Performance Modeling and Prediction of Parallel and Distributed Computing Systems: A Survey of the State of the Art *

Sabri Pllana, Ivona Brandic and Siegfried Benkner
University of Vienna, Institute of Scientific Computing
Nordbergstrasse 15, 1090 Vienna, Austria
T. +43 1 4277 39411, F. +43 1 4277 9394
{pllana,brandic,sigi}@par.univie.ac.at

Abstract

Performance is one of the key features of parallel and distributed computing systems. Therefore, in the past a significant research effort was invested in the development of approaches for performance modeling and prediction of parallel and distributed computing systems. In this paper we identify the trends, contributions, and drawbacks of the state of the art approaches. We describe a wide range of the performance modeling approaches that spans from the high-level mathematical modeling to the detailed instruction-level simulation. For each approach we describe how the program and machine are modeled and estimate the model development and evaluation effort, the efficiency, and the accuracy. Furthermore, we present an overall evaluation of the presented approaches.

1 Introduction

The solution of resource-demanding scientific and engineering computational problems involves the execution of programs on parallel and distributed computing machines, in order to solve large problems or to reduce the time to solution for a single problem [9]. However, the development of this kind of computing systems is an expensive and time-consuming endeavor. For instance, the development cost of the Earth Simulator Center (ESC) [7] was about US$350 million [21], and its development took about five years. While the life of parallel and distributed computing machines is commonly up to five years long, the life of parallel and distributed programs is up to 30 years [6]. Therefore, it is important to have means for the performance evaluation of programs not only on existing machines, but also on machines that are under development or being planned.

*The work described in this paper was partially supported by the Austrian Science Fund (FWF) under the project AURORA.

Because the performance is a key indicator of computing systems, the performance evaluation was a preoccupation of many computer scientists in the past [1, 17, 10, 13, 15]. The commonly used techniques for the performance evaluation of computing systems include: measurement, mathematical modeling, and simulation. Each of these techniques has its limitations. Measurement techniques require that the system under study is available for experimentation, and their applicability is limited to only existing systems. Mathematical performance models usually lack the system's structural information, and therefore, are not suitable for the model-based performance analysis. The model-based performance analysis involves the modification of structure of the model to reflect system structural changes, in order to predict what would be the performance of the system under study if its structure is changed. Detailed simulation models demand large computational and storage resources, and their evaluation may be so slow that the performance assessment of real-world programs is impractical.

In this paper we systematically present a collection of the state of the art approaches for performance modeling and prediction of parallel and distributed computing systems. The range of the approaches for performance modeling and prediction that we cover spans from the high-level mathematical modeling to the detailed instruction-level simulation. For each approach we describe how the program and machine are modeled and estimate the model development and evaluation effort, the efficiency and the accuracy. In addition, we evaluate the suitability of the presented approaches for the model-based performance analysis of parallel and distributed computing systems.

The rest of this paper is organized as follows. Preliminaries are described in Section 2. Section 3 describes a collection of approaches for performance modeling and prediction of computing systems. An evaluation of the described approaches is presented in Section 4. Finally, Section 5 presents some concluding remarks.

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

2 Preliminaries

The terms *workload*, *machine*, and *system* are used frequently in the literature, but not always with the same meaning. In the context of this paper the term *workload* indicates a distributed and parallel program, whereas the term *machine* indicates a distributed and parallel computing architecture. The term *system* indicates a computing system. In our approach, the workload model and the machine model are considered as integral parts of the computing system model (see Figure 1).

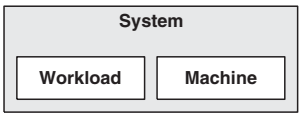

Figure 1. Computing system model.

In this paper, for each performance modeling and prediction approach, we describe how the workload and machine are modeled. In addition, the following properties of approaches are discussed:

Model development effort indicates the endeavor of the user of a specific approach for building the model.

Model evaluation effort indicates the time and the computing resources that are needed to evaluate the model.

Efficiency of the approach considers the effort needed to develop and evaluate the model. For instance, the efficiency of an approach is high if the model development and evaluation effort is low.

Accuracy indicates the exactness of an approach. For estimation of the accuracy is commonly performed a comparison of prediction results with measurement results.

3 Performance Modeling and Prediction Approaches

In this section we present some of the most relevant approaches for performance modeling and prediction of parallel and distributed computing systems.

3.1 A1: PAL

This approach for the performance modeling and prediction of distributed computing systems is commonly used by the Performance and Architecture Laboratory (PAL) at Los Alamos National Laboratory [14, 15].

The PAL approach expresses the execution time of a program on a machine as a parameterized mathematical model. The model parameters characterize the problem size S, and computational and communicational capabilities of the machine M. The program execution time $T_{ProgExec}$ is estimated as follows,

$$T_{ProgExec}(S,M) = T_{Comp}(S,M) + T_{Comm}(S,M) + \\ + T_{MemCont}(S,M) \quad (1)$$

where T_{Comp} is the computation time, T_{Comm} is the communication time, and $T_{MemCont}$ is the time spent for memory contention within a multiprocessor node. By expressing the performance of the whole program with a mathematical expression the structural information (for instance the control flow) of the program is not preserved. For this reason, this approach may not be suitable for the model-based performance evaluation of various program designs.

Workload model. The development of the workload model is based on a detailed analysis of the source code of the program (such as counting computation and communication operations). The modeling procedure results with a parameterized mathematical model (see Equation 1).

Machine model. Machine is not modeled separately. Machine is characterized by a set of parameters such as number of processors per node, number of nodes, number of communication links per node, communication latency and bandwidth, and sequential processing capacity. These parameters serve as input for the model that predicts the program execution time.

Model development effort. PAL modeling approach requires a careful analysis of the program control flow and data structures. Furthermore, computer measurement techniques are used for determining the values of various machine parameters such as communication latencies and bandwidths, sequential processing capacity, and memory contention. Therefore, the modeling effort for this approach is considered to be high.

Model evaluation effort. The modeling procedure results with a mathematical expression, which is suitable for fast evaluation. Therefore, the model evaluation effort for this approach is low.

Efficiency. We consider that the efficiency of this approach is medium, since the model development effort is high but the model evaluation effort is low.

Accuracy. In [15] authors of PAL approach report performance prediction results for various systems with average accuracy between 5% and 11%. We should emphasize that the performance effects of the overlapping of computation and communication phases are not considered. Therefore, models that are built based on PAL approach may provide accurate results only for that class of programs for which the overlapping of computation and communication does not affects significantly the overall program performance.

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

3.2 A2: Performance Prophet

Performance Prophet [18] is a tool for performance modeling and prediction of parallel and distributed computing systems. This tool provides a graphical user interface, which alleviates the problem of specification and modification of the performance model. The user specifies graphically the performance model using the Unified Modeling Language (UML) [17, 16]. Afterwards, Performance Prophet automatically transforms the performance model from UML to C++ and evaluates it by simulation.

Performance Prophet is suitable for exploring a large set of possible program's versions, because of its efficiency of model building and evaluation. The rapid performance model evaluation capability of Performance Prophet is due to a methodology that involves model simplification, and the combination of mathematical modeling with discrete event simulation. The aim is to combine the model evaluation efficiency of mathematical performance models with the structure awareness of simulation models. The behavior of the whole computing system is split-up into *action states* and *waiting states*. Mathematical modeling is used for modeling the performance behavior of action states, whereas the performance behavior of waiting states is simulated.

Workload model. The workload model is specified graphically based on UML.

Machine model. The machine model for clusters of SMP's is composed automatically based on the user-specified parameters such as the number of nodes and the number of processors per node.

Model development effort. Commonly, the model is composed rapidly from the existing building blocks. In this case the model development effort is low. However, if the user needs to develop new building blocks, then the model development effort is increased significantly. Therefore, in the general case the model development effort is medium.

Model evaluation effort. The time needed for model evaluation is reduced, by (1) simplifying the model, and by (2) combining mathematical modeling and discrete event simulation. Therefore, the model evaluation effort is low. In a case study presented in [18], the model evaluation with Performance Prophet on a single processor workstation was several thousand times faster than the execution time of the real program on a real cluster.

Efficiency. We consider that the efficiency of this approach is high, since the model development effort is medium but the model evaluation effort is low.

Accuracy. The authors have assessed the accuracy of Performance Prophet by modeling and simulating a real-world material science program that comprises about 15,000 lines of code [18]. The average prediction accuracy was about 7%.

3.3 A3: POEMS

The aim of Performance Oriented End-to-end Modeling System (POEMS) [1] project (period of performance 1997-2000) was to develop an environment for performance modeling of parallel computing systems.

POEMS proposed a methodology for the evaluation of system model using multiple evaluation tools. The model of system is composed of *component models*. POEMS authors state that each component of the system model may be evaluated by the corresponding evaluation tool; the output of a tool serves as input for the subsequent tool [1]. We consider that in the general case the component models may be of different kinds and at different levels of abstraction, and therefore the output of an evaluation tool may not be *interpretable* for the subsequent evaluation tool.

Workload model. POEMS devised a graphical representation for parallel programs, which is based on task graphs. Each node of the task graph may represent a set of parallel tasks. Edges of the task graph may represent data flow or task precedence. However, POEMS does not provide the corresponding tool-support for graphical model composition.

Machine model. The processor and the memory subsystem are simulated with SimpleScalar [4], the network is simulated based on Parsec simulation language, and I/O subsystem is simulated with PIOSIM [3].

Model development effort. A relevant outcome of the POEMS project is an automatic task graph generator for High Performance Fortran (HPF) programs [2]. The tool-support is provided by an extended version of *dHPF* [5] compiler. This approach supports the automatic model development for HPF programs. However, the automatic development of the machine model is not adequately addressed. The claim that the machine model may be automatically composed from existing components is not of particular practical relevance, if these components are not already developed.

Model evaluation effort. It is difficult to estimate, because of the large spectrum of addressed abstraction levels and proposed model evaluation tools. For equation solvers the model evaluation effort is low. But, for detailed simulators the model evaluation effort may be very high both in terms of the time and computing resources needed to evaluate the model.

Efficiency. This approach aims to incorporate a large spectrum of the performance models from the high-level mathematical models to the instruction-level simulators. Therefore, it is hard to estimate the efficiency of this approach.

Accuracy. It may be anywhere from low to high. It depends on the model abstraction level, system modeling process, and the used model evaluation tools.

3.4 A4: POSE

Parallel Object-oriented Simulation Environment (POSE) [23] is a parallel discrete event simulator [8]. POSE is used for simulation of the performance behavior of programs that are executed on large-scale machines such as IBM BlueGene [24, 11]. The BlueGene/L [11], which is listed as number one in the list of 500 hundred most powerful computers in the world (TOP500 [22], November 2006), contains 131,072 processors. A detailed simulation of such large machines may require processing and memory resources that are not available on a singe processor machine. Therefore, such simulations are executed on multiprocessor machines.

POSE is implemented based on Charm++ [12], which is a parallel C++ library. The simulation entities of POSE are represented as Charm++ objects. Each object has a data member for tracking the simulation time and a set of methods for event handling. Based on POSE and the Charm++ runtime environment a specific simulator that simulates the BlueGene/L machine was developed [24]. The rest of this section provides a discussion of properties of the BlueGene/L simulator.

Workload model. The BlueGene/L simulator is an execution-driven simulator. It supports the simulation of programs that are written based on Charm++ or MPI.

Machine model. Machine is modeled as a set of interconnected nodes. Each node may have a set of processors.

Model development effort. Charm++ or MPI programs may be used as input for the simulator. In this case the workload modeling effort is low. However, in order to evaluate different program versions the user has to restructure the source of the program. This process may be time consuming for the real-world programs. The effort for the development of a detailed machine model that supports execution-driven simulation is high.

Model evaluation effort. Commonly sequential discrete event simulators are about 10 times faster than parallel discrete event simulators if a single-processor machine is used for model evaluation. This is due to the synchronization and communication overhead of parallel discrete event simulators. If a machine with more than 10 processors is available for model evaluation, then parallel simulators outperform sequential simulators (see [23]).

Efficiency. We consider that the efficiency of this approach is low, since the effort for the model development and evaluation is high.

Accuracy. We were unable to find a comparison of prediction results of BlueGene/L simulator with measurement results on the real BlueGene/L machine. The prediction accuracy of this approach may be anywhere from low to high, depending on the fidelity of the machine model.

3.5 A5: RSIM

Rice Simulator for ILP Multiprocessors (RSIM) is a simulator of cache-coherent non-uniform memory access (CC-NUMA) shared-memory machines [19, 10]. A distinguishing feature of RSIM is the ability to simulate processors that use instruction-level parallelism (ILP). ILP processors are capable of executing multiple instructions in parallel.

RSIM supports SPARC processors [20]. As input for RSIM simulator may serve programs that are compiled and linked (that is *executables*) on SPARC/Solaris systems. During the program simulation, RSIM interprets the executable of the program. The output of RSIM includes the number of executed cycles, and statistics on the utilization of components of the machine.

RSIM comprises a detailed (that is a cycle-level) machine model that allows the analysis of the performance effects of architectural parameters. Therefore, it is suitable to evaluate various designs of CC-NUMA shared-memory machines. However, because the simulation of the program execution with RSIM is very slow (several thousands times slower than the program execution on the real machine), it is not suitable for evaluation of various designs of real-world programs.

Workload model. The executable of the program serves as a workload. The RSIM simulator takes as input the programs that are compiled and linked on SPARC/Solaris systems.

Machine model. The main components of the machine model include processors, the memory hierarchy (that is L1 cache, L2 cache, local and remote memory), and the interconnection network.

Model development effort. The development of the workload model is straitforward. Basically, it involves program compiling and linking. But, the effort for development of the cycle-level machine model is high.

Model evaluation effort. Authors of RSIM report that several thousands times more time was needed to simulate a program with RSIM, than to execute the program on a real machine. For instance, for a program that performs *LU matrix decomposition* the RSIM simulation was 7100 times slower than the program execution on the real machine. Therefore, the model evaluation effort for this approach is high.

Efficiency. We consider that the efficiency of this approach is low, since the effort for the model development and evaluation is high.

Accuracy. The model of the computer architecture that is used by RSIM does not match exactly the architecture of any existing machine. This makes difficult the task of estimation of the prediction accuracy of RSIM, since it is not possible to compare simulation results with results that are obtained from the real computing system.

Table 1. A summary of the properties of approaches for the performance modeling and prediction of parallel and distributed computing systems.

ID	Mod.Dev.	Mod.Eval.	Efficiency	Accuracy
A1	High	Low	Medium	Medium
A2	Medium	Low	High	Medium
A3	Medium	Medium	Medium	Medium
A4	High	High	Low	Medium
A5	High	High	Low	High

4 Evaluation

In this section we evaluate the suitability of the approaches, which we described in Section 3, for the model-based performance analysis of parallel and distributed computing systems. The time spent for the model development and evaluation should be short, in order to explore a large set of design alternatives within a reasonable time. The approaches should provide performance prediction results with an *accuracy* that makes possible the *comparison* of various design alternatives. We evaluate the approaches based on the following properties: (1) model development effort, (2) model evaluation effort, (3) efficiency, and (4) accuracy. The efficiency of an approach is deduced based on the model development and evaluation effort.

Table 1 shows an estimation of values of the properties of approaches. The values of properties are obtained from our reasoning about each approach in Section 3. Approaches are arranged in the table in the order of their appearance in Section 3.

Figure 2 depicts the assessment results of approaches based on the values of their properties (see Table 1).

Figure 2(a) provides an assessment of approaches by considering the model development and evaluation effort. Approaches $A4$ and $A5$ received the grade *poor*; $A1$ and $A3$ received the grade *good*; and $A2$ received the grade *very good*.

Figure 2(b) provides an assessment of approaches by considering the accuracy and efficiency. $A4$ received the grade *sufficient*; $A1$, $A3$ and $A5$ received the grade *good*; and $A2$ received the grade *very good*.

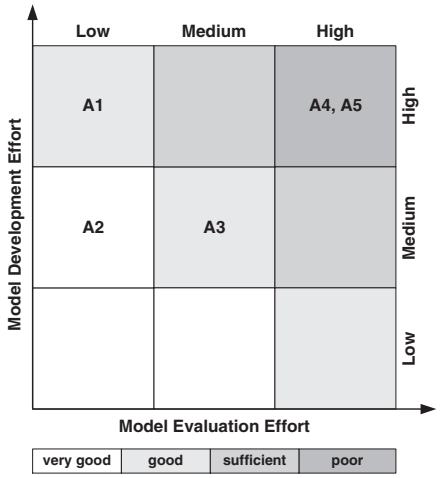

Figure 2. The assessment of approaches based on (a) model development effort and model evaluation effort, (b) accuracy and efficiency.

5 Conclusions

Performance is a key indicator of parallel and distributed computing systems. Therefore, the performance evaluation of computing systems was a preoccupation of many computer scientists in the past. In this paper we have systematically presented the state of the art approaches for performance modeling and prediction of parallel and distributed computing systems. We have discussed the contributions and drawbacks of each approach. In what follows we summarize some of the issues that we have identified.

Most of approaches for the performance modeling of parallel and distributed programs are of limited use to support performance-oriented software engineering because of the following reasons: (1) the use of a notation that is not based on widely accepted standards, and (2) the requirement that the software engineer has a thorough un-

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

derstanding of the underlying performance modeling technique. Some approaches aim to bridge this gap between the performance modeling and the software engineering by incorporating UML.

Most of approaches are able to cope only with small programs such as matrix-vector multiplication. There are several reasons for the lack of scalability: (1) a very complex code analysis is used during the workload modeling that does not scale up to the size and complexity of the real-world programs, (2) a detailed machine model is used that is so slow that makes impractical the simulation of real-world programs, or (3) for the model evaluation are required very large resources (processors and memory) that may not be available. Some approaches have addressed this issue by using model simplification techniques, combination of mathematical modeling with discrete event simulation, and by using a simple machine simulation model.

Performance models that represent the whole program and machine as a symbolic expression lack the structural information. Consequently, it is difficult to identify the part of system that is responsible for the suboptimal performance. Some approaches support the development of performance models at various levels of abstraction. For instance, for workload modeling are used UML activity diagrams. An activity may represent a single instruction, or larger blocks of the program (for instance a *loop*), or the whole program. Furthermore, some approaches use discrete-event simulation to describe the structure of system and the interaction among its components.

In future we plan to supplement our survey by including other approaches for performance modeling and prediction of parallel and distributed computing systems.

References

[1] V. Adve, R. Bagrodia, J. Browne, E. Deelman, A. Dube, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. Teller, and M. Vernon. POEMS: End-to-End Performance Design of Large Parallel Adaptive Computational Systems. *IEEE Transactions on Software Engineering*, 26:1027–1048, November 2000.

[2] V. Adve, R. Bagrodia, E. Deelman, T. Phan, and R. Sakellariou. Compiler-Supported Simulation of Highly Scalable Parallel Applications. In *Proceedings of SC99*, Portland, Oregon, USA, 1999. ACM Press.

[3] R. Bagrodia, S. Docy, and A. Kahn. Parallel Simulation of Parallel File Systems and I/O Programs. In *SC97 (SuperComputing 97)*, San Jose, CA, 1997. ACM.

[4] D. Burger and T. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-1997-1342, Computer Sciences Department, University of Wisconsin-Madison, 1997.

[5] dHPF Compiler. Department of Computer Science, Rice University, http://www.cs.rice.edu/ dsystem/dhpf/.

[6] J. Dongarra. An Overview of High-Performance Computing and Challenges for the Future. http://www.netlib.org/utk/people/JackDongarra/talks.htm, May 9 2006.

[7] The Earth Simulator Center. http://www.es.jamstec.go.jp/esc/eng/.

[8] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Willey, 2000.

[9] S. Graham, M. Snir, and C. Patterson. *Getting Up to Speed: The Future of Supercomputing*. The National Academies Press, 2004.

[10] C. Hughes, V. Pai, P. Ranganathan, and S. Adve. RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors. *IEEE Computer*, 35(2):40–49, February 2002.

[11] IBM Research: Blue Gene Project. http://www.research.ibm.com/bluegene/.

[12] L. Kale and S. Krishnan. *Parallel Programming using C++*, chapter Charm++: Parallel Programming with Message-Driven Objects. MIT Press, 1996.

[13] H. Karatza. *Applied System Simulation: Methodologies and Applications*, chapter Simulation of Parallel and Distributed Systems Scheduling, Concepts, Issues and Approaches. Springer, 2003.

[14] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings. Predictive Performance and Scalability Modeling of a Large-scale Application. In *ACM/IEEE conference on Supercomputing (SC 2001)*, Denver, CO, USA, November 2001. ACM.

[15] D. Kerbyson, A. Hoisie, and H. Wasserman. Use of Predictive Performance Modeling During Large-Scale System Installation. *Parallel Processing Letters*, 15(4), December 2005.

[16] Object Management Group (OMG). UML 2.0 Superstructure Specification. http://www.omg.org, August 2005.

[17] S. Pllana and T. Fahringer. UML Based Modeling of Performance Oriented Parallel and Distributed Applications. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, California, USA, December 2002. IEEE.

[18] S. Pllana and T. Fahringer. PerformanceProphet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs. In *The 2005 International Conference on Parallel Processing (ICPP-05). Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems.*, Oslo, Norway, June 2005. IEEE Computer Society.

[19] Rice Simulator for ILP Multiprocessors (RSIM). http://rsim.cs.uiuc.edu/rsim/.

[20] SPARC International. http://www.sparc.org/.

[21] Simulating the Planet Earth. http://www.nec.com/global/features/index9/index.html.

[22] TOP500 Supercomputer Sites. http://www.top500.org/.

[23] T. Wilmarth, G. Zheng, E. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, and L. Kale. Performance Prediction using Simulation of Large-scale Interconnection Networks in POSE. In *2005 Workshop on Principles of Advanced and Disctributed Simulation (PADS)*, Monterey, California, June 2005. IEEE Computer Society.

[24] G. Zheng, G. Kakulapati, and L. Kale. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, New Mexico, USA, April 2004. IEEE Computer Society.