. It is important to keep all of this information because of the impact that each has on the performance of the application. For example, it is well known that compiler options significantly impact the performance of an application. This impact is illustrated below in Table 1 that gives the timing results for matrix multiply on POWER2 Super Chip using IBM xlf Fortran compiler and KAP* preprocessor [3]. It is assumed that applications may be developed using multiple languages, such as C, C++, Fortran77 and Fortran90. The Executable Information consists of the following entities: Executable, Modules, Module_Information, Functions, Function_Information, Model_Templates, Model_Information, Compiler, Library and Control_Flow. Data is placed into these entities when a new executable is generated.
- Run Information: includes all of the entities related to running an executable, which includes the system information, inputs used for execution and the data and time of the run. This system may be a single processor, single parallel machine or distributed system. The system used for execution is described

Table 1. *Effect of optimizers and precompilers on naive* 512×512 *matrix-matrix multiply.*

Compiling Options	Runtime (seconds)
none	48.89
-O2	39.15
-O3	37.69
-Pk	13.74
-O3 -Pk	1.61

in terms of the different resources in the system and the interconnection of these resources. The system entities does not make any assumptions about the memory system or the interconnection between processors, which is very important and significant. By describing a system in terms of resources and interconnections of resources we are able to characterize existing systems such as the SGI Origin or Sony PlayStation2 or future systems that are still being researched such as the HTMT [8]. Further the system entities allow for easy characterization of distributed systems, which have multiple interconnects. The Run Information consists of the entities Run, Inputs, Systems, Connection_Information and Resource_Types. Data is placed into these entities for each run of a given executable.

- Performance Statistics Information: includes all of the entities related to the raw performance data collected during execution. Performance statistics are collected at the granularity specified by the user. The default granularity is at the lowest level, the basic unit, resulting in performance information being available at the level of the application, function and basic unit. For the function-level performance statistics, we keep up with information by the caller of a function. Hence if two functions A and B call a third function C, the performance statistics for C are distinguished by that associated with function A calling function C and function B calling function C. This is included to aid in identifing and tracing bottlenecks in the application. Further, we collect statistics about data structures, including cache misses and number of accesses, when available. Such statistics can be collected using PAPI [13], which provides an interface to the hardware counters. The Performance Statistics Information consists of the entities Application_Performance, Function_Performance, Basic_Unit_Performance and Data_Structure_Performance. Data is placed into these entities for each run of an executable.

The Prophesy Database uses the PostgreSQL [15], which is supported on many platforms. The use of the Dezign tool to develop the PD schema, however, allows for ease of use of other database systems such as Sybase, DB2, Informix and MySQL. We use Perl scripts to enter the performance data directly into the Prophesy database.

3 Prophesy Database Usage

In this section, we discuss how the PD can be used to generate performance models, identify the best implementation, and allow for one to obtain insight into methods to improve the performance of the application on a given distributed system. The Prophesy infrastructure includes automatic instrumentation and automatic model development, resulting in the processes described below being fully automated.

Performance models are important for understanding the performance of the algorithms used in an application and predicting the performance of an application on different systems. To generate a performance model with the PD, the following information is used: executable information, run information and performance statistics. Prophesy currently supports two distinct model types: (1) Parameterization, which is based on system information and the given application, and (2) Curve Fitting, which is based on the empirical data. Parameterization utilizes the system data (e.g., communication and computation parameters for the various resources in the system, obtained from the systems database), the control flow information, the pure functions identified in the application, and the model templates for the pure functions. For the case when a pure function is not used in the application, a user can specify the model type (e.g., linear, affine, quadratic, cubic, etc.) to be used for the full application or the different modules, as determined by the desired level of granularity needed for the performance model. A user can request that this model, which is based on system and application parameters, be compared to some empirical data, if desired. Curve Fitting uses the empirical data found in the database to generate the model. Users are prompted to identify the range of data to be used for the curve fitting. Such models have been developed for some small kernels such as matrix-matrix multiplication and conjugate gradient.

To identify the best implementation of a function, such as FFT or conjugate gradient, one would query the PD with the desired pure function name (selected from a pull down menu) and the given system (also selected from a pull down menu). The result of this query is a pointer to the implementation that has the best execution time, a pointer to the owner, and the compiler and compiler flags needed to get the results. Hence, a user can take this information and contact the owner of the application with the efficient function to get the actual code. This code can then be easily used in an application.

The Prophesy infrastructure is targeted to the community in general. It is our goal that many application developers will use the Prophesy system such that the high performance computing community can benefit from each other. In this way, a user can query the Prophesy system to get information such as the performance of many different applications executed on a particular system, such as a linux cluster or SGI Origin. For such a query, the PD would return the results of all the applications executed on the desired system. This information could be used to generate scalability plots, efficiency plots, or plots about a particular component such as I/O or the memory subsystem. The plots would provide insights about different classes of applications. Such plots have been generated for systems such as a linux cluster, for which we have significant data about different applications.

4 Related Work

There exist different approaches to organizing performance data by using database techniques. For example, Snodgrass [19] developed a relational approach to monitoring complex systems by storing the information processed by a monitor into a historical database. The basic idea is to use historical databases to formalize dynamic information. The SIEVE (Spreadsheet based Interactive Event Visualization Environment) system [18] maintains dependence graph information in a static database and tracefile information in a dynamic data base. Users may select columns from spreadsheet and associate those with graphical objects for display. However, it depends on those instrumentations and tracefiles. The PDS (Performance Database Server) system [7, 10] was specifically designed with a simple tabular format that involves displaying the data in rows (machine configuration) and columns (numbers). It logically organizes data according to the benchmarks themselves: a Linpack table, a Perfect table, etc. In contrast, the Prophesy database is focused on data collection of the complete system, keeping information such as compiler type and operating system. Further, detailed information is kept such as data motion within the processor as well as between processors. Lastly, the Prophesy system is a complete system, which includes automatic instrumentation as well as automated performance model development.

Significant work has been done with developing performance tools such as Pablo [16], AIMS [20], or Paradyn [11]. These tools, while significant, have a different focus than that of Prophesy. Prophesy includes the automatic generation of performance models; the models may be linear or nonlinear. The aforementioned tools provide the mechanisms for collecting the data, but do include a database of performance data or generate the models. The model development step, which is very time consuming, is left to the user.

Performance analysis environments, in particular PACE [9] and POEMS [14], are being developed. These environments focus on performance predication. PACE represents the application, computational resource requirement and communication patterns in their $CHIP^3$ language. The $CHIP^3$ scripts are compiled and evaluated to generate a performance prediction very quickly. POEMS evaluates the end-to-end performance of a problem solving environment, consisting of application software, runtime and operating system software and hardware architecture. The analytical models with POEMS include deterministic task graph analysis, LogP [4] and LoGPC [12] models. These models are generally coarse grain, representing asymptotic performance. In contrast, the focus of Prophesy is on detailed, analytical model development; Prophesy explores nonlinear as well as linear models with different levels of granularity. Further, Prophesy complements the PACE and POEMS environments by providing a framework for developing models that can be added to their various libraries. In addition, Prophesy is based on advanced database and web technology, allowing users from anywhere to access the performance data, add performance data, or utilize the automated processes.

5 Summary

This paper presents the design and development of a platform-independent Prophesy performance database that allows for the recording of performance data, system features and application details in order to improve performance of scientific computing applications and to reduce high cost of the iterative application software development. It shows that the historical performance database is an appropriate model for processing dynamic performance data of scientific computing applications, and provides a convenient and powerful management system that guides the performance data processing but does not constrain the representation of performance data. We hope that the PD can provide computer system vendors with valuable feedback on system bottlenecks that can be eliminated in future databases.

The Prophesy system is applicable to single processor, parallel machine and distributed system environments. The PD is hierarchical, from which different granularities of performance information may be obtained. We illustrated the use of the PD with discussing how the PD can be used to generate performance models, identify the best implementation, and allow for one to obtain insight into methods to improve the performance of the application on a given distributed system. The PD allows users to gain needed insights into application performance based upon their experience as well as that of others. Currently, we are populating the PD with performance information from some of the PACI applications and NAS serial and parallel benchmarks. Our goal is to have the PD populated by results from numerous research groups, thereby allowing us to gain insights from each other's experiences.

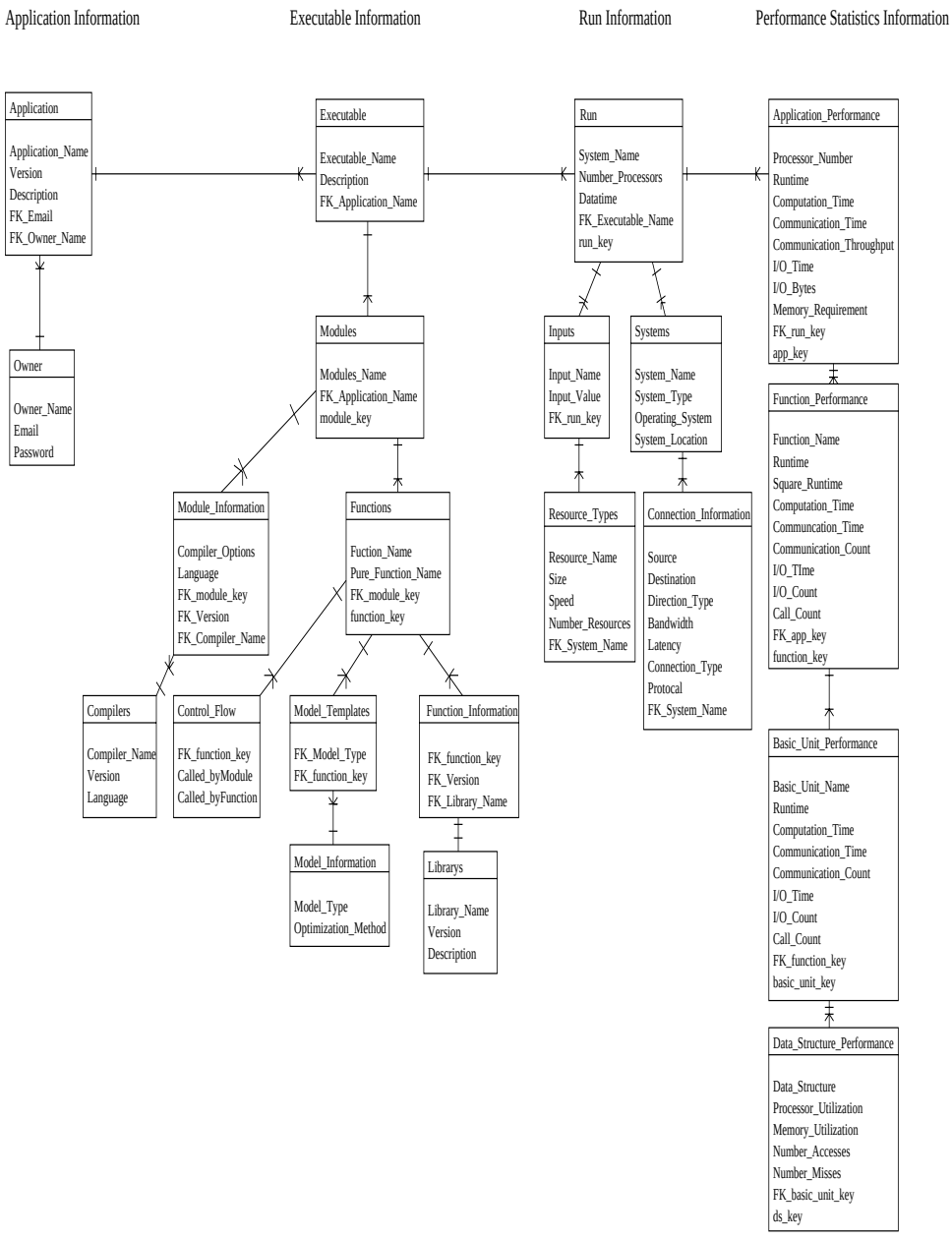

Figure 3. *Framework of Prophesy Database Schema*

Bibliography

[1] D. Bailey, T. Harris, et al., *The NAS Parallel Benchmarks*, Tech. Report NAS-95- 020, Dec. 1995. See also http://science.nas.nasa.gov/Software/NPB/.

[2] T. Boutell, *CGI Programming in C & Perl*, Addison-Wesley Developers Press, 1996.

[3] Cornell Theory Center, http://www.tc.cornell.edu/Edu/Talks/Performance/SingleProcPerf/.

[4] D. Culler, R. Karp, D. Patterson, et al., *LogP: Towards a realistic model of parallel computation*, Proc. of 4th ACM SIGPLAN Conf. on Para. Prog. Pract. and Exp., 1993.

[5] J. Geisler and V. Taylor, *Performance coupling: A methodology for predicting application performance using kernel performance*, Proc. of the 9th SIAM Conference on Parallel Processing for Scientific Computing, March 1999.

[6] Heraut Automatisering, *DeZign for database*, version 2.3, http://www.heraut.demon.nl/dezign/dezign.html.

[7] R. Hockney and M. Berry, *Public International Benchmarks for Parallel Computers*, PARKBENCH Committee: Report-1, February 7, 1994.

[8] J. N. Amaral, G. R. Gao, P. Merkey, T. Sterling, Z. Ruiz and S. Ryan, *Performance Prediction for the HTMT: A Programming Example*, Proc. of the Third PETAFLOP workshop, Feb. 22, 1999.

[9] D. Kerbyson, J. Harper, et al., *PACE: A toolset to investigate and predict performance in parallel systems*, Proc. of European Parallel Tools Meeting, Oct. 1996.

[10] B. H. LaRose, *The Development and Implementation of a Performance Database Server*, Master Thesis, University of Tennessee at Knoxville, August 1993.

[11] B. P. Miller, M. D. Callaghan, et al., *The Paradyn parallel performance measurement tools*, IEEE Computer, Vol. 28 (11), Nov. 1995.

[12] C. A. Moritz, and M. I. Frank, *LoGPC: modeling network contention in message-passing programs*, Proc. of Intern. Conf. on Meas. and modeling of computer systems, June 1998.

[13] S. Browne, J. Dongarra, N. Garner, K. London and P. Mucci, *A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters*, Proc. of Supercomputing 2000, Nov. 2000.

[14] *POEMS Performance Environment*, http://www.cs.utexas.edu/users/poems.

[15] *PostgreSQL 7.0.2*, http://www.PostgreSQL.org/docs/

[16] D. A. Reed, R. A. Aydt, et al., *Scalable performance analysis: The Pablo performance analysis environment*, Proc. of the Scalable Parallel Libraries Conference, Oct. 1993.

[17] R. L. Schwartz and T. Christiansen, *Learning Perl*, O'Reilly & Associates, Inc., 1997.

[18] S. R. Sarukkai, and D. Gannon, *SIEVE: A performance debugging environment for parallel program*, Journal of Parallel and Distributed Computing 18 (1993), 147-168.

[19] R. Snodgrass, *A relational approach to monitoring complex systems*, ACM Transactions on Computer Systems, Vol. 6, No. 2 (1988), 157-196.

[20] J. C. Yan, S. R. Sarukkai, and P. Mehra, *Performance measurement, visualization and modeling of parallel and distributed programs using the AIMS toolkit*, Software Practice and Experience, Vol. 25 (4), April 1995.

[21] Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling Lan, Rick Stevens, Mark Hereld and Ivan R. Judson, *Prophesy: An Infrastructure for Analyzing and Modeling the Performance of Parallel and Distributed Applications*, Proc. of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC's 2000), IEEE Computer Society Press, Pittsburgh, August 2000.